

# Database Concepts

Ninth Edition



## Chapter 1

### Getting Started

# Learning Objectives (1 of 2)

- Understand the importance of databases in Internet Web applications and mobile apps
- Understand the nature and characteristics of databases
- Understand the potential problems with lists
- Understand the reasons for using a database
- Understand how using related tables helps you avoid the problems of using lists
- Know the components of a database system
- Learn the elements of a database
- Learn the purpose of a database management system (DBMS)
- Understand the functions of a database application

# Learning Objectives (2 of 2)

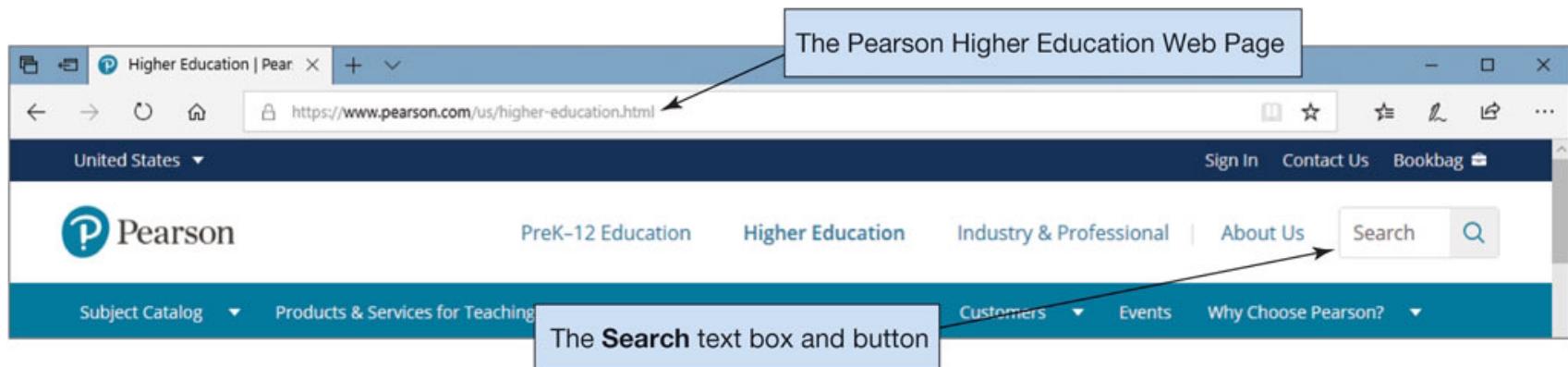
- Introduce Web database applications
- Introduce data warehouses and business intelligence (BI) systems
- Introduce Big Data and cloud computing

# The Importance of Databases in the Internet and Mobile App World

## Understand the importance of databases in Internet Web applications and mobile apps

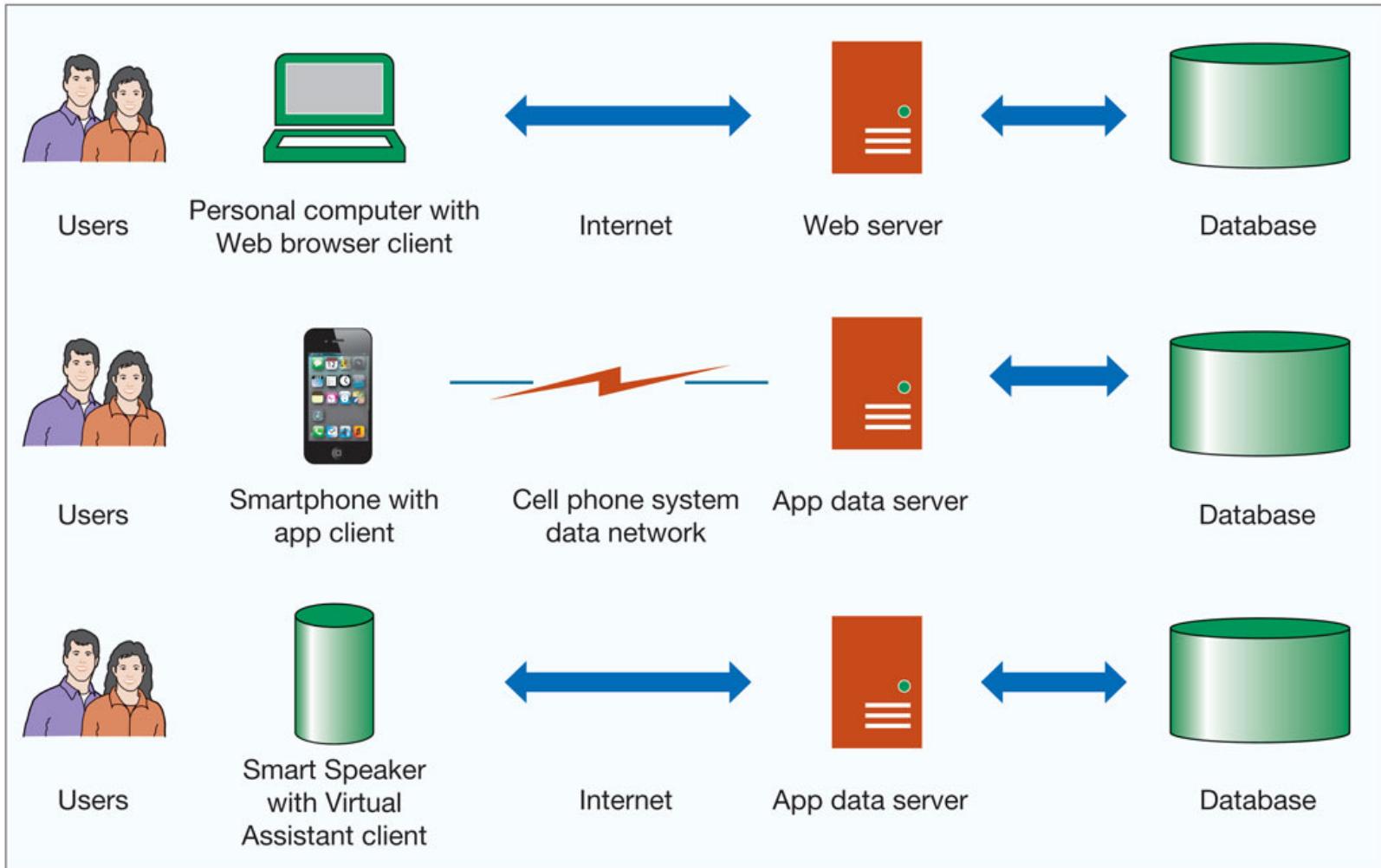
- The Internet was created as the ARPANET in 1969 and grew to connect LANs.
- It became known and used when the World Wide Web (Web) became accessible in 1993.
- In the early 2000s, Web 2.0 sites started to appear allowing users to add content.
- The Internet of Things (IoT) is the latest development which allows all types of devices to connect with each other.
- All of these items depend upon databases.

# Figure 1.1(a) Searching a Database in a Web Browser



Courtesy of Pearson Education

# Figure 1.2 The Internet and Mobile Device World



# Why Use a Database?

## Understand the nature and characteristics of databases

- The reason databases are used is to keep track of things.
- Databases store more complicated information than simple lists like a spreadsheet.

# Figure 1.7 The Student with Advisor and Department List

## Understand the potential problems with lists

- A major problem with using lists include:
  - **Modification problems:** Redundancy and multiple themes can create modification problems such as deleting, updating, and inserting records as seen in the figure below.

A	B	C	D	E	F	G	H	
1	SID	StudentLastName	StudentFirstName	StudentEmail	AdviserLastName	AdviserEmail	Department	AdminLastName
2	S0023	Andrews	Matthew	Matthew.Andrews@ourcampus.edu	Baker	Linda.Baker@ourcampus.edu	Accounting	Smith
3	S0065	Fischer	Douglas	Douglas.Fisher@ourcampus.edu	Baker	Linda.Baker@ourcampus.edu	Accounting	Smith
4	S0083	Hwang	Terry	Terry.Hwang@ourcampus.edu	Taing	Susan.Taing@ourcampus.edu	Accounting	Smith
5	S0132	Thompson	James	James.Thompson@ourcampus.edu	Taing	Susan.Taing@ourcampus.edu	InfoSystems	Rogers
6	S0154	Brisbon	Lisa	Lis.Brisbon@ourcampus.edu	Valdez	Richard.Valdez@ourcampus.edu	Chemistry	Chaplin
7	S0167	Lai	Tzu	Tzu.Lai@ourcampus.edu	Yeats	Bill.Yeats@ourcampus.edu	InfoSystems	Rogers
8	S0212	Marino	Chip	Chip.Marino@ourcampus.edu	Tran	Ken.Tran@ourcampus.edu	InfoSystems	Rogers
9	???	???	???	???	???	???	Biology	Kelly

If Adviser **Baker** is changed to **Taing**, we need to change *AdviserEmail* as well. If changed to **Valdez**, we need to change *AdviserEmail*, *Department*, and *AdminLastName*.

Deleted row—Student and Adviser data lost

Inserted row—both Student and Adviser data missing

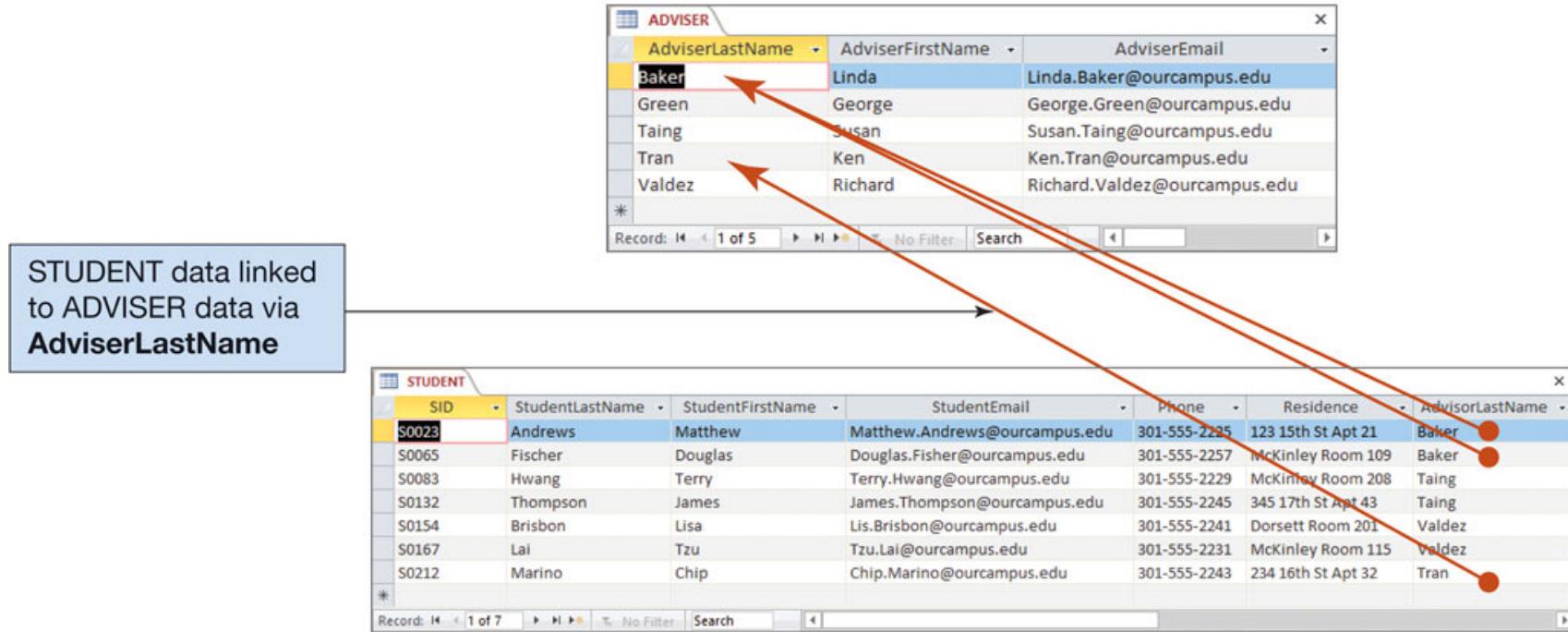
Excel 2019, Windows 10, Microsoft Corporation.

# Using Relational Database Tables

**Understand how using related tables helps you avoid the problems of using lists.**

- **Relational model** is a methodology used as a solution for database design.
- A **relational database** contains a collection of separate tables.
- A **table** holds data about only one theme.
- Each **column**, also known as fields, in a table stores a characteristic common to all rows in a table. An example is StudentNumber.
- A **row** in a table, also known as a record, has data about an occurrence. An example would be all the information on one student.

# Figure 1.8 The Advisor and Student Tables



Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.9 Modifying the Advisor and Student Tables

Changed data—data remains consistent

Inserted data—no STUDENT data required

Deleted data—no ADVISER data lost

AdviserLastName	AdviserFirstName	AdviserEmail
Baker	Linda	Linda.Baker@ourcampus.edu
Green	George	George.Green@ourcampus.edu
Taing	Susan	Sue.Taing@ourcampus.edu
Tran	Ken	Ken.Tran@ourcampus.edu
Valdez	Richard	Richard.Valdez@ourcampus.edu
Yeats	Bill	Bill.Yeats@ourcampus.edu
*		

SID	StudentLastName	StudentFirstName	StudentEmail	Phone	Residence	AdvisorLastName
S0023	Andrews	Matthew	Matthew.Andrews@ourcampus.edu	301-555-2225	123 15th St Apt 21	Baker
S0065	Fischer	Douglas	Douglas.Fisher@ourcampus.edu	301-555-2257	McKinley Room 109	Baker
S0083	Hwang	Terry	Terry.Hwang@ourcampus.edu	301-555-2229	McKinley Room 208	Taing
S0132	Thompson	James	James.Thompson@ourcampus.edu	301-555-2245	345 17th St Apt 43	Taing
S0154	Brisbon	Lisa	Lis.Brisbon@ourcampus.edu	301-555-2241	Dorsett Room 201	Valdez
S0167	Lai	Tzu	Tzu.Lai@ourcampus.edu	301-555-2231	McKinley Room 115	Valdez
S0212	Marino	Chip	Chip.Marino@ourcampus.edu	301-555-2243	234 16th St Apt 32	Tran
*						

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.10 The Department, Advisor, and Student Tables

Can insert DEPARTMENT data as needed—no ADVISER or STUDENT data required

DepartmentName	DepartmentPhone	AdminLastName	AdminFirstName	AdminEmail
Accounting	301-557-1011	Smith	Shawna	Shawna.Smith@ourcampus.edu
Biology	301-557-1021	Kelly	Chris	Chris.Kelly@ourcampus.edu
Chemistry	301-557-1031	Chaplin	Robin	Robin.Chaplin@ourcampus.edu
InfoSystems	301-557-1041	Rogers	Aaron	Aaron.Rogers@ourcampus.edu

Can change STUDENT Adviser name as needed—new value is linked to its own data

AdviserLastName	AdviserFirstName	AdviserEmail	Department
Baker	Linda	Linda.Baker@ourcampus.edu	Accounting
Green	George	George.Green@ourcampus.edu	Biology
Taing	Susan	Sue.Taing@ourcampus.edu	Accounting
Tran	Ken	Ken.Tran@ourcampus.edu	InfoSystems
Valdez	Richard	Richard.Valdez@ourcampus.edu	Chemistry
Yeats	Bill	Bill.Yeats@ourcampus.edu	InfoSystems

Can delete STUDENT data as needed—no DEPARTMENT or ADVISER data lost

SID	StudentLastName	StudentFirstName	StudentEmail	Phone	Residence	AdvisorLastName
S0023	Andrews	Matthew	Matthew.Andrews@ourcampus.edu	301-555-2225	123 15th St Apt 21	Baker
S0065	Fischer	Douglas	Douglas.Fisher@ourcampus.edu	301-555-2257	McKinley Room 109	Baker
S0083	Hwang	Terry	Terry.Hwang@ourcampus.edu	301-555-2229	McKinley Room 208	Taing
S0132	Thompson	James	James.Thompson@ourcampus.edu	301-555-2245	345 17th St Apt 43	Taing
S0154	Brisbon	Lisa	Lis.Brisbon@ourcampus.edu	301-555-2241	Dorsett Room 201	Valdez
S0167	Lai	Tzu	Tzu.Lai@ourcampus.edu	301-555-2231	McKinley Room 115	Valdez
S0212	Marino	Chip	Chip.Marino@ourcampus.edu	301-555-2243	234 16th St Apt 32	Tran

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.11 The Art Course List with Modification Problems

	A	B	C	D	E	F	G
1	CustomerLastName	CustomerFirstName	Phone	CourseDate	AmountPaid	Course	Fee
2	Johnson	Ariel	206-567-1234	10/1/2019	\$250.00	Adv Pastels	\$500.00
3	Green	Robin	425-678-8765	9/15/2019	\$350.00	Beg Oils	\$350.00
4	Jackson	Charles	360-789-3456	10/1/2019	\$500.00	Adv Pastels	\$500.00
5	Johnson	Ariel	206-567-1234	3/15/2019	\$350.00	Int Pastels	\$350.00
6	Pearson	Jeffery	206-567-2345	10/1/2019	\$500.00	Adv Pastels	\$500.00
7	Sears	Miguel	360-789-4567	9/15/2019	\$350.00	Beg Oils	\$350.00
8	Kyle	Leah	425-678-7654	11/15/2019	\$250.00	Adv Pastels	\$500.00
9	Myers	Lynda	360-789-5678	10/15/2019	\$0.00	Beg Oils	\$350.00

How to enter the fee for a new course?

Consequences of changing this date?

Consequences of deleting this row?

```
graph LR; A[How to enter the fee for a new course?] --> G9[Fee]; B[Consequences of changing this date?] --> D5[CourseDate]; C[Consequences of deleting this row?] --> R4[Row 4];
```

Excel 2019, Windows 10, Microsoft Corporation.

# Figure 1.12 The Art Course Database Tables

The figure displays three Microsoft Access database tables: CUSTOMER, COURSE, and ENROLLMENT. Red arrows point from three callout boxes containing notes to specific fields in the tables.

- CUSTOMER Table:** Shows customer information with fields CustomerNumber, CustomerLastName, CustomerFirstName, and Phone. A red arrow points to the CustomerNumber field.
- COURSE Table:** Shows course offerings with fields CourseNumber, Course, CourseDate, and Fee. A red arrow points to the CourseNumber field.
- ENROLLMENT Table:** Shows enrollment details with fields CustomerNumber, CourseNumber, and AmountPaid. Two red arrows point to the CustomerNumber and CourseNumber fields.

**Annotations:**

- Can change COURSE CourseDate without problems
- Can insert new COURSE data as needed
- Can delete ENROLLMENT rows as needed—no adverse consequences

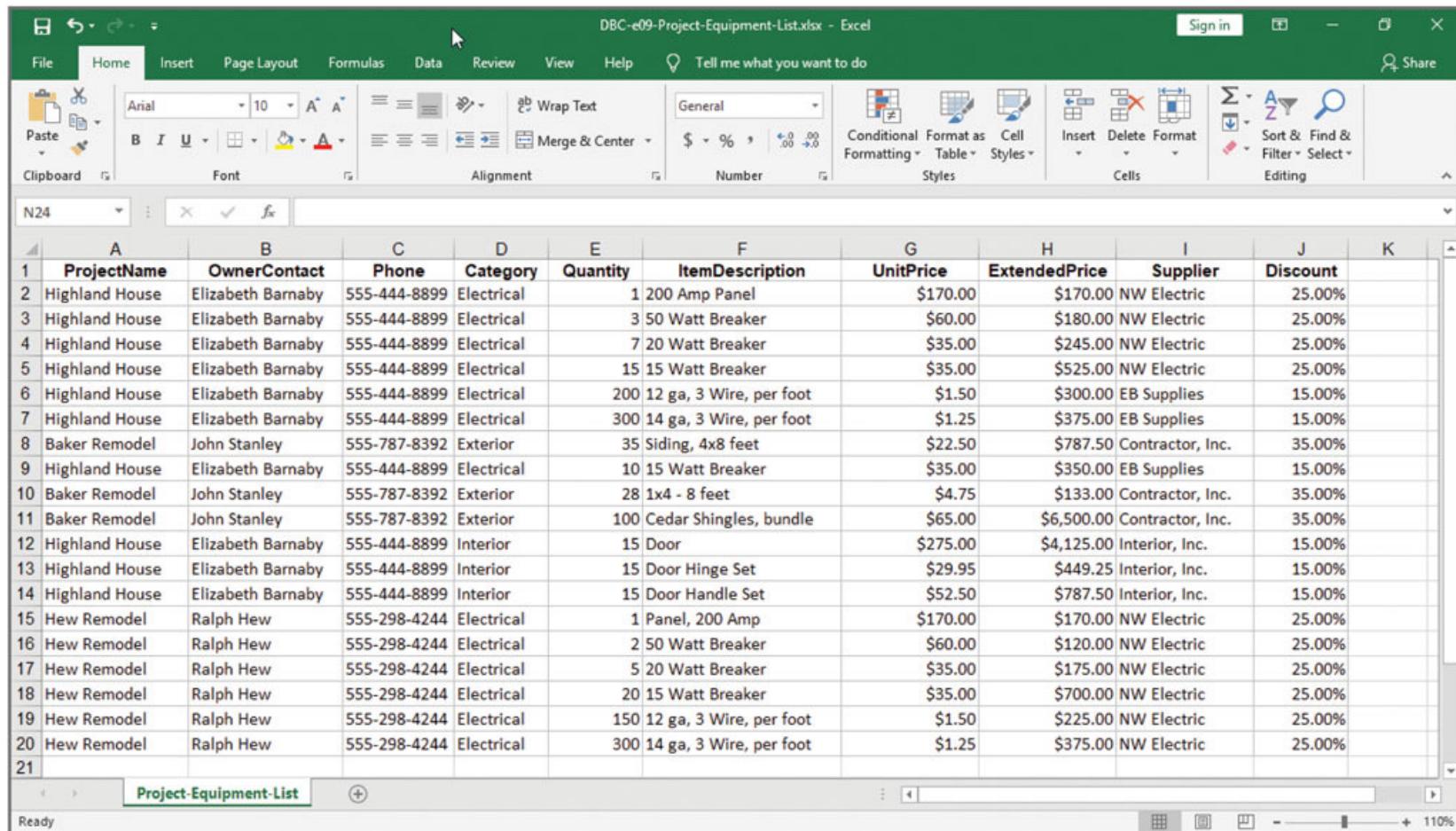
CustomerNumber	CustomerLastName	CustomerFirstName	Phone
1	Johnson	Ariel	206-567-1234
2	Green	Robin	425-678-8765
3	Jackson	Charles	360-789-3456
4	Pearson	Jeffery	206-567-2345
5	Sears	Miguel	360-789-4567
6	Kyle	Leah	425-678-7654
7	Myers	Lynda	360-789-5678

CourseNumber	Course	CourseDate	Fee
1	Adv Pastels	10/1/2019	\$500.00
2	Beg Oils	9/15/2019	\$350.00
3	Int Pastels	3/15/2019	\$350.00
4	Beg Oils	10/15/2019	\$350.00
5	Adv Pastels	11/15/2019	\$500.00

CustomerNumber	CourseNumber	AmountPaid
1	1	\$250.00
1	3	\$350.00
2	2	\$350.00
3	1	\$500.00
4	1	\$500.00
5	2	\$350.00
6	5	\$250.00
7	4	\$0.00
0	0	\$0.00

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.13 The Project Equipment List as a Spreadsheet



The screenshot shows a Microsoft Excel 2019 window titled "DBC-e09-Project-Equipment-List.xlsx - Excel". The ribbon is visible at the top with tabs for File, Home, Insert, Page Layout, Formulas, Data, Review, View, and Help. The Home tab is selected. The Excel interface includes toolbars for Clipboard, Font, Alignment, Number, Styles, Cells, and Editing. The main area displays a data table with 21 rows and 11 columns. The columns are labeled A through K. The data includes project names like Highland House and Baker Remodel, owner contacts, phone numbers, categories (Electrical, Exterior, Interior), quantities, item descriptions, unit prices, extended prices, suppliers, and discounts. The table is styled with bold headers and some merged cells.

	A	B	C	D	E	F	G	H	I	J	K
1	ProjectName	OwnerContact	Phone	Category	Quantity	ItemDescription	UnitPrice	ExtendedPrice	Supplier	Discount	
2	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	1	200 Amp Panel	\$170.00	\$170.00	NW Electric	25.00%	
3	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	3	50 Watt Breaker	\$60.00	\$180.00	NW Electric	25.00%	
4	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	7	20 Watt Breaker	\$35.00	\$245.00	NW Electric	25.00%	
5	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	15	15 Watt Breaker	\$35.00	\$525.00	NW Electric	25.00%	
6	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	200	12 ga, 3 Wire, per foot	\$1.50	\$300.00	EB Supplies	15.00%	
7	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	300	14 ga, 3 Wire, per foot	\$1.25	\$375.00	EB Supplies	15.00%	
8	Baker Remodel	John Stanley	555-787-8392	Exterior	35	Siding, 4x8 feet	\$22.50	\$787.50	Contractor, Inc.	35.00%	
9	Highland House	Elizabeth Barnaby	555-444-8899	Electrical	10	15 Watt Breaker	\$35.00	\$350.00	EB Supplies	15.00%	
10	Baker Remodel	John Stanley	555-787-8392	Exterior	28	1x4 - 8 feet	\$4.75	\$133.00	Contractor, Inc.	35.00%	
11	Baker Remodel	John Stanley	555-787-8392	Exterior	100	Cedar Shingles, bundle	\$65.00	\$6,500.00	Contractor, Inc.	35.00%	
12	Highland House	Elizabeth Barnaby	555-444-8899	Interior	15	Door	\$275.00	\$4,125.00	Interior, Inc.	15.00%	
13	Highland House	Elizabeth Barnaby	555-444-8899	Interior	15	Door Hinge Set	\$29.95	\$449.25	Interior, Inc.	15.00%	
14	Highland House	Elizabeth Barnaby	555-444-8899	Interior	15	Door Handle Set	\$52.50	\$787.50	Interior, Inc.	15.00%	
15	Hew Remodel	Ralph Hew	555-298-4244	Electrical	1	Panel, 200 Amp	\$170.00	\$170.00	NW Electric	25.00%	
16	Hew Remodel	Ralph Hew	555-298-4244	Electrical	2	50 Watt Breaker	\$60.00	\$120.00	NW Electric	25.00%	
17	Hew Remodel	Ralph Hew	555-298-4244	Electrical	5	20 Watt Breaker	\$35.00	\$175.00	NW Electric	25.00%	
18	Hew Remodel	Ralph Hew	555-298-4244	Electrical	20	15 Watt Breaker	\$35.00	\$700.00	NW Electric	25.00%	
19	Hew Remodel	Ralph Hew	555-298-4244	Electrical	150	12 ga, 3 Wire, per foot	\$1.50	\$225.00	NW Electric	25.00%	
20	Hew Remodel	Ralph Hew	555-298-4244	Electrical	300	14 ga, 3 Wire, per foot	\$1.25	\$375.00	NW Electric	25.00%	
21											

Excel 2019, Windows 10, Microsoft Corporation.

# Figure 1.14 The Project Equipment Database Tables

The figure displays four Microsoft Access database tables:

- PROJECT**: Stores project information. Primary key is ProjectID.
- ITEM**: Stores item details. Primary key is ItemNumber.
- QUOTE**: Stores quote details. Primary key is QuoteID. It has a many-to-one relationship with the PROJECT table based on ProjectID, and a many-to-one relationship with the ITEM table based on ItemNumber.
- SUPPLIER**: Stores supplier information. Primary key is SupplierID.

Relationships shown by red arrows:

- A red arrow points from the ProjectID column in the PROJECT table to the ProjectID column in the QUOTE table, indicating a one-to-many relationship.
- A red arrow points from the ItemNumber column in the ITEM table to the ItemNumber column in the QUOTE table, indicating a many-to-one relationship.
- A red arrow points from the SupplierID column in the QUOTE table to the SupplierID column in the SUPPLIER table, indicating a many-to-one relationship.

**PROJECT** Table Data:

ProjectID	ProjectName	OwnerContact	Phone
1	Highland House	Elizabeth Barnaby	555-444-8899
2	Baker Remodel	John Stanley	555-787-8392
3	Hew Remodel	Ralph Hew	555-298-4244

**ITEM** Table Data:

ItemNumber	ItemDescription	Category
1100	200 Amp Panel	Electrical
1200	50 Watt Breaker	Electrical
1300	20 Watt Breaker	Electrical
1400	15 Watt Breaker	Electrical
1500	12 ga, 3 Wire, per foot	Electrical
1550	14 ga, 3 Wire, per foot	Electrical
1600	Siding, 4x8 feet	Exterior
1700	1x4 - 8 feet	Exterior
1800	Cedar Shingles, bundle	Exterior
2000	Door	Interior
2100	Door Hinge Set	Interior
2200	Door Handle Set	Interior

**QUOTE** Table Data:

QuoteID	ProjectID	ItemNumber	SupplierID	Quantity	UnitPrice	ExtendedPrice
1	1	1100	1	1	\$170.00	\$170.00
2	1	1200	1	3	\$60.00	\$180.00
3	1	1300	1	7	\$35.00	\$245.00
4	1	1400	1	15	\$35.00	\$525.00
5	1	1500	2	200	\$1.50	\$300.00
6	1	1500	2	300	\$1.25	\$375.00
7	2	1600	3	35	\$22.50	\$787.50
8	1	1400	2	10	\$35.00	\$350.00
9	2	1700	3	28	\$4.75	\$133.00
10	2	1800	3	100	\$65.00	\$6,500.00
11	1	2000	4	15	\$275.00	\$4,125.00
12	1	2100	4	15	\$29.95	\$449.25
13	1	2200	4	15	\$52.50	\$787.50
14	3	1100	1	1	\$170.00	\$170.00
15	3	1200	1	2	\$60.00	\$120.00
16	3	1300	1	5	\$35.00	\$175.00
17	3	1400	1	20	\$35.00	\$700.00
18	3	1500	1	150	\$1.50	\$225.00
19	3	1550	1	300	\$1.25	\$375.00

**SUPPLIER** Table Data:

SupplierID	Supplier	Discount
1	NW Electric	25.00%
2	EB Supplies	15.00%
3	Contractor, Inc.	35.00%
4	Interior, Inc.	15.00%

Access 2019, Windows 10, Microsoft Corporation.

# How do I Process Relational Tables Using SQL?

- The leading technique for data definition and manipulation is **Structured Query Language (SQL)**.
- **SQL** is an international standard for creating, processing, and querying databases and their tables.
- Using **SQL** you can:
  - Reconstruct lists from their underlying tables,
  - Query for specific data conditions,
  - Perform calculations on data in tables, and
  - Insert, update, and delete data.

# SQL Art Course Database Example

**Understand how using related tables helps you avoid the problems of using lists**

- Using SQL, the following code will combine the three tables in the Art Course Database as seen in Figure 1.12:

```
SELECT      CUSTOMER.CustomerLastName,  
            CUSTOMER.CustomerFirstName, CUSTOMER.Phone,  
            COURSE.CourseDate, ENROLLMENT.AmountPaid, COURSE.Course,  
            COURSE.Fee  
FROM        CUSTOMER, ENROLLMENT, COURSE  
WHERE       CUSTOMER.CustomerNumber = ENROLLMENT.CustomerNumber  
AND         COURSE.CourseNumber = ENROLLMENT.CourseNumber;
```

- The results of running the code above can be seen in Figure 1.15 on the next page.

# Figure 1.15 Results of the SQL Query to Recreate the Art Course List Data

Art Course List							
CustomerLastName	CustomerFirstName	Phone	CourseDate	AmountPaid	Course	Fee	
Johnson	Ariel	206-567-1234	10/1/2019	\$250.00	Adv Pastels	\$500.00	
Johnson	Ariel	206-567-1234	3/15/2019	\$350.00	Int Pastels	\$350.00	
Green	Robin	425-678-8765	9/15/2019	\$350.00	Beg Oils	\$350.00	
Jackson	Charles	360-789-3456	10/1/2019	\$500.00	Adv Pastels	\$500.00	
Pearson	Jeffery	206-567-2345	10/1/2019	\$500.00	Adv Pastels	\$500.00	
Sears	Miguel	360-789-4567	9/15/2019	\$350.00	Beg Oils	\$350.00	
Kyle	Leah	425-678-7654	11/15/2019	\$250.00	Adv Pastels	\$500.00	
Myers	Lynda	360-789-5678	10/15/2019	\$0.00	Beg Oils	\$350.00	

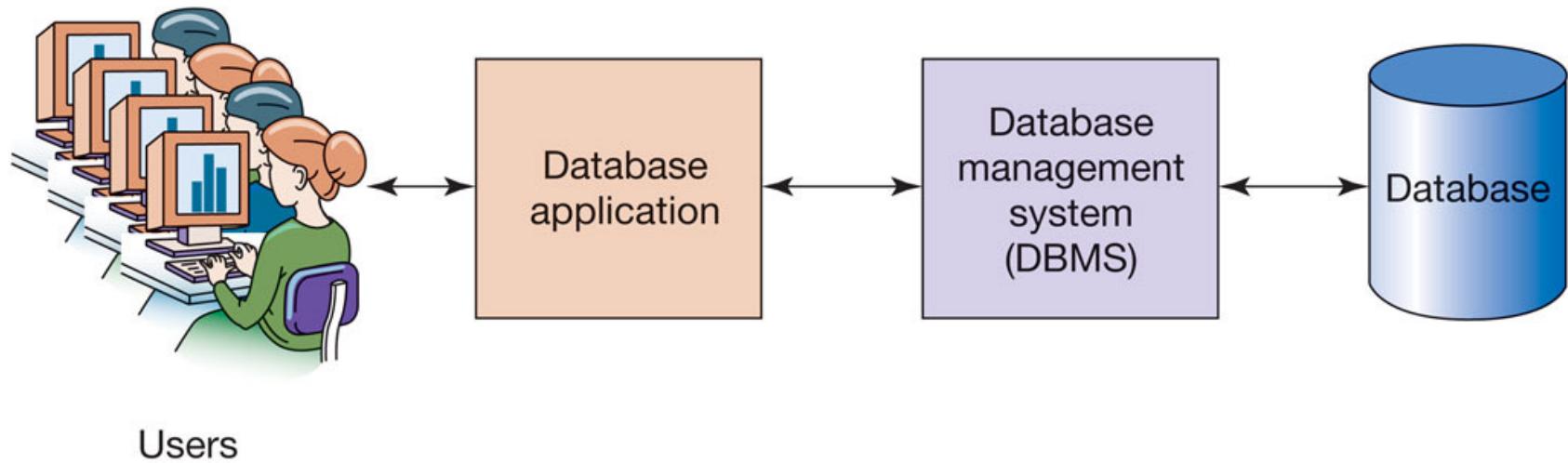
Access 2019, Windows 10, Microsoft Corporation

# What is a Database System?

## Know the components of a database system

- A **database system** has four components consisting of:
  1. Users,
  2. Database application
  3. Database management system (DBMS), and
  4. Database.

# Figure 1.17 Components of a Database System

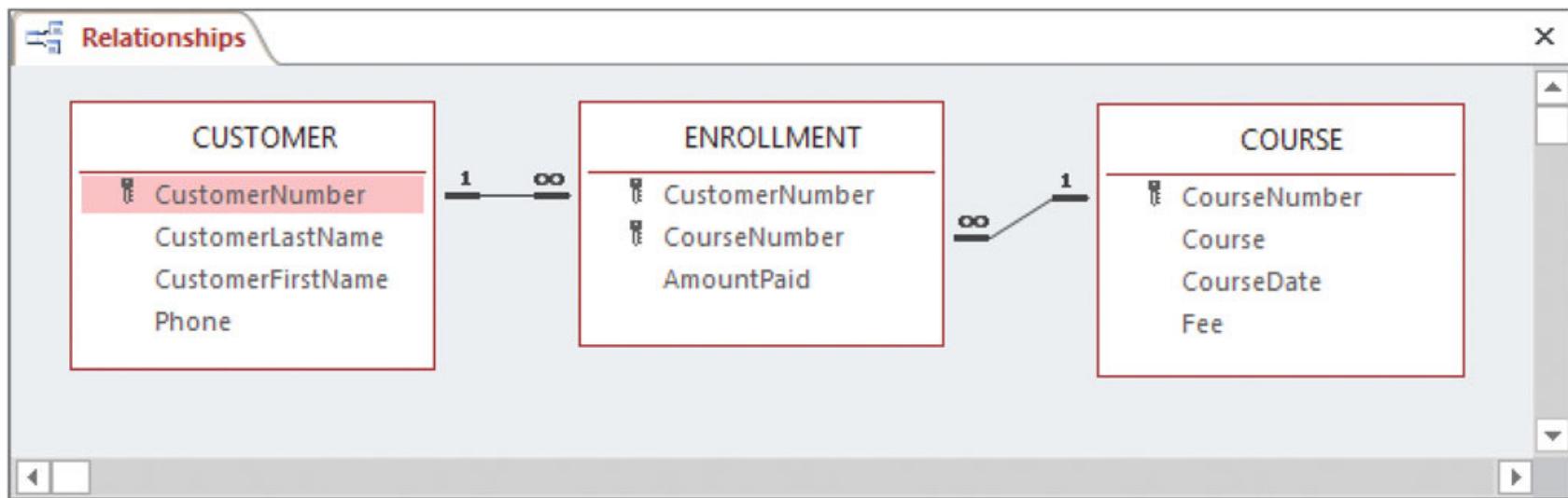


# The Database

## Know the components of a database system

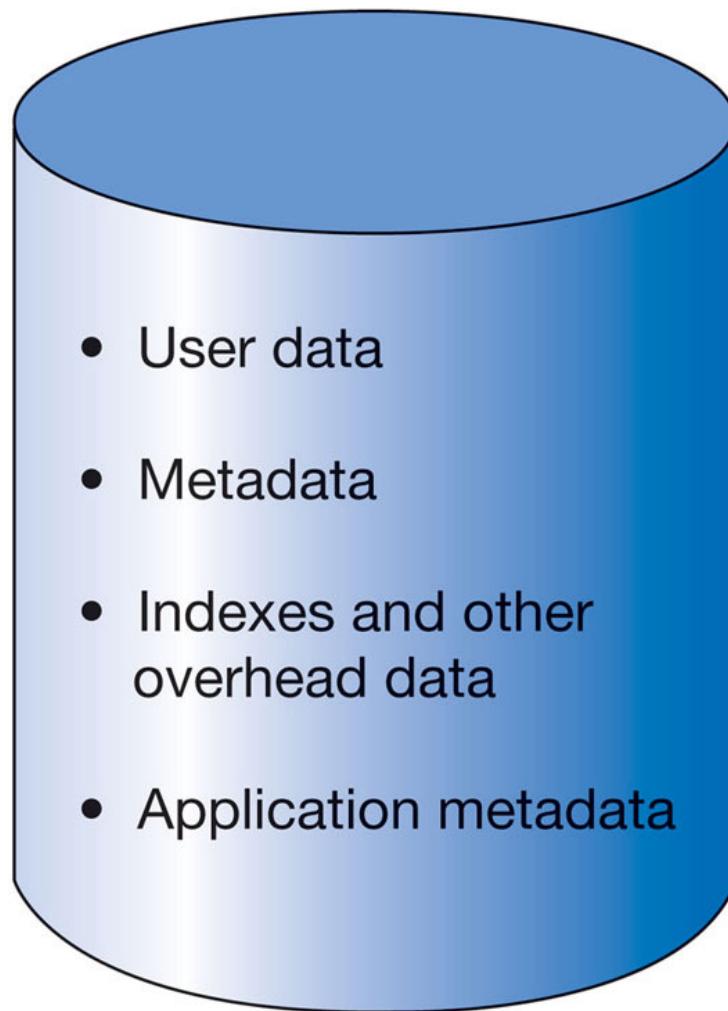
- A **user** of a database system will:
  - Employ a database application to keep track of things
  - Use forms to read, enter, and query data
  - Produce reports
- A **database** is a self-describing collection of related tables:
  - **Self-describing** means a description of the structure of the database is contained with the database itself.
  - **Metadata** is data about the structure of the database.

# Figure 1.18 Example Metadata: A Relationship Diagram for the Art Course Tables in Figure 1.12



Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.19 Database Contents



# Figure 1.20 Functions of a DBMS

## Learn to purpose of a database management system (DBMS)

- The purpose of a DBMS is to create, process, and administer databases and are licensed from a vendor.
- The functions of a DBMS are shown in the following slide:

- Create database
- Create tables
- Create supporting structures (e.g., indexes)
- Read database data
- Modify (insert, update, or delete) database data
- Maintain database structures
- Enforce rules
- Control concurrency
- Provide security
- Perform backup and recovery

# Referential Integrity Constraints

## Learn the purpose of a database management system (DBMS)

- **Referential integrity constraints** are rules enforced by the DBMS to ensure values of a column in one table are valid when compared to values in another table.
  - For example, in the Art Course database what would happen if a user mistakenly entered 9 for CustomerNumber in the ENROLLMENT table?
    - Since 9 does not exist in that table it would cause errors and not execute.
    - To prevent this situation, the DBMS enforces the rule that if a CustomerNumber is entered in the ENROLLMENT table, must also exist in the CUSTOMER table.

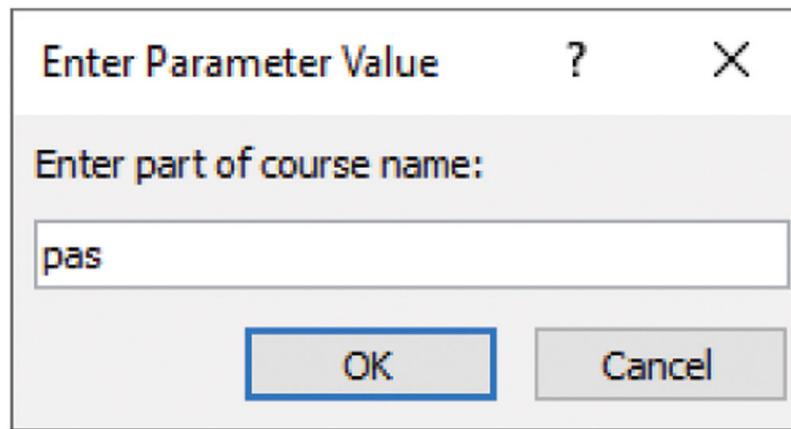
# Figure 1.21 Functions of Database Application Programs

- Create and process forms
- Process user queries
- Create and process reports
- Execute application logic
- Control application

# Figure 1.22 Example Data Entry Form

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.23 (a) Query Parameter Form, and (b) Query Results



Course Parameter Query						
CustomerLastName	CustomerFirstName	Course	CourseDate	Fee	AmountPaid	Amount Due
Jackson	Charles	Adv Pastels	10/1/2019	\$500.00	\$500.00	\$0.00
Johnson	Ariel	Int Pastels	3/15/2019	\$350.00	\$350.00	\$0.00
Johnson	Ariel	Adv Pastels	10/1/2019	\$500.00	\$250.00	\$250.00
Kyle	Leah	Adv Pastels	11/15/2019	\$500.00	\$250.00	\$250.00
Pearson	Jeffery	Adv Pastels	10/1/2019	\$500.00	\$500.00	\$0.00
*						

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.24 Example Report

## Course Enrollment Report

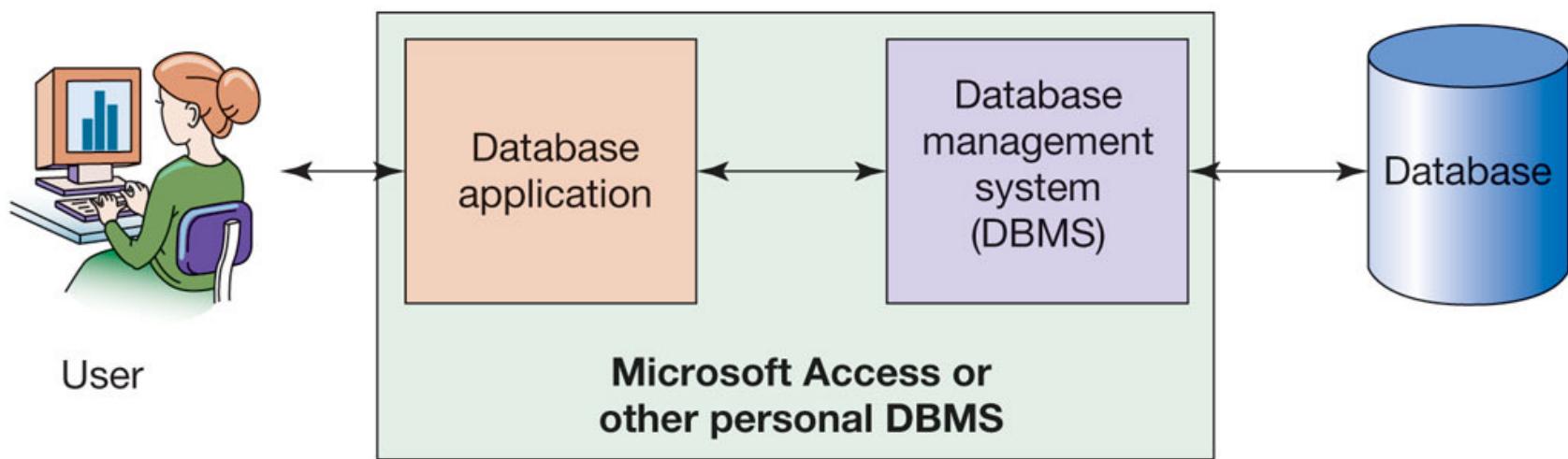
Course	CourseDate	CustomerLastName	CustomerFirstName	Phone	Fee	AmountPaid	AmountDue
<b>Adv Pastels</b>							
	10/1/2019	Jackson	Charles	360-789-3456	\$500.00	\$500.00	\$0.00
		Johnson	Ariel	206-567-1234	\$500.00	\$250.00	\$250.00
		Pearson	Jeffery	206-567-2345	\$500.00	\$500.00	\$0.00
<b>Beg Oils</b>							
	11/15/2019	Kyle	Leah	425-678-7654	\$500.00	\$250.00	\$250.00
<b>Int Pastels</b>							
	9/15/2019	Green	Robin	425-678-8765	\$350.00	\$350.00	\$0.00
		Sears	Miguel	360-789-4567	\$350.00	\$350.00	\$0.00
	10/15/2019	Myers	Lynda	360-789-5678	\$350.00	\$0.00	\$350.00
	3/15/2019	Johnson	Ariel	206-567-1234	\$350.00	\$350.00	\$0.00

Access 2019, Windows 10, Microsoft Corporation.

# Personal vs. Enterprise-Class Database Systems

- A **personal database system** is used by only one person and would include:
  - Only a few tables containing only a few hundred rows of data
  - Use only one computer with one user at a time
- An **enterprise database system is used international organizations with thousands** of concurrent users and would include:
  - Hundreds of tables with millions of rows of data
  - In use 24/7 24 hours a day

# Figure 1.25 Personal Database System



# Figure 1.26 SQL Generated by Microsoft Access Query

The SQL has  
been arranged to  
make it easy to read

**Art Course List**

```
SELECT CUSTOMER.CustomerLastName,  
       CUSTOMER.CustomerFirstName,  
       CUSTOMER.Phone,  
       COURSE.CourseDate,  
       ENROLLMENT.AmountPaid,  
       COURSE.Course,  
       COURSE.Fee  
  FROM CUSTOMER, ENROLLMENT, COURSE  
 WHERE (((CUSTOMER.CustomerNumber)=[ENROLLMENT].[CustomerNumber]))  
   AND ((COURSE.CourseNumber)=[ENROLLMENT].[CourseNumber]));
```

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.27 Microsoft Access 2019

The database name **Art-Course-Database**

The table object **CUSTOMER** is displayed under the **Tables** section of All Access Objects

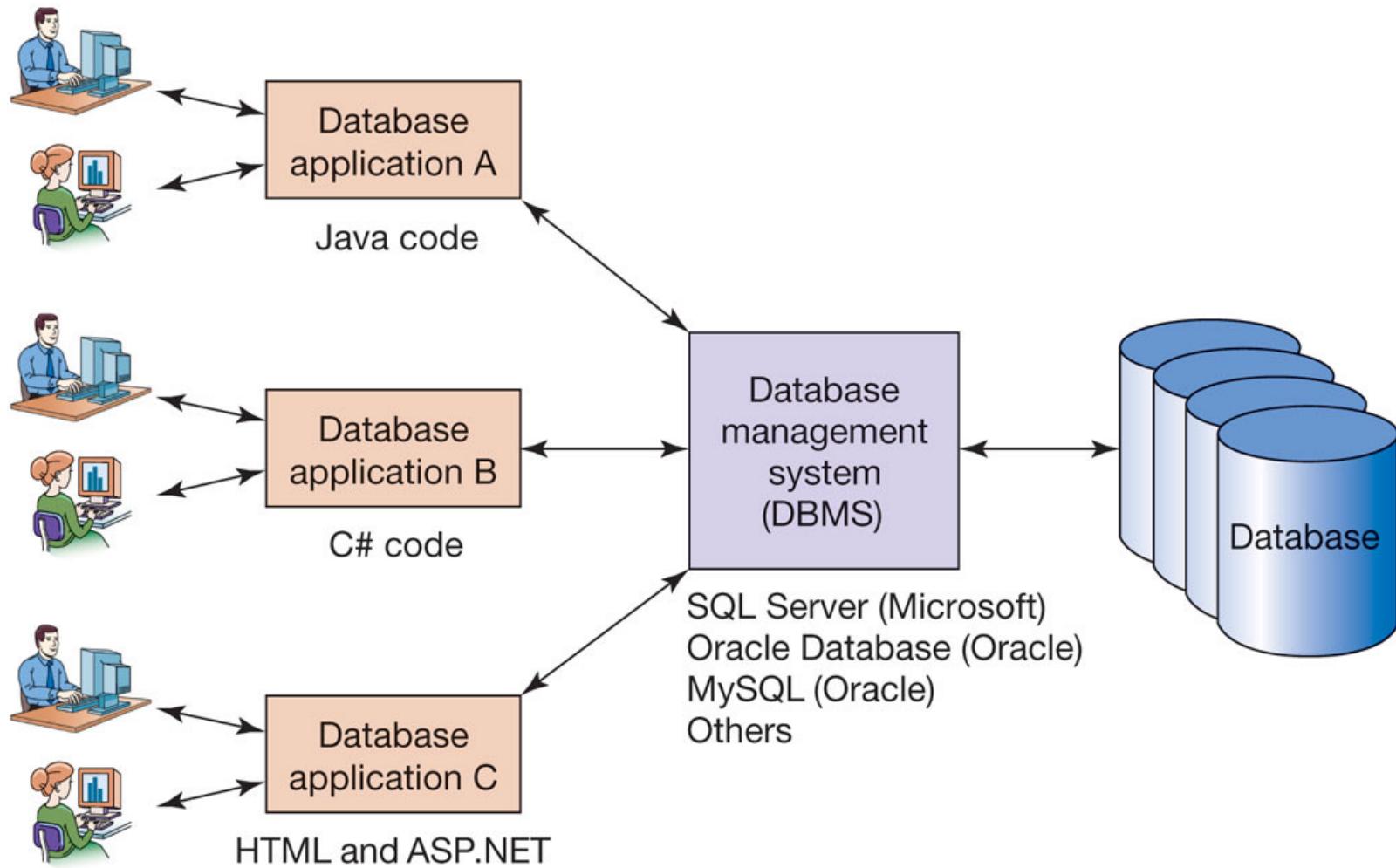
The query object **Art Course List** stores the query itself

The query results in table format

CustomerLastName	CustomerFirstName	Phone	CourseDate	AmountPaid	Course	Fee
Johnson	Ariel	206-567-1234	10/1/2019	\$250.00	Adv Pastels	\$500.00
Johnson	Ariel	206-567-1234	3/15/2019	\$350.00	Int Pastels	\$350.00
Green	Robin	425-678-8765	9/15/2019	\$350.00	Beg Oils	\$350.00
Jackson	Charles	360-789-3456	10/1/2019	\$500.00	Adv Pastels	\$500.00
Pearson	Jeffery	206-567-2345	10/1/2019	\$500.00	Adv Pastels	\$500.00
Sears	Miguel	360-789-4567	9/15/2019	\$350.00	Beg Oils	\$350.00
Kyle	Leah	425-678-7654	11/15/2019	\$250.00	Adv Pastels	\$500.00
Myers	Lynda	360-789-5678	10/15/2019	\$0.00	Beg Oils	\$350.00

Access 2019, Windows 10, Microsoft Corporation.

# Figure 1.28 Enterprise-Class Database System



# Figure 1.29 MySQL 8.0

The screenshot shows the MySQL Workbench interface with several annotations:

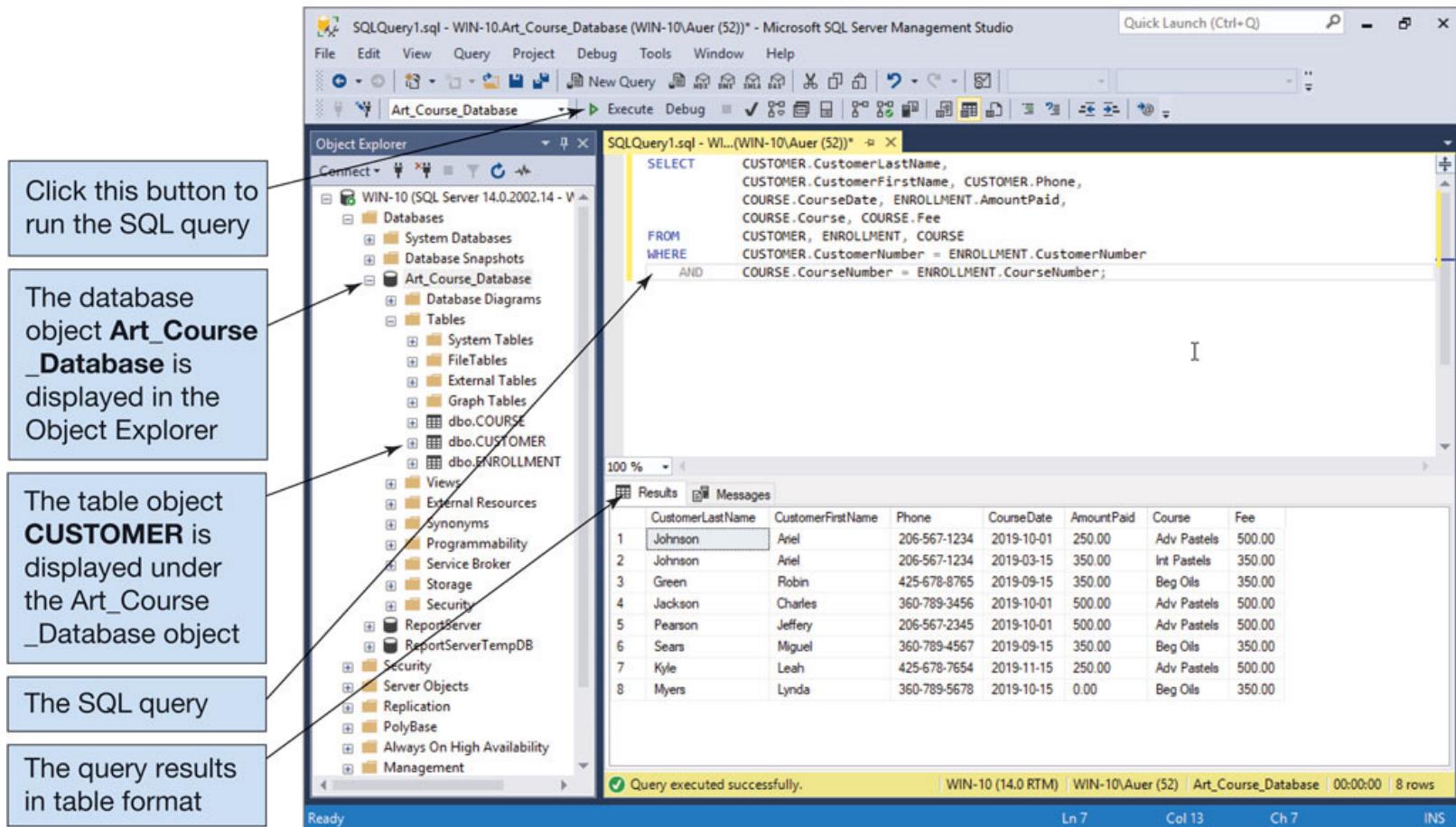
- Click this button to run the SQL query**: Points to the "Run" button in the toolbar.
- The SQL query**: Points to the SQL editor area containing the following query:

```
SELECT CUSTOMER.CustomerLastName,
       CUSTOMER.CustomerFirstName, CUSTOMER.Phone,
       COURSE.CourseDate, ENROLLMENT.AmountPaid,
       COURSE.Course, COURSE.Fee
  FROM CUSTOMER, ENROLLMENT, COURSE
 WHERE CUSTOMER.CustomerNumber=ENROLLMENT.CustomerNumber
   AND COURSE.CourseNumber=ENROLLMENT.CourseNumber;
```
- The database object art\_course\_database is displayed in the Object Browser**: Points to the "art\_course\_database" entry in the Object Browser under the "Tables" category.
- The table object CUSTOMER is displayed under the art\_course\_database object**: Points to the "CUSTOMER" table entry under "art\_course\_database" in the Object Browser.
- The query results in table format**: Points to the "Result Grid" showing the query results:

	CustomerLastName	CustomerFirstName	Phone	CourseDate	AmountPaid	Course	Fee
1	Johnson	Ariel	206-567-1234	2019-10-01	250.00	Adv Pastels	500.00
2	Jackson	Charles	360-789-3456	2019-10-01	500.00	Adv Pastels	500.00
3	Pearson	Jeffery	206-567-2345	2019-10-01	500.00	Adv Pastels	500.00
4	Green	Robin	425-678-8765	2019-09-15	350.00	Beg Oils	350.00
5	Sears	Miguel	360-789-4567	2019-09-15	350.00	Beg Oils	350.00
6	Johnson	Ariel	206-567-1234	2019-03-15	350.00	Int Pastels	350.00
7	Myers	Lynda	360-789-5678	2019-10-15	0.00	Beg Oils	350.00
8	Kyle	Leah	425-678-7654	2019-11-15	250.00	Adv Pastels	500.00

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Figure 1.30 Microsoft SQL Server 2017



SQL Server 2017, SQL Server Management Studio, Microsoft Corporation.

# Figure 1.31 Oracle DatabaseXE

The database object **Art\_Course\_Database** is displayed in the Oracle Connections browser

Click this button to run the SQL query

The table object **CUSTOMER** is displayed in the Tables objects

The SQL query

The query results in table format

The screenshot shows the Oracle SQL Developer interface. The 'Connections' browser on the left lists the 'Art\_Course\_Database'. Under 'Tables (Filtered)', the 'CUSTOMER' table is selected. The 'Worksheet' tab in the center contains an SQL query:

```
SELECT CUSTOMER.CustomerLastName,
       CUSTOMER.CustomerFirstName, CUSTOMER.Phone,
       COURSE.CourseDate, ENROLLMENT.AmountPaid,
       COURSE.Course, COURSE.Fee
  FROM CUSTOMER, ENROLLMENT, COURSE
 WHERE CUSTOMER.CustomerNumber=ENROLLMENT.CustomerNumber
   AND COURSE.CourseNumber=ENROLLMENT.CourseNumber;
```

The 'Query Result' tab at the bottom displays the execution results in a table format:

	CUSTOMERLASTNAME	CUSTOMERFIRSTNAME	PHONE	COURSEDATE	AMOUNTPAID	COURSE	Fee
1	Johnson	Ariel	206-567-1234	15-MAR-19	350	Int Pastels	350
2	Johnson	Ariel	206-567-1234	01-OCT-19	250	Adv Pastels	500
3	Green	Robin	425-678-8765	15-SEP-19	350	Beg Oils	350
4	Jackson	Charles	360-789-3456	01-OCT-19	500	Adv Pastels	500
5	Pearson	Jeffery	206-567-2345	01-OCT-19	500	Adv Pastels	500
6	Sears	Miguel	360-789-4567	15-SEP-19	350	Beg Oils	350
7	Kyle	Leah	425-678-7654	15-NOV-19	250	Adv Pastels	500
8	Myers	Lynda	360-789-5678	15-OCT-19	0	Beg Oils	350

Annotations with arrows point from the callout boxes to specific elements: one arrow points from the 'CUSTOMER' entry in the 'Tables' list to the 'CUSTOMER' table icon; another arrow points from the 'Run' button in the toolbar to the 'AND' clause in the SQL query; and a third arrow points from the 'CustomerNumber' entry in the 'Tables' list to the 'CustomerNumber' column in the 'Query Result' table.

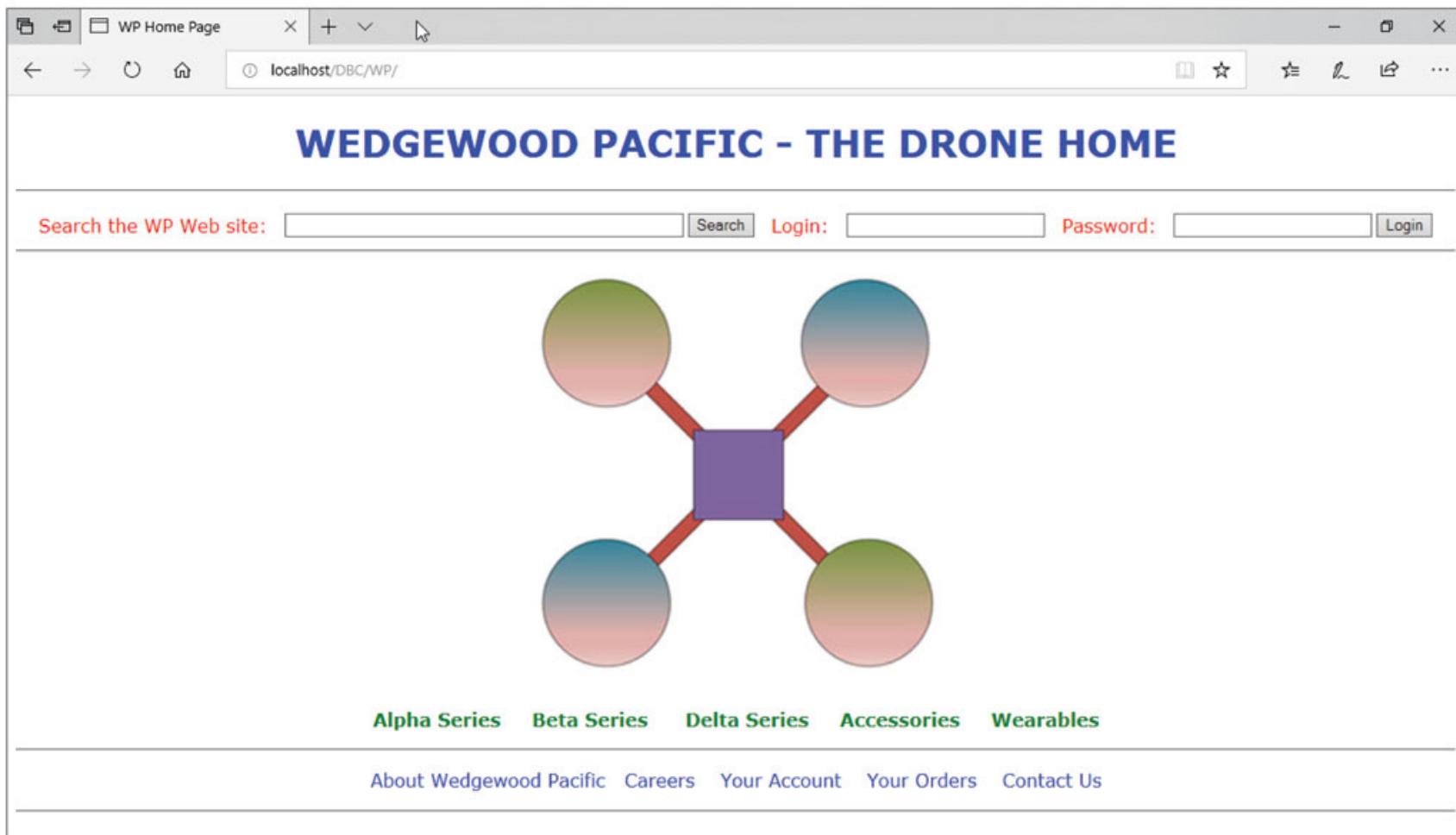
Oracle DatabaseXE, SQL Developer 18.4, Oracle Corporation.

# What is a Web Database Application?

## Introduce Web database applications

- A **Web database application** is an application with a Web user interface that is dependent on a database to store the data needed by the application (see Figure 1.32).
- An application programming interface (API) is a programming language such as PHP or JavaScript to connect to a DBMS allowing the sending of SQL commands to the DBMS and then to receive them back.

# Figure 1.32 The WP Web Page



Microsoft Edge, Windows 10, Microsoft Corporation.

# What are Data Warehouses and Business Intelligence (BI) Systems?

## Introduce data warehouses and business intelligence (BI) systems

- **Transactions** are purchases bought online that are recorded in a company's database, also referred to as an **online transaction processing (OLTP)** database.
- Data analysis is done on an organization's **online analytical processing (OLAP)** database and is used for research.
- A **business intelligence system** consists of tools used to analyze and report on company data.

# What is Big Data?

## Introduce Big Data and cloud computing

- **Big Data** is the current term for the enormous datasets generated by Web and mobile applications.
- **Nonrelational databases** are used to store Big Data (also known as NoSQL).

# Figure 1.33 ArangoDB

The screenshot shows the ArangoDB web interface. On the left, there's a sidebar with links for Collections, Queries (which is selected), Graphs, Services, Logs, Support, Help Us, and Get Enterprise. The main area has tabs for Editor, Running Queries, and Slow Query History. In the Editor tab, there's a code editor with the following AQL query:

```
1 FOR c IN COURSES  
2 RETURN c
```

A callout box points to this code with the text: "The query written in ArangoDB query language (AQL)". Below the code editor is a table with the following data:

_key	_id	_rev	CourseNumber	Course	CourseDate	Fee	Enrollments
17183940	COURSES/17183940	_WB-BLay-B	2	Beg Oils	9/15/2019	350	[{"CustomerNumber":2,"CustomerLastName":"Green","CustomerFirstName":"Robin","Phone":"425-678-8765","AmountPaid":350}, {"CustomerNumber":5,"CustomerLastName":"Sears","CustomerFirstName":"Miguel","Phone":"360-789-4567","AmountPaid":350}]
17183944	COURSES/17183944	_WB-BLay-F	4	Beg Oils	10/15/2019	350	[{"CustomerNumber":7,"CustomerLastName":"Myers","CustomerFirstName":"Lynda","Phone":"360-789-5678","AmountPaid":0}]
17183936	COURSES/17183936	_WB-BLay_-	1	Adv Pastels	10/1/2019	500	[{"CustomerNumber":1,"CustomerLastName":"Johnson","CustomerFirstName":"Ariel","Phone":"206-567-1234","AmountPaid":250}, {"CustomerNumber":3,"CustomerLastName":"Jackson","CustomerFirstName":"Charles","Phone":"360-789-3456","AmountPaid":500}, {"CustomerNumber":4,"CustomerLastName":"Pearson","CustomerFirstName":"Jeffery","Phone":"206-567-2345","AmountPaid":500}]
17183942	COURSES/17183942	_WB-BLay-D	3	Int Pastels	3/15/2019	350	[{"CustomerNumber":1,"CustomerLastName":"Johnson","CustomerFirstName":"Ariel","Phone":"206-567-1234","AmountPaid":350}]
17183946	COURSES/17183946	_WB-BLay-H	5	Adv Pastels	11/15/2019	500	[{"CustomerNumber":6,"CustomerLastName":"Kyle","CustomerFirstName":"Leah","Phone":"425-678-7654","AmountPaid":250}]

A callout box points to the table with the text: "The query results in table format—note the extra columns \_key, \_id, and \_rev". At the bottom right of the interface, there are buttons for Download, CSV, and Copy To Editor.

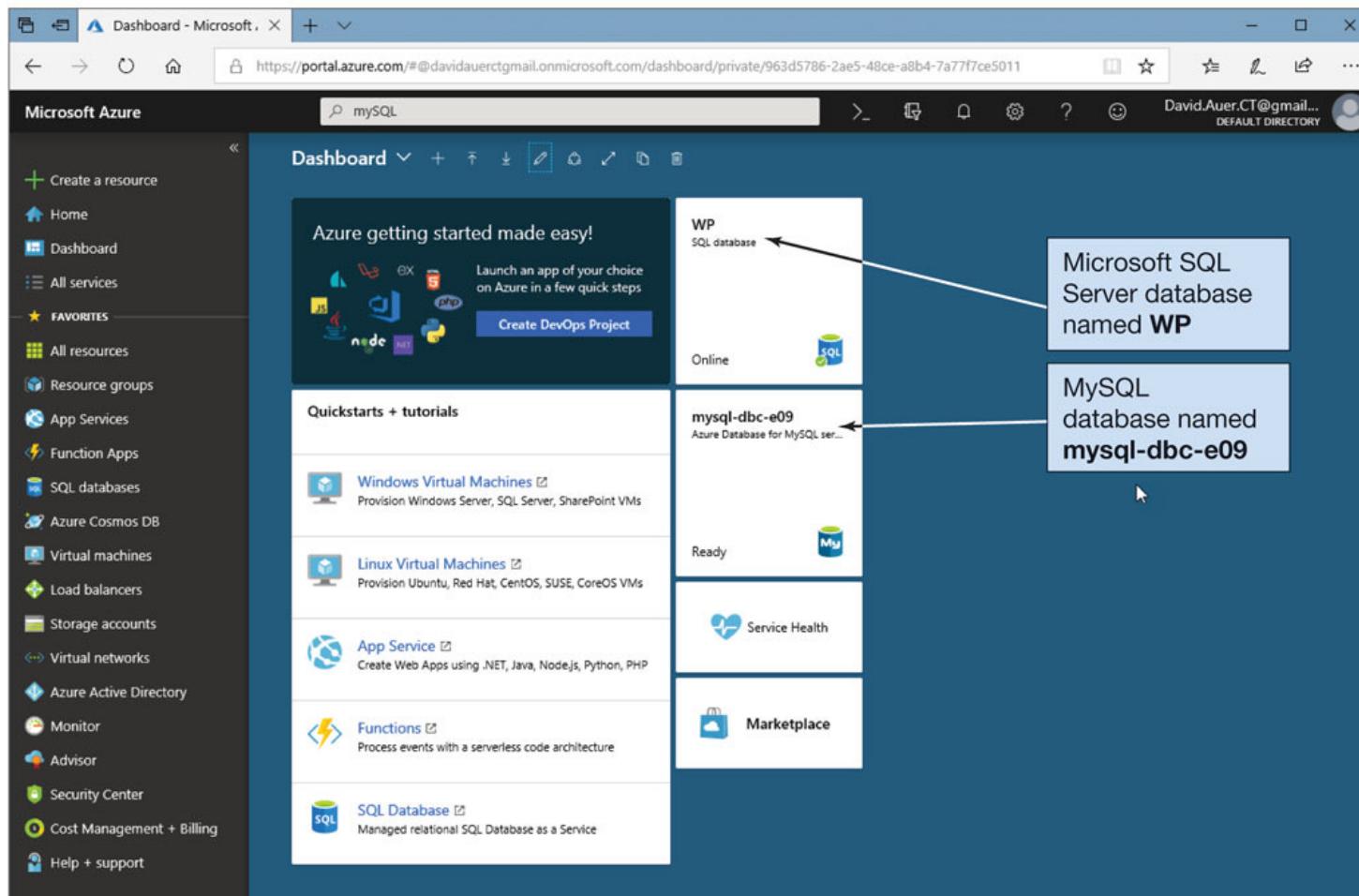
Courtesy of ArangoDB GmbH.

# What is Cloud Computing?

## Introduce Big Data and cloud computing

- **Cloud Computing** is the use of another company's hardware to conduct business via an Internet connection or through a Web server.
- Examples include:
  - Amazon Web Services (AWS)
  - Google Cloud Platform

# Figure 1.34 Microsoft Azure Cloud Service



Azure, Microsoft Edge, Microsoft Corporation.

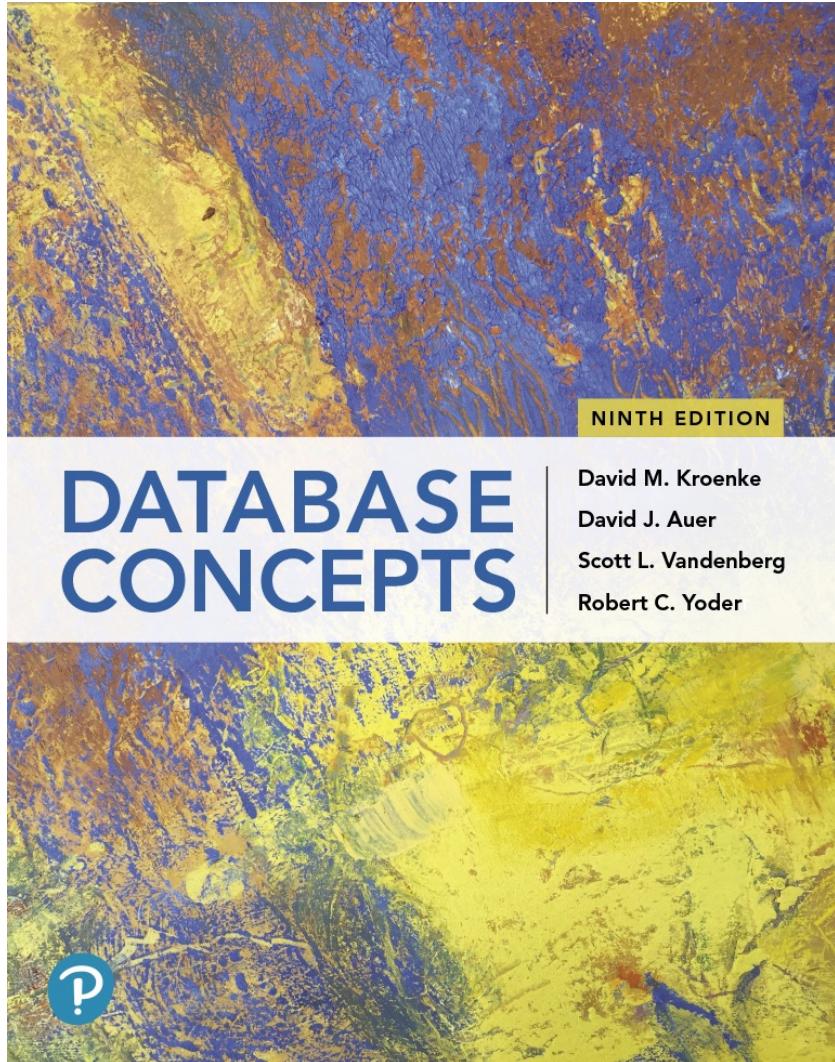
# Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

# Database Concepts

Ninth Edition



## Chapter 2

### The Relational Model

# Learning Objectives

- Learn the conceptual foundation of the relational model
- Understand how relations differ from nonrelational tables
- Learn basic relational terminology
- Learn the meaning and importance of keys, foreign keys, and related terminology
- Understand how foreign keys represent relationships
- Learn the purpose and use of surrogate keys
- Learn the meaning of functional dependencies
- Learn to apply a process for normalizing relations

# Relational Terms

## Learn the conceptual foundation of the relational model

- An **entity** is something of importance to the user that needs to be represented in a database.
  - In an entity-relationship model (discussed in Chapter 4), entities are restricted to things that can be represented by a single table.
- A **relation** is a two-dimensional table consisting of rows and columns that has the characteristics shown in Figure 2.1 on the next slide.

# Figure 2.1 Characteristics of a Relation

1. Rows contain data about an entity
2. Columns contain data about attributes of the entity
3. Cells of the table hold a single value
4. All entries in a column are of the same kind
5. Each column has a unique name
6. The order of the columns is unimportant
7. The order of the rows is unimportant
8. No two rows may hold identical sets of data values

# Figure 2.2 Sample EMPLOYEE Relation

EmployeeNumber	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420

# Figure 2.3 Nonrelational Table-Multiple Entries per Cell

EmployeeNumber	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310, 360-285-8391, 206-723-8392
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420, 360-285-8491

# Figure 2.4 Nonrelational Table-Order of Rows Matters and Kind of Column Entries Differs in Email

EmployeeNumber	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310
				Fax:	360-285-8391
				Home:	206-723-8392
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420
				Fax:	360-285-8491

# Figure 2.5 Relation with Variable-Length Column Values

EmployeeNumber	FirstName	LastName	Department	EmailAddress	Phone	Comments
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110	Company CEO. Completed her MBA in June, 2015.
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120	
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210	Partner at Able, Bandalone and Conerstone. Company counsel on a retainer basis.
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310	
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320	
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410	
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420	

# Presenting Relation Structures

## Learn basic relational terminology

- When writing out relation structures use the following format:
  - Relation names are written first in all caps (if two words then use an underscore between them) and they are always singular.
  - A column name is written with the first letter capitalized (if two words, then run them together and capitalize the first letter of each word)
- A **database schema** is the design on which a database and its associated applications are built.

## Figure 2.6 Equivalent Sets of Terms

Table	Row	Column
File	Record	Field
Relation	Tuple	Attribute

# Types of Keys

**Learn the meaning and importance of keys, foreign keys, and related terminology**

- A **key** is one or more columns of a relation that is used to identify a row.
  - A key can be unique (primary key) or nonunique (foreign key)
- A **composite key** contains two or more attributes.
- **Candidate keys** are keys that uniquely identify each row in a relation.
- A **primary key** is a candidate key that is chosen as the key that the DMBS will use to uniquely identify each row in a relation.
  - The primary key in a relation will be underlined.

# Example of a Relation

## Learn the meaning and importance of keys, foreign keys, and related terminology

- Consider the following example of a relation:
- EMPLOYEE (EmployeeNumber, FirstName, LastName, Department, EmailAddress, Phone)
- Notice the following in the example above:
  - The relation is in all caps
  - All the attributes are within parentheses
  - The primary key consists of two words run together with the first letter of each word capitalized. It is also underlined.

# Figure 2.7 Defining a Primary Key in Microsoft Access 2019

The key symbol indicates which column or columns are being used as the primary key

Primary Key button

Field Name	Data Type	Description
CustomerNumber	AutoNumber	A surrogate key for the CUSTOMER table
CustomerLastName	Short Text	CUSTOMER Last Name
CustomerFirstName	Short Text	CUSTOMER First Name
Phone	Short Text	The ten-digit North American phone number in the format: nnn-nnn-nnnn

Field Properties

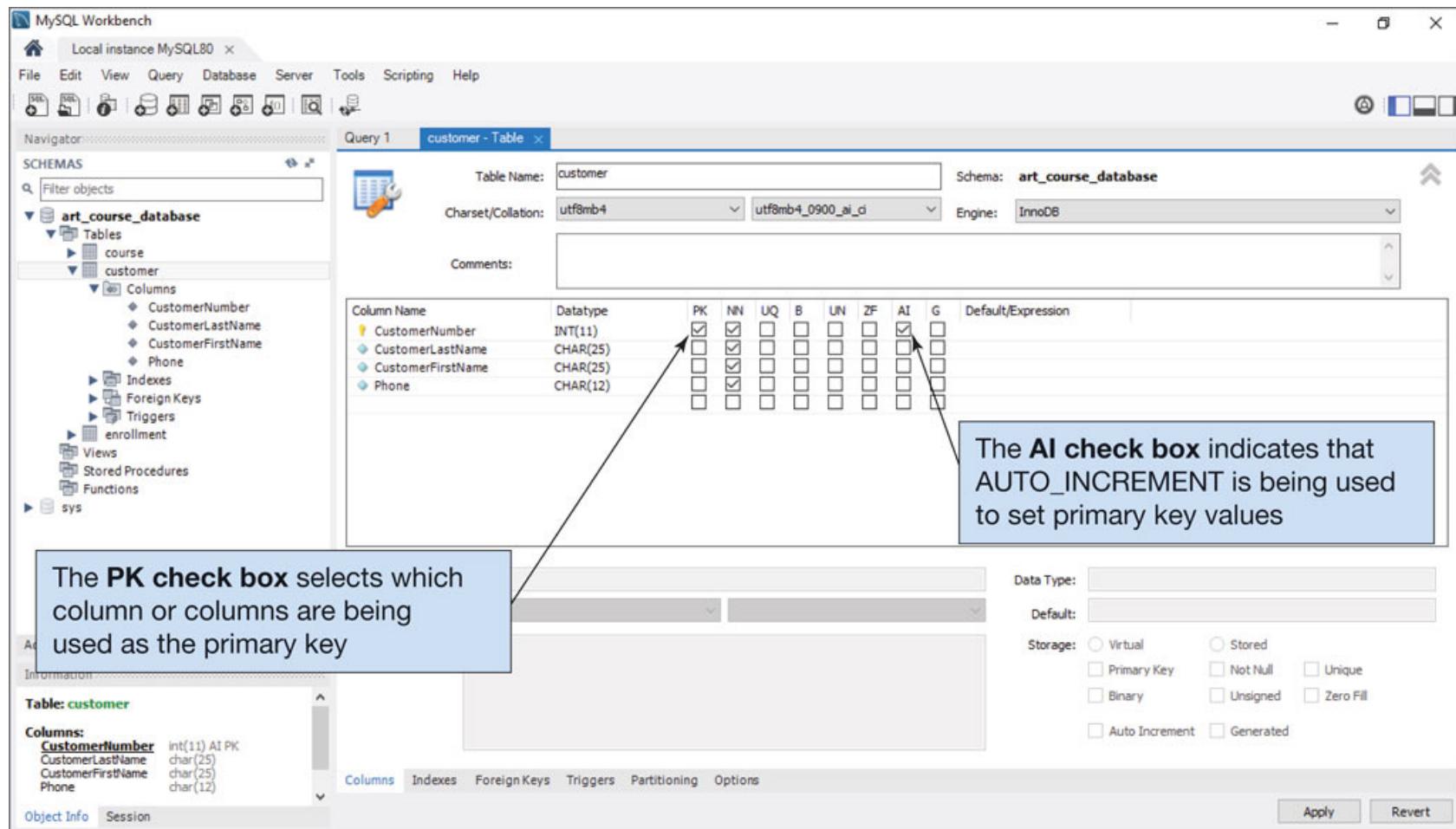
General Lookup

Field Size: Long Integer  
New Values: Increment  
Format:  
Caption:  
Indexed: Yes (No Duplicates)  
Text Align: General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Access 2019, Windows 10, Microsoft Corporation

# Figure 2.8 Defining a Primary Key in MySQL 8.0



MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 2.9 Defining a Key in Microsoft SQL Server 2017

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database structure is visible, including the CUSTOMER table under the Art\_Course\_Database. The Table Designer is open for the CUSTOMER table, displaying columns: CustomerNumber (int, PK, not null), CustomerLastName (char(25)), CustomerFirstName (char(25)), and Phone (char(12)). The CustomerNumber column is highlighted with a key symbol. In the Column Properties grid, the 'Identity Specification' section is selected, showing 'Is Identity' set to Yes, 'Identity Increment' set to 1, and 'Identity Seed' set to 1.

The key symbol indicates which column or columns are being used as the primary key

The Is Identity setting

The Identity Increment setting

The Identity Seed setting

Column Name	Data Type	Allow Nulls
CustomerNumber	int	<input type="checkbox"/>
CustomerLastName	char(25)	<input type="checkbox"/>
CustomerFirstName	char(25)	<input type="checkbox"/>
Phone	char(12)	<input type="checkbox"/>

Setting	Value
Is Identity	Yes
Identity Increment	1
Identity Seed	1
Indexable	Yes
Is Columnset	No

SQL Server 2017, SQL Server Management Studio, Microsoft Corporation

# Figure 2.10 Defining a Primary Key in Oracle Database XE

The screenshot shows the Oracle SQL Developer interface with the following details:

- Connections:** Art\_Course\_Database is selected.
- Tables:** CUSTOMER is selected.
- Constraints:** The CUSTOMER table has one primary key constraint, CUSTOMER\_PK, and four check constraints (SYS\_C007084 through SYS\_C007087).
- Columns:** The CUSTOMERNUMBER column is listed as the first column in the primary key constraint.
- Sequences:** Sequences used for primary key values are listed: SEQCOURSEID and SEQCUSTOMERID.

Annotations explain the following:

- The constraint type **Primary\_Key** indicates which constraints are being used to create the primary key.
- This constraint creates the primary key for CUSTOMER.
- The columns used in the primary key constraint CUSTOMER\_PK are shown in the Columns pane.
- Sequences used to define primary key values are shown here.

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEF
1 CUSTOMER_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT
2 SYS_C007084	Check	"CUSTOMERNUMBER" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT
3 SYS_C007085	Check	"CUSTOMERLASTNAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT
4 SYS_C007086	Check	"CUSTOMERFIRSTNAME" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT
5 SYS_C007087	Check	"PHONE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT

Oracle Database XE, SQL Developer 18.4, Oracle Corporation

# Surrogate Keys

## Learn the purpose and use of surrogate keys

- A **surrogate key** is a column with a unique, DBMS-assigned identifier that has been added to a table to be the primary key.
  - The ideal surrogate key is short, numeric and never changes.
- Suppose we have the following table:

**PROPERTY (Street, City, State, ZIP, OwnerID)**

  - Notice that it takes the street, city, state, and zip to uniquely identify a row in a table.
- Now lets use a surrogate key as a unique identifier!

**PROPERTY (PropertyID, Street, City, State, ZIP, OwnerID)**

# Foreign Keys

## Learn how foreign keys represent relationships

- A **foreign key** is a primary key of another relation that has been placed in the current relation to represent a relationship between two tables.
  - It is represented in a relation by italics as seen in the example below.

EMPLOYEE (EmployeeNumber, FirstName, LastName, *Department*,  
EmailAddress, Phone)

DEPARTMENT (DepartmentName, BudgetCode, OfficeNumber,  
DepartmentPhone)

# Referential Integrity

## Understand how foreign keys represent relationships

- A **referential integrity constraint** states that every value of a foreign key must match a value of an existing primary key.
- In the relationship between EMPLOYEE and DEPARTMENT seen on the previous slide, the department attribute located in the EMPLOYEE table is the foreign key and whatever value is placed in that column, the same value **MUST** exist in the Department attribute in the DEPARTMENT table.

# Figure 2.11 Enforcing Referential Integrity in Microsoft Access 2019

The relationship is between CUSTOMER and ENROLLMENT—the foreign key CustomerNumber in ENROLLMENT references the primary key CustomerNumber in CUSTOMER

Use this check box to enforce referential integrity in this relationship

Relationship Tools

Design

Tell me what you want to do

File Home Create External Data Database Tools Help

Relationships

Clear Layout

Relationship Report

Tools

Hide Table

Show Table

Direct Relationships

All Relationships

Close

Relationships

CUSTOMER

ENROLLMENT

COURSE

CustomerNumber  
CustomerLastName  
CustomerFirstName  
Phone

CustomerNumber  
CourseNumber  
AmountPaid

CourseNumber  
Course  
CourseDate  
Fee

Navigation Pane

Ready

Table/Query: CUSTOMER Related Table/Query: ENROLLMENT

CustomerNum CustomerNum

CustomerNum

Enforce Referential Integrity

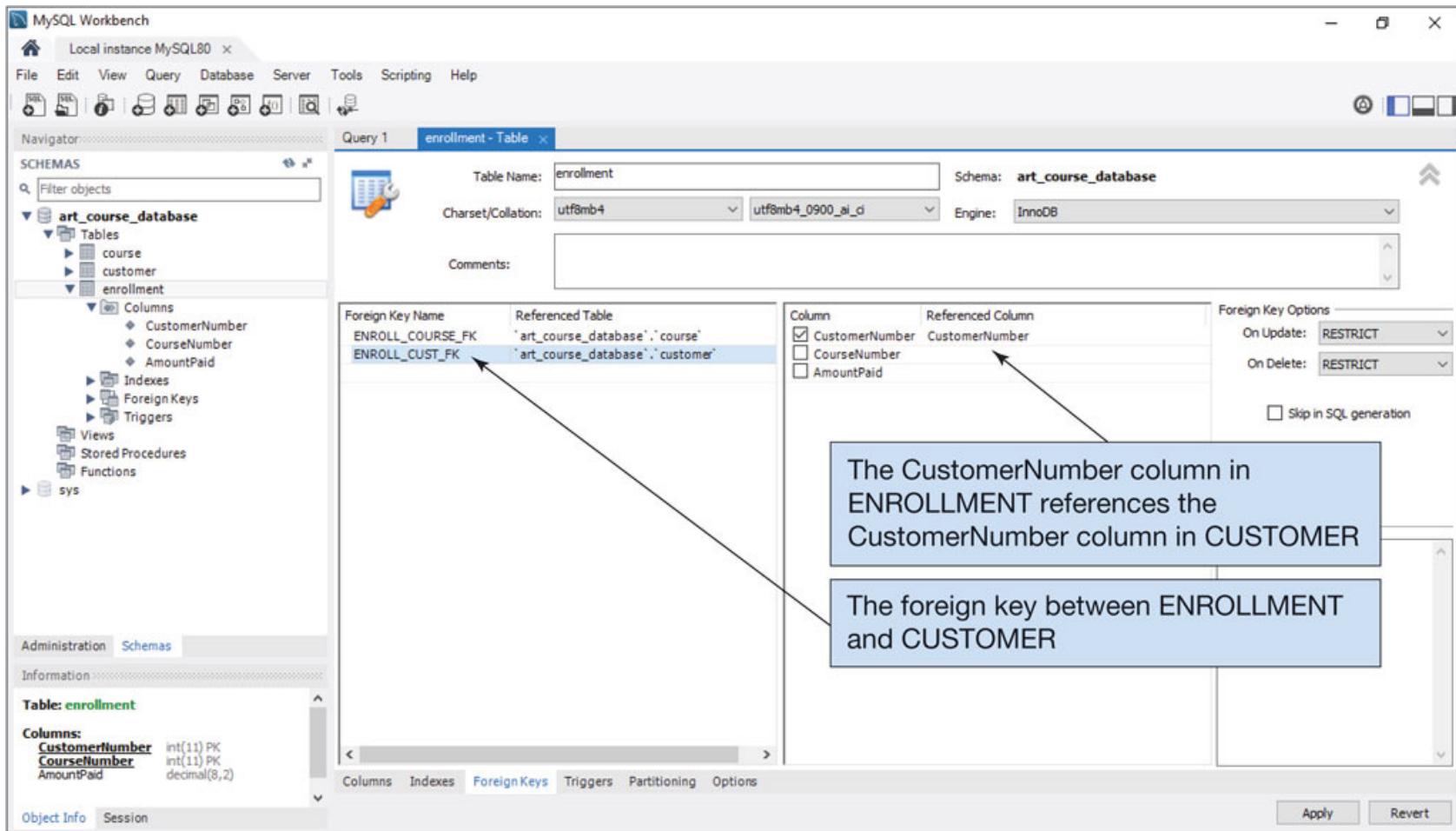
Cascade Update Related Fields

Cascade Delete Related Records

Relationship Type: One-To-Many

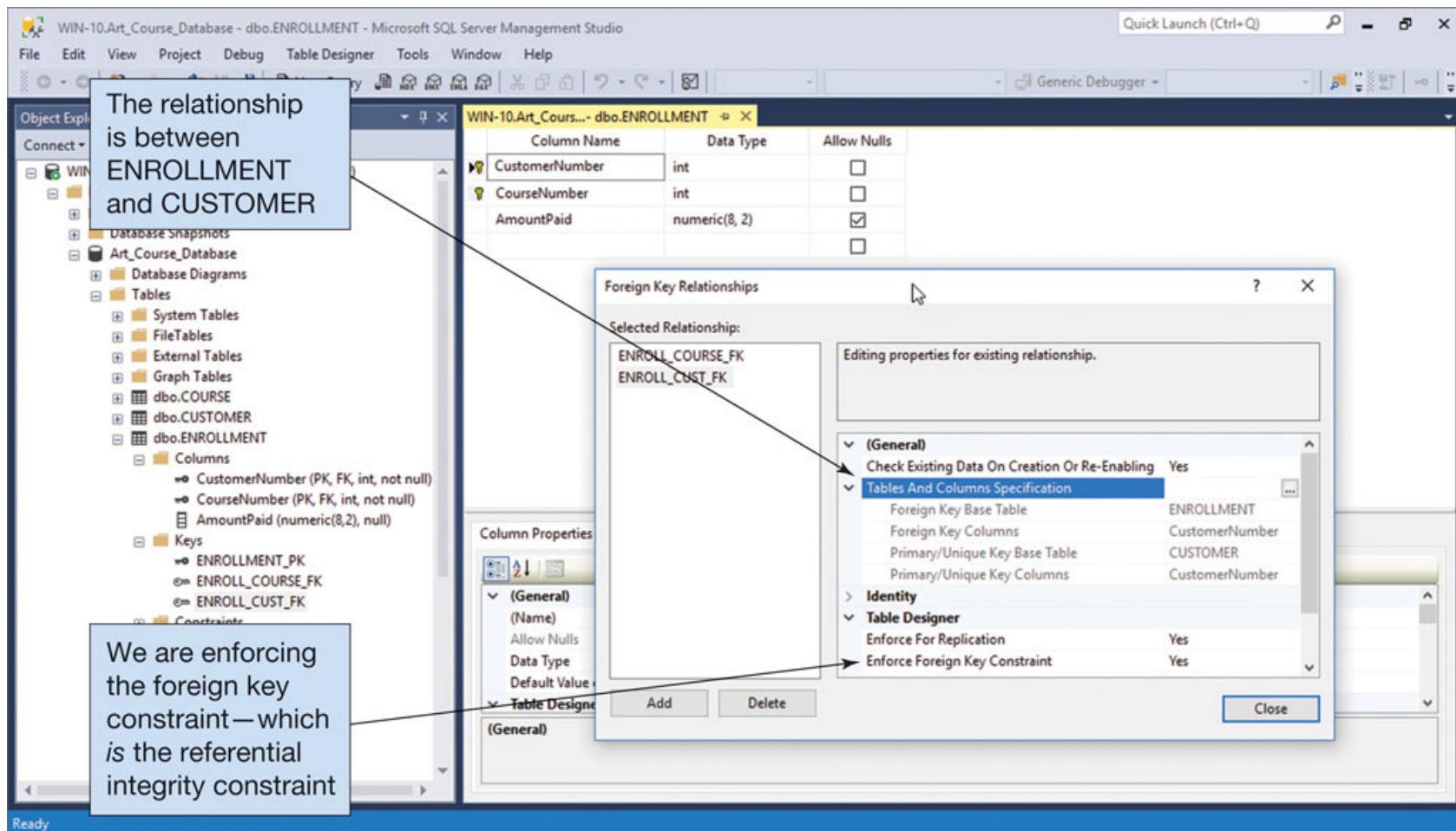
Access 2019, Windows 10, Microsoft Corporation

# Figure 2.12 Enforcing Referential Integrity in MySQL 8.0



MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 2.13 Enforcing Referential Integrity in Microsoft SQL Server 2017



SQL Server 2017, SQL Server Management Studio, Microsoft Corporation

# Figure 2.14 Enforcing Referential Integrity in Oracle Database XE

The screenshot shows the Oracle SQL Developer interface with the 'ENROLLMENT' table selected in the 'Constraints' tab. The constraints listed are:

CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NAME	DELETE_RULE	STATUS	DEFERR
1 ENROLLMENT_PK	Primary_Key	(null)	(null)	(null)	(null)	(null)	ENABLED	NOT DEF
2 ENROLL_COURSE_FK	Foreign_Key	(null)	ACD_USER	COURSE	COURSE_PK	CASCADE	ENABLED	NOT DEF
3 ENROLL_CUST_FK	Foreign_Key	(null)	ACD_USER	CUSTOMER	CUSTOMER_PK	NO ACTION	ENABLED	NOT DEF
4 SYS_C007094	Check	"CUSTOMERNUMBER" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEF
5 SYS_C007095	Check	"COURSENOMBRE" IS NOT NULL	(null)	(null)	(null)	(null)	ENABLED	NOT DEF

Three callout boxes provide additional information:

- The constraint type **Foreign\_Key** indicates which constraints are being used to create the foreign keys.
- This constraint creates the foreign key relationship is between ENROLLMENT and CUSTOMER.
- The columns used in the foreign key constraint ENROLL\_CUST\_FK are shown in the Columns pane.

Oracle Database XE, SQL Developer 18.4, Oracle Corporation

# The NULL Value

## Learn the meaning and importance of keys, foreign keys, and related terminology

- A **null value** is a missing value in a cell in a relation.
- The problem with null values is that it is ambiguous:
  - Is it that no value is appropriate,
  - Is it known, but not entered, or
  - Is it unknown, thus not entered.
- You can eliminate null values by requiring an attribute value.

## Figure 2.15 Sample ITEM Relation and Data

ItemNumber	ItemName	Color	Quantity
110	T-shirt, Small	Red	15
111	T-shirt, Small	Blue	25
115	T-shirt, Small	Green	22
120	T-shirt, Medium	Red	45
125	T-shirt, Medium	Green	52
310	Baseball Cap, Fits All	Red	38
311	Baseball Cap, Fits All	Blue	54
400	Spring Hat		120

# Functional Dependencies

## Learn the meaning of functional dependencies

- A **functional dependency** occurs when a candidate key determines all the other attributes in a relation.
  - In other words, all the attributes in a relation are functionally dependent on the candidate key.
  - A dependency is shown with the determinant on the left and then an arrow showing the attribute(s) that depend on it, as shown below.

$\text{CustomerNumber} \rightarrow (\text{CustomerLastName}, \text{CustomerFirstName}, \text{Phone})$

# Normalization

## Learn to apply a process for normalizing relations

- **Normalization** is the process of (or set of steps for) breaking a table or relation with more than one theme into a set of tables such that each has only one theme.
- Relational design principles for a well-formed relation:
  - Every determinate must be a candidate key
  - Any relation that is not well formed should be broken into two or more relations that are well formed
- A relation is in first normal form (1NF) if it:
  - Has characteristics listed in Figure 2.1
  - Has a defined primary key
  - No repeating groups

# Normalization Process

## Learn to apply a process for normalizing relations

- Steps in the normalization process are as follows:
  1. Identify all the candidate keys of the relation.
  2. Identify all the functional dependencies in the relation.
  3. Examine the determinants of the functional dependencies. If any determinant is not a candidate key, the relation is not well formed. In this case:
    - a. Place the columns of the functional dependency in a new relation of their own.
    - b. Make the determinant of the functional dependency the primary key of the new relation.
    - c. Leave a copy of the determinant as a foreign key in the original relation.
    - d. Create a referential integrity constraint between the original and the new relation.
  4. Repeat Step 3 until every determinant of every relation is a candidate key.

# Figure 2.17 Sample PRESCRIPTION Relation and Data

PrescriptionNumber	Date	Drug	Dosage	CustomerName	CustomerPhone	CustomerEmailAddress
P10001	10/17/2019	DrugA	10mg	Smith, Alvin	575-523-2233	ASmith@somewhere.com
P10003	10/17/2019	DrugB	35mg	Rhodes, Jeff	575-645-3455	JRhodes@somewhere.com
P10004	10/17/2019	DrugA	20mg	Smith, Sarah	575-523-2233	SSmith@somewhere.com
P10007	10/18/2019	DrugC	20mg	Frye, Michael	575-645-4566	MFrye@somewhere.com
P10010	10/18/2019	DrugB	30mg	Rhodes, Jeff	575-645-3455	JRhodes@somewhere.com

# Figure 2.18 Normalized CUSTOMER and PRESCRIPTION Relations and Data

CustomerEmailAddress	CustomerName	CustomerPhone
ASmith@somewhere.com	Smith, Alvin	575-523-2233
JRhodes@somewhere.com	Rhodes, Jeff	575-645-3455
MFrye@somewhere.com	Frye, Michael	575-645-4566
SSmith@somewhere.com	Smith, Sarah	575-523-2233

PrescriptionNumber	Date	Drug	Dosage	CustomerEmailAddress
P10001	10/17/2019	DrugA	10mg	ASmith@somewhere.com
P10003	10/17/2019	DrugB	35mg	JRhodes@somewhere.com
P10004	10/17/2019	DrugA	20mg	SSmith@somewhere.com
P10007	10/18/2019	DrugC	20mg	MFrye@somewhere.com
P10010	10/18/2019	DrugB	30mg	JRhodes@somewhere.com

# Figure 2.19 Sample STU\_DORM Relation and Data

StudentNumber	LastName	FirstName	DormName	DormCost
100	Smith	Terry	Stephens	3,500.00
200	Johnson	Jeff	Alexander	3,800.00
300	Abernathy	Susan	Horan	4,000.00
400	Smith	Susan	Alexander	3,800.00
500	Wilcox	John	Stephens	3,500.00
600	Webber	Carl	Horan	4,000.00
700	Simon	Carol	Stephens	3,500.00

# Figure 2.20 Normalized STU\_DORM and DORM Relations and Data

StudentNumber	LastName	FirstName	DormName
100	Smith	Terry	Stephens
200	Johnson	Jeff	Alexander
300	Abernathy	Susan	Horan
400	Smith	Susan	Alexander
500	Wilcox	John	Stephens
600	Webber	Carl	Horan
700	Simon	Carol	Stephens

DormName	DormCost
Alexander	3,800.00
Horan	4,000.00
Stephens	3,500.00

# Figure 2.21 Sample EMPLOYEE Relation and Data

EmployeeNumber	FirstName	LastName	Department	EmailAddress	DepartmentPhone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8100
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8100
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8200
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8300
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8300
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8400
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8400

# Figure 2.22 Normalized EMPLOYEE and DEPARTMENT Relations and Data

EmployeeNumber	FirstName	LastName	Department	EmailAddress
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com
104	George	Smith	Human Resources	George.Smith@ourcompany.com
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com

Department	DepartmentPhone
Administration	360-285-8100
Finance	360-285-8400
Human Resources	360-285-8300
Legal	360-285-8200

## Figure 2.23 Sample MEETING Relation and Data

Attorney	ClientNumber	ClientName	Date	Duration
Boxer	1000	ABC, Inc	11/5/2019	2.00
Boxer	2000	XYZ Partners	11/5/2019	5.50
James	1000	ABC, Inc	11/7/2019	3.00
Boxer	1000	ABC, Inc	11/9/2019	4.00
Wu	3000	Malcomb Zoe	11/11/2019	7.00

# Figure 2.24 Normalized MEETING and CLIENT Relations and Data

Attorney	ClientNumber	Date	Duration
Boxer	1000	11/5/2019	2.00
Boxer	2000	11/5/2019	5.50
James	1000	11/7/2019	3.00
Boxer	1000	11/9/2019	4.00
Wu	3000	11/11/2019	7.00

ClientNumber	ClientName
1000	ABC, Inc
2000	XYZ Partners
3000	Malcomb Zoe

# Eliminating Anomalies from Multivalued Dependencies

## Learn to apply a process for normalizing relations

- When modification problems are due to functional dependencies and we then normalize relations to BCNF, we eliminate these anomalies.
- Anomalies can also arise from another kind of dependency—the multivalued dependency.
  - A **multivalued dependency** occurs when a determinant is matched with a particular set of values as seen below.

$\text{EmployeeName} \rightarrow\rightarrow \text{EmployeeDegree}$

$\text{EmployeeName} \rightarrow\rightarrow \text{EmployeeSibling}$

$\text{PartKitName} \rightarrow\rightarrow \text{Part}$

## Figure 2.25 Examples of Multivalued Dependencies

EMPLOYEE\_DEGREE

EmployeeName	EmployeeDegree
Chau	BS
Green	BS
Green	MS
Green	PhD
Jones	AA
Jones	BA

EMPLOYEE\_SIBLING

EmployeeName	EmployeeSibling
Chau	Eileen
Chau	Jonathan
Green	Nikki
Jones	Frank
Jones	Fred
Jones	Sally

PARTKIT\_PART

PartKitName	Part
Bike Repair	Screwdriver
Bike Repair	Tube Fix
Bike Repair	Wrench
First Aid	Aspirin
First Aid	Bandaids
First Aid	Elastic Bands
First Aid	Ibuprofin
Toolbox	Drill and drill bits
Toolbox	Hammer
Toolbox	Saw
Toolbox	Screwdriver
Toolbox	Wrench

# A relation that is not normalized

Patient #	Surgeon #	Surg. date	Patient Name	Patient Addr	Surgeon	Surgery	Postop drug	side effec
1111	145 311	Jan 1, 1995; June 12, 1995	John White	15 New St. New York, NY	Beth Little Michael Diamond	Gallstone s removal; Kidney stones removal	Penicillin, none-	rash none
1234	243 467	Apr 5, 1994 May 10, 1995	Mary Jones	10 Main St. Rye, NY	Charles Field Patricia Gold	Eye Cataract removal Thrombos is removal	Tetracyclin e none	Fever none
2345	189	Jan 8, 1996	Charles Brown	Dogwood Lane Harrison, NY	David Rosen	Open Heart Surgery	Cephalosp orin	none
4876	145	Nov 5, 1995	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecyst ectomy	Demicillin	none
5123	145	May 10, 1995	Paul Kosher	Blind Brook Mamaronec k, NY	Beth Little	Gallstone s Removal	none	none
6845	243	Apr 5, 1994 Dec 15, 1984	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replace ment Eye cataract removal	Tetracyclin e	Fever

# First Normal Form

- . First Normal Form (1NF)
  - each cell has only one value, and all entries in a column are of the same kind

## First Normal Form

Patient #	Surgeon #	Surgery Date	Patient Name	Patient Addr	Surgeon Name	Surgery	Drug admin	Side Effects
1111	145	01-Jan-95	John White	15 New St. New York, NY	Beth Little	Gallstone s removal	Penicillin	rash
1111	311	12-Jun-95	John White	15 New St. New York, NY	Michael Diamond	Kidney stones removal	none	none
1234	243	05-Apr-94	Mary Jones	10 Main St. Rye, NY	Charles Field	Eye Cataract removal	Tetracyclin	Fever
1234	467	10-May-95	Mary Jones	10 Main St. Dogwood Lane Harrison, NY	Patricia Gold	Thrombos is removal	none	none
2345	189	08-Jan-96	Charles Brown		David Rosen	Open Heart Surgery	Cephalosp orin	none
4876	145	05-Nov-95	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecyst ectomy	Demicillin	none
5123	145	10-May-95	Paul Kosher	Blind Brook Mamaronec k, NY	Beth Little	Gallstone s Removal	none	none
6845	243	05-Apr-94	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replace ment	Tetracyclin	Fever
6845	243	15-Dec-84	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye cataract removal	none	none

## Second Normal Form

- . Each table is in 1NF and all non-key attributes are determined by the entire primary key
  - usually used in tables with a multiple-field primary key (composite key)
  - each non-key field relates to the entire primary key
  - any field that does not relate to the primary key is placed in a separate table
  - **MAIN POINT –**
    - eliminate redundant data in a table
    - Create separate tables for sets of values that apply to multiple records

# Example:

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY	WAREHOUSE ADDRESS
P0010	Warehouse A	400	1608 New Field Road
P0010	Warehouse B	543	4141 Greenway Drive
P0010	Warehouse C	329	171 Pine Lane
P0020	Warehouse B	200	4141 Greenway Drive
P0020	Warehouse D	278	800 Massey Street

PART\_STOCK TABLE

PART (Primary Key)	WAREHOUSE (Primary Key)	QUANTITY
P0010	Warehouse A	400
P0010	Warehouse B	543
P0010	Warehouse C	329
P0020	Warehouse B	200
P0020	Warehouse D	278

WAREHOUSE TABLE

WAREHOUSE (Primary Key)	WAREHOUSE_ADDRESS
Warehouse A	1608 New Field Road
Warehouse B	4141 Greenway Drive
Warehouse C	171 Pine Lane
Warehouse D	800 Massey Street

## 3<sup>rd</sup> Normal Form

- . Third Normal Form (3NF)
  - each table is in 2NF and no non-key attributes are determined by another non-key attribute

# Example:

**EMPLOYEE\_DEPARTMENT TABLE**

<b>EMPNO (Primary Key)</b>	<b>FIRSTNAME</b>	<b>LASTNAME</b>	<b>WORKDEPT</b>	<b>DEPTNAME</b>
000290	John	Parker	E11	Operations
000320	Ramlal	Mehta	E21	Software Support
000310	Maude	Setright	E11	Operations

**EMPLOYEE TABLE**

<b>EMPNO (Primary Key)</b>	<b>FIRSTNAME</b>	<b>LASTNAME</b>	<b>WORKDEPT</b>
000290	John	Parker	E11
000320	Ramlal	Mehta	E21
000310	Maude	Setright	E11

**DEPARTMENT TABLE**

<b>DEPTNO (Primary Key)</b>	<b>DEPTNAME</b>
E11	Operations
E21	Software Support

1

∞

# Boyce-Codd Normal Form

- each table is in 3NF and all determinants are candidate keys.
- Most 3NF relations are also BCNF relations.
- A 3NF relation is NOT in BCNF if:
  - Candidate keys in the relation are composite keys (they are not single attributes)
  - There is more than one candidate key in the relation, and
  - The keys are not disjoint, that is, some attributes in the keys are common
- A table is in Boyce-Codd normal form (BCNF) if every determinant in the table is a candidate key.
  - (A determinant is any attribute whose value determines other values with a row.)
- If a table contains only one candidate key, the 3NF and the BCNF are equivalent.
- BCNF is a special case of 3NF

## Exercises: Normalize the following table up to 3<sup>rd</sup> Normal form

. Grade (ClassName, Section, Term, Grade, StudentNumber, StudentName, ProfessorName, Department, ProfessorEmailAddress)

# Exercises: Normalize the following tables up to the Third normal form

Student#	Advisor#	Advisor	Adv-Room	Class1	Class2	Class3
1022	10	Susan Jones	412	101-07	143-01	159-02
4123	12	Anne Smith	216	101-07	159-02	214-01

# Exercises: Normalize the following tables up to the Third normal form

	A	B	C	D	E	F	G	H	I	J	K
1	PetName	PetType	PetBreed	PetDOB	OwnerLastName	OwnerFirstName	OwnerPhone	OwnerEmail	Service	Date	Charge
2	King	Dog	Std. Poodle	27-Feb-16	Downs	Marsha	201-823-5467	Marsha.Downs@somewhere.com	Ear Infection	17-Aug-18	\$ 65.00
3	Teddy	Cat	Cashmier	1-Feb-15	James	Richard	201-735-9812	Richard.James@somewhere.com	Nail Clip	5-Sep-18	\$ 27.50
4	Fido	Dog	Std. Poodle	17-Jul-17	Downs	Marsha	201-823-5467	Marsha.Downs@somewhere.com			
5	AJ	Dog	Collie Mix	5-May-17	Frier	Liz	201-823-6578	Liz.Frier@somewhere.com	One-year Shots	5-May-18	\$ 42.50
6	Cedro	Cat	Unknown	6-Jun-14	James	Richard	201-735-9812	Richard.James@somewhere.com	Nail Clip	5-Sep-18	\$ 27.50
7	Woolley	Cat	Unknown	???	James	Richard	201-735-9812	Richard.James@somewhere.com	Skin Infection	3-Oct-18	\$ 35.00
8	Buster	Dog	Border Collie	11-Dec-13	Trent	Miles	201-634-7865	Miles.Trent@somewhere.com	Laceration Repair	5-Oct-18	\$ 127.00
9	Jiddah	Cat	Abyssinian	1-Jul-10	Evans	Hilary	201-634-2345	Hilary.Evans@somewhere.com	Booster Shots	4-Nov-18	\$ 111.00

# Copyright



**This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.**

# Database Concepts

Ninth Edition



## Chapter 3

### Structured Query Language

# Learning Objectives

- Learn basic SQL statements for creating database structures
- Learn basic SQL statements for adding data to a database
- Learn basic SQL SELECT statements and options for processing a single table
- Learn basic SQL SELECT statements for processing multiple tables with subqueries
- Learn basic SQL SELECT statements for processing multiple tables with joins
- Learn basic SQL statements for modifying and deleting data from a database
- Learn basic SQL statements for modifying and deleting database tables and constraints

# Structured Query Language

## Learn basic SQL statements for creating database structures

- **Structured Query Language (SQL)** is not a complete programming language; rather, it is a data sublanguage
  - Developed by IBM in the late 1970's
  - Endorsed and adopted by ANSI in 1992
  - Endorsed as a standard by the International Organization for Standardization (ISO)
  - It is text oriented
  - Can be used by MS Access using Query by Example (QBE)

# SQL Statement Categories

## Learn basic SQL statements for creating database structures

- SQL statement categories are as follows:
  - Data definition language (DDL)
    - Statements used to create database structures
  - Data manipulation language (DML)
    - Statements used to query, insert, modify and delete data
  - SQL/Persistent stored modules (SQL/PSM)
    - Statements to extend SQL by adding procedural programming capabilities
  - Transaction control language (TCL)
    - Statements used to mark transaction boundaries
  - Data control language (DCL)
    - Statements used to grant and revoke database permissions

# **Wedgewood Pacific Example**

## **Learn basic SQL statements for creating database structures**

- This chapter uses the Wedgewood Pacific (WP) company as an example.
  - Founded in 1957 in Seattle, WA
  - Manufacture and sell consumer drone aircraft
  - In January, 2018, the FAA said that over one million drones had been registered
  - WP's database has four tables (Department, Employee, Project, and Assignment)

# Wedgewood Pacific Relations

## Learn basic SQL statements for creating database structures

- The following relations are used for the Wedgewood Pacific (WP) example used in this chapter:

DEPARTMENT (DepartmentName, BudgetCode, OfficeNumber, DepartmentPhone)

EMPLOYEE (EmployeeNumber, FirstName, LastName, *Department*, Position,  
*Supervisor*, OfficePhone, EmailAddress)

PROJECT (ProjectID, ProjectName, *Department*, MaxHours, StartDate, EndDate)

ASSIGNMENT (ProjectID, EmployeeNumber, HoursWorked)

# WP's Referential Integrity Constraints

## Learn basic SQL statements for creating database structures

- A referential integrity constraint is used to link (or reference) relations. This means that a foreign key in a relation must also exist in the relation in which it serves as the primary key. WP's referential integrity constraints:

Department in EMPLOYEE must exist in DepartmentName in DEPARTMENT

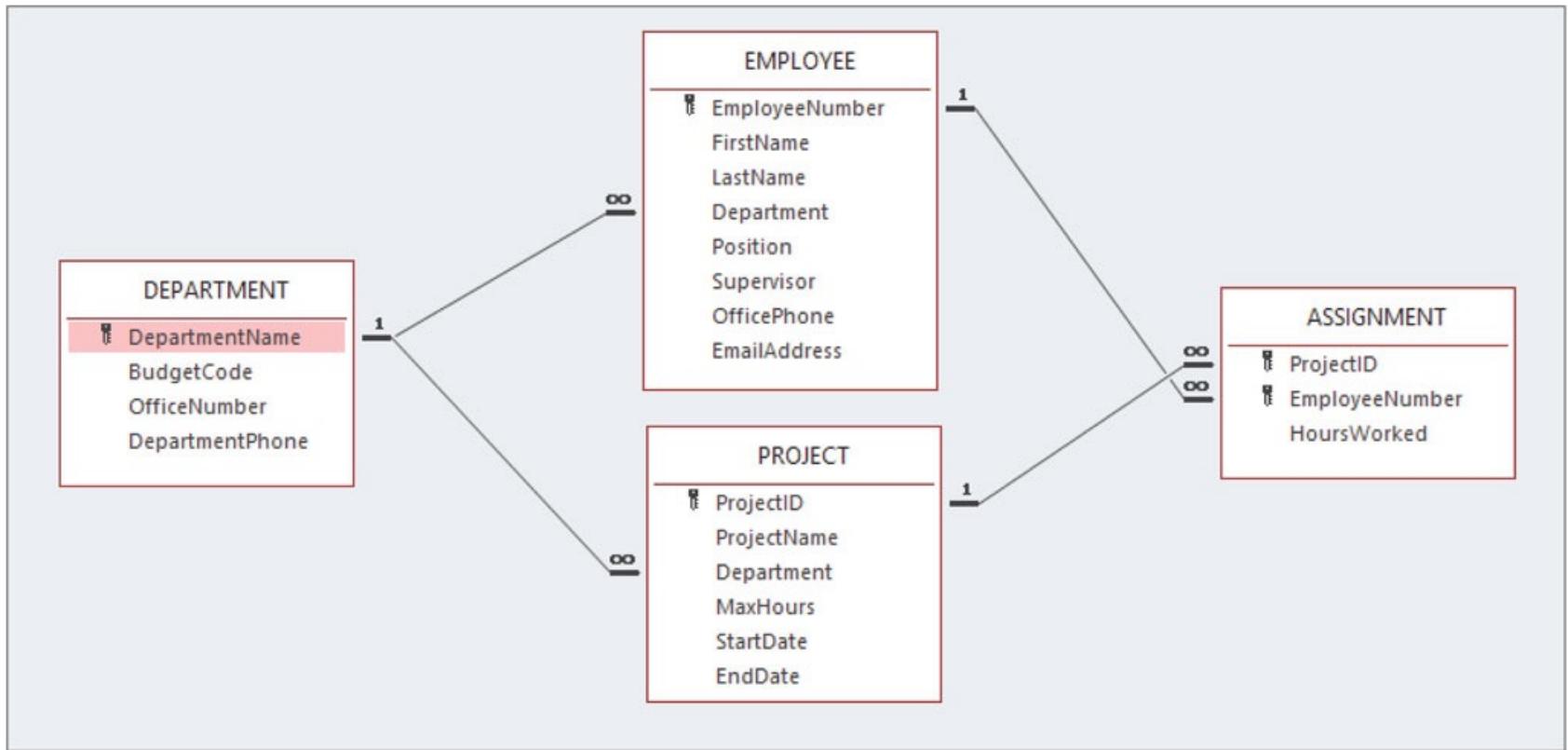
Supervisor in EMPLOYEE must exist in EmployeeNumber in EMPLOYEE

Department in PROJECT must exist in DepartmentName in DEPARTMENT

ProjectID in ASSIGNMENT must exist in ProjectID in PROJECT

EmployeeNumber in ASSIGNMENT must exist in EmployeeNumber in EMPLOYEE

# Figure 3.1(a) Database Column Characteristics for the WP Database



Access 2019, Windows 10, Microsoft Corporation

# Figure 3.1(b) DEPARTMENT Table

Column Name	Type	Key	Required	Remarks
DepartmentName	Short Text (35)	Primary Key	Yes	
BudgetCode	Short Text (30)	No	Yes	
OfficeNumber	Short Text (15)	No	Yes	
DepartmentPhone	Short Text (12)	No	Yes	

# Figure 3.1(c) EMPLOYEE Table

Column Name	Type	Key	Required	Remarks
EmployeeNumber	AutoNumber	Primary Key	Yes	Surrogate Key
FirstName	Short Text (25)	No	Yes	
LastName	Short Text (25)	No	Yes	
Department	Short Text (35)	Foreign Key	Yes	Links to DepartmentName in DEPARTMENT
Position	Short Text (35)	No	No	
Supervisor	Number	Foreign Key	No	Long Integer. Links to EmployeeNumber in EMPLOYEE
OfficePhone	Short Text (12)	No	No	
EmailAddress	Short Text (100)	No	Yes	

# Figure 3.1(d) PROJECT Table

ColumnName	Type	Key	Required	Remarks
ProjectID	Number	Primary Key	Yes	Long Integer
ProjectName	Short Text (50)	No	Yes	
Department	Short Text (35)	Foreign Key	Yes	Links to DepartmentName in DEPARTMENT
MaxHours	Number	No	Yes	Double
StartDate	Date	No	No	
EndDate	Date	No	No	

# Figure 3.1(e) ASSIGNMENT Table

Column Name	Type	Key	Required	Remarks
ProjectID	Number	Primary Key, Foreign Key	Yes	Long Integer Links to ProjectID in PROJECT
EmployeeNumber	Number	Primary Key, Foreign Key	Yes	Long Integer Links to EmployeeNumber in EMPLOYEE
HoursWorked	Number	No	No	Double

# Figure 3.2(a) DEPARTMENT Table

Sample Data for the WP Database

DepartmentName	BudgetCode	OfficeNumber	DepartmentPhone
Administration	BC-100-10	BLDG01-210	360-285-8100
Legal	BC-200-10	BLDG01-220	360-285-8200
Human Resources	BC-300-10	BLDG01-230	360-285-8300
Finance	BC-400-10	BLDG01-110	360-285-8400
Accounting	BC-500-10	BLDG01-120	360-285-8405
Sales and Marketing	BC-600-10	BLDG01-250	360-285-8500
InfoSystems	BC-700-10	BLDG02-210	360-285-8600
Research and Development	BC-800-10	BLDG02-250	360-285-8700
Production	BC-900-10	BLDG02-110	360-285-8800

# Figure 3.2(b) Sample Data for the WP Database: EMPLOYEE Table (1 of 2)

Employee Number	FirstName	LastName	Department	Position	Supervisor	OfficePhone	EmailAddress
1	Mary	Jacobs	Administration	CEO		360-285-8110	<a href="mailto:Mary.Jacobs@WP.com">Mary.Jacobs@WP.com</a>
2	Rosalie	Jackson	Administration	Admin Assistant	1	360-285-8120	<a href="mailto&gt;Rosalie.Jackson@WP.com">Rosalie.Jackson@WP.com</a>
3	Richard	Bandalone	Legal	Attorney	1	360-285-8210	<a href="mailto:Richard.Bandalone@WP.com">Richard.Bandalone@WP.com</a>
4	George	Smith	Human Resources	HR3	1	360-285-8310	<a href="mailto:George.Smith@WP.com">George.Smith@WP.com</a>
5	Alan	Adams	Human Resources	HR1	4	360-285-8320	<a href="mailto:Alan.Adams@WP.com">Alan.Adams@WP.com</a>
6	Ken	Evans	Finance	CFO	1	360-285-8410	<a href="mailto:Ken.Evans@WP.com">Ken.Evans@WP.com</a>
7	Mary	Abernathy	Finance	FA3	6	360-285-8420	<a href="mailto&gt;Mary.Abernathy@WP.com">Mary.Abernathy@WP.com</a>
8	Tom	Caruthers	Accounting	FA2	6	360-285-8430	<a href="mailto:Tom.Caruthers@WP.com">Tom.Caruthers@WP.com</a>
9	Heather	Jones	Accounting	FA2	6	360-285-8440	<a href="mailto:Heather.Jones@WP.com">Heather.Jones@WP.com</a>
10	Ken	Numoto	Sales and Marketing	SM3	1	360-285-8510	<a href="mailto:Ken.Numoto@WP.com">Ken.Numoto@WP.com</a>

# Figure 3.2(b) Sample Data for the WP Database: EMPLOYEE Table (2 of 2)

Employee Number	FirstName	LastName	Department	Position	Supervisor	OfficePhone	EmailAddress
11	Linda	Granger	Sales and Marketing	SM2	10	360-285-8520	<a href="mailto:Linda.Granger@WP.com">Linda.Granger@WP.com</a>
12	James	Nestor	InfoSystems	CIO	1	360-285-8610	<a href="mailto:James.Nestor@WP.com">James.Nestor@WP.com</a>
13	Rick	Brown	InfoSystems	IS2	12		<a href="mailto:Rick.Brown@WP.com">Rick.Brown@WP.com</a>
14	Mike	Nguyen	Research and Development	CTO	1	360-285-8710	<a href="mailto:Mike.Nguyen@WP.com">Mike.Nguyen@WP.com</a>
15	Jason	Sleeman	Research and Development	RD3	14	360-285-8720	<a href="mailto:Jason.Sleeman@WP.com">Jason.Sleeman@WP.com</a>
16	Mary	Smith	Production	OPS3	1	360-285-8810	<a href="mailto:Mary.Smith@WP.com">Mary.Smith@WP.com</a>
17	Tom	Jackson	Production	OPS2	16	360-285-8820	<a href="mailto:Tom.Jackson@WP.com">Tom.Jackson@WP.com</a>
18	George	Jones	Production	OPS2	17	360-285-8830	<a href="mailto:George.Jones@WP.com">George.Jones@WP.com</a>
19	Julia	Hayakawa	Production	OPS1	17		<a href="mailto:Julia.Hayakawa@WP.com">Julia.Hayakawa@WP.com</a>
20	Sam	Stewart	Production	OPS1	17		<a href="mailto:Sam.Stewart@WP.com">Sam.Stewart@WP.com</a>

# Figure 3.2(c) Sample Data for the WP Database: PROJECT Table

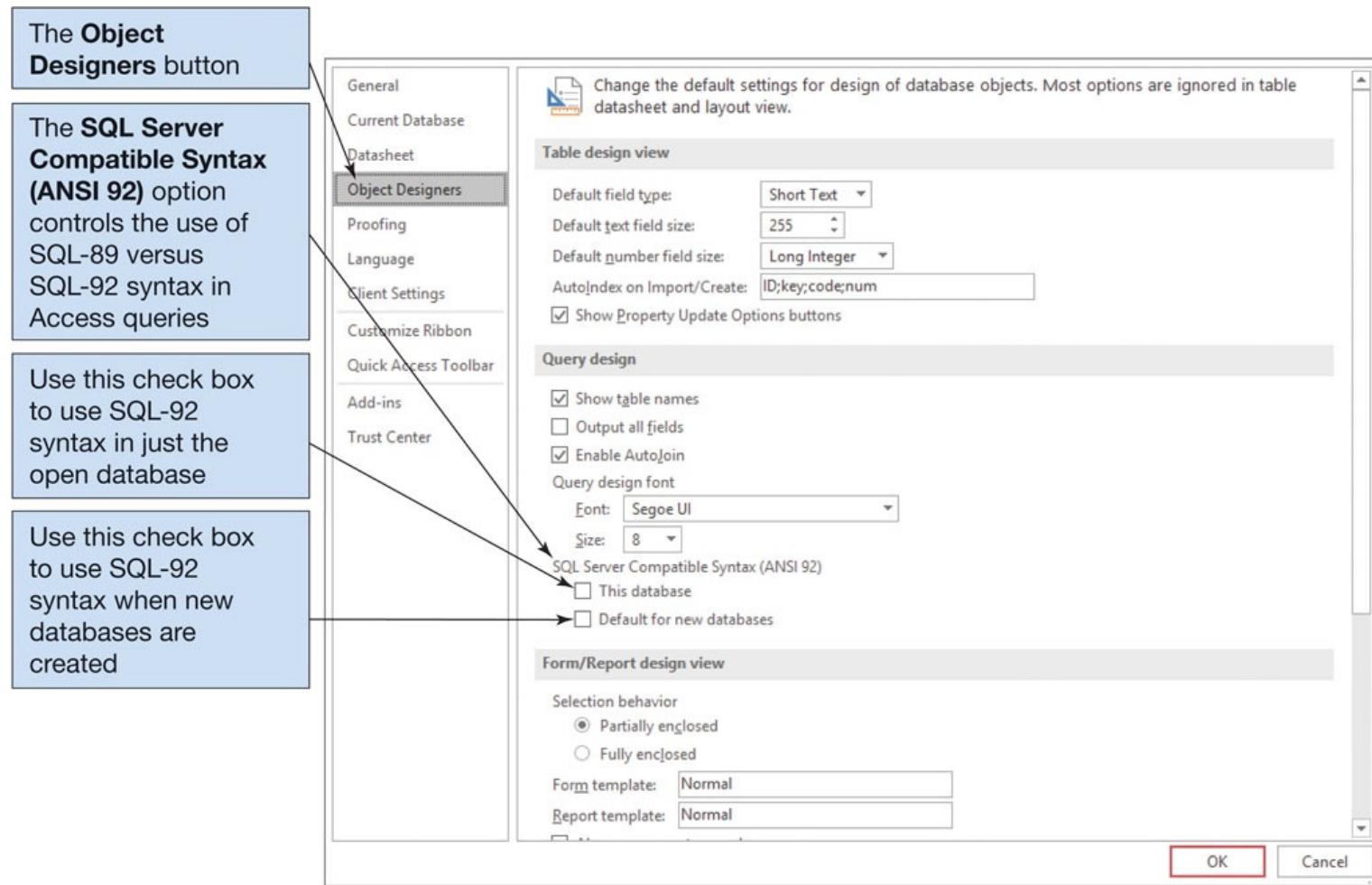
ProjectID	ProjectName	Department	MaxHours	StartDate	EndDate
1000	2019 Q3 Production Plan	Production	100.00	05/10/19	06/15/19
1100	2019 Q3 Marketing Plan	Sales and Marketing	135.00	05/10/19	06/15/19
1200	2019 Q3 Portfolio Analysis	Finance	120.00	07/05/19	07/25/19
1300	2019 Q3 Tax Preparation	Accounting	145.00	08/10/19	10/15/19
1400	2019 Q4 Production Plan	Production	100.00	08/10/19	09/15/19
1500	2019 Q4 Marketing Plan	Sales and Marketing	135.00	08/10/19	09/15/19
1600	2019 Q4 Portfolio Analysis	Finance	140.00	10/05/19	

# Figure 3.2(d) Sample Data for the WP Database: ASSIGNMENT Table

ProjectID	EmployeeNumber	HoursWorked
1000	1	30.00
1000	6	50.00
1000	10	50.00
1000	16	75.00
1000	17	75.00
1100	1	30.00
1100	6	75.00

Partial List of Data

# Figure 3.3 The Microsoft Access 2019 Options Object Designers Page



Access 2019, Windows 10, Microsoft Corporation

# SQL for Data Definition Language (DDL)

## Learn basic SQL statements for creating database structures

- The SQL data definition statements include:
  - **CREATE**
    - to create database objects
  - **ALTER**
    - to modify the structure and/or characteristics of database objects
  - **DROP**
    - to delete database objects
  - **TRUNCATE**
    - to delete table data while keeping the structure

# SQL CREATE TABLE Statement

## Learn basic SQL statements for creating database structures

- The SQL CREATE TABLE statement format is as follows:

```
CREATE TABLE NewTableName (
    ColumnName      DataType      OptionalColumnConstraints,
    ColumnName      DataType      OptionalColumnConstraints,
    ColumnName      DataType      OptionalColumnConstraints,
    optional table constraints
    ...
);
```

# Figure 3.4 SQL CREATE TABLE Statements

```
CREATE TABLE DEPARTMENT (
    DepartmentName      Char(35)          PRIMARY KEY,
    BudgetCode          Char(30)          NOT NULL,
    OfficeNumber        Char(15)          NOT NULL,
    DepartmentPhone    Char(12)          NOT NULL
);

CREATE TABLE EMPLOYEE (
    EmployeeNumber     Int               PRIMARY KEY,
    FirstName          Char(25)         NOT NULL,
    LastName           Char(25)         NOT NULL,
    Department         Char(35)         NOT NULL DEFAULT 'Human Resources',
    Position            Char(35)         NULL,
    Supervisor          Int              NULL,
    OfficePhone        Char(12)         NULL,
    EmailAddress       VarChar(100)    NOT NULL UNIQUE
);

CREATE TABLE PROJECT (
    ProjectID          Int               PRIMARY KEY,
    ProjectName        Char(50)         NOT NULL,
    Department         Char(35)         NOT NULL,
    MaxHours           Numeric(8,2)    NOT NULL DEFAULT 100,
    StartDate          Date             NULL,
    EndDate            Date             NULL
);
```

# Create a new database:

- Create the following table using SQL in the query window of MS Access:

```
CREATE TABLE DEPARTMENT (
    DepartmentName      Char (35)      PRIMARY KEY,
    BudgetCode          Char (30)      NOT NULL,
    OfficeNumber        Char (15)      NOT NULL,
    DepartmentPhone    Char (12)      NOT NULL
);
```

# Create the second table using the following method.

- DO NOT ADD THE AUTO INCREMENT

```
);  
  
CREATE TABLE EMPLOYEE(  
EmployeeNumber      Int          NOT NULL AUTO_INCREMENT,  
FirstName           Char(25)     NOT NULL,  
LastName            Char(25)     NOT NULL,  
Department          Char(35)     NOT NULL DEFAULT 'Human Resources',  
Position             Char(35)     NULL,  
Supervisor           Int          NULL,  
OfficePhone          Char(12)     NULL,  
EmailAddress         VarChar(100) NOT NULL UNIQUE,  
CONSTRAINT          EMPLOYEE_PK PRIMARY KEY(EmployeeNumber),  
CONSTRAINT          EMP_DEPART_FK FOREIGN KEY(Department)  
                      REFERENCES DEPARTMENT(DepartmentName)  
                      ON UPDATE CASCADE,  
CONSTRAINT          EMP_SUPER_FK FOREIGN KEY(Supervisor)  
                      REFERENCES EMPLOYEE(EmployeeNumber)  
);
```

# Create the remaining tables

```
CREATE TABLE PROJECT (
    ProjectID      Int          NOT NULL,
    ProjectName    Char(50)     NOT NULL,
    Department     Char(35)     NOT NULL,
    MaxHours       Numeric(8,2) NOT NULL DEFAULT 100,
    StartDate      Date        NULL,
    EndDate        Date        NULL,
    CONSTRAINT     PROJECT_PK  PRIMARY KEY (ProjectID),
    CONSTRAINT     PROJ_DEPART_FK FOREIGN KEY(Department)
                    REFERENCES DEPARTMENT(DepartmentName)
                    ON UPDATE CASCADE
);

CREATE TABLE ASSIGNMENT (
    ProjectID      Int          NOT NULL,
    EmployeeNumber Int          NOT NULL,
    HoursWorked    Numeric(6,2) NULL,
    CONSTRAINT     ASSIGNMENT_PK PRIMARY KEY (ProjectID, EmployeeNumber),
    CONSTRAINT     ASSIGN_PROJ_FK FOREIGN KEY (ProjectID)
                    REFERENCES PROJECT (ProjectID)
                    ON UPDATE NO ACTION
                    ON DELETE CASCADE,
    CONSTRAINT     ASSIGN_EMP_FK FOREIGN KEY (EmployeeNumber)
                    REFERENCES EMPLOYEE (EmployeeNumber)
                    ON UPDATE NO ACTION
                    ON DELETE NO ACTION
);
```

# Data Types

- There are many data types available in all DBMS products. To see these you can refer to page 146 in your text.
- Char for character
- Numeric for numeric data
- Dates for date

# Figure 3.6 Creating Primary Keys with SQL Table Constraints

```
CREATE TABLE DEPARTMENT(
    DepartmentName      Char(35)      NOT NULL,
    BudgetCode          Char(30)      NOT NULL,
    OfficeNumber        Char(15)      NOT NULL,
    DepartmentPhone    Char(12)      NOT NULL,
    CONSTRAINT          DEPARTMENT_PK PRIMARY KEY(DepartmentName)
);

CREATE TABLE EMPLOYEE(
    EmployeeNumber     Int          NOT NULL AUTO_INCREMENT,
    FirstName          Char(25)     NOT NULL,
    LastName           Char(25)     NOT NULL,
    Department         Char(35)     NOT NULL DEFAULT 'Human Resources',
    Position           Char(35)     NULL,
    Supervisor          Int          NULL,
    OfficePhone        Char(12)     NULL,
    EmailAddress       VarChar(100) NOT NULL UNIQUE,
    CONSTRAINT          EMPLOYEE_PK PRIMARY KEY(EmployeeNumber)
);

CREATE TABLE PROJECT (
    ProjectID          Int          NOT NULL,
    ProjectName        Char(50)     NOT NULL,
    Department         Char(35)     NOT NULL,
    MaxHours           Numeric(8,2) NOT NULL DEFAULT 100,
    StartDate          Date         NULL,
    EndDate            Date         NULL,
    CONSTRAINT          PROJECT_PK PRIMARY KEY (ProjectID)
);

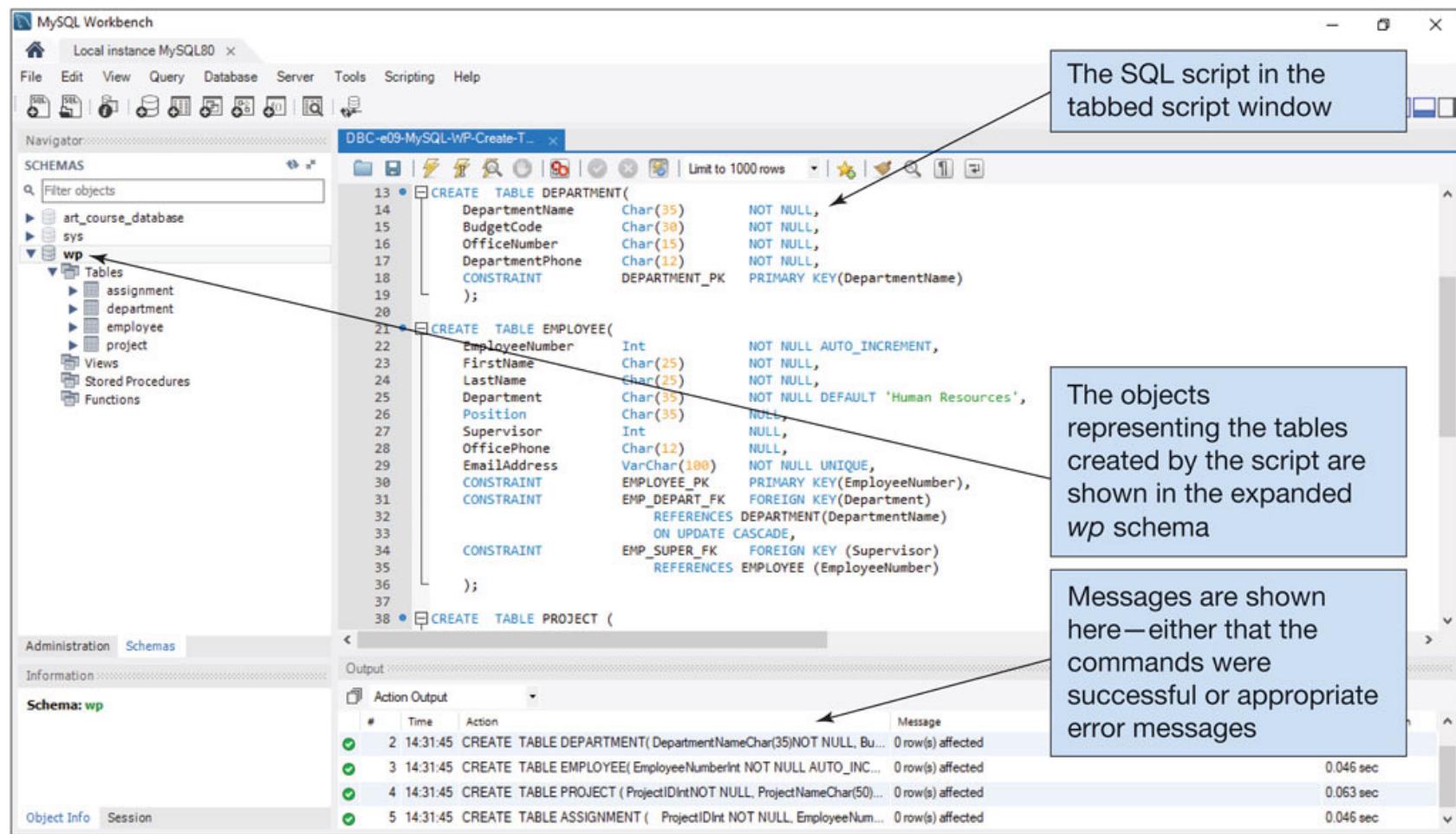
CREATE TABLE ASSIGNMENT (
    ProjectID          Int          NOT NULL,
    EmployeeNumber     Int          NOT NULL,
    HoursWorked        Numeric(6,2) NULL,
    CONSTRAINT          ASSIGNMENT_PK PRIMARY KEY (ProjectID, EmployeeNumber)
);
```

## Figure 3.7 Creating Foreign Keys with SQL Table Constraints

```
CREATE TABLE DEPARTMENT(
    DepartmentName      Char(35)      NOT NULL,
    BudgetCode          Char(30)      NOT NULL,
    OfficeNumber        Char(15)      NOT NULL,
    DepartmentPhone    Char(12)      NOT NULL,
    CONSTRAINT          DEPARTMENT_PK PRIMARY KEY(DepartmentName)
);

CREATE TABLE EMPLOYEE(
    EmployeeNumber     Int          NOT NULL AUTO_INCREMENT,
    FirstName          Char(25)      NOT NULL,
    LastName           Char(25)      NOT NULL,
    Department         Char(35)      NOT NULL DEFAULT 'Human Resources',
    Position           Char(35)      NULL,
    Supervisor          Int          NULL,
    OfficePhone        Char(12)      NULL,
    EmailAddress       VarChar(100)  NOT NULL UNIQUE,
    CONSTRAINT          EMPLOYEE_PK PRIMARY KEY(EmployeeNumber),
    CONSTRAINT          EMP_DEPART_FK FOREIGN KEY(Department)
                            REFERENCES DEPARTMENT(DepartmentName)
                            ON UPDATE CASCADE,
    CONSTRAINT          EMP_SUPER_FK FOREIGN KEY(Supervisor)
                            REFERENCES EMPLOYEE(EmployeeNumber)
);
CREATE TABLE PROJECT (
    ProjectID          Int          NOT NULL,
    ProjectName        Char(50)      NOT NULL,
    Department         Char(35)      NOT NULL,
    MaxHours           Numeric(8,2)  NOT NULL DEFAULT 100,
    StartDate          Date         NULL,
    EndDate            Date         NULL,
    CONSTRAINT          PROJECT_PK PRIMARY KEY (ProjectID),
    CONSTRAINT          PROJ_DEPART_FK FOREIGN KEY(Department)
                            REFERENCES DEPARTMENT(DepartmentName)
                            ON UPDATE CASCADE
);
CREATE TABLE ASSIGNMENT (
    ProjectID          Int          NOT NULL,
    EmployeeNumber     Int          NOT NULL,
    HoursWorked        Numeric(6,2)  NULL,
    CONSTRAINT          ASSIGNMENT_PK PRIMARY KEY (ProjectID, EmployeeNumber),
    CONSTRAINT          ASSIGN_PROJ_FK FOREIGN KEY (ProjectID)
                            REFERENCES PROJECT (ProjectID)
                            ON UPDATE NO ACTION
                            ON DELETE CASCADE,
    CONSTRAINT          ASSIGN_EMP_FK FOREIGN KEY (EmployeeNumber)
                            REFERENCES EMPLOYEE (EmployeeNumber)
                            ON UPDATE NO ACTION
                            ON DELETE NO ACTION
);
```

# Figure 3.8 Processing the SQL CREATE TABLE Statements Using MySQL 8.0



MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 3.9 Processing the SQL CREATE TABLE Statements Using MS SQL Server 2017

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, under the 'Tables' node for the 'WP' database, several tables are listed: ASSIGNMENT, DEPARTMENT, EMPLOYEE, and PROJECT. A callout points to the 'dbo' prefix, explaining it stands for 'database owner'. The main window displays two CREATE TABLE statements:

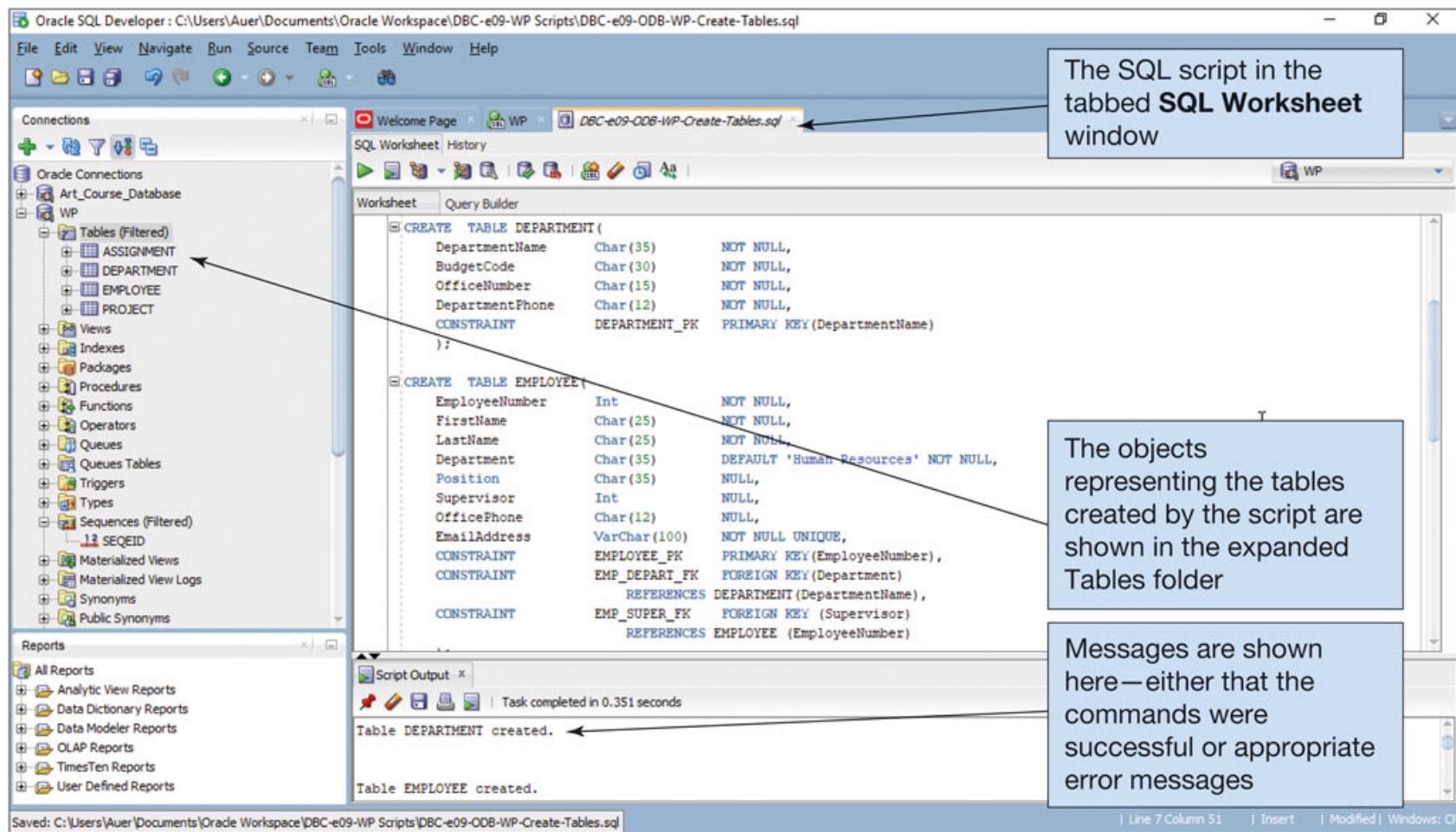
```
CREATE TABLE DEPARTMENT(
    DepartmentName Char(35) NOT NULL,
    BudgetCode Char(30) NOT NULL,
    OfficeNumber Char(15) NOT NULL,
    DepartmentPhone Char(12) NOT NULL,
    CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DepartmentName)
);

CREATE TABLE EMPLOYEE(
    EmployeeNumber Int NOT NULL IDENTITY (1, 1),
    FirstName Char(25) NOT NULL,
    LastName Char(25) NOT NULL,
    Department Char(35) NOT NULL DEFAULT 'Human Resources',
    Position Char(35) NULL,
    Supervisor Int NULL,
    OfficePhone Char(12) NULL,
    EmailAddress VarChar(100) NOT NULL UNIQUE,
    CONSTRAINT EMPLOYEE_PK PRIMARY KEY(EmployeeNumber),
    CONSTRAINT EMP_DEPART_FK FOREIGN KEY(Department) REFERENCES DEPARTMENT(DepartmentName)
        ON UPDATE CASCADE,
    CONSTRAINT EMP_SUPER_FK FOREIGN KEY (Supervisor) REFERENCES EMPLOYEE (EmployeeNumber)
);
```

A callout points to the 'Messages' pane at the bottom left, which displays the message 'Commands completed successfully.' Another callout points to the status bar at the bottom right, which shows 'Query executed successfully.'

SQL Server 2017, SQL Server Management Studio, Microsoft Corporation

# Figure 3.10 Processing the SQL CREATE TABLE Statements Using Oracle Database XE



Oracle Database XE, SQL Developer 18.4, Oracle Corporation

# SQL for Data Manipulation (DML) – Inserting Data

- SQL DML is used to query databases and to modify data in the tables.
- There are three possible data modification operations:
  - INSERT (adding data to a relation)
  - UPDATE (modifying data in a relation)
  - DELETE (deleting data in a relation)

# Inserting Data

## Learn basic SQL SELECT for adding data to a database

```
/* *** SQL-INSERT-CH03-01 *** */  
INSERT INTO DEPARTMENT VALUES('Administration',  
'BC-100-10', 'BLDG01-210, '360-285-8100');
```

The order of the columns on the INSERT command does not matter as long as the values match the order of the columns as seen below.

```
/* *** SQL-INSERT-CH03-05 *** */  
INSERT INTO EMPLOYEE(Address, FirstName, LastName,  
Department, Position, OfficePhone)  
VALUES(  
'Mary.Jacobs@WP.com', 'Mary', 'Jacobs',  
'Administration', 'CEO', '360-285-8110');
```

# The SQL Query Framework

## Learn basic SQL SELECT statements and options for processing a single table

- The basic framework for the SQL Query statement has three statements as follows:
  - the **SQL SELECT** clause
    - specifies which *columns* are to be listed in the query results
  - the **SQL FROM** clause
    - specifies which *tables* are to be used in the query
  - the **SQL WHERE** clause
    - specifies which *rows* are to be listed in the query results

# Reading Specified Columns from a Single Table

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-01 *** */
```

```
SELECT ProjectID, ProjectName, Department, MaxHours, StartDate, EndDate  
FROM PROJECT;
```

	ProjectID	ProjectName	Department	MaxHours	StartDate	EndDate
▶	1000	2019 Q3 Production Plan	Production	100.00	2019-05-10	2019-06-15
	1100	2019 Q3 Marketing Plan	Sales and Marketing	135.00	2019-05-10	2019-06-15
	1200	2019 Q3 Portfolio Analysis	Finance	120.00	2019-07-05	2019-07-25
	1300	2019 Q3 Tax Preparation	Accounting	145.00	2019-08-10	2019-10-15
	1400	2019 Q4 Production Plan	Production	100.00	2019-08-10	2019-09-15
	1500	2019 Q4 Marketing Plan	Sales and Marketing	135.00	2019-08-10	2019-09-15
	1600	2019 Q4 Portfolio Analysis	Finance	140.00	2019-10-05	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries Displaying All Columns

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-02 *** */
```

```
SELECT      *
FROM        PROJECT;
```

	ProjectID	ProjectName	Department	MaxHours	StartDate	EndDate
▶	1000	2019 Q3 Production Plan	Production	100.00	2019-05-10	2019-06-15
	1100	2019 Q3 Marketing Plan	Sales and Marketing	135.00	2019-05-10	2019-06-15
	1200	2019 Q3 Portfolio Analysis	Finance	120.00	2019-07-05	2019-07-25
	1300	2019 Q3 Tax Preparation	Accounting	145.00	2019-08-10	2019-10-15
	1400	2019 Q4 Production Plan	Production	100.00	2019-08-10	2019-09-15
	1500	2019 Q4 Marketing Plan	Sales and Marketing	135.00	2019-08-10	2019-09-15
	1600	2019 Q4 Portfolio Analysis	Finance	140.00	2019-10-05	NULL

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries Specifying Column Order

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-02 *** */  
SELECT ProjectName, Department, MaxHours  
FROM PROJECT;
```

	ProjectName	Department	MaxHours
▶	2019 Q3 Production Plan	Production	100.00
	2019 Q3 Marketing Plan	Sales and Marketing	135.00
	2019 Q3 Portfolio Analysis	Finance	120.00
	2019 Q3 Tax Preparation	Accounting	145.00
	2019 Q4 Production Plan	Production	100.00
	2019 Q4 Marketing Plan	Sales and Marketing	135.00
	2019 Q4 Portfolio Analysis	Finance	140.00

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 3.12 SQL Query Results in the MySQL Workbench

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Query' is selected. Below it, the 'Query 1' tab is active, containing the following SQL statement:

```
1 •  SELECT ProjectName, Department, MaxHours
2   FROM  PROJECT;
```

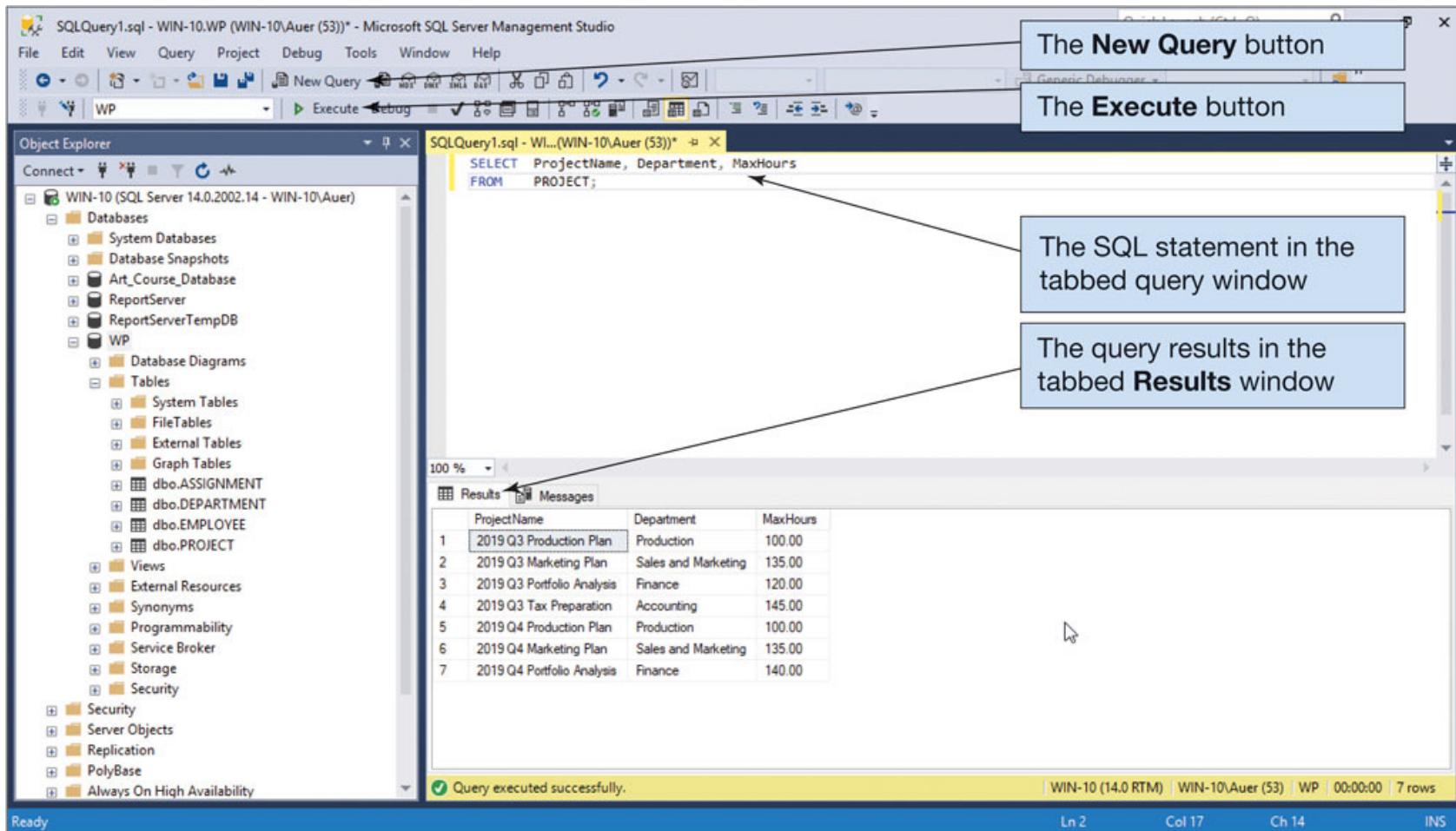
An arrow points from the 'Execute' button (a blue triangle icon) to the 'Execute' button label. Another arrow points from the 'Query 1' tab to the 'Query' statement. A third arrow points from the 'PROJECT 1' tab to the 'Results Grid' tab, which displays the following data:

ProjectName	Department	MaxHours
2019 Q3 Production Plan	Production	100.00
2019 Q3 Marketing Plan	Sales and Marketing	135.00
2019 Q3 Portfolio Analysis	Finance	120.00
2019 Q3 Tax Preparation	Accounting	145.00
2019 Q4 Production Plan	Production	100.00
2019 Q4 Marketing Plan	Sales and Marketing	135.00
2019 Q4 Portfolio Analysis	Finance	140.00

Annotations in callout boxes point to the 'Execute' button, the 'Query' statement, and the results grid.

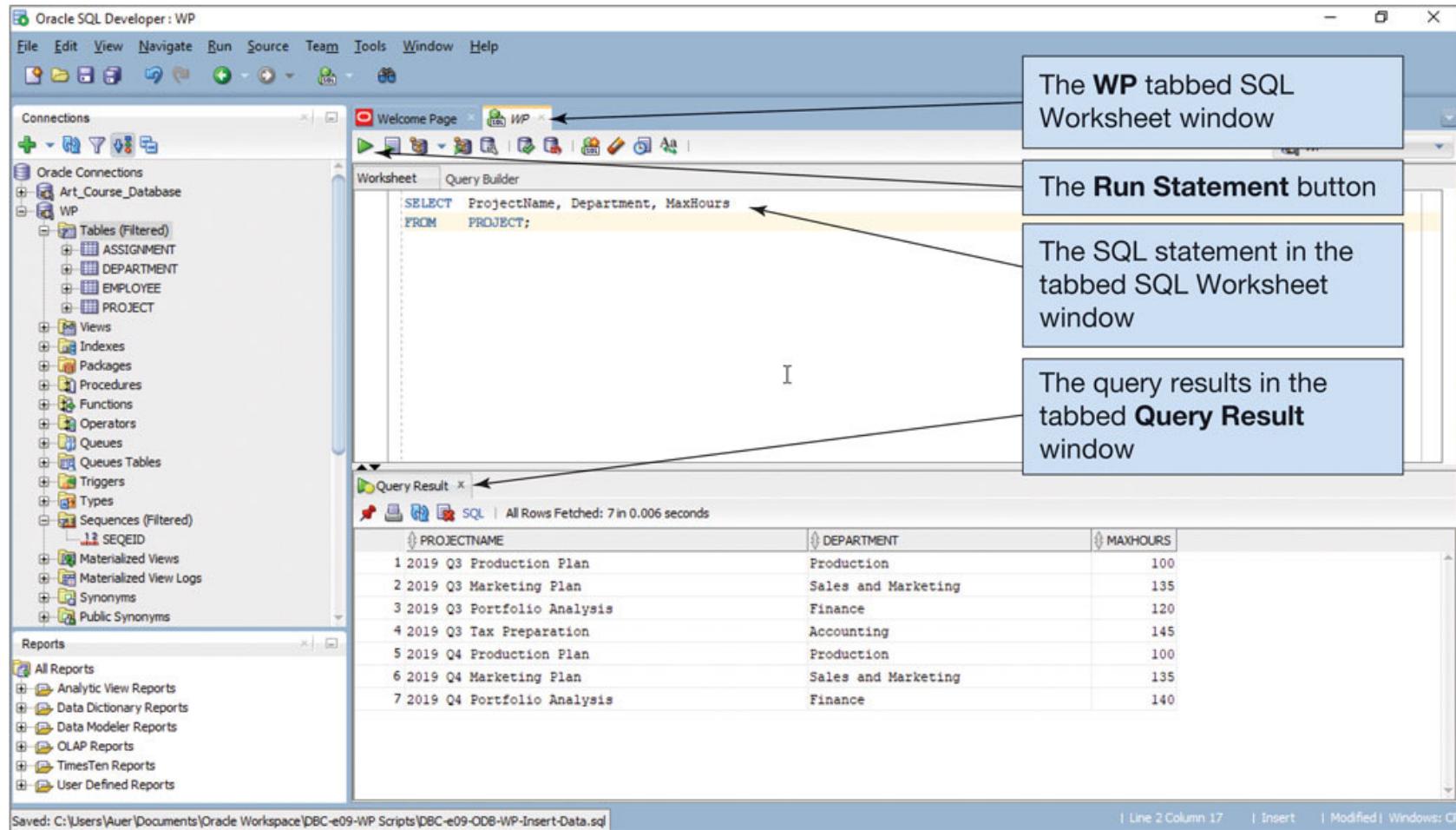
MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 3.13 SQL Query Results in the Microsoft SQL Server Management Studio



SQL Server 2017, SQL Server Management Studio, Microsoft Corporation

# Figure 3.14 SQL Query Results in the Oracle SQL Developer



Oracle Database XE, SQL Developer 18.4, Oracle Corporation

# Reading Specified Rows from a Single Table

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-05 *** */  
SELECTDepartment  
FROM PROJECT;
```

	Department
▶	Accounting
	Finance
	Finance
	Production
	Production
	Sales and Marketing
	Sales and Marketing

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries The DISTINCT Keyword

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-05 *** */  
SELECT Department  
FROM PROJECT;
```

	Department
▶	Accounting
	Finance
	Production
	Sales and Marketing

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries The WHERE Clause

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-07 *** */  
SELECT *  
FROM PROJECT  
WHERE Department= 'Finance';
```

	ProjectID	ProjectName	Department	MaxHours	StartDate	EndDate
▶	1200	2019 Q3 Portfolio Analysis	Finance	120.00	2019-07-05	2019-07-25
	1600	2019 Q4 Portfolio Analysis	Finance	140.00	2019-10-05	NULL

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Figure 3.15 SQL Comparison Operators

## SQL Comparison Operators

Operator	Meaning
=	Is equal to
<>	Is NOT Equal to
<	Is less than
>	Is greater than
<=	Is less than OR equal to
>=	Is greater than OR equal to
IN	Is equal to one of a set of values
NOT IN	Is NOT equal to any of a set of values
BETWEEN	Is within a range of numbers (includes the end points)
NOT BETWEEN	Is NOT within a range of numbers (includes the end points)
LIKE	Matches a set of characters
NOT LIKE	Does NOT match a set of characters
IS NULL	Is equal to NULL
IS NOT NULL	Is NOT equal to NULL

# Single Table Queries Using Comparison Operators

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-09 *** */  
SELECT*  
FROM PROJECT  
WHERE MaxHours > 135;
```

	ProjectID	ProjectName	Department	MaxHours	StartDate	EndDate
▶	1300	2019 Q3 Tax Preparation	Accounting	145.00	2019-08-10	2019-10-15
	1600	2019 Q4 Portfolio Analysis	Finance	140.00	2019-10-05	NULL

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries Reading Specified Columns and Rows

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-10 *** */  
SELECT FirstName, LastName, Department, OfficePhone  
FROM EMPLOYEE  
WHERE Department = 'Accounting';
```

	FirstName	LastName	Department	OfficePhone
▶	Tom	Caruthers	Accounting	360-285-8430
	Heather	Jones	Accounting	360-285-8440

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation

# Single Table Queries Sorting the Results of a Query

## Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-11 *** */  
SELECT FirstName, LastName,  
       Department, OfficePhone  
  FROM EMPLOYEE  
 ORDER BY Department;
```

By default, SQL Server sorts in ascending order. If you need the sort in descending order it would be:

```
ORDER BY Department DESC;
```

	FirstName	LastName	Department	OfficePhone
▶	Tom	Caruthers	Accounting	360-285-8430
	Heather	Jones	Accounting	360-285-8440
	Mary	Jacobs	Administration	360-285-8110
	Rosalie	Jackson	Administration	360-285-8120
	Ken	Evans	Finance	360-285-8410
	Mary	Abernathy	Finance	360-285-8420
	George	Smith	Human Resources	360-285-8310
	Alan	Adams	Human Resources	360-285-8320
	James	Nestor	InfoSystems	360-285-8610
	Rick	Brown	InfoSystems	NULL
	Richard	Bandalone	Legal	360-285-8210
	Mary	Smith	Production	360-285-8810
	Tom	Jackson	Production	360-285-8820
	George	Jones	Production	360-285-8830
	Julia	Hayakawa	Production	NULL
	Sam	Stewart	Production	NULL
	Mike	Nguyen	Research and Development	360-285-8710
	Jason	Sleeman	Research and Development	360-285-8720
	Ken	Numoto	Sales and Marketing	360-285-8510
	Linda	Granger	Sales and Marketing	360-285-8520

# Single Table Queries Sorting by Multiple Columns

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-13 *** */  
SELECT FirstName, LastName, Department, OfficePhone  
FROM EMPLOYEE  
ORDER BY Department DESC, LastName ASC;
```

The result is:

	FirstName	LastName	Department	OfficePhone
▶	Linda	Granger	Sales and Marketing	360-285-8520
	Ken	Numoto	Sales and Marketing	360-285-8510
	Mike	Nguyen	Research and Development	360-285-8710
	Jason	Sleeman	Research and Development	360-285-8720
	Julia	Hayakawa	Production	NULL
	Tom	Jackson	Production	360-285-8820
	George	Jones	Production	360-285-8830
	Mary	Smith	Production	360-285-8810
	Sam	Stewart	Production	NULL
	Richard	Bandalone	Legal	360-285-8210
	Rick	Brown	InfoSystems	NULL
	James	Nestor	InfoSystems	360-285-8610
	Alan	Adams	Human Resources	360-285-8320
	George	Smith	Human Resources	360-285-8310
	Mary	Abernathy	Finance	360-285-8420
	Ken	Evans	Finance	360-285-8410
	Rosalie	Jackson	Administration	360-285-8120
	Mary	Jacobs	Administration	360-285-8110
	Tom	Caruthers	Accounting	360-285-8430
	Heather	Jones	Accounting	360-285-8440

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Single Table Queries SQL WHERE Clause Options

Learn basic SQL SELECT statements and options for processing a single table

- Three options for the SQL WHERE clauses are:
  - compound clauses
  - ranges
  - wildcards
- An example of the compound clause is below:

```
/* *** SQL-Query-CH03-14 *** */  
SELECT FirstName, LastName, Department, OfficePhone  
FROM EMPLOYEE  
WHERE Department = 'Accounting'  
AND OfficePhone = '360-285-8430';
```

# Figure 3.16 SQL Logical Operators

## SQL Logical Operators

Operator	Meaning
AND	Both arguments are TRUE
OR	One or the other or both of the arguments are TRUE
NOT	Negates the associated operator

# Single Table Queries: SQL WHERE Clauses Using Ranges of Values

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-19 *** */  
SELECT FirstName, LastName, Department, OfficePhone  
FROM EMPLOYEE  
WHERE EmployeeNumber >= 2  
    AND EmployeeNumber <= 5;
```

	FirstName	LastName	Department	OfficePhone
▶	Rosalie	Jackson	Administration	360-285-8120
	Richard	Bandalone	Legal	360-285-8210
	George	Smith	Human Resources	360-285-8310
	Alan	Adams	Human Resources	360-285-8320

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Single Table Queries: SQL WHERE Clauses Using Wildcards

**Learn basic SQL SELECT statements and options for processing a single table**

When using an underscore it means that any character can occur in the spot occupied by the underscore.

```
/* *** SQL-Query-CH03-21 *** */  
SELECT *  
FROM   PROJECT  
WHERE  ProjectName LIKE '2019 Q_ Portfolio Analysis';
```

Notice below that an underscore is used for each space needed.

```
/* *** SQL-Query-CH03-22 *** */  
SELECT *  
FROM   EMPLOYEE  
WHERE  OfficePhone LIKE '360-285-88__';
```

# Single Table Queries: SQL WHERE Clauses That Use NULL Values

**Learn basic SQL SELECT statements and options for processing a single table**

To include or exclude rows that contain NULL values, we use the IS NULL or IS NOT NULL comparison values.

```
/* *** SQL-Query-CH03-26 *** */
```

```
SELECT FirstName, LastName, Department, OfficePhone  
FROM EMPLOYEE  
WHERE OfficePhone IS NULL;
```

```
/* *** SQL-Query-CH03-27 *** */
```

```
SELECT FirstName, LastName, Department, OfficePhone  
FROM MPLOYEE  
WHERE OfficePhone IS NOT NULL;
```

# **Single Table Queries: SQL Queries That Perform Calculations**

**Learn basic SQL SELECT statements and options for processing a single table**

- It is possible to perform certain types of calculations in SQL query statements.
  - One group of calculations involves the use of SQL built-in functions.
  - Another group involves simple arithmetic operations on the columns in the SELECT statement.

# Figure 3.17 SQL Built-in Aggregate Functions

## SQL Built-in Aggregate Functions

Function	Meaning
COUNT(*)	Count the number of rows in the table
COUNT ({Name})	Count the number of rows in the table where column {Name} IS NOT NULL
SUM	Calculate the sum of all values (numeric columns only)
AVG	Calculate the average of all values (numeric columns only)
MIN	Calculate the minimum value of all values
MAX	Calculate the maximum value of all values

# Single Table Queries: Using SQL Built-in Aggregate Functions

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-28 *** */  
SELECT COUNT(*)  
FROM PROJECT;
```

```
/* *** SQL-Query-CH03-29 *** */  
SELECT COUNT(*) AS NumberOfProjects  
FROM PROJECT;
```

NumberOfProjects	
▶	7

# Single Table Queries: Using Multiple Aggregate Functions in an SQL Statement

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-32 *** */  
SELECT  SUM(MaxHours) AS TotalMaxHours,  
        AVG(MaxHours) AS AverageMaxHours,  
        MIN(MaxHours) AS MinimumMaxHours,  
        MAX(MaxHours) AS MaximumMaxHours  
FROM    PROJECT  
WHERE   ProjectID <= 1200;
```

	TotalMaxHours	AverageMaxHours	MinimumMaxHours	MaximumMaxHours
▶	355.00	118.333333	100.00	135.00

# Single Table Queries: Using Expressions in SQL SELECT Statements

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-35 *** */  
SELECT ProjectID, ProjectName, MaxHours,  
       (24.50 * MaxHours) AS MaxProjectCost  
FROM   PROJECT;
```

	ProjectID	ProjectName	MaxHours	MaxProjectCost
▶	1000	2019 Q3 Production Plan	100.00	2450.0000
	1100	2019 Q3 Marketing Plan	135.00	3307.5000
	1200	2019 Q3 Portfolio Analysis	120.00	2940.0000
	1300	2019 Q3 Tax Preparation	145.00	3552.5000
	1400	2019 Q4 Production Plan	100.00	2450.0000
	1500	2019 Q4 Marketing Plan	135.00	3307.5000
	1600	2019 Q4 Portfolio Analysis	140.00	3430.0000

# **Single Table Queries: Grouping Rows Using SQL SELECT Statements**

**Learn basic SQL SELECT statements and options for processing a single table**

- In SQL, you can use the SQL GROUP BY clause to group rows by common values.
  - This is a powerful feature, but it can be difficult to understand.
  - What do we mean by a group? Consider the EMPLOYEE table in Figure 3.18 on the next slide, where we show the employees grouped by which department they work in.
  - Because there are 9 departments, we have the employees divided into 9 groups.

# Figure 3.18 Department Groups in the EMPLOYEE Table

EmployeeNumber	FirstName	LastName	Department	Position	Supervisor	OfficePhone	EmailAddress
1	Mary	Jacobs	Administration	CEO	HULL	360-285-8110	Mary.Jacobs@WP.com
2	Rosalie	Jackson	Administration	Admin Assistant	1	360-285-8120	Rosalie.Jackson@WP.com
3	Richard	Bandalone	Legal	Attorney	1	360-285-8210	Richard.Bandalone@WP.com
4	George	Smith	Human Resources	HR3	1	360-285-8310	George.Smith@WP.com
5	Alan	Adams	Human Resources	HR1	4	360-285-8320	Alan.Adams@WP.com
6	Ken	Evans	Finance	CFO	1	360-285-8410	Ken.Evans@WP.com
7	Mary	Abernathy	Finance	FA3	6	360-285-8420	Mary.Abernathy@WP.com
8	Tom	Caruthers	Accounting	FA2	6	360-285-8430	Tom.Caruthers@WP.com
9	Heather	Jones	Accounting	FA2	6	360-285-8440	Heather.Jones@WP.com
10	Ken	Numoto	Sales and Marketing	SM3	1	360-285-8510	Ken.Numoto@WP.com
11	Linda	Granger	Sales and Marketing	SM2	10	360-285-8520	Linda.Granger@WP.com
12	James	Nestor	InfoSystems	CIO	1	360-285-8610	James.Nestor@WP.com
13	Rick	Brown	InfoSystems	IS2	12	HULL	Rick.Brown@WP.com
14	Mike	Nguyen	Research and Development	CTO	1	360-285-8710	Mike.Nguyen@WP.com
15	Jason	Sleeman	Research and Development	RD3	14	360-285-8720	Jason.Sleeman@WP.com
16	Mary	Smith	Production	OPS3	1	360-285-8810	Mary.Smith@WP.com
17	Tom	Jackson	Production	OPS2	16	360-285-8820	Tom.Jackson@WP.com
18	George	Jones	Production	OPS2	17	360-285-8830	George.Jones@WP.com
19	Julia	Hayakawa	Production	OPS1	17	HULL	Julia.Hayakawa@WP.com
20	Sam	Stewart	Production	OPS1	17	HULL	Sam.Stewart@WP.com

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Single Table Queries: Using the GROUP BY Clause

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-36 *** */  
SELECT Department, Count(*) AS NumberOfEmployees  
FROM EMPLOYEE  
GROUP BY Department;
```

	Department	NumberOfEmployees
▶	Accounting	2
	Administration	2
	Finance	2
	Human Resources	2
	InfoSystems	2
	Legal	1
	Production	5
	Research and Development	2
	Sales and Marketing	2

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Single Table Queries: Using the GROUP BY Clause with the HAVING Clause

Learn basic SQL SELECT statements and options for processing a single table

```
/* *** SQL-Query-CH03-36 *** */  
SELECT Department, Count(*) AS NumberOfEmployees  
FROM EMPLOYEE  
GROUP BY Department  
HAVING COUNT(*) > 1;
```

	Department	NumberOfEmployees
▶	Accounting	2
	Administration	2
	Finance	2
	Human Resources	2
	InfoSystems	2
	Production	5
	Research and Development	2
	Sales and Marketing	2

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Multiple Table Queries: SQL For Data Manipulation (DML)

## Learn basic SQL SELECT Statements for processing multiple tables with subqueries

- The queries considered so far have involved data from a single table. However, at times, more than one table must be processed to obtain the desired information. SQL provides two different techniques for querying data from multiple tables:
  - the SQL Subquery
  - the SQL Join

# Multiple Table Queries: Querying with Subqueries

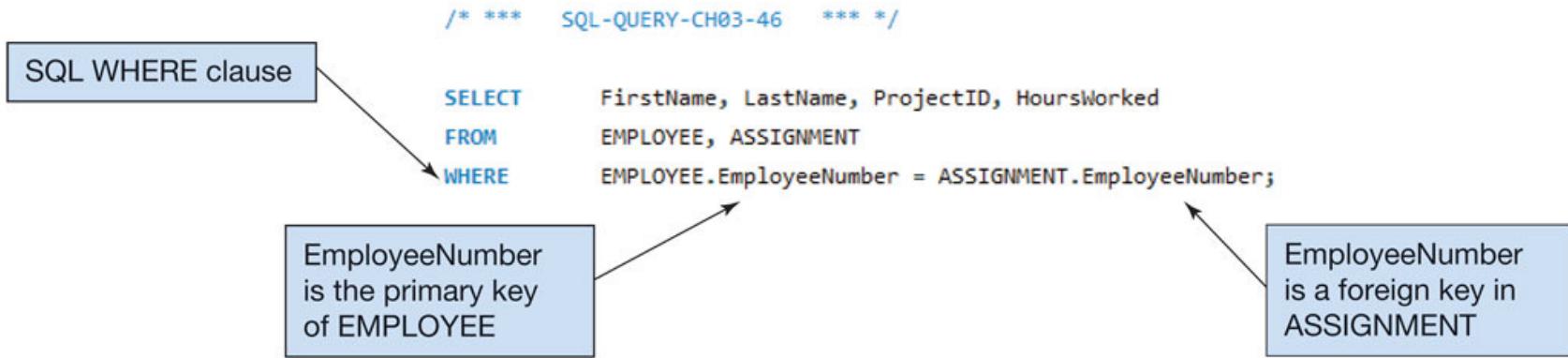
Learn basic SQL SELECT Statements for processing multiple tables with subqueries

```
/* *** SQL-Query-CH03-41 *** */  
SELECT FirstName, LastName  
FROM EMPLOYEE  
WHERE EmployeeNumber IN  
(SELECT DISTINCT EmployeeNumber  
FROM ASSIGNMENT  
WHERE HoursWorked > 50);
```

	FirstName	LastName
▶	Ken	Evans
	Ken	Numoto
	Linda	Granger
	Mary	Smith
	Tom	Jackson

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Figure 3.19 Using Primary Key and Foreign Key Values in the SQL WHERE Clause in an Implicit SQL JOIN



MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Multiple Table Queries: Querying with Joins (1 of 2)

Learn basic SQL SELECT Statements for processing multiple tables with Joins

```
/* *** SQL-Query-CH03-46 *** */  
SELECT FirstName, LastName, ProjectID, HoursWorked  
FROM EMPLOYEE, ASSIGNMENT  
WHERE EMPLOYEE.EmployeeNumber = ASSIGNMENT.EmployeeNumber;
```

	FirstName	LastName	ProjectID	HoursWorked
▶	Mary	Jacobs	1000	30.00
	Mary	Jacobs	1100	30.00

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Multiple Table Queries: Querying with Joins (2 of 2)

## Learn basic SQL SELECT Statements for processing multiple tables with Joins

```
/* *** SQL-Query-CH03-46 *** */  
SELECT FirstName, LastName, ProjectID, HoursWorked  
FROM EMPLOYEE, ASSIGNMENT  
WHERE EMPLOYEE.EmployeeNumber = ASSIGNMENT.EmployeeNumber;
```

	FirstName	LastName	ProjectID	HoursWorked
▶	Mary	Jacobs	1000	30.00
	Mary	Jacobs	1100	30.00
	Mary	Jacobs	1400	30.00
	Mary	Jacobs	1500	30.00

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

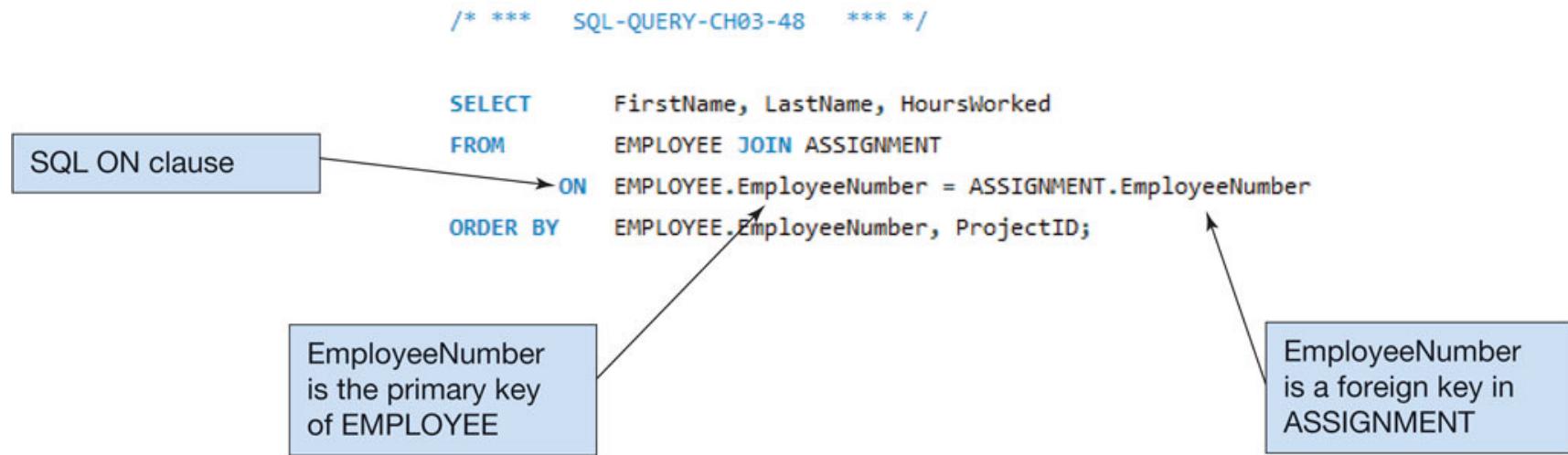
# Multiple Table Queries: Using the JOIN ON with the GROUP BY

## Learn basic SQL SELECT Statements for processing multiple tables with Joins

The JOIN ON syntax still requires a statement of primary key to foreign key as shown below:

```
/* *** SQL-Query-CH03-49 *** */  
SELECT FirstName, LastName, SUM(HoursWorked) AS TotalHoursWorked  
FROM EMPLOYEE AS E JOIN ASSIGNMENT AS A  
    ON E.EmployeeNumber = A.EmployeeNumber  
GROUP BY LastName, FirstName  
ORDER BY LastName, FirstName;
```

# Figure 3.22 Using Primary Key and Foreign Key Values in the SQL ON Clause in an Explicit SQL Join



MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Multiple Table Queries: Comparing Subqueries and Joins

**Learn basic SQL SELECT Statements for processing multiple tables with Joins**

- Subqueries and joins both process multiple tables, but they differ slightly.
  - A subquery can only be used to retrieve data from the top table
  - A join can be used to obtain data from any number of tables.

# Multiple Table Queries: SQL Inner Joins

## Learn basic SQL SELECT Statements for processing multiple tables with Joins

- SQL Inner Join is also referred to as an SQL equijoin.
- An Inner Join only displays data from the rows that match based on join conditions:
  - if a row has a value that does not match the WHERE clause condition, that row will not be included in the join result

# Figure 3.25 Types of SQL JOINS

STUDENT			LOCKER	
StudentPK	StudentName	LockerFK	LockerPK	LockerType
1	Adams	NULL		
2	Buchanan	NULL		
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half

StudentPK	StudentName	LockerFK	LockerPK	LockerType
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half

StudentPK	StudentName	LockerFK	LockerPK	LockerType
1	Adams	NULL	NULL	NULL
2	Buchanan	NULL	NULL	NULL
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half

StudentPK	StudentName	LockerFK	LockerPK	LockerType
3	Carter	10	10	Full
4	Ford	20	20	Full
5	Hoover	30	30	Half
6	Kennedy	40	40	Full
7	Roosevelt	50	50	Full
8	Truman	60	60	Half
NULL	NULL	NULL	70	Full
NULL	NULL	NULL	80	Full
NULL	NULL	NULL	90	Half

# **SQL for Data Manipulation (DML)– Data Modification and Deletion**

**Learn basic SQL for modifying and deleting  
data from a database**

- The SQL DML contains commands for the three possible data modifications operations:
  - Insert
  - Modify
  - Delete

# Modifying Data Example 1

## Learn basic SQL for modifying and deleting data from a database

```
/* *** SQL-UPDATE-CH03-01 *** */  
UPDATE      EMPLOYEE  
SET          OfficePhone = '360-285-8620'  
WHERE        EmployeeNumber = 13;
```

To see the results, we use the following command:

```
/* *** SQL-Query-CH03-58 *** */  
SELECT      *  
FROM        EMPLOYEE  
WHERE        EmployeeNumber = 13;
```

	EmployeeNumber	FirstName	LastName	Department	Position	Supervisor	OfficePhone	EmailAddress
▶	13	Rick	Brown	InfoSystems	IS2	12	360-285-8620	Rick.Brown@WP.com

MySQL Community Server 8.0, MySQL Workbench, Oracle Corporation.

# Modifying Data Example 2

## Learn basic SQL for modifying and deleting data from a database

```
/* *** EXAMPLE CODE - DO NOT RUN *** */  
/* *** SQL-UPDATE-CH03-03 *** */  
UPDATE      EMPLOYEE  
SET          Department = 'Finance', OfficePhone = '360-285-8420'  
WHERE        EmployeeNumber = 9;
```

# Deleting Data

## Learn basic SQL for modifying and deleting data from a database

```
/* *** EXAMPLE CODE - DO NOT RUN *** */  
/* *** SQL-DELETE-CH03-01 *** */  
DELETE  
FROM      PROJECT  
WHERE     Department = 'Sales and Marketing';
```

The example above is a valid statement, but note that if you fail to include the WHERE clause then you will have just deleted all records in the table.

# **SQL For Data Definition (DDL) – Table and Constraint Modification and Deletion**

**Learn basic SQL statements for modifying and  
deleting database tables and constraints**

- Two of the most useful data definition SQL statements are:
  - the SQL DROP TABLE statement
  - the SQL ALTER TABLE statement

# The SQL DROP TABLE Statement

**Learn basic SQL statements for modifying and deleting database tables and constraints**

```
/* *** SQL-DROP-TABLE-CH03-01 *** */
DROP TABLE ASSIGNMENT;
```

# The SQL ALTER TABLE Statement

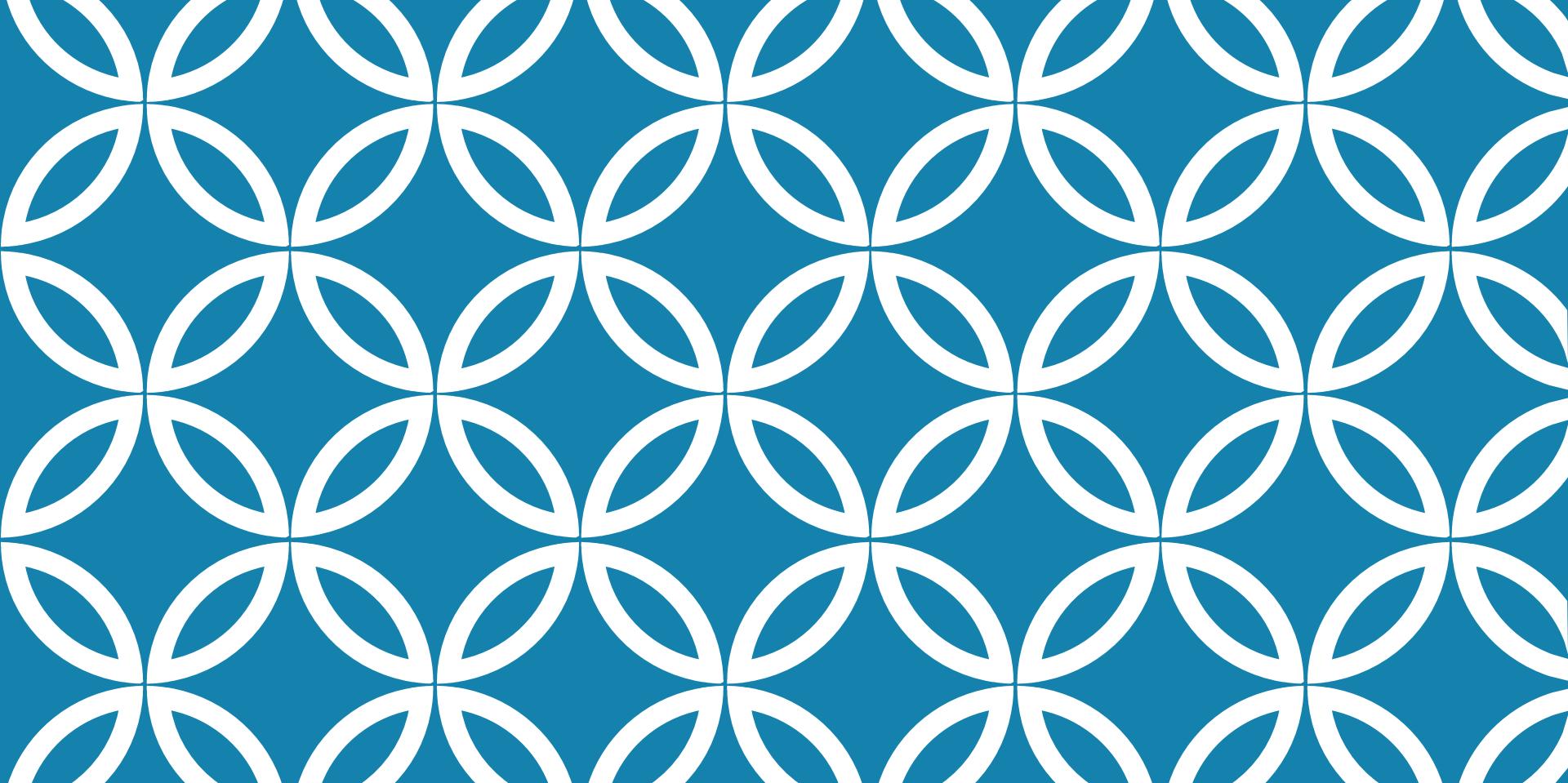
Learn basic SQL statements for modifying and deleting database tables and constraints

```
/* *** SQL-ALTER-TABLE-CH03-01 *** */  
ALTER TABLE ASSIGNMENT  
    DROP CONSTRAINT ASSIGN_EMP_FK;
```

# Copyright



**This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.**



# MORE SQL DATA DEFINITION

Database Systems

# IN THIS LECTURE

## More SQL

- DROP TABLE
- ALTER TABLE
- INSERT, UPDATE, and DELETE
- Data dictionary
- Sequences

## For more information

- Connolly and Begg chapters 5 and 6

# CREATING TABLES

From last lecture...

- CREATE TABLE
- Columns
  - Data types
  - [NOT] NULL, DEFAULT values
- Constraints
  - Primary keys
  - Unique columns
  - Foreign keys

```
CREATE TABLE  
  <name> (  
    <col-def-1>,  
    <col-def-2>,  
    :  
    <col-def-n>,  
    <constraint-1>,  
    :  
    <constraint-k>)
```

# DELETING TABLES

To delete a table use

**DROP TABLE**

**[ IF EXISTS ]**

**<name>**

Example:

**DROP TABLE Module**

**BE CAREFUL** with any SQL statement with DROP in it

- You will delete any information in the table as well
- You won't normally be asked to confirm
- There is no easy way to undo the changes

# CHANGING TABLES

Sometimes you want to change the structure of an existing table

- One way is to **DROP** it then rebuild it
- This is dangerous, so there is the **ALTER TABLE** command instead

**ALTER TABLE** can

- Add a new column
- Remove an existing column
- Add a new constraint
- Remove an existing constraint

# ALTERING COLUMNS

To add or remove columns use

```
ALTER TABLE <table>
ADD COLUMN <col>
```

```
ALTER TABLE <table>
DROP COLUMN <name>
```

Examples

```
ALTER TABLE Student
ADD COLUMN
Degree VARCHAR(50)
```

```
ALTER TABLE Student
DROP COLUMN Degree
```

# ALTERING CONSTRAINTS

To add or remove columns use

```
ALTER TABLE <table>
    ADD CONSTRAINT
        <definition>
```

```
ALTER TABLE <table>
    DROP CONSTRAINT
        <name>
```

Examples

```
ALTER TABLE Module
    ADD CONSTRAINT
        ck UNIQUE (title)
```

```
ALTER TABLE Module
    DROP CONSTRAINT ck
```

# INSERT, UPDATE, DELETE

**INSERT** - add a row to a table

**UPDATE** - change row(s) in a table

**DELETE** - remove row(s) from a table

**UPDATE** and **DELETE** use '**WHERE clauses**' to specify which rows to change or remove

BE CAREFUL with these - an incorrect **WHERE clause** can destroy lots of data

# INSERT

**INSERT INTO**

**<table>**

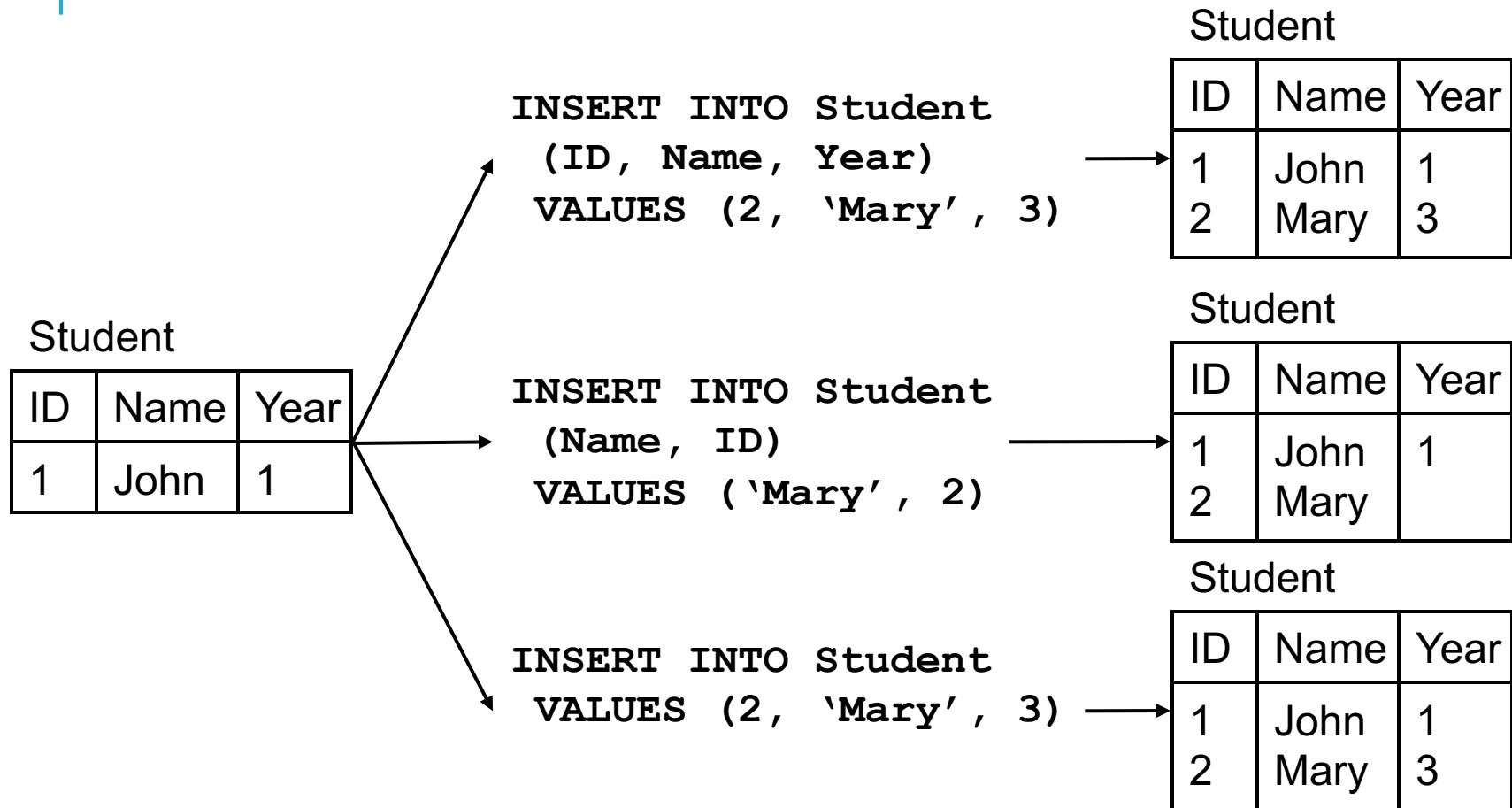
**(col1, col2, ...)**

**VALUES**

**(val1, val2, ...)**

- The number of columns and values must be the same
- If you are adding a value to every column, you don't have to list them
- SQL doesn't require that all rows are different (unless a constraint says so)

# INSERT



# UPDATE

**UPDATE <table>**

**SET col1 = val1**

**[ ,col2 = val2...]**

**[WHERE**

**<condition>]**

- All rows where the condition is true have the columns set to the given values
- If no condition is given all rows are changed so BE CAREFUL
- Values are constants or can be computed from columns

# UPDATE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
UPDATE Student  
SET Year = 1,  
    Name = 'Jane'  
WHERE ID = 4
```

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Jane	1

```
UPDATE Student  
SET Year = Year + 1
```

Student

ID	Name	Year
1	John	2
2	Mark	4
3	Anne	3
4	Mary	3

# **DELETE**

**Removes all rows which satisfy the condition**

```
DELETE FROM  
<table>  
[WHERE  
<condition>]
```

- If no condition is given then ALL rows are deleted - BE CAREFUL
- Some versions of SQL also have **TRUNCATE TABLE <T>** which is like **DELETE FROM <T>** but it is quicker as it doesn't record its actions

# DELETE

Student

ID	Name	Year
1	John	1
2	Mark	3
3	Anne	2
4	Mary	2

```
DELETE FROM  
Student  
WHERE Year = 2
```

Student

ID	Name	Year
1	John	1
2	Mark	3

```
DELETE FROM Student  
or  
TRUNCATE TABLE Student
```

Student

ID	Name	Year

# SELECT

The SQL command you will use most often

- Queries a set of tables and returns results as a table
- Lots of options, we will look at many of them
- Usually more than one way to do any given query

# SQL SELECT OVERVIEW

**SELECT**

[DISTINCT | ALL] <column-list>

FROM <table-names>

[WHERE <condition>]

[ORDER BY <column-list>]

[GROUP BY <column-list>]

[HAVING <condition>]

( [ ] - optional, | - or )

# SIMPLE SELECT

```
SELECT <columns>  
FROM <table>
```

**<columns>** can be

- A single column
- A comma-separated list of columns
- \* for ‘all columns’

Given a table Student with columns

- stuID
- stuName
- stuAddress
- stuYear

# SAMPLE SELECTS

```
SELECT * FROM Student
```

<i>stuID</i>	<i>stuName</i>	<i>stuAddress</i>	<i>stuYear</i>
1	Anderson	15 High St	1
2	Brooks	27 Queen's Rd	3
3	Chen	Lenton Hall	1
4	D' Angelo	Derby Hall	1
5	Evans	Lenton Hall	2
6	Franklin	13 Elm St	3
7	Gandhi	Lenton Hall	1
8	Harrison	Derby Hall	1

# SAMPLE SELECTS

```
SELECT stuName FROM Student
```

stuName
Anderson
Brooks
Chen
D' Angelo
Evans
Franklin
Gandhi
Harrison

# SAMPLE SELECTS

```
SELECT stuName, stuAddress  
FROM Student
```

<i>stuName</i>	<i>stuAddress</i>
Anderson	15 High St
Brooks	27 Queen's Rd
Chen	Lenton Hall
D' Angelo	Derby Hall
Evans	Lenton Hall
Franklin	13 Elm St
Gandhi	Lenton Hall
Harrison	Derby Hall

# BEING CAREFUL

## When using DELETE and UPDATE

- You need to be careful to have the right WHERE clause
- You can check it by running a SELECT statement with the same WHERE clause first

Before running

**DELETE FROM Student**

**WHERE Year = 3**

run

**SELECT \* FROM Student**

**WHERE Year = 3**

# ORACLE DATA DICTIONARY

To find out what tables and sequences you have defined use

```
SELECT table_name  
      FROM user_tables
```

- The user\_tables table is maintained by Oracle
- It has *lots* of columns, so don't use

```
SELECT * FROM user_tables
```

# ORACLE DATA DICTIONARY

To find the details of a table use

```
DESCRIBE <table name>
```

**Example:**

```
SQL> DESCRIBE Student;
```

Name	Null?	Type
STUID	NOT NULL	NUMBER (38)
STUNAME	NOT NULL	VARCHAR2 (50)
STUADDRESS		VARCHAR2 (50)
STUYEAR		NUMBER (38)

# EXERCISE

Track

cID	Num	Title	Time	aID
1	1	Reason	239	1
1	2	I love you	410	1
1	3	Breathless	217	1
1	4	I am no 4	279	1
2	1	Lion	362	1
2	2	Catman	417	2

CD

cID	Title	Price
1	Mix	19.99
2	Compilation	11.99

Artist

aID	Name
1	Jango
2	Ranna

1. Create the tables above using the create table query
2. Add the appropriate constraints.
3. Insert data using the insert query
4. Demonstrate a delete and edit query

# PROVIDE THE SQL STATEMENTS FOR THE FOLLOWING

1. All tracks having the number 1.
2. Track with the title “Lion”
3. CD with the price of below 12 dollars.
4. Artist named Jango
5. Track with time range of 200 to 300

# Oracle Data Definition Language (DDL)

\*Data types

\* Constraints

# Oracle

- ▶ Learn about Data Definition Language (DDL) statements to work with the structure of an Oracle database table.
  - ▶ Various data types used in defining columns in a database table.
  - ▶ Integrity and value constraints
  - ▶ Viewing, modifying, and removing a table structure.

# NAMING RULES AND CONVENTIONS

- ▶ A table is an object that can store data in an Oracle database.
- ▶ When you create a table, you must specify
  1. the table name,
  2. the name of each column,
  3. the data type of each column,
  4. and the size of each column.

# NAMING RULES AND CONVENTIONS

- ▶ Oracle provides you with different constraints
  - ▶ to specify a primary or a composite key for the table,
  - ▶ to define a foreign key in a table that references a primary key in another table,
  - ▶ to set data validation rules for each column,
  - ▶ to specify whether a column allows NULL values,
  - ▶ and to specify if a column should have unique values only.

# Data Types

- ▶ When a table is created, each column in the table is assigned a data type.
- ▶ Some important data types:
  - ▶ Varchar2
  - ▶ Char
  - ▶ Number

# Varchar2

- ▶ The VARCHAR2 type is a character data type to store **variable-length** alphanumeric data in a column.
- ▶ The size is specified within parentheses, for example, VARCHAR2(20).
- ▶ If the data are smaller than the specified size, only the data value is stored, and trailing spaces are not added to the value.
- ▶ VARCHAR2 is the most appropriate type for a column whose values do not have a fixed length.

# Char

- ▶ The CHAR type is a character data type to store **fixed-length** alphanumeric data in a column.
- ▶ The CHAR data type uses the storage more efficiently and processes data faster than the VARCHAR2 type.

# Number

- ▶ The NUMBER data type is used to store negative, positive, integer, fixed-decimal, and floating-point numbers.
- ▶ When a number type is used for a column, its **precision** and **scale** can be specified.
  - ▶ Precision is the total number of significant digits in the number, both to the left and to the right of the decimal point.
  - ▶ Scale is the total number of digits to the right of the decimal point.

# Number -- integer

- ▶ An **integer** is a whole number without any decimal part.
- ▶ The data type for it would be defined as NUMBER(3), where 3 represents the maximum number of digits.

# Number - fixed-point

- ▶ decimal number has a specific number of digits to the right of the decimal point.
- ▶ The PRICE column has values in dollars and cents, which requires two decimal places - for example, values like 2.95, 3.99, 24.99, and so on.
- ▶ If it is defined as NUMBER(4,2), the first number specifies the precision and the second number the scale.

# Number - floating-point

- ▶ A **floating-point** decimal number has a variable number of decimal places
- ▶ To define such a column, do not specify the scale or precision along with the NUMBER type.
- ▶ By defining a column as a floating-point number, a value can be stored in it with very high precision

# Numbers

Data Type	Description	MAX Size - Oracle 9i	MAX Size - Oracle 10g	MAX Size - Oracle 11g	MAX Size - Oracle 12c	MAX Size - PL/SQL
NUMBER(p,s)	Numeric data, with a precision of p and scale of s.	Precision p ranges from 1 to 38, and scale ranges from -84 to 127	Precision p ranges from 1 to 38, and scale ranges from -84 to 127	Precision p ranges from 1 to 38, and scale ranges from -84 to 127	Precision p ranges from 1 to 38, and scale ranges from -84 to 127	Precision p ranges from 1 to 38, and scale ranges from -84 to 127
BINARY_FLOAT	32-bit, single-precision floating point number	From 1.17549E-38F to 3.40282E+38F	n/a			
BINARY_DOUBLE	64-bit, double-precision floating point number	From 2.22507485850720E-308 to 1.79769313486231E+308	n/a			
BOOLEAN	True, False, or NULL	n/a	n/a	n/a	n/a	n/a
PLS_INTEGER	Signed integer.	n/a	n/a	n/a	n/a	From -2,147,483,647 to 2,147,483,647
BINARY_INTEGER	Signed integer. Older, slower version of PLS_INTEGER	n/a	n/a	n/a	n/a	From -2,147,483,647 to 2,147,483,647
INTEGER	Translated to NUMBER(38)	n/a	n/a	n/a	n/a	
FLOAT	Translated to NUMBER	1 to 126 binary digits, and up to 22 bytes	1 to 126 binary digits, and up to 22 bytes	1 to 126 binary digits, and up to 22 bytes	1 to 126 binary digits, and up to 22 bytes	
DECIMAL	Translated to NUMBER	n/a	n/a	n/a	n/a	

# Characters/Text

Data Type	Description	MAX Size - Oracle 9i	MAX Size - Oracle 10g	MAX Size - Oracle 11g	MAX Size - Oracle 12c	MAX Size - PL/SQL
CHAR(size)	Fixed length character with a length of size.	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	32,767 bytes. Default and minimum is 1 byte
NCHAR(size)	Fixed length national character with a length of size.	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	2000 bytes. Default and minimum is 1 byte	32,767 bytes. Default and minimum is 1 byte
VARCHAR	Deprecated and only used for backward compatibility.					
VARCHAR2(size)	Variable length character string with a maximum length of size bytes	4000 bytes. Minimum is 1 byte	4000 bytes. Minimum is 1 byte	4000 bytes. Minimum is 1 byte	32,767 bytes	32,767 bytes. Minimum is 1 byte
NVARCHAR2(size)	Variable length national character string with a maximum length of size bytes	4000 bytes. Minimum is 1 byte	4000 bytes. Minimum is 1 byte	4000 bytes. Minimum is 1 byte	32,767 bytes	32,767 bytes. Minimum is 1 byte
LONG	Variable length character string. Larger than VARCHAR2. Deprecated	2 GB	2 GB	2 GB	2 GB	32,760 bytes.
RAW(size)	Raw binary data of length size	2000 bytes	2000 bytes	2000 bytes	32,767 bytes	32,767 bytes
LONG RAW	Raw binary data of variable length. Deprecated	2 GB	2 GB	2 GB	2 GB	32,760 bytes.

# Types of Constraints

There are two types of constraints:

1. *Integrity constraints*: define both the primary key and the foreign key with the table and primary key it references.
2. *Value constraints*: define if NULL values are disallowed, if UNIQUE values are required, and if only certain set of values are allowed in a column.

# Naming a Constraint

- ▶ The general convention used for naming constraints is  
*<table name>\_<column name>\_<constraint type>*
  
- ▶ *table name* is the name of the table where the constraint is being defined,
- ▶ *column name* is the name of the column to which the constraint applies,
- ▶ and *constraint type* is an abbreviation used to identify the constraint's type.

# Naming a Constraint

For example, a constraint name *emp\_deptno\_fk* refers to:

- ▶ a constraint in table EMP on column DeptNo of type foreign key. A constraint name *dept\_deptno\_pk* is for a primary key constraint in table DEPT on column DeptNo.

# Popular Constraint abbreviations

- ▶ Primary Key        pk
- ▶ Foreign Key       fk
- ▶ Unique              uk
- ▶ Check               ck
- ▶ Not Null           nn

# Defining a Constraint

- ▶ A constraint can be created at the same time the table is created, or it can be added to the table afterward. There are two levels where a constraint is defined:
  - ▶ Column level.
  - ▶ Table level.

# Column level

- ▶ A column-level constraint references a single column and is defined along with the definition of the column.
- ▶ Any constraint can be defined at the column level except for a FOREIGN KEY and COMPOSITE primary key constraints.

Column datatype [CONSTRAINT constraint\_name] constraint\_type

Example:

Building VARCHAR2(7) CONSTRAINT location\_building\_nn NOT NULL

# Table level

- ▶ A table-level constraint references one or more columns and is defined separately from the definitions of the columns.
- ▶ Normally, it is written after all columns are defined.
- ▶ All constraints can be defined at the table level except for the NOT NULL constraint.

[CONSTRAINT constraint\_name] constraint\_typ (Column, . . .),

Example:

CONSTRAIN location\_roomid\_pk PRIMARY KEY(Roomid)

# The Primary Key Constraint

- ▶ The PRIMARY KEY constraint is also known as the **entity integrity constraint**
- ▶ It creates a primary key for the table. A table can have only one primary key constraint.
- ▶ If a table uses more than one column as its primary key (i.e., a composite key), the key can only be declared at the table level.

# The Primary Key Constraint

- ▶ At the column level, the constraint is defined by

```
DeptId NUMBER (2) CONSTRAINT dept_deptid_pk PRIMARY  
KEY,
```

- ▶ At the table level, the constraint is defined by

```
CONSTRAINT dept_deptid_pk PRIMARY KEY(DeptId),
```

# The FOREIGN KEY Constraint

- ▶ The FOREIGN KEY constraint is also known as the **referential integrity constraint**.
- ▶ It uses a column or columns as a foreign key, and it establishes a relationship with the primary key of the same or another table.

# The FOREIGN KEY Constraint

- ▶ To establish a foreign key in a table, the other referenced table and its primary key must already exist.
- ▶ Foreign key and referenced primary key columns need not have the same name, but a foreign key value **must match** the value in the parent table's primary key value or be NULL

# The FOREIGN KEY Constraint

- ▶ At the table level ONLY

```
CONSTRAINT student_facultyid_fk FOREIGN  
KEY(FacultyId)
```

```
REFERENCES faculty (FacultyId),
```

# The NOT NULL Constraint

- ▶ The NOT NULL constraint ensures that the column has a value and the value is not a null value
- ▶ A space or a numeric zero is not a null value
- ▶ At the column level **ONLY**, the constraint is defined by:

```
Name VARCHAR2(15) CONSTRAINT faculty_name_nn NOT  
NULL,
```

# The UNIQUE Constraint

- ▶ The UNIQUE constraint requires that every value in a column or set of columns be unique.
- ▶ At the table level, the constraint is defined by  
`CONSTRAINT dept_deptname_uk UNIQUE(DeptName),`
- ▶ At the column level, the constraint is defined by:  
`DeptName VARCHAR2(12) CONSTRAINT  
dept_deptname_uk UNIQUE,`

# The CHECK Constraint

- ▶ The CHECK constraint defines a condition that every row must satisfy
- ▶ At the column level, the constraint is defined by  
**DeptId NUMBER(2) CONSTRAINT dept\_deptid\_cc**  
**CHECK((DeptId >= 10) and (DeptId <= 99)),**
- ▶ At the table level, the constraint is defined by:  
**CONSTRAINT dept\_deptid\_cc**  
**CHECK((DeptId >= 10) and (DeptId <= 99)),**

# CREATING AN ORACLE TABLE

A table is created as soon as the CREATE statement is successfully executed by the Oracle server. The general syntax of CREATE TABLE statement is

*CREATE TABLE [schema.] tablename*

*(column1 datatype [CONSTRAINT constraint\_name]  
constraint\_type . . . ,*

*(column2 datatype [CONSTRAINT constraint\_name]  
constraint\_type ,*

*[CONSTRAINT constraint\_name] constraint\_type  
(column, . . . ), . . . );*

# Create Table example

```
CREATE TABLE student
  (StudentId CHAR (5),
   Last      VARCHAR2 (15) CONSTRAINT student_last_nn NOT NULL,
   First     VARCHAR2 (15) CONSTRAINT student_first_nn NOT NULL,
   Street    VARCHAR2 (25),
   City      VARCHAR2 (15),
   State     CHAR (2) DEFAULT 'NJ',
   Zip       CHAR (5),
   StartTerm CHAR (4),
   BirthDate DATE,
   FacultyId NUMBER (3),
   MajorId   NUMBER (3),
   Phone     CHAR (10),
   CONSTRAINT student_studentid_pk PRIMARY KEY (StudentID));
```

# Viewing a Table's Structure

- ▶ The SQL\*Plus command to view a table's structure is **DESCRIBE**, which does not need a semicolon at the end because it is not a SQL statement.

```
SQL> DESCRIBE student
```

# Adding a New Column to an Existing Table

- ▶ The general syntax to add a column to an existing table is

*ALTER TABLE tablename*

*ADD columnname datatype;*

```
SQL> ALTER TABLE student  
2 ADD SocialSecurity CHAR(9);  
Table altered.  
SQL>
```

# Modifying an Existing Column

- ▶ The general syntax to modify an existing column is

*ALTER TABLE tablename*

*MODIFY columnname newdatatype;*

where *newdatatype* is the new data type or the new size for the column.

```
SQL> ALTER TABLE student  
2  MODIFY SocialSecurity VARCHAR2(11);  
Table altered.  
SQL>
```

# Adding a Constraint

- ▶ To add a constraint using ALTER TABLE, the syntax for table level constraint is used. The general syntax of ALTER TABLE is

*ALTER TABLE tablename*

*ADD [CONSTRAINT constraint\_name] constraint\_type  
(column, ...),*

```
SQL> ALTER TABLE COURSE
  2  ADD CONSTRAINT COURSE_PREREQ_FK FOREIGN KEY (PREREQ)
  3      REFERENCES COURSE(COURSEID) ;
Table altered.
SQL>
```

# Displaying Table Information

- ▶ When a user creates a table or many tables in the database, Oracle tracks them using its own data dictionary

- ▶ Viewing a User's Table Names

```
SELECT TABLE_NAME FROM USER_TABLES;
```

- ▶ To display all information:

```
SELECT * FROM USER_TABLES;
```

# Dropping a Column

- ▶ The general syntax is

```
ALTER TABLE tablename DROP COLUMN columnname;
```

# Dropping a Table

- ▶ The general syntax is  
***DROP TABLE tablename [CASCADE CONSTRAINTS];***
- ▶ For example,  
**DROP TABLE sample;**
- ▶ Oracle displays a “Table dropped” message when a table is successfully dropped.
- ▶ If you add optional CASCADE CONSTRAINTS clause, it removes foreign key references to the table also.

# Oracle SQL

## SQL SELECT Statements

# Objectives

- Identify keywords, mandatory clauses, and optional clauses in a SELECT statement
- Select and view all columns of a table
- Select and view one column of a table
- Display multiple columns of a table
- Use a WHERE clause to restrict the rows returned by a query
- Create a search condition using mathematical comparison operators
- Use the BETWEEN...AND comparison operator to identify records within a range of values
- Specify a list of values for a search condition using the IN comparison operator

# Objectives (continued)

- Use a column alias to clarify the contents of a particular column
- Perform basic arithmetic operations in the SELECT clause
- Remove duplicate lists using either the DISTINCT or UNIQUE keyword
- Use concatenation to combine fields, literals, and other data
- Search for patterns using the LIKE comparison operator
- Identify the purpose of the % and \_ wildcard characters
- Join multiple search conditions using the appropriate logical operator
- Perform searches for NULL values
- Specify the order for the presentation of query results using an ORDER BY clause

# Create the JustLee Database

- Use the provided script to create the database so you can follow the chapter examples
- Verify table contents using the DESCRIBE command

# SELECT Statement Syntax

- SELECT statements are used to retrieve data from the database
- A SELECT statement is referred to as a query
- Syntax gives the basic structure, or rules, for a command
- Optional clauses and keywords are shown in brackets

# SELECT Statement Syntax (continued)

```
SELECT [DISTINCT | UNIQUE] (*, columnname [ AS alias], ...)  
      FROM      tablename  
      [WHERE      condition]  
      [GROUP BY group_by_expression]  
      [HAVING    group_condition]  
      [ORDER BY columnname];
```

# SELECT Statement Syntax (continued)

- SELECT and FROM clauses are required
- SELECT clause identifies column(s)
- FROM clause identifies table(s)
- Each clause begins with a keyword

# Selecting All Data in a Table

- Substitute an asterisk for the column names in a SELECT clause

The screenshot shows a SQL development environment with the following components:

- Enter SQL Statement:** A text area containing the SQL code:

```
SELECT *
FROM customers;
```
- Results:** A grid displaying the results of the query. The columns are labeled CUSTOMER#, LASTNAME, FIRSTNAME, and ADDRESS. The data is as follows:

CUSTOMER#	LASTNAME	FIRSTNAME	ADDRESS
1	MORALES	BONITA	P.O. BOX 651
2	THOMPSON	RYAN	P.O. BOX 9835
3	SMITH	LEILA	P.O. BOX 66
4	PIERSON	THOMAS	69821 SOUTH A'
5	GIRARD	CINDY	P.O. BOX 851

# Selecting One Column from a Table

- Enter column name in SELECT clause

The screenshot shows a SQL development environment with the following components:

- Enter SQL Statement:** A text input field containing the SQL query:

```
SELECT title  
FROM books;
```
- Toolbar:** Buttons for **Results**, **Script Output**, **Explain**, and **Autocomplete**.
- Results:** A table displaying the output of the query:

AZ	TITLE
1	BODYBUILD IN 10 MINUTES A DAY
2	REVENGE OF MICKEY

# Selecting Multiple Columns from a Table

- Separate column names with a comma

Enter SQL Statement:

```
SELECT title, pubdate
  FROM books;
```

Results:

	AZ TITLE	AZ PUBDATE
1	BODYBUILD IN 10 MINUTES A DAY	21-JAN-05
2	REVENGE OF MICKEY	14-DEC-05
3	BUILDING A CAR WITH TOOTHPICKS	18-MAR-06

# Operations within the SELECT Statement

- Column alias can be used for column headings
- Perform arithmetic operations
- Suppress duplicates
- Concatenate data

# Using Column Aliases

- List the alias after the column heading
- AS keyword is optional
- Enclose in double quotation marks:
  - If it contains blank space(s)
  - If it contains special symbol(s)
  - To retain case

# Column Alias Example

Enter SQL Statement:

```
SELECT title AS "Title of Book", category
FROM books;
```

Results:

	Title of Book	Category
1	BODYBUILD IN 10 MINUTES A DAY	FITNESS
2	REVENGE OF MICKEY	FAMILY LIFE
3	BUILDING A CAR WITH TOOTHPICKS	CHILDREN
4	DATABASE IMPLEMENTATION	COMPUTER
5	COOKING WITH MUSHROOMS	COOKING
6	HOLY GRAIL OF ORACLE	COMPUTER
7	HANDCRANKED COMPUTERS	COMPUTER
8	E-BUSINESS THE EASY WAY	COMPUTER
9	PAINLESS CHILD-REARING	FAMILY LIFE
10	THE WOK WAY TO COOK	COOKING
11	BIG BEAR AND LITTLE DOVE	CHILDREN
12	HOW TO GET FASTER PIZZA	SELF HELP
13	HOW TO MANAGE THE MANAGER	BUSINESS
14	SHORTEST POEMS	LITERATURE

# Using Arithmetic Operations

- Arithmetic operations
  - Executed left to right
  - Multiplication and division are solved first
  - Addition and subtraction are solved last
  - Override order with parentheses

# Example Arithmetic Operation with Column Alias

Enter SQL Statement:

```
SELECT title, retail-cost profit
FROM books;
```

Results:

	TITLE	PROFIT
1	BODYBUILD IN 10 MINUTES A DAY	12.2
2	REVENGE OF MICKEY	7.8
3	BUILDING A CAR WITH TOOTHPICKS	22.15
4	DATABASE IMPLEMENTATION	24.55
5	COOKING WITH MUSHROOMS	7.45
6	HOLY GRAIL OF ORACLE	28.7
7	HANDCRANKED COMPUTERS	3.2
8	E-BUSINESS THE EASY WAY	16.6
9	PAINLESS CHILD-REARING	41.95
10	THE WOK WAY TO COOK	9.75
11	BIG BEAR AND LITTLE DOVE	3.63
12	HOW TO GET FASTER PIZZA	12.1
13	HOW TO MANAGE THE MANAGER	16.55
14	SHORTEST POEMS	18.1

# NULL Values

Enter SQL Statement:

```
SELECT title, retail, discount, retail-discount
FROM books;
```

Results:

	TITLE	RETAIL	DISCOUNT	RETAIL-DISCOUNT
1	BODYBUILD IN 10 MINUTES A DAY	30.95	(null)	(null)
2	REVENGE OF MICKEY	22	(null)	(null)
3	BUILDING A CAR WITH TOOTHPICKS	59.95	3	56.95
4	DATABASE IMPLEMENTATION	55.95	(null)	(null)
5	COOKING WITH MUSHROOMS	19.95	(null)	(null)
6	HOLY GRAIL OF ORACLE	75.95	3.8	72.15
7	HANDCRANKED COMPUTERS	25	(null)	(null)
8	E-BUSINESS THE EASY WAY	54.5	(null)	(null)
9	PAINLESS CHILD-REARING	89.95	4.5	85.45
10	THE WOK WAY TO COOK	28.75	(null)	(null)
11	BIG BEAR AND LITTLE DOVE	8.95	(null)	(null)
12	HOW TO GET FASTER PIZZA	29.95	1.5	28.45
13	HOW TO MANAGE THE MANAGER	31.95	(null)	(null)
14	SHORTEST POEMS	39.95	(null)	(null)

# Using DISTINCT and UNIQUE

- Enter DISTINCT or UNIQUE after SELECT keyword to suppress duplicates

The screenshot shows a SQL development environment with the following components:

- Enter SQL Statement:** A text input field containing the SQL query: `SELECT DISTINCT state FROM customers;`
- Toolbar:** A horizontal bar with five tabs: **Results**, **Script Output**, **Explain**, **Autotrace**, and **DBMS Output**. The **Results** tab is selected.
- Results:** A table displaying the results of the query. The table has one column labeled **STATE**. The data is as follows:

STATE
NJ
CA
WY
MA
GA
IL
MI
NY
FL
ID
WA
TX

# Using Concatenation

- You can combine data with a string literal
- Use the concatenation operator, ||
- It allows the use of column aliases

# Concatenation Example

Enter SQL Statement:

```
SELECT firstname || ' ' || lastname "Customer Name"
  FROM customers;
```

Results | Script Output | Explain | Autotrace | DBMS Output

Results:

	Customer Name
1	BONITA MORALES
2	RYAN THOMPSON
3	LEILA SMITH
4	THOMAS PIERSON
5	CINDY GIRARD
6	MESHIA CRUZ
7	TAMMY GIANA
8	KENNETH JONES
9	JORGE PEREZ
10	JAKE LUCAS
11	REESE MCGOVERN
12	WILLIAM MCKENZIE
13	NICHOLAS NGUYEN
14	JASMINE LEE
15	STEVE SCHELL
16	MICHELL DAUM
17	BECCA NELSON
18	GREG MONTIASA
19	JENNIFER SMITH
20	KENNETH FALAH

# WHERE Clause Syntax

- A WHERE clause is used to retrieve rows based on a stated condition
- Requires:
  - Column name
  - Comparison operator
  - Value or column for comparison
- Values are case sensitive

# WHERE Clause Example

- List WHERE clause after FROM clause
- Enclose nonnumeric data in single quotes

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

- Enter SQL Statement:** A text area containing the SQL query:

```
SELECT lastname, state
  FROM customers
 WHERE state = 'FL';
```
- Results:** A table displaying the query results:

	LASTNAME	STATE
1	MORALES	FL
2	SMITH	FL
3	NGUYEN	FL
4	SCHELL	FL

# Comparison Operators

- Indicate how the data should relate to the given search value

The screenshot shows a SQL development environment with the following components:

- Enter SQL Statement:** A text input field containing the SQL query:

```
SELECT title, retail
FROM books
WHERE retail > 55;
```
- Toolbar:** A horizontal bar with five tabs: **Results**, **Script Output**, **Explain**, **Autotrace**, and **DBMS Output**. The **Results** tab is selected.
- Results:** A table displaying the output of the query. The table has two columns: **TITLE** and **RETAIL**. The data is as follows:

	TITLE	RETAIL
1	BUILDING A CAR WITH TOOTHPICKS	59.95
2	DATABASE IMPLEMENTATION	55.95
3	HOLY GRAIL OF ORACLE	75.95
4	PAINLESS CHILD-REARING	89.95

# Arithmetic Comparison Operators

## COMPARISON OPERATORS

### Mathematical Comparison Operators

= Equality or “equal to”—for example, cost = 55.95

> Greater than—for example, cost > 20

< Less than—for example, cost < 20

<>, !=, or ^= Not equal to—for example, cost <> 55.95 or cost != 55.95 or cost ^=55.95

<= Less than or equal to—for example, cost <= 20

>= Greater than or equal to—for example, cost >= 20

# Other Comparison Operators

## Other Comparison Operators

[NOT] BETWEEN x AND y	Used to express a range—for example, searching for numbers BETWEEN 5 and 10. The optional NOT is used when searching for numbers that are NOT BETWEEN 5 AND 10.
[NOT] IN(x,y,...)	Similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses—for example, Pubid IN (1, 4, 5) will return only books with a publisher id of 1, 4, or 5. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5.
[NOT] LIKE	Used when searching for patterns if you are not certain how something is spelled—for example, title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.
IS [NOT] NULL	Used to search for records that do not have an entry in the specified field—for example, Shipdate IS NULL. Include the optional NOT to find records that do have an entry in the field—for example, Shipdate IS NOT NULL.

# BETWEEN...AND Operator

- Finds values in a specified range

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text area containing the SQL query:

```
SELECT title, pubid
  FROM books
 WHERE pubid BETWEEN 1 AND 3;
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, and DBMS Output.
- Results:** A table displaying the output of the query:

	TITLE	PUBID
1	REVENGE OF MICKEY	1
2	BUILDING A CAR WITH TOOTHPICKS	2
3	DATABASE IMPLEMENTATION	3
4	HOLY GRAIL OF ORACLE	3
5	HANDCRANKED COMPUTERS	3
6	E-BUSINESS THE EASY WAY	2
7	HOW TO MANAGE THE MANAGER	1

# IN Operator

- Returns records that match a value in a specified list
- List must be in parentheses
- Values are separated by commas

# IN Operator Example

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text input field containing the following SQL query:

```
SELECT title, pubid
FROM books
WHERE pubid IN (1,2,5);
```
- Toolbar:** A horizontal bar with five tabs: **Results**, **Script Output**, **Explain**, **Autotrace**, and **DBMS Output**. The **Results** tab is selected.
- Results:** A table displaying the query results. The table has two columns: **TITLE** and **PUBID**. The data is as follows:

	TITLE	PUBID
1	REVENGE OF MICKEY	1
2	BUILDING A CAR WITH TOOTHPICKS	2
3	E-BUSINESS THE EASY WAY	2
4	PAINLESS CHILD-REARING	5
5	BIG BEAR AND LITTLE DOVE	5
6	HOW TO MANAGE THE MANAGER	1
7	SHORTEST POEMS	5

# LIKE Operator

- Performs pattern searches
- Used with wildcard characters
  - Underscore (\_) for exactly one character in the indicated position
  - Percent sign (%) represents any number of characters

# LIKE Operator Example

The screenshot shows a SQL development environment with the following components:

- Enter SQL Statement:** A text area containing the SQL query:

```
SELECT lastname
  FROM customers
 WHERE lastname LIKE 'P%';
```
- Toolbar:** A row of buttons for different operations: Results, Script Output, Explain, Autotrace, and DBMS Output.
- Results:** A table displaying the query results:

LASTNAME
PIERSON
PEREZ

# Logical Operators

- Used to combine conditions
- Evaluated in order of NOT, AND, OR
  - NOT – reverses meaning
  - AND – both conditions must be TRUE
  - OR – at least one condition must be TRUE

# AND Logical Operator Example

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text input field containing the following SQL query:

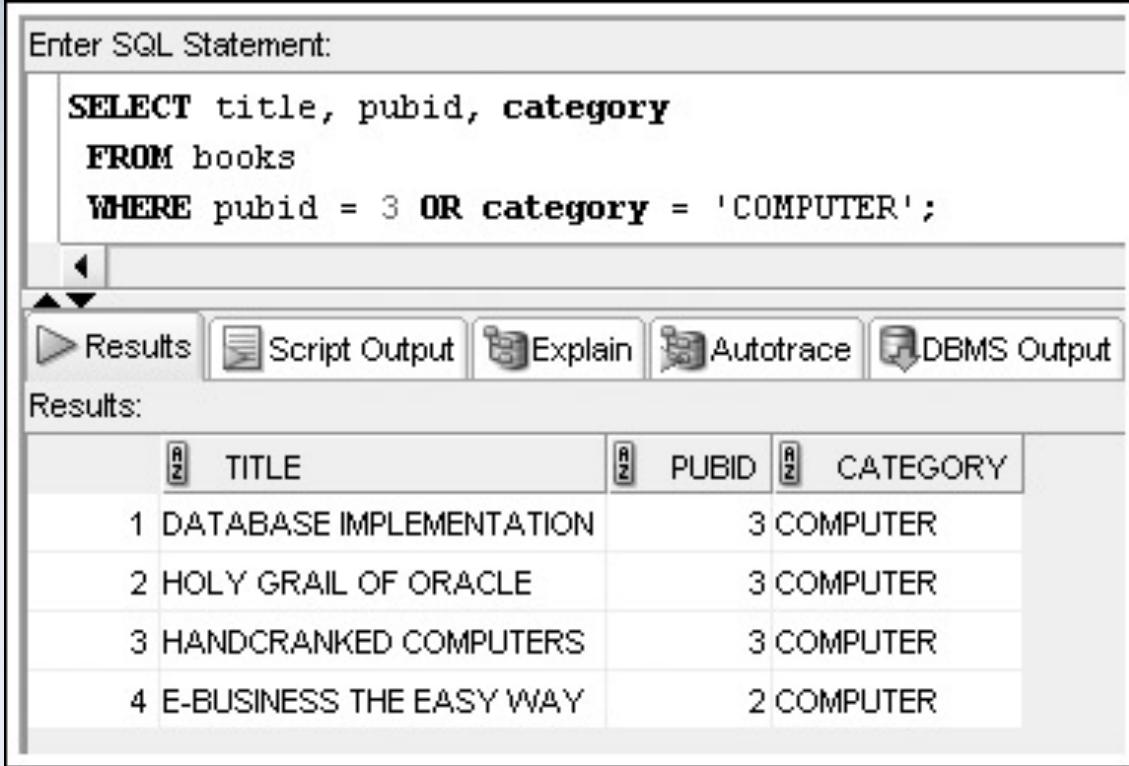
```
SELECT title, pubid, category
FROM books
WHERE pubid = 3 AND category = 'COMPUTER';
```
- Toolbar:** A horizontal bar with several icons: a play button (Results), a script icon (Script Output), a magnifying glass icon (Explain), a folder icon (Autotrace), and a DBMS icon (DBMS Output).
- Results:** A table displaying the query results. The table has three columns: TITLE, PUBID, and CATEGORY. The data is as follows:

	TITLE	PUBID	CATEGORY
1	DATABASE IMPLEMENTATION	3	COMPUTER
2	HOLY GRAIL OF ORACLE	3	COMPUTER
3	HANDCRANKED COMPUTERS	3	COMPUTER

# OR Logical Operator Example

Enter SQL Statement:

```
SELECT title, pubid, category
  FROM books
 WHERE pubid = 3 OR category = 'COMPUTER';
```

Results: 

	AZ TITLE	AZ PUBID	AZ CATEGORY
1	DATABASE IMPLEMENTATION		3 COMPUTER
2	HOLY GRAIL OF ORACLE		3 COMPUTER
3	HANDCRANKED COMPUTERS		3 COMPUTER
4	E-BUSINESS THE EASY WAY		2 COMPUTER

# Multiple Logical Operators

- Resolved in order of NOT, AND, OR

Enter SQL Statement:

```
SELECT *
  FROM books
 WHERE category = 'FAMILY LIFE'
   OR pubid = 4
   AND cost > 15;
```

Results:

#	ISBN	Title	PUBDATE	PUBID	COST	RETAIL	DISCOUNT	Category
1	1059831198	BODYBUILD IN 10 MINUTES A DAY	21-JAN-05	4	18.75	30.95	(null)	FITNESS
2	0401140733	REVENGE OF MICKEY	14-DEC-05	1	14.2	22	(null)	FAMILY LIFE
3	2491748320	PAINLESS CHILD-REARING	17-JUL-04	5	48	89.95	4.5	FAMILY LIFE
4	0299282519	THE WOK WAY TO COOK	11-SEP-04	4	19	28.75	(null)	COOKING
5	0132149871	HOW TO GET FASTER PIZZA	11-NOV-06	4	17.85	29.95	1.5	SELF HELP

# Multiple Logical Operators

- Use parentheses to override the order of evaluation

Enter SQL Statement:

```
SELECT *
  FROM books
 WHERE (category = 'FAMILY LIFE'
        OR pubid = 4)
        AND cost > 15;
```

Results:

#	ISBN	Title	PUBDATE	PUBID	COST	RETAIL	DISCOUNT	Category
1	1059831198	BODYBUILD IN 10 MINUTES A DAY	21-JAN-05	4	18.75	30.95	(null)	FITNESS
2	2491748320	PAINLESS CHILD-REARING	17-JUL-04	5	48	89.95	4.5	FAMILY LIFE
3	0299282519	THE WOK WAY TO COOK	11-SEP-04	4	19	28.75	(null)	COOKING
4	0132149871	HOW TO GET FASTER PIZZA	11-NOV-06	4	17.85	29.95	1.5	SELF HELP

# Resolving Multiple Types of Operators

1. Arithmetic operators
2. Comparison operators
3. Logical operators

# Treatment of NULL Values

- Absence of data
- Requires use of IS NULL operator

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text area containing the SQL query:

```
SELECT order#, shipdate
FROM orders
WHERE shipdate IS NULL;
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, and DBMS Output.
- Results:** A table displaying the query results:

	ORDER#	SHIPDATE
1	1012	(null)
2	1015	(null)
3	1016	(null)
4	1018	(null)
5	1019	(null)
6	1020	(null)

# Treatment of NULL Values (continued)

- A common error is using = NULL, which does not raise an Oracle error but also does not return any rows

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

- Enter SQL Statement:** A text area containing the SQL query:

```
SELECT order#, shipdate
FROM orders
WHERE shipdate = NULL;
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, and DBMS Output.
- Results:** A table header row with columns ORDER# and SHIPDATE.
- Status:** Below the results table, the text "No rows returned" is displayed, with an arrow pointing upwards from the text towards the table.

# ORDER BY Clause Syntax

- The ORDER BY clause presents data in sorted order
- Ascending order is default
- Use DESC keyword to override column default
- 255 columns maximum

# ORDER BY Clause Syntax

## Sort Sequence

- In ascending order, values will be listed in the following sequence:
  - Numeric values
  - Character values
  - NULL values
- In descending order, sequence is reversed

# ORDER BY Example

Enter SQL Statement:

```
SELECT lastname, firstname, state, city
  FROM customers
 WHERE state IN('FL','CA')
 ORDER BY state DESC, city;
```

Results Script Output Explain Autotrace DBMS Output

Results:

	LASTNAME	FIRSTNAME	STATE	CITY
1	NGUYEN	NICHOLAS	FL	CLERMONT
2	MORALES	BONITA	FL	EASTPOINT
3	SCHELL	STEVE	FL	MIAMI
4	SMITH	LEILA	FL	TALLAHASSEE
5	DAUM	MICHELL	CA	BURBANK
6	PEREZ	JORGE	CA	BURBANK
7	THOMPSON	RYAN	CA	SANTA MONICA

# ORDER BY Can Reference Column Position

Enter SQL Statement:

```
SELECT lastname, firstname, state, city
  FROM customers
 WHERE state IN('FL','CA')
 ORDER BY 3 DESC, 4;
```

Results:

	LASTNAME	FIRSTNAME	STATE	CITY
1	NGUYEN	NICHOLAS	FL	CLERMONT
2	MORALES	BONITA	FL	EASTPOINT
3	SCHELL	STEVE	FL	MIAMI
4	SMITH	LEILA	FL	TALLAHASSEE
5	DAUM	MICHELL	CA	BURBANK
6	PEREZ	JORGE	CA	BURBANK
7	THOMPSON	RYAN	CA	SANTA MONICA

# Summary

- A basic query in Oracle 11g SQL includes the SELECT and FROM clauses, the only mandatory clauses in a SELECT statement
- To view all columns in the table, specify an asterisk (\*) or list all of the column names individually in the SELECT clause
- To display a specific column or set of columns, list the column names in the SELECT clause (in the order in which you want them to appear)
- When listing column names in the SELECT clause, a comma must separate column names

# Summary (continued)

- A column alias can be used to clarify the contents of a particular column; if the alias contains spaces or special symbols, or if you want to display the column with any lowercase letters, you must enclose the column alias in double quotation marks (" ")
- Indicate the table name following the FROM keyword
- Basic arithmetic operations can be performed in the SELECT clause
- NULL values indicate an absence of a value

# Summary (continued)

- To remove duplicate listings, include either the DISTINCT or UNIQUE keyword
- To specify which table contains the desired columns, you must list the name of the table after the keyword FROM
- Use vertical bars (||) to combine, or concatenate, fields, literals, and other data

# Summary (continued)

- The WHERE clause can be included in a SELECT statement to restrict the rows returned by a query to only those meeting a specified condition
- When searching a nonnumeric field, the search values must be enclosed in single quotation marks
- Comparison operators are used to indicate how the record should relate to the search value
- The BETWEEN...AND comparison operator is used to search for records that fall within a certain range of values

# Summary (continued)

- The LIKE comparison operator is used with the percent and underscore symbols (%) and (\_) to establish search patterns
- Logical operators such as AND and OR can be used to combine several search conditions
- When using the AND operator, all conditions must be TRUE for a record to be returned in the results
  - However, with the OR operator, only one condition must be TRUE
- A NULL value is the absence of data, not a field with a blank space entered

# Summary (continued)

- Use the IS NULL comparison operator to match NULL values; the IS NOT NULL comparison operator finds records that do not contain NULL values in the indicated column
- You can sort the results of queries by using an ORDER BY clause; when used, the ORDER BY clause should be listed last in the SELECT statement
- By default, records are sorted in ascending order; entering DESC directly after the column name sorts the records in descending order
- A column does not have to be listed in the SELECT clause to serve as a basis for sorting

# Oracle SQL

*Joining Data from Multiple Tables*

# Objectives

- Identify a Cartesian join
- Create an equality join using the WHERE clause
- Create an equality join using the JOIN keyword
- Create a non-equality join using the WHERE clause
- Create a non-equality join using the JOIN...ON approach

# Objectives (continued)

- Create a self-join using the WHERE clause
- Create a self-join using the JOIN keyword
- Distinguish an inner join from an outer join
- Create an outer join using the WHERE clause
- Create an outer join using the OUTER keyword
- Use set operators to combine the results of multiple queries

# Purpose of Joins

- Joins are used to link tables and reconstruct data in a relational database
- Joins can be created through:
  - Conditions in a WHERE clause
  - Use of JOIN keywords in FROM clause

# Cartesian Joins

- Created by omitting joining condition in the WHERE clause or through CROSS JOIN keywords in the FROM clause
- Results in every possible row combination ( $m * n$ )

# Cartesian Join Example: Omitted Condition

Enter SQL Statement:

```
SELECT title, name
  FROM books, publisher;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	TITLE	NAME
1	BODYBUILD IN 10 MINUTES A DAY	PRINTING IS US
2	REVENGE OF MICKEY	PRINTING IS US
3	BUILDING A CAR WITH TOOTHPICKS	PRINTING IS US
4	DATABASE IMPLEMENTATION	PRINTING IS US
5	COOKING WITH MUSHROOMS	PRINTING IS US
6	HOLY GRAIL OF ORACLE	PRINTING IS US
7	HANDCRANKED COMPUTERS	PRINTING IS US
8	E-BUSINESS THE EASY WAY	PRINTING IS US
9	PAINLESS CHILD-REARING	PRINTING IS US
10	THE WOK WAY TO COOK	PRINTING IS US
11	BIG BEAR AND LITTLE DOVE	PRINTING IS US
12	HOW TO GET FASTER PIZZA	PRINTING IS US
13	HOW TO MANAGE THE MANAGER	PRINTING IS US
14	SHORTEST POEMS	PRINTING IS US
15	BODYBUILD IN 10 MINUTES A DAY	PUBLISH OUR WAY

Partial output shown

# Equality Joins

- Link rows through equivalent data that exists in both tables
- Created by:
  - Creating equivalency condition in the WHERE clause
  - Using NATURAL JOIN, JOIN...USING, or JOIN...ON keywords in the FROM clause

# Equality Joins: WHERE Clause Example

Enter SQL Statement:

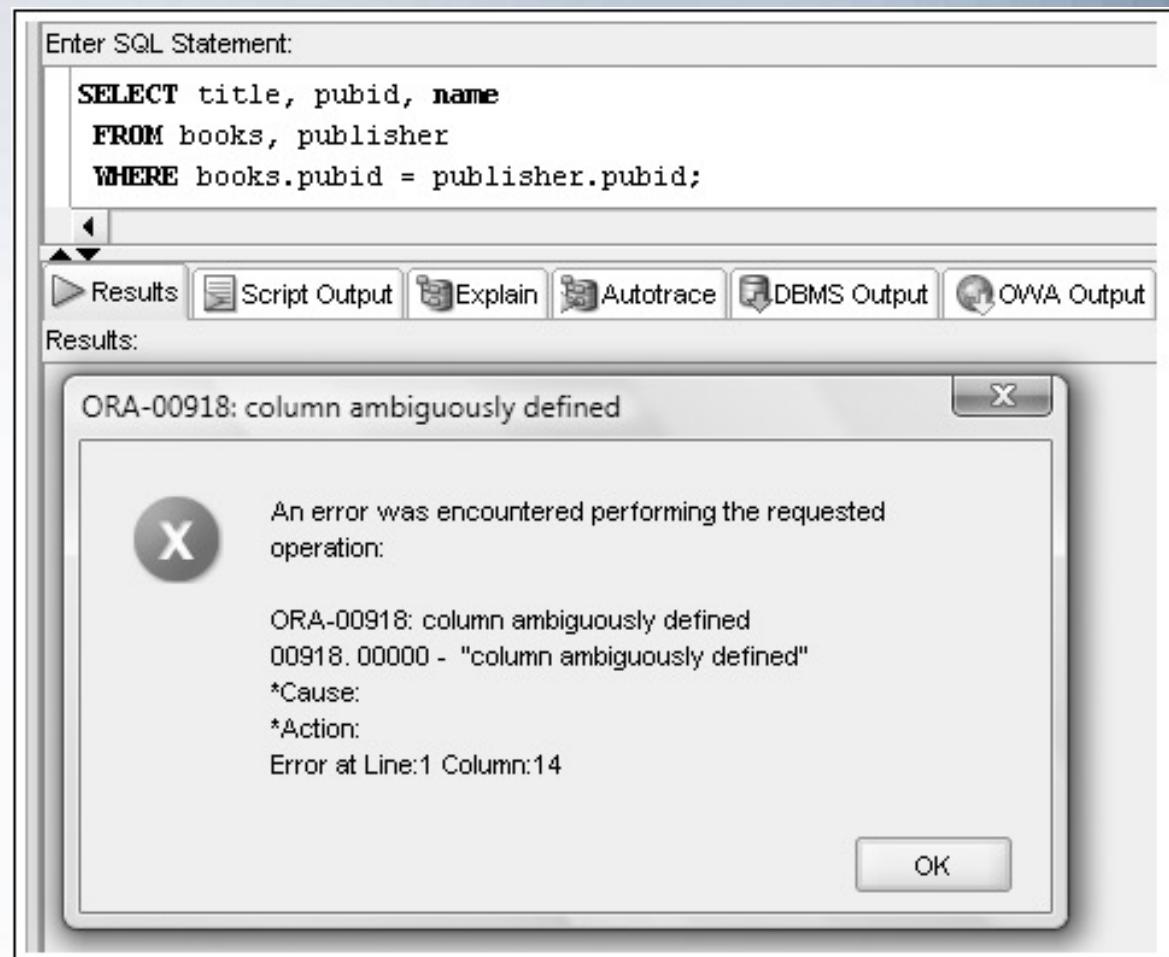
```
SELECT title, name
  FROM books, publisher
 WHERE books.pubid = publisher.pubid;
```

Results:

R TITLE	R NAME
1 BODYBUILD IN 10 MINUTES A DAY	READING MATERIALS INC.
2 REVENGE OF MICKEY	PRINTING IS US
3 BUILDING A CAR WITH TOOTHPICKS	PUBLISH OUR WAY
4 DATABASE IMPLEMENTATION	AMERICAN PUBLISHING
5 COOKING WITH MUSHROOMS	READING MATERIALS INC.
6 HOLY GRAIL OF ORACLE	AMERICAN PUBLISHING
7 HANDCRANKED COMPUTERS	AMERICAN PUBLISHING
8 E-BUSINESS THE EASY WAY	PUBLISH OUR WAY
9 PAINLESS CHILD-REARING	REED-N-RITE
10 THE WOK WAY TO COOK	READING MATERIALS INC.
11 BIG BEAR AND LITTLE DOVE	REED-N-RITE
12 HOW TO GET FASTER PIZZA	READING MATERIALS INC.
13 HOW TO MANAGE THE MANAGER	PRINTING IS US
14 SHORTEST POEMS	REED-N-RITE

# Qualifying Column Names

- Columns in both tables must be qualified



# WHERE Clause Supports Join and Other Conditions

Enter SQL Statement:

```
SELECT title, books.pubid, name
  FROM books, publisher
 WHERE books.pubid = publisher.pubid
   AND publisher.pubid = 4;
```

Results: Results Script Output Explain Autotrace DBMS Output OWA Output

AZ TITLE	AZ PUBID	AZ NAME
1 BODYBUILD IN 10 MINUTES A DAY	4	READING MATERIALS INC.
2 COOKING WITH MUSHROOMS	4	READING MATERIALS INC.
3 THE WOK WAY TO COOK	4	READING MATERIALS INC.
4 HOW TO GET FASTER PIZZA	4	READING MATERIALS INC.

# Joining More Than Two Tables

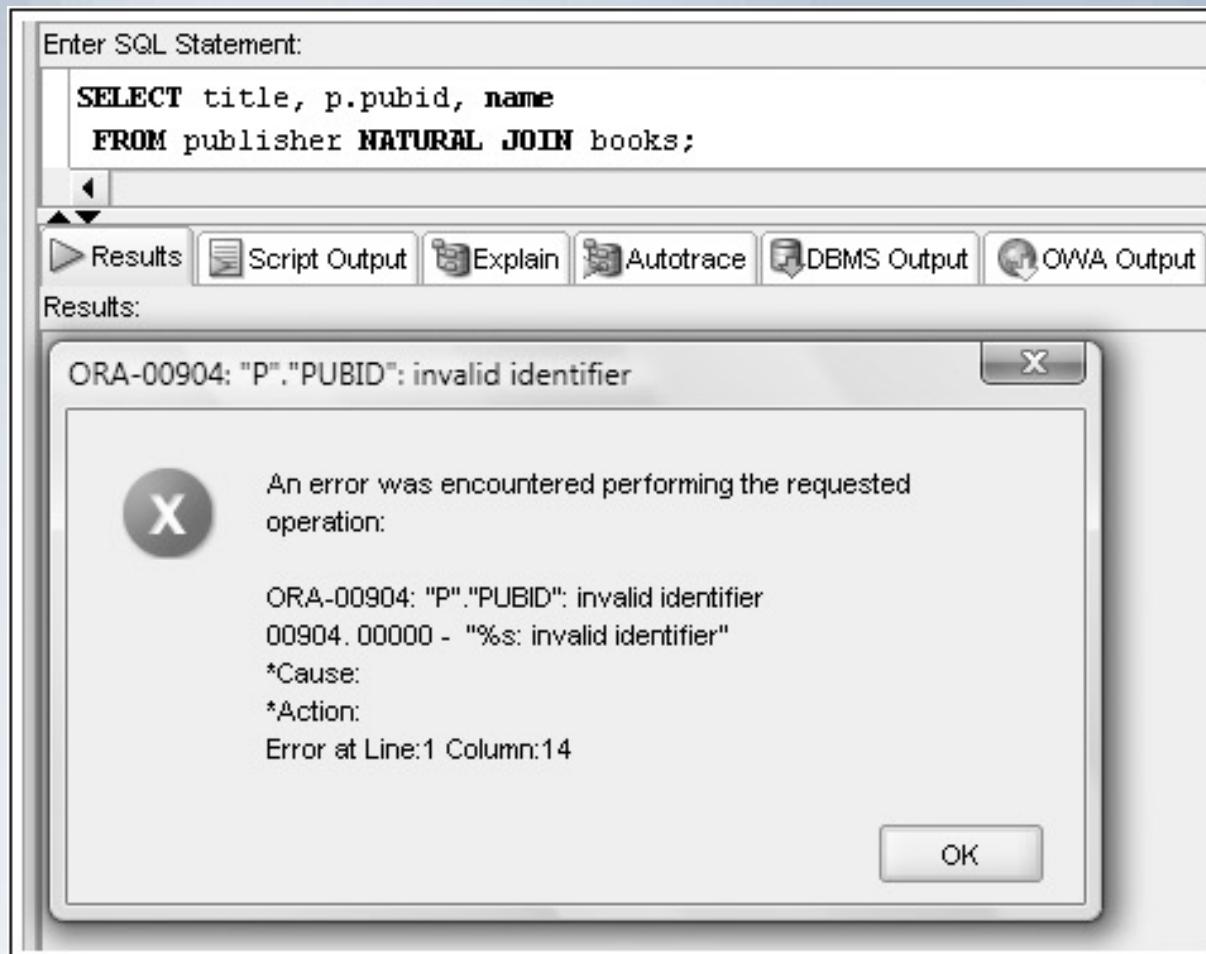
- Joining four tables requires three join conditions

```
Enter SQL Statement:  
  
SELECT c.lastname, c.firstname, b.title  
FROM customers c, orders o, orderitems oi, books b  
WHERE c.customer# = o.customer#  
    AND o.order# = oi.order#  
    AND oi.isbn = b.isbn  
ORDER BY lastname, firstname;
```

# Equality Joins: NATURAL JOIN

```
Enter SQL Statement:  
|  
| SELECT title, pubid, name  
| FROM publisher NATURAL JOIN books;  
|  
|
```

# No Qualifiers with a NATURAL JOIN



# Equality Joins: JOIN...USING

```
Enter SQL Statement:  
SELECT b.title, pubid, p.name  
  FROM publisher p JOIN books b  
        USING (pubid);
```

# Equality Joins: JOIN...ON

- Required if column names are different

The image shows a screenshot of a SQL editor window. At the top, there is a label "Enter SQL Statement:". Below it is a text input field containing the following SQL code:

```
SELECT b.title, b.pubid, p.name
FROM publisher2 p JOIN books b
    ON p.id = b.pubid;
```

# JOIN Keyword Overview

- Use JOIN...USING when tables have one or more columns in common
- Use JOIN...ON when same named columns are not involved or a condition is needed to specify a relationship other than equivalency (next section)
- Using the JOIN keyword frees the WHERE clause for exclusive use in restricting rows

# Oracle SQL

*Joining Data from Multiple Tables*

# Non-Equality Joins

- In WHERE clause, use any comparison operator other than the equal sign
- In FROM clause, use JOIN...ON keywords with a non-equivalent condition

# Non-Equality Joins: WHERE Clause Example

Enter SQL Statement:

```
SELECT b.title, p.gift
  FROM books b, promotion p
 WHERE b.retail BETWEEN p.minretail AND p.maxretail;
```

# Non-Equality Joins: JOIN...ON Example

Enter SQL Statement:

```
SELECT b.title, p.gift
  FROM books b JOIN promotion p
    ON b.retail BETWEEN p.minretail AND p.maxretail;
```

# Self-Joins

- Used to link a table to itself
- Requires the use of table aliases
- Requires the use of a column qualifier

# Customer Table Example

Customer 1003 (Leila Smith) has referred two customers (Tammy Giana and Jorge Perez)							
CUSTOMER#	LASTNAME	FIRSTNAME	ADDRESS	CITY	STATE	ZIP	REFERRED
1001	MORALES	BONITA	P.O. BOX 651	EASTPOINT	FL	32328	
1002	THOMPSON	RYAN	P.O. BOX 9835	SANTA MONICA	CA	90404	
1003	SMITH	LEILA	P.O. BOX 66	TALLAHASSEE	FL	32306	
1004	PIERSON	THOMAS	69821 SOUTH AVENUE	BOISE	ID	83707	
1005	GIRARD	CINDY	P.O. BOX 851	SEATTLE	WA	98115	
1006	CRUZ	MESHIA	82 DIRT ROAD	ALBANY	NY	12211	
1007	GINA	TAMMY	9153 MAIN STREET	AUSTIN	TX	78710	1003
1008	JONES	KENNETH	P.O. BOX 137	CHEYENNE	WY	82003	
1009	PEREZ	JORGE	P.O. BOX 8564	BURBANK	CA	91510	1003
1010	LUCAS	JAKE	114 EAST SAVANNAH	ATLANTA	GA	30314	
1011	MCGOVERN	REESE	P.O. BOX 18	CHICAGO	IL	60606	
1012	MCKENZIE	WILLIAM	P.O. BOX 971	BOSTON	MA	02110	
1013	NGUYEN	NICHOLAS	357 WHITE EAGLE AVE.	CLERMONT	FL	34711	1006

Customer 1006  
(Meshia Cruz) has  
referred one customer  
(Nicholas Nguyen)

# Self-Joins: WHERE Clause Example

Enter SQL Statement:

```
SELECT r.firstname, r.lastname, c.lastname "Referred"
  FROM customers c, customers r
 WHERE c.referred = r.customer#;
```

Results Script Output Explain Autotrace DBMS Output OWA Output

Results:

	FIRSTNAME	LASTNAME	Referred
1	LEILA	SMITH	SMITH
2	LEILA	SMITH	PEREZ
3	LEILA	SMITH	GIANA
4	MESHIA	CRUZ	NGUYEN
5	JAKE	LUCAS	DAUM

# Self-Joins: JOIN...ON Example

Enter SQL Statement:

```
SELECT r.firstname, r.lastname, c.lastname "Referred"
  FROM customers c JOIN customers r
    ON c.referred = r.customer#;
```

Results: Results Script Output Explain Autotrace DBMS Output OWA Output

	FIRSTNAME	LASTNAME	Referred
1	LEILA	SMITH	SMITH
2	LEILA	SMITH	PEREZ
3	LEILA	SMITH	GIANA
4	MESHIA	CRUZ	NGUYEN
5	JAKE	LUCAS	DAUM

# Outer Joins

- Use outer joins to include rows that do not have a match in the other table
- In WHERE clause, include outer join operator (+) immediately after the column name of the table with missing rows to add NULL rows
- In FROM clause, use FULL, LEFT, or RIGHT with OUTER JOIN keywords

# Outer Joins: WHERE Clause Example

Enter SQL Statement:

```
SELECT c.lastname, c.firstname, o.order#
  FROM customers c, orders o
 WHERE c.customer# = o.customer#(+)
 ORDER BY c.lastname, c.firstname;
```

# Outer Joins: OUTER JOIN Keyword Example

Enter SQL Statement:

```
SELECT c.lastname, c.firstname, o.order#
  FROM customers c LEFT OUTER JOIN orders o
    USING (customer#)
 ORDER BY c.lastname, c.firstname;
```



# Left Outer joins explained

id	title	category
1	ASSASSIN'S CREED: EMBERS	Animations
2	Real Steel(2012)	Animations
3	Alvin and the Chipmunks	Animations
4	The Adventures of Tin Tin	Animations
5	Safe (2012)	Action
6	Safe House(2012)	Action
7	GIA	18+
8	Deadline 2009	18+
9	The Dirty Picture	18+
10	Marley and me	Romance

id	first_name	last_name	movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2
3	Susan	Davidson	5
4	Jenny	Adrianna	8
6	Lee	Pong	10

title	first_name	last_name
ASSASSIN'S CREED: EMBERS	Adam	Smith
Real Steel(2012)	Ravi	Kumar
Safe (2012)	Susan	Davidson
Deadline(2009)	Jenny	Adrianna
Marley and me	Lee	Pong
Alvin and the Chipmunks	NULL	NULL
The Adventures of Tin Tin	NULL	NULL
Safe House(2012)	NULL	NULL
GIA	NULL	NULL
The Dirty Picture	NULL	NULL

Note: Null is returned for non-matching rows on right

# Right Outer Join

first_name	last_name	title
Adam	Smith	ASSASSIN'S CREED: EMBERS
Ravi	Kumar	Real Steel(2012)
Susan	Davidson	Safe (2012)
Jenny	Adrianna	Deadline(2009)
Lee	Pong	Marley and me
NULL	NULL	Alvin and the Chipmunks
NULL	NULL	The Adventures of Tin Tin
NULL	NULL	Safe House(2012)
NULL	NULL	GIA
NULL	NULL	The Dirty Picture

Note: Null is returned for non-matching rows on left

# Set Operators

- Used to combine the results of two or more SELECT statements

Set Operator	Description
UNION	Returns the results of both queries and removes duplicates
UNION ALL	Returns the results of both queries but includes duplicates
INTERSECT	Returns only the rows included in the results of both queries
MINUS	Subtracts the second query's results if they're also returned in the first query's results

# Set Operators: UNION Example

Enter SQL Statement:

```
SELECT ba.authorid
  FROM books b JOIN bookauthor ba
    USING (isbn)
 WHERE category = 'FAMILY LIFE'
UNION
SELECT ba.authorid
  FROM books b JOIN bookauthor ba
    USING (isbn)
 WHERE category = 'CHILDREN';
```

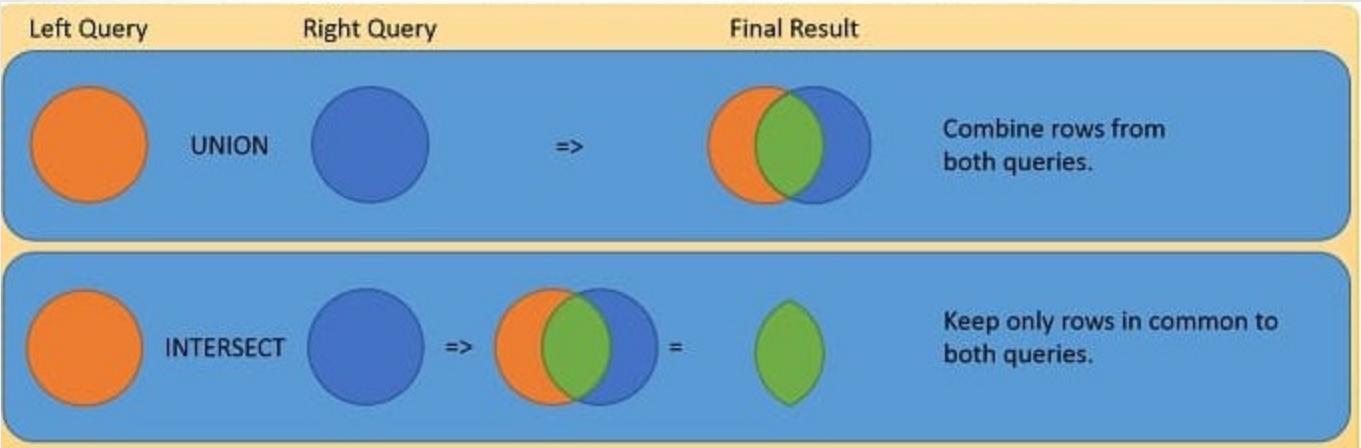
Results: Script Output Explain Autotrace DBMS Output OWA Output

AUTHORID
1 B100
2 F100
3 J100
4 K100
5 R100

# Set Operators: INTERSECT Example

Enter SQL Statement:

```
SELECT customer#
  FROM customers
INTERSECT
SELECT customer#
  FROM orders;
```



# Set Operators: MINUS Example

Enter SQL Statement:

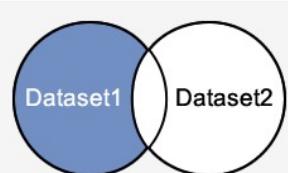
```
SELECT customer#
  FROM customers
MINUS
SELECT customer#
  FROM orders;
```

Results Script Output Explain Autotrace DBMS Output

Results:

	CUSTOMER#
1	1002
2	1006
3	1009
4	1012
5	1013
6	1016

Minus Query



TechOnTheNet.com

**Explanation:** The MINUS query will return the records in the blue shaded area. These are the records that exist in Dataset1 and not in Dataset2.

# Summary

- Data stored in multiple tables regarding a single entity can be linked together through the use of joins
- A Cartesian join between two tables returns every possible combination of rows from the tables; the resulting number of rows is always  $m * n$
- An equality join is created when the data joining the records from two different tables are an exact match
- A non-equality join establishes a relationship based upon anything other than an equal condition
- Self-joins are used when a table must be joined to itself to retrieve needed data

# Summary (continued)

- Inner joins are categorized as being equality, non-equality, or self-joins
- An outer join is created when records need to be included in the results without having corresponding records in the join tables
  - The record is matched with a NULL record so it will be included in the output
- Set operators such as UNION, UNION ALL, INTERSECT, and MINUS can be used to combine the results of multiple queries

# Oracle 11g: SQL

*Group Functions*

# Objectives

- Differentiate between single-row and multiple-row functions
- Use the SUM and AVG functions for numeric calculations
- Use the COUNT function to return the number of records containing non-NULL values
- Use COUNT(\*) to include records containing NULL values
- Use the MIN and MAX functions with nonnumeric fields

# Objectives (continued)

- Determine when to use the GROUP BY clause to group data
- Identify when the HAVING clause should be used
- List the order of precedence for evaluating WHERE, GROUP BY, and HAVING clauses
- State the maximum depth for nesting group functions
- Nest a group function inside of a single-row function

# Objectives (continued)

- Calculate the standard deviation and variance of a set of data, using the STDDEV and VARIANCE functions
- Use composite columns and concatenated groupings in grouping operations

# Group Functions

- Return one result per group of rows processed
- Are also called multiple-row and aggregate functions
- All group functions ignore NULL values except COUNT(\*)
- Use DISTINCT to suppress duplicate values

# Added Clauses

```
SELECT * |columnname, columnname...
FROM tablename
[WHERE condition]
[GROUP BY columnname, columnname...]
[HAVING group condition];
```



# SUM Function

- Calculates total amount stored in a numeric column for a group of rows

Enter SQL Statement:

```
SELECT SUM((paideach-cost)*quantity) "Total Profit"
FROM orderitems JOIN books USING (isbn)
WHERE order# = 1007;
```

Results: Script Output Explain Autotrace DBMS Output OWA Output

	Total Profit
1	119.48

# AVG Function

- Calculates the average of numeric values in a specified column

The screenshot shows the Oracle SQL Developer interface. In the top panel, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT AVG(retail-cost) "Average Profit"  
FROM books  
WHERE category = 'COMPUTER';
```

Below the SQL statement, there is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is highlighted, indicating it is the active tab.

In the bottom panel, under the "Results:" heading, there is a table with one row. The table has two columns: "Average Profit" and a numerical value. The table is displayed as follows:

Average Profit	18.2625
1	18.2625

# COUNT Function

- Two purposes
  - Count non-NULL values
  - Count total records, including those with NULL values

# COUNT Function – Non-NULL Values

- Include column name in argument to count number of occurrences

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT COUNT(DISTINCT category)
FROM books;
```

Below the code editor is a toolbar with five buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is highlighted, indicating it is the active tab.

In the bottom-left pane, under the heading "Results:", there is a table with one row. The table has two columns: the first column contains the value "1", and the second column contains the value "8". The header of the table is "COUNT(DISTINCTCATEGORY)".

	COUNT(DISTINCTCATEGORY)
1	8

# COUNT Function – NULL Values

- Include asterisk in argument to count number of rows

The screenshot shows a SQL developer interface. In the top panel, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT COUNT(*)
FROM orders
WHERE shipdate IS NULL;
```

Below the code editor is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is highlighted.

The results pane below the toolbar displays the output of the query:

COUNT(*)
1
6

# MAX Function

- Returns largest value

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a text input field labeled "Enter SQL Statement:" containing the following SQL query:

```
SELECT MAX(retail-cost) "Highest Profit"  
FROM books;
```

Below the SQL statement, there is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is currently selected.

In the bottom-left pane, under the heading "Results:", there is a table with one row. The table has two columns: "Highest Profit" and a numerical value. The table is displayed in a grid format with a header row and a data row.

Highest Profit
1 41.95

# MIN Function

- Returns the smallest value

The screenshot shows the Oracle SQL Developer interface. In the top-left panel, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT MIN(pubdate)  
FROM books;
```

Below the SQL statement, there is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output". The "Results" button is highlighted, indicating it is the active tab.

In the bottom panel, under the "Results:" heading, there is a table with one row. The table has two columns: the first column contains the number "1", and the second column contains the date "09-MAY-03". The header of the table is "MIN(PUBDATE)".

	MIN(PUBDATE)
1	09-MAY-03

# Datatypes

- The COUNT, MIN, and MAX functions can be used on values with character, numeric, and date datatypes

# Grouping Data

- GROUP BY clause
  - Used to group data
  - Must be used for any individual column in the SELECT clause with a group function
  - Cannot reference column aliases

# GROUP BY Example

Enter SQL Statement:

```
SELECT category, TO_CHAR(AVG(retail-cost), '999.99') "Profit"
FROM books
GROUP BY category;
```

Results:

Category	Profit
COMPUTER	18.26
COOKING	8.60
CHILDREN	12.89
LITERATURE	18.10
BUSINESS	16.55
FITNESS	12.20
FAMILY LIFE	24.88
SELF HELP	12.10

# Common Error

- A common error is missing a GROUP BY clause for nonaggregated columns in the SELECT clause

The screenshot shows a SQL developer interface with a query editor and a results window. The query editor contains the following SQL statement:

```
Enter SQL Statement:  
SELECT category, TO_CHAR(AVG(retail-cost), '999.99') "Profit"  
FROM books;
```

The results window displays an error message:

ORA-00937: not a single-group group function

An error was encountered performing the requested operation:

ORA-00937: not a single-group group function  
00937. 00000 - "not a single-group group function"  
\*Cause:  
\*Action:  
Error at Line:1 Column:7

OK

# Restricting Aggregated Output

- HAVING clause serves as the WHERE clause for grouped data

Enter SQL Statement:

```
SELECT category, TO_CHAR(AVG(retail-cost), '999.99') "Profit"
  FROM books
 GROUP BY category
 HAVING AVG(retail-cost) > 15;
```

Results:

	CATEGORY	Profit
1	COMPUTER	18.26
2	LITERATURE	18.10
3	BUSINESS	16.55
4	FAMILY LIFE	24.88

# Restricting Aggregated Output (continued)

- When included in the same SELECT statement, the clauses are evaluated in the order of:
  - WHERE
  - GROUP BY
  - HAVING

# Restricting Aggregated Output (continued)

Enter SQL Statement:

```
SELECT category, TO_CHAR(AVG(retail-cost), '999.99') "Profit"
  FROM books
 WHERE pubdate > '01-JAN-05'
 GROUP BY category
 HAVING AVG(retail-cost) > 15;
```

Results:

CATEGORY	Profit
1 COMPUTER	16.17
2 LITERATURE	18.10

# Nesting Functions

- Inner function is resolved first
  - Maximum nesting depth: 2

# Statistical Group Functions

- Based on normal distribution
- Includes:
  - STDDEV
  - VARIANCE

# STDDEV Function

Enter SQL Statement:

```
SELECT category, COUNT(*), TO_CHAR(AVG(retail-cost),'999.99') "Avg",
       TO_CHAR(STDDEV(retail-cost),'999.9999') "Stddev"
  FROM books
 GROUP BY category;
```

Results:

	Category	Count(*)	Avg	Stddev
1	COMPUTER	4	18.26	11.2267
2	COOKING	2	8.60	1.6263
3	CHILDREN	2	12.89	13.0956
4	LITERATURE	1	18.10	.0000
5	BUSINESS	1	16.55	.0000
6	FITNESS	1	12.20	.0000
7	FAMILY LIFE	2	24.88	24.1477
8	SELF HELP	1	12.10	.0000

# VARIANCE Function

- Determines data dispersion within a group

Enter SQL Statement:

```
SELECT category, TO_CHAR(VARIANCE(retail-cost), '999.99') "Var",
       MIN(retail-cost) "Min", MAX(retail-cost) "Max"
  FROM books
 GROUP BY category;
```

Results:

	Category	Var	Min	Max
1	COMPUTER	126.04	3.2	28.7
2	COOKING	2.65	7.45	9.75
3	CHILDREN	171.50	3.63	22.15
4	LITERATURE	.00	18.1	18.1
5	BUSINESS	.00	16.55	16.55
6	FITNESS	.00	12.2	12.2
7	FAMILY LIFE	583.11	7.8	41.95
8	SELF HELP	.00	12.1	12.1

# Summary

- The AVG, SUM, STDDEV, and VARIANCE functions are used only with numeric fields
- The COUNT, MAX, and MIN functions can be applied to any datatype
- The AVG, SUM, MAX, MIN, STDDEV, and VARIANCE functions all ignore NULL values
- By default, the AVG, SUM, MAX, MIN, COUNT, STDDEV, and VARIANCE functions include duplicate values

# Summary (continued)

- The GROUP BY clause is used to divide table data into groups
- If a SELECT clause contains both an individual field name and a group function, the field name must also be included in a GROUP BY clause
- The HAVING clause is used to restrict groups in a group function
- Group functions can be nested to a depth of only two. The inner function is always performed first, using the specified grouping. The results of the inner function are used as input for the outer function.

# Summary (continued)

- The STDDEV and VARIANCE functions are used to perform statistical analyses on a set of data

# Oracle: SQL

*Selected Single-Row Functions*

# Objectives

- Use the UPPER, LOWER, and INITCAP functions to change the case of field values and character strings
- Manipulate character substrings with the SUBSTR and INSTR functions
- Nest functions inside other functions
- Determine the length of a character string using the LENGTH function
- Use the LPAD and RPAD functions to pad a string to a certain width
- Use the LTRIM and RTRIM functions to remove specific characters strings
- Substitute character string values with the REPLACE

# Objectives (continued)

- Round and truncate numeric data using the ROUND
- Return the remainder only of a division operation using the MOD function
- Use the ABS function to set numeric values as positive
- Calculate the number of months between two dates using the MONTHS\_BETWEEN function
- Manipulate date data using the ADD\_MONTHS, NEXT\_DAY, LAST\_DAY, and TO\_DATE functions

# Objectives (continued)

- Using the NVL function
- Display dates and numbers in a specific format with the TO\_CHAR function

# Terminology

- Function – predefined block of code that accepts arguments
- Single-row function – returns one row of results for each record processed
- Multiple-row function – returns one result per group of data processed (covered in the next chapter)

# Types of Functions

Type of Function	Functions
Case conversion functions	UPPER, LOWER, INITCAP
Character manipulation functions	SUBSTR, INSTR, LENGTH, LPAD/RPAD, LTRIM/RTRIM, REPLACE, TRANSLATE, CONCAT
Numeric functions	ROUND, TRUNC, MOD, ABS, POWER
Date functions	MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, TO_DATE, ROUND, TRUNC, CURRENT_DATE
Regular expressions	REGEXP_LIKE, REGEXP_SUBSTR
Other functions	NVL, NVL2, NULLIF, TO_CHAR, DECODE, CASE expression, SOUNDEX, TO_NUMBER

# Case Conversion Functions

- Case conversion functions alter the case of data stored in a column or character string
  - Used in a SELECT clause, they alter the appearance of the data in the results
  - Used in a WHERE clause, they alter the value for comparison

# LOWER Function

- Used to convert characters to lowercase letters

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there is a status bar with the text "ORA-00001: unique constraint (SYS\_C001034) violated" and "Rows: 1". Below this is a toolbar with icons for Undo, Redo, New, Open, Save, Print, Copy, Paste, Find, Replace, Cut, Copy, Paste, Find, Replace, and a Help icon.

The main area is a SQL editor window titled "Enter SQL Statement:" containing the following SQL code:

```
SELECT firstname, lastname
  FROM customers
 WHERE LOWER(lastname) = 'nelson';
```

Below the SQL statement is a toolbar with five buttons: "Results", "Script Output", "Explain", "Autotrace", and "DBMS Output".

The bottom section is titled "Results:" and displays a table with two columns: "FIRSTNAME" and "LASTNAME". The table contains one row with values "BECCA" and "NELSON".

# UPPER Function

- Used to convert characters to uppercase letters
- It can be used in the same way as the LOWER function
  - To affect the display of characters, it is used in a SELECT clause
  - To modify the case of characters for a search condition, it is used in a WHERE clause
- The syntax for the UPPER function is  
**UPPER(c)**
  - Where c is the character string or field to be converted into uppercase characters

# INITCAP Function

- Used to convert characters to mixed case

```
select INITCAP(city) from customers|
```

# Character Manipulation Functions

- Character manipulation functions manipulate data by extracting substrings, counting the number of characters, replacing strings, etc.

# SUBSTR Function

- Used to return a substring, or portion of a string

Enter SQL Statement:

```
SELECT DISTINCT zip, SUBSTR(zip, 1, 3), SUBSTR(zip, -3, 2)
FROM customers
WHERE SUBSTR(zip, -3, 2) < 30;
```

Results: Script Output Explain Autotrace DBMS Output OWA Output

	ZIP	SUBSTR(ZIP,1,3)	SUBSTR(ZIP,-3,2)
1	98115	981	11
2	12211	122	21
3	82003	820	00
4	02110	021	11
5	49006	490	00
6	31206	312	20
7	33111	331	11

# LENGTH Function

- Used to determine the number of characters in a string

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text input field containing the SQL query.
- SQL Query:**

```
SELECT DISTINCT LENGTH(address)
  FROM customers
 ORDER BY LENGTH(address) DESC;
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, and DBMS Output.
- Results:** A table displaying the results of the query.
- Table Data:**

	LENGTH(ADDRESS)
1	20
2	18
3	17
4	16
5	13
6	12
7	11

# LPAD and RPAD Functions

- Used to pad, or fill in, a character string to a fixed width

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

- Enter SQL Statement:** The query is:

```
SELECT firstname, LPAD(firstname, 12, ' '), LPAD(firstname, 12, '*')
FROM customers
WHERE firstname LIKE 'J%';
```
- Results:** The results are displayed in a grid:

	FIRSTNAME	LPAD(FIRSTNAME,12,"")	LPAD(FIRSTNAME,12,"*")
1	JORGE	JORGE	*****JORGE
2	JAKE	JAKE	*****JAKE
3	JASMINE	JASMINE	****JASMINE
4	JENNIFER	JENNIFER	***JENNIFER

# LTRIM and RTRIM Functions

- Used to remove a specific string of characters

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

- Enter SQL Statement:** The query is:

```
SELECT lastname, address, LTRIM(address, 'P.O. BOX')
FROM customers
WHERE state = 'FL';
```
- Results Tab:** The results show the output of the query:

	LASTNAME	ADDRESS	LTRIM(ADDRESS,'P.O.BOX')
1	MORALES	P.O. BOX 651	651
2	SMITH	P.O. BOX 66	66
3	NGUYEN	357 WHITE EAGLE AVE.	357 WHITE EAGLE AVE.
4	SCHELL	P.O. BOX 677	677

# REPLACE Function

- Substitutes a string with another specified string

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT address, REPLACE(address, 'P.O.', 'POST OFFICE')
  FROM customers
 WHERE state = 'FL';
```

Below the SQL statement, there is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output".

The bottom pane is titled "Results:" and displays a table with two columns: "ADDRESS" and "REPLACE(ADDRESS,'P.O.','POSTOFFICE')". The table contains four rows of data:

ADDRESS	REPLACE(ADDRESS,'P.O.','POSTOFFICE')
1 P.O. BOX 651	POST OFFICE BOX 651
2 P.O. BOX 66	POST OFFICE BOX 66
3 357 WHITE EAGLE AVE.	357 WHITE EAGLE AVE.
4 P.O. BOX 677	POST OFFICE BOX 677

# CONCAT Function

- Used to concatenate two character strings

The screenshot shows a SQL development environment with the following interface elements:

- Enter SQL Statement:** A text area containing the following SQL code:

```
SELECT firstname, lastname,
       CONCAT('Customer number: ', customer#) "Number"
  FROM customers
 WHERE state = 'FL';
```
- Toolbar:** A row of buttons for navigating results: **Results**, **Script Output**, **Explain**, **Autotrace**, **DBMS Output**, and **OWA Output**.
- Results:** A table displaying the output of the query. The table has three columns: FIRSTNAME, LASTNAME, and Number. The data is as follows:

	FIRSTNAME	LASTNAME	Number
1	BONITA	MORALES	Customer number: 1001
2	LEILA	SMITH	Customer number: 1003
3	NICHOLAS	NGUYEN	Customer number: 1013
4	STEVE	SCHELL	Customer number: 1015

# Number Functions

- Allow for manipulation of numeric data
  - ROUND
  - MOD
  - ABS

# ROUND Function

- Used to round numeric columns to a stated precision

Enter SQL Statement:

```
SELECT title, retail, ROUND(retail,1), ROUND(retail,0), ROUND(retail,-1)
FROM books;
```

Results:

	TITLE	RETAIL	ROUND(RETAIL,1)	ROUND(RETAIL,0)	ROUND(RETAIL,-1)
1	BODYBUILD IN 10 MINUTES A DAY	30.95	31	31	30
2	REVENGE OF MICKEY	22	22	22	20
3	BUILDING A CAR WITH TOOTHPICKS	59.95	60	60	60
4	DATABASE IMPLEMENTATION	55.95	56	56	60
5	COOKING WITH MUSHROOMS	19.95	20	20	20

# MOD Function

```
select mod(5,2) from customers
```

# ABS Function

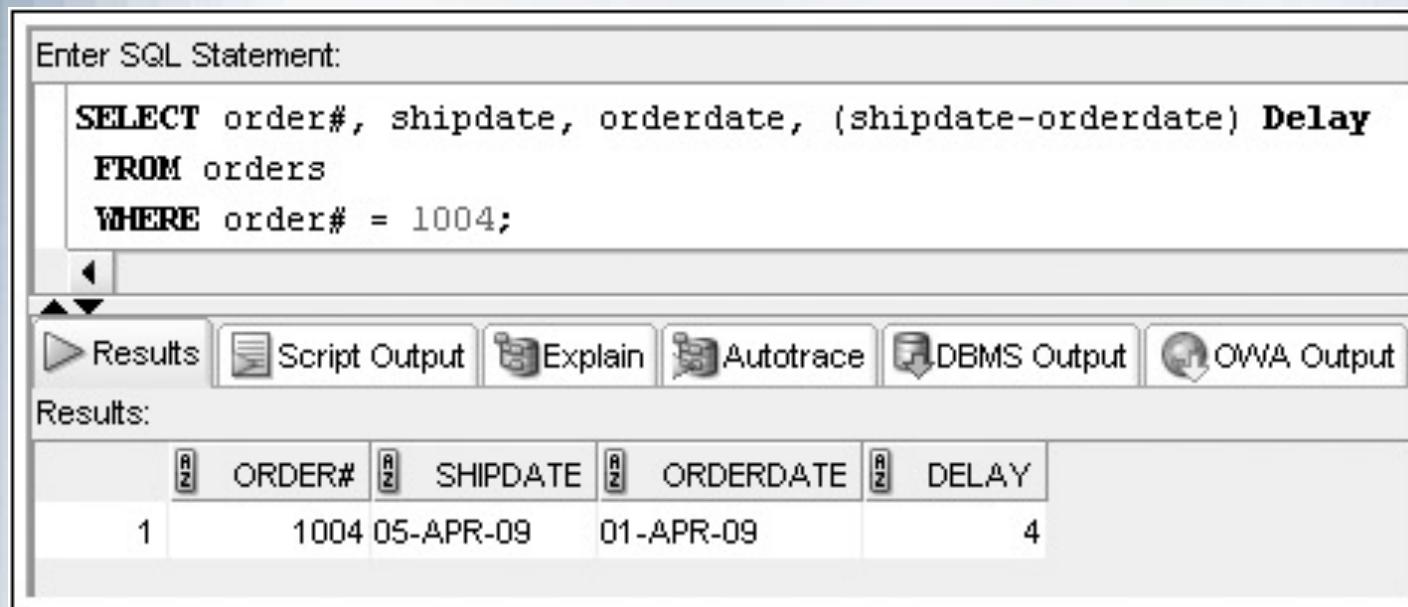
```
SELECT Abs(-243.5) AS AbsNum;
```

# Date Functions

- Used to perform date calculations or format date values
- Subtract date for number of days difference

Enter SQL Statement:

```
SELECT order#, shipdate, orderdate, (shipdate-orderdate) Delay
  FROM orders
 WHERE order# = 1004;
```

Results: 

ORDER#	SHIPDATE	ORDERDATE	DELAY
1	1004 05-APR-09	01-APR-09	4

# MONTHS\_BETWEEN Function

- Determines the number of months between two dates

Enter SQL Statement:

```
SELECT title, MONTHS_BETWEEN(orderdate,pubdate) MTHS
  FROM books JOIN orderitems USING (isbn)
            JOIN orders USING (order#)
 WHERE order# = 1004;
```

Results: Results Script Output Explain Autotrace DBMS Output OWA Output

	TITLE	MTHS
1	PAINLESS CHILD-REARING	56.48387096774193548387096774193548387097

# ADD\_MONTHS Function

- Adds a specified number of months to a date

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT title, pubdate, ADD_MONTHS('01-DEC-08',18) "Renegotiate Date",
       ADD_MONTHS(pubdate,84) "Drop Date"
  FROM books
 WHERE category = 'COMPUTER'
 ORDER BY "Renegotiate Date";
```

Below the code editor, there is a toolbar with several tabs: "Results", "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output". The "Results" tab is currently selected.

Under the "Results" tab, the output is displayed in a table format:

	TITLE	PUBDATE	Renegotiate Date	Drop Date
1	DATABASE IMPLEMENTATION	04-JUN-03	01-JUN-10	04-JUN-10
2	HOLY GRAIL OF ORACLE	31-DEC-05	01-JUN-10	31-DEC-12
3	HANDCRANKED COMPUTERS	21-JAN-05	01-JUN-10	21-JAN-12
4	E-BUSINESS THE EASY WAY	01-MAR-06	01-JUN-10	01-MAR-13

# NEXT\_DAY Function

- Determines the next occurrence of a specified day of the week after a given date

The screenshot shows the Oracle SQL Developer interface. In the top panel, there is a text input field labeled "Enter SQL Statement:" containing the following SQL code:

```
SELECT order#, orderdate, NEXT_DAY(orderdate, 'MONDAY')
FROM orders
WHERE order# = 1018;
```

Below the SQL statement, there is a toolbar with several buttons: "Results", "Script Output", "Explain", "Autotrace", "DBMS Output", and "OWA Output". The "Results" button is currently selected.

The bottom panel displays the results of the query execution. The results are presented in a table with three columns:

	ORDER#	ORDERDATE	NEXT_DAY(ORDERDATE,'MONDAY')
1	1018	05-APR-09	06-APR-09

# TO\_DATE Function

- Converts various date formats to the internal format (DD-MON-YY) used by Oracle 11g

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

- Enter SQL Statement:** A text area containing the following SQL query:

```
SELECT order#, orderdate, shipdate
FROM orders
WHERE orderdate = TO_DATE('March 31, 2009','Month DD, YYYY');
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output.
- Results:** A table displaying the output of the query. The table has columns: ORDER#, ORDERDATE, and SHIPDATE. The data is as follows:

	ORDER#	ORDERDATE	SHIPDATE
1	1000	31-MAR-09	02-APR-09
2	1001	31-MAR-09	01-APR-09
3	1002	31-MAR-09	01-APR-09

# NVL Function

- Substitutes a value for a NULL value

```
SELECT NVL(supplier_city, 'n/a')  
FROM suppliers;
```

# TO\_CHAR Function

- Converts dates and numbers to a formatted character string

The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there is a status bar with the text "ORA-00001: unique constraint (SYS\_C001034) violated" and "Rows: 1". Below this is a toolbar with icons for Undo, Redo, New, Open, Save, Print, Copy, Paste, Find, Replace, Cut, Copy, Paste, Find, Replace, and a Help icon.

The main area is a SQL editor window titled "Enter SQL Statement:". It contains the following SQL code:

```
SELECT title,
       TO_CHAR(pubdate, 'MONTH DD, YYYY') "Publication Date",
       TO_CHAR(retail, '$999.99') "Retail Price"
  FROM books
 WHERE isbn = 0401140733;
```

Below the SQL statement are several tabs: Results (selected), Script Output, Explain, Autotrace, DBMS Output, and OWA Output. The "Results" tab displays the query output:

AZ TITLE	AZ Publication Date	AZ Retail Price
1 REVENGE OF MICKEY	DECEMBER 14, 2005	\$22.00

# CASE Expression

```
select order#, shipcost,  
  
CASE  
  
WHEN shipcost > 5 then 'Expensive'  
WHEN shipcost > 3 then 'Ok'  
ELSE 'cheap'  
  
END  
  
from orders
```

Results	Explain	Describe	Saved SQL	History
<b>ORDER#</b> <b>SHIPCOST</b> <b>CASEWHENSHIPCOST&gt;5THEN'EXPENSIVE'WHENSHIPCOST&gt;3THEN</b>				
1000	2	cheap		
1001	3	cheap		
1002	3	cheap		
1003	4	Ok		
1004	-	cheap		
1005	2	cheap		
1006	2	chean		

# Oracle: SQL

*Subqueries*

# Objectives

- Determine when using a subquery is appropriate
- Identify which clauses can contain subqueries
- Distinguish between an outer query and a subquery
- Use a single-row subquery in a WHERE clause
- Use a single-row subquery in a HAVING clause
- Use a single-row subquery in a SELECT clause

# Objectives (continued)

- Distinguish between single-row and multiple-row comparison operators
- Use a multiple-row subquery in a WHERE clause
- Use a multiple-row subquery in a HAVING clause
- Use a multiple-column subquery in a WHERE clause

# Objectives (continued)

- Create an inline view using a multiple-column subquery in a FROM clause
- Compensate for NULL values in subqueries
- Distinguish between correlated and uncorrelated subqueries
- Nest a subquery inside another subquery
- Use a subquery in a DML action
- Process multiple DML actions with a MERGE statement

# Subqueries and Their Uses

- Subquery – a query nested inside another query
- Used when a query is based on an unknown value
- Requires SELECT and FROM clauses
- Must be enclosed in parentheses
- Place on right side of comparison operator

# Single-Row Subqueries

- Can only return one result to the outer query
- Operators include =, >, <, >=, <=, <>

# Single-Row Subquery in a WHERE Clause

- Used for comparison against individual data

Enter SQL Statement:

```
SELECT category, title, cost
  FROM books
 WHERE cost >
    (SELECT cost
      FROM books
     WHERE title = 'DATABASE IMPLEMENTATION')
   AND category = 'COMPUTER'
```

Results: Script Output Explain Autotrace DBMS Output

	CATEGORY	TITLE	COST
1	COMPUTER	HOLY GRAIL OF ORACLE	47.25
2	COMPUTER	E-BUSINESS THE EASY WAY	37.9

# Single-Row Subquery in a HAVING Clause

- Required when returned value is compared to grouped data

Enter SQL Statement:

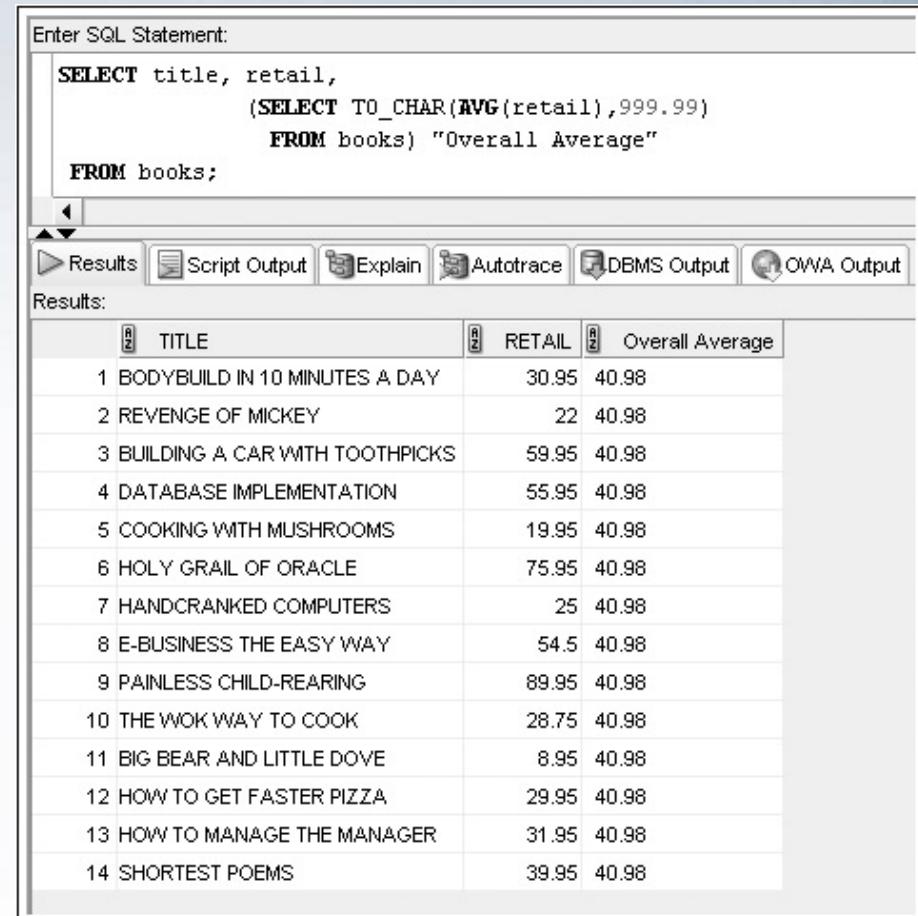
```
SELECT category, AVG(retail-cost) "Average Profit"
  FROM books
 GROUP BY category
 HAVING AVG(retail-cost) > (SELECT AVG(retail-cost)
                                FROM books
                               WHERE category = 'LITERATURE');
```

Results:

CATEGORY	Average Profit
1 COMPUTER	18.2625
2 FAMILY LIFE	24.875

# Single-Row Subquery in a SELECT Clause

- Replicates subquery value for each row displayed



The screenshot shows a SQL development environment with the following details:

- Enter SQL Statement:** The input field contains the following SQL query:

```
SELECT title, retail,
       (SELECT TO_CHAR(AVG(retail),999.99)
        FROM books) "Overall Average"
  FROM books;
```
- Results:** Below the input field is a results grid with the following data:

	TITLE	RETAIL	Overall Average
1	BODYBUILD IN 10 MINUTES A DAY	30.95	40.98
2	REVENGE OF MICKEY	22	40.98
3	BUILDING A CAR WITH TOOTHPICKS	59.95	40.98
4	DATABASE IMPLEMENTATION	55.95	40.98
5	COOKING WITH MUSHROOMS	19.95	40.98
6	HOLY GRAIL OF ORACLE	75.95	40.98
7	HANDCRANKED COMPUTERS	25	40.98
8	E-BUSINESS THE EASY WAY	54.5	40.98
9	PAINLESS CHILD-REARING	89.95	40.98
10	THE WOK WAY TO COOK	28.75	40.98
11	BIG BEAR AND LITTLE DOVE	8.95	40.98
12	HOW TO GET FASTER PIZZA	29.95	40.98
13	HOW TO MANAGE THE MANAGER	31.95	40.98
14	SHORTEST POEMS	39.95	40.98

# Multiple-Row Subqueries

- Return more than one row of results
- Require use of IN, ANY, ALL, or EXISTS operators

# ANY and ALL Operators

- Combine with arithmetic operators

OPERATOR	DESCRIPTION
>ALL	More than the highest value returned by the subquery
<ALL	Less than the lowest value returned by the subquery
<ANY	Less than the highest value returned by the subquery
>ANY	More than the lowest value returned by the subquery
=ANY	Equal to any value returned by the subquery (same as IN)

**FIGURE 12-11** Descriptions of ALL and ANY operator combinations

# Difference between Any and All

<b>Id</b>	<b>Age</b>	<b>Salary</b>
A	25	2000
B	33	4000
C	23	8500
D	27	6000
E	24	4150

Select \* from customers where age > ANY(Select age from customers where salary > 5000)

Any will be true if any of the subquery values meet the conditions

The inner query returned 23, 27

The outer query is then compared to the results if any of them either greater than 23 or 27

What records it will return?

# Difference between Any and All

<b>Id</b>	<b>Age</b>	<b>Salary</b>
A	25	2000
B	33	4000
C	23	8500
D	27	6000
E	24	4150

Select \* from customers where age > ALL(Select age from customers where salary > 5000)

All will be true if all of the subquery values meet the conditions

The inner query returned 23, 27

The outer query is then compared to the results if they are greater than 23 and 27

What records it will return?

# Multiple-Row Subquery in a WHERE Clause (continued)

Enter SQL Statement:

```
SELECT title, retail
  FROM books
 WHERE retail < ANY (SELECT retail
                        FROM books
                      WHERE category = 'COOKING');
```

Results:

	TITLE	RETAIL
1	BIG BEAR AND LITTLE DOVE	8.95
2	COOKING WITH MUSHROOMS	19.95
3	REVENGE OF MICKEY	22
4	HANDCRANKED COMPUTERS	25

# Multiple-Row Subquery in a HAVING Clause

Enter SQL Statement:

```
SELECT order#, SUM(quantity*paideach)
FROM orderitems
HAVING SUM(quantity*paideach) >ALL (SELECT SUM(quantity*paideach)
                                         FROM customers JOIN orders USING (customer#)
                                         JOIN orderitems USING (order#)
                                         WHERE state = 'FL'
                                         GROUP BY order#)

GROUP BY order#;
```

Results: Results Script Output Explain Autotrace DBMS Output OWA Output

ORDER#	SUM(QUANTITY*PAIDEACH)
1	117.4
2	111.9
3	335.85
4	170.9
5	166.4

# Multiple-Row Subquery in a WHERE Clause

Enter SQL Statement:

```
SELECT title, retail, category
  FROM books
 WHERE retail IN (SELECT MAX(retail)
                      FROM books
                     GROUP BY category)
 ORDER BY category;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	TITLE	RETAIL	CATEGORY
1	HOW TO MANAGE THE MANAGER	31.95	BUSINESS
2	BUILDING A CAR WITH TOOTHPICKS	59.95	CHILDREN
3	HOLY GRAIL OF ORACLE	75.95	COMPUTER
4	THE WOK WAY TO COOK	28.75	COOKING
5	PAINLESS CHILD-REARING	89.95	FAMILY LIFE
6	BODYBUILD IN 10 MINUTES A DAY	30.95	FITNESS
7	SHORTEST POEMS	39.95	LITERATURE
8	HOW TO GET FASTER PIZZA	29.95	SELF HELP

Note: Could use IN operator or =ANY

# Multiple-Column Subqueries

- Return more than one column in results
- Can return more than one row
- Column list on the left side of operator must be in parentheses
- Use the IN operator for WHERE and HAVING clauses

# Multiple-Column Subquery in a FROM Clause

- Creates a temporary table

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

- Enter SQL Statement:** A text area containing the following SQL code:

```
SELECT b.title, b.retail, a.category, a.cataverage
FROM books b, (SELECT category, AVG(retail) cataverage
                FROM books
               GROUP BY category) a
 WHERE b.category = a.category
   AND b.retail > a.cataverage;
```
- Toolbar:** Buttons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output.
- Results:** A table displaying the query results with columns: TITLE, RETAIL, CATEGORY, and CATAVERAGE. The data is as follows:

	TITLE	RETAIL	CATEGORY	CATAVERAGE
1	E-BUSINESS THE EASY WAY	54.5	COMPUTER	52.85
2	HOLY GRAIL OF ORACLE	75.95	COMPUTER	52.85
3	DATABASE IMPLEMENTATION	55.95	COMPUTER	52.85
4	THE WOK WAY TO COOK	28.75	COOKING	24.35
5	BUILDING A CAR WITH TOOTHPICKS	59.95	CHILDREN	34.45
6	PAINLESS CHILD-REARING	89.95	FAMILY LIFE	55.975

# Multiple-Column Subquery in a WHERE Clause

- Returns multiple columns for evaluation

Enter SQL Statement:

```
SELECT title, retail, category
  FROM books
 WHERE (category, retail) IN (SELECT category, MAX(retail)
                                FROM books
                               GROUP BY category)

 ORDER BY category;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	A Z TITLE	A Z RETAIL	A Z CATEGORY
1	HOW TO MANAGE THE MANAGER	31.95	BUSINESS
2	BUILDING A CAR WITH TOOTHPICKS	59.95	CHILDREN
3	HOLY GRAIL OF ORACLE	75.95	COMPUTER
4	THE WOK WAY TO COOK	28.75	COOKING
5	PAINLESS CHILD-REARING	89.95	FAMILY LIFE
6	BODYBUILD IN 10 MINUTES A DAY	30.95	FITNESS
7	SHORTEST POEMS	39.95	LITERATURE
8	HOW TO GET FASTER PIZZA	29.95	SELF HELP

# Using the exist condition

To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname  
FROM Customers  
WHERE EXISTS (SELECT *  
               FROM Orders  
              WHERE Customers.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

# Using the Not Exists condition

Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname  
FROM Customer  
WHERE NOT EXISTS (SELECT *  
                   FROM Orders  
                   WHERE Customers.customer_id = Orders.c_id);
```

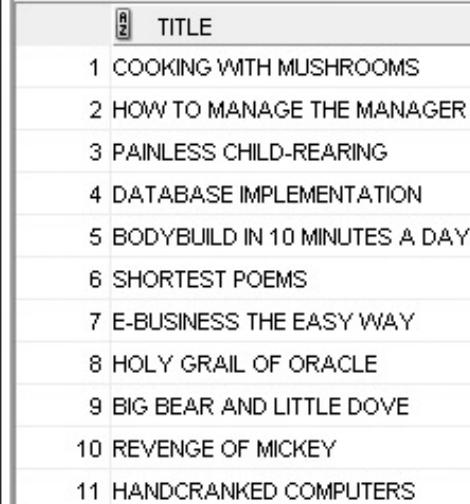
Output:

<b>lname</b>	<b>fname</b>
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

# The Exist Condition

Enter SQL Statement:

```
SELECT title
  FROM books
 WHERE EXISTS (SELECT isbn
                  FROM orderitems
                 WHERE books.isbn = orderitems.isbn);
```

Results: 

TITLE
1 COOKING WITH MUSHROOMS
2 HOW TO MANAGE THE MANAGER
3 PAINLESS CHILD-REARING
4 DATABASE IMPLEMENTATION
5 BODYBUILD IN 10 MINUTES A DAY
6 SHORTEST POEMS
7 E-BUSINESS THE EASY WAY
8 HOLY GRAIL OF ORACLE
9 BIG BEAR AND LITTLE DOVE
10 REVENGE OF MICKEY
11 HANDCRANKED COMPUTERS

# Nested Subqueries

- Maximum of 255 subqueries if nested in the WHERE clause
- No limit if nested in the FROM clause
- Innermost subquery is resolved first, then the next level, etc.

# Nested Subqueries (continued)

- Innermost is resolved first (A), then the second level (B), then the outer query (C)

The screenshot shows the Oracle SQL Developer interface with a SQL statement editor and a results grid.

**SQL Statement:**

```
SELECT customer#, lastname, firstname
  FROM customers JOIN orders USING(customer#) C
 WHERE order# IN (SELECT order#
                   FROM orderitems B
                   GROUP BY order#
                   HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                           FROM orderitems A
                           GROUP BY order#));
```

**Results:**

CUSTOMER#	LASTNAME	FIRSTNAME
1	1007	GIANA TAMMY
2	1017	NELSON BECCA

# Subquery in a DML action

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

The screenshot shows a SQL development environment with the following details:

- Enter SQL Statement:** The main input field contains the following SQL code:

```
UPDATE employees
SET bonus = (SELECT AVG(bonus)
              FROM employees)
WHERE empno = 8844;
```
- Toolbar:** Below the input field is a toolbar with several buttons: Results, Script Output, Explain, Autotrace, and DBMS Output.
- Status Bar:** At the bottom of the interface, the status bar displays the message "1 rows updated".

# Summary

- A subquery is a complete query nested in the SELECT, FROM, HAVING, or WHERE clause of another query
  - The subquery must be enclosed in parentheses and have a SELECT and a FROM clause, at a minimum
- Subqueries are completed first; the result of the subquery is used as input for the outer query
- A single-row subquery can return a maximum of one value
- Single-row operators include =, >, <, >=, <=, and <>
- Multiple-row subqueries return more than one row of results

# Summary (continued)

- Operators that can be used with multiple-row subqueries include IN, ALL, ANY, and EXISTS
- Multiple-column subqueries return more than one column to the outer query
- NULL values returned by a multiple-row or multiple-column subquery will not present a problem if the IN or =ANY operator is used
- Correlated subqueries reference a column contained in the outer query
- Subqueries can be nested to a maximum depth of 255 subqueries in the WHERE clause of the parent query

# Summary (continued)

- With nested subqueries, the innermost subquery is executed first, then the next highest level subquery is executed, and so on, until the outermost query is reached

# Database Concepts

Ninth Edition



## Chapter 4

Data Modeling and the Entity-Relationship Model

# Learning Objectives (1 of 2)

- Learn the basic steps of systems analysis and design
- Learn the basic stages of database development
- Understand the purpose and role of a data model
- Know the principle components of the E-R data model
- Understand how to interpret traditional E-R diagrams
- Understand how to interpret the Information Engineering (IE) model's Crow's Foot E-R diagrams
- Learn to construct E-R diagrams
- Learn the purpose of a database management system (DBMS)
- Know how to represent 1:1, 1:N, N:M, and binary relationships with the E-R model

# Learning Objectives (2 of 2)

- Understand two types of weak entities and how to use them
- Understand nonidentifying and identifying relationships and know how to use them
- Know how to represent subtype entities with the E-R model
- Know how to represent recursive relationships with the E-R model
- Learn how to create an E-R diagram from source documents

# Data and Information

## Learn the basic steps of system analysis and design

- **Data** is defined as recorded facts and numbers
- **Information** is defined as:
  - Knowledge derived from data
  - Data presented in a meaningful context
  - Data processed by summing, ordering, averaging, grouping, comparing, or other similar operations

# What is an Information System?

## Learn the basic steps of system analysis and design

- A **System** is defined as a set of components that interact to achieve some purpose or goal
- An **information system** is a system that has the goal of producing information and is composed of the following components:
  - Hardware
  - Software
  - Data
  - Procedures
  - People

# Figure 4.1 The Five-Component Information System Framework

Hardware	Software	Data	Procedures	People
----------	----------	------	------------	--------

# Figure 4.2 A Generalized Business Process

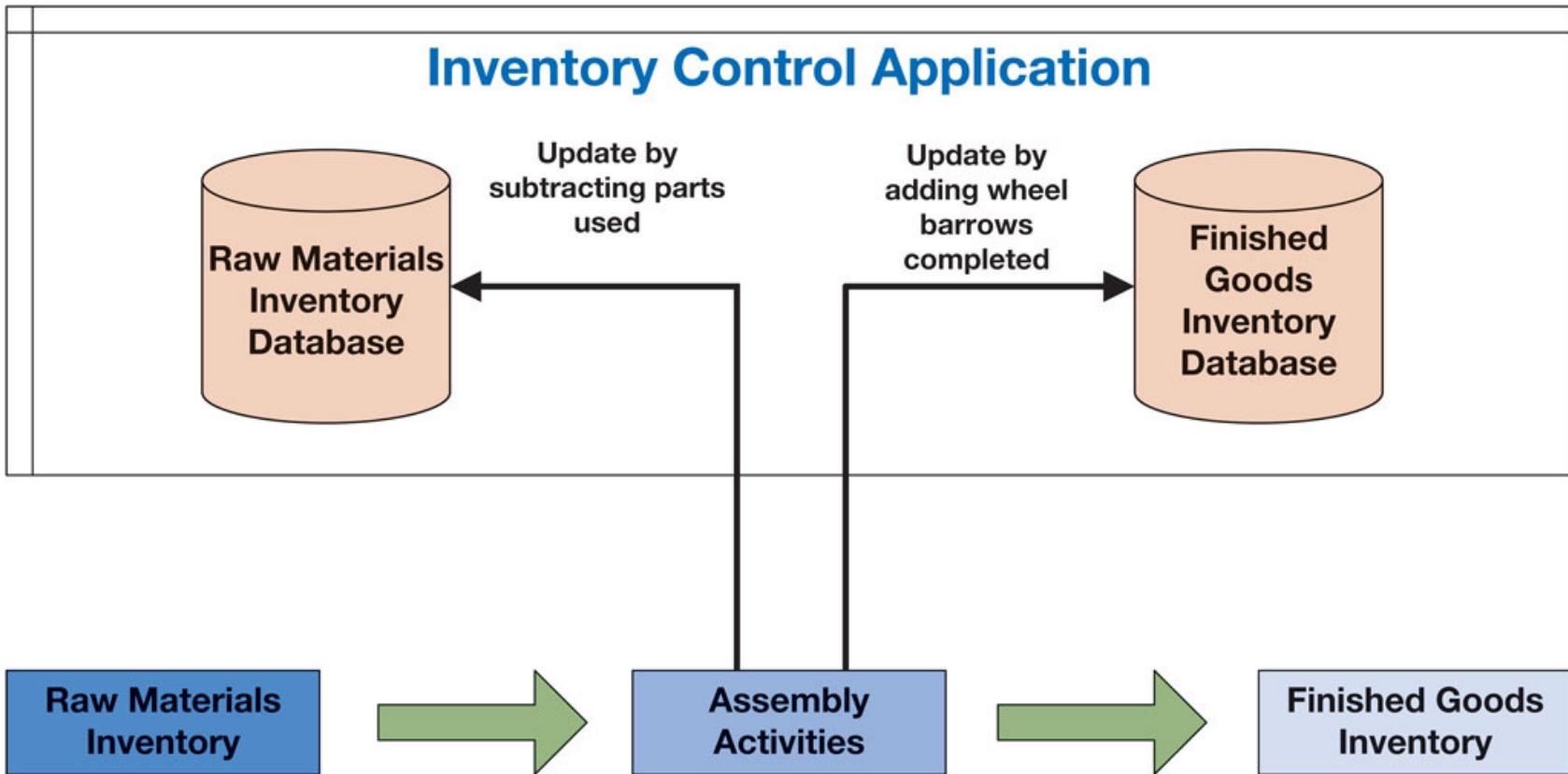
A **business process** is a set of activities that transform input into outputs



# Figure 4.3 The Manufacturing Process



# Figure 4.4 The Manufacturing Process with Supporting Information System

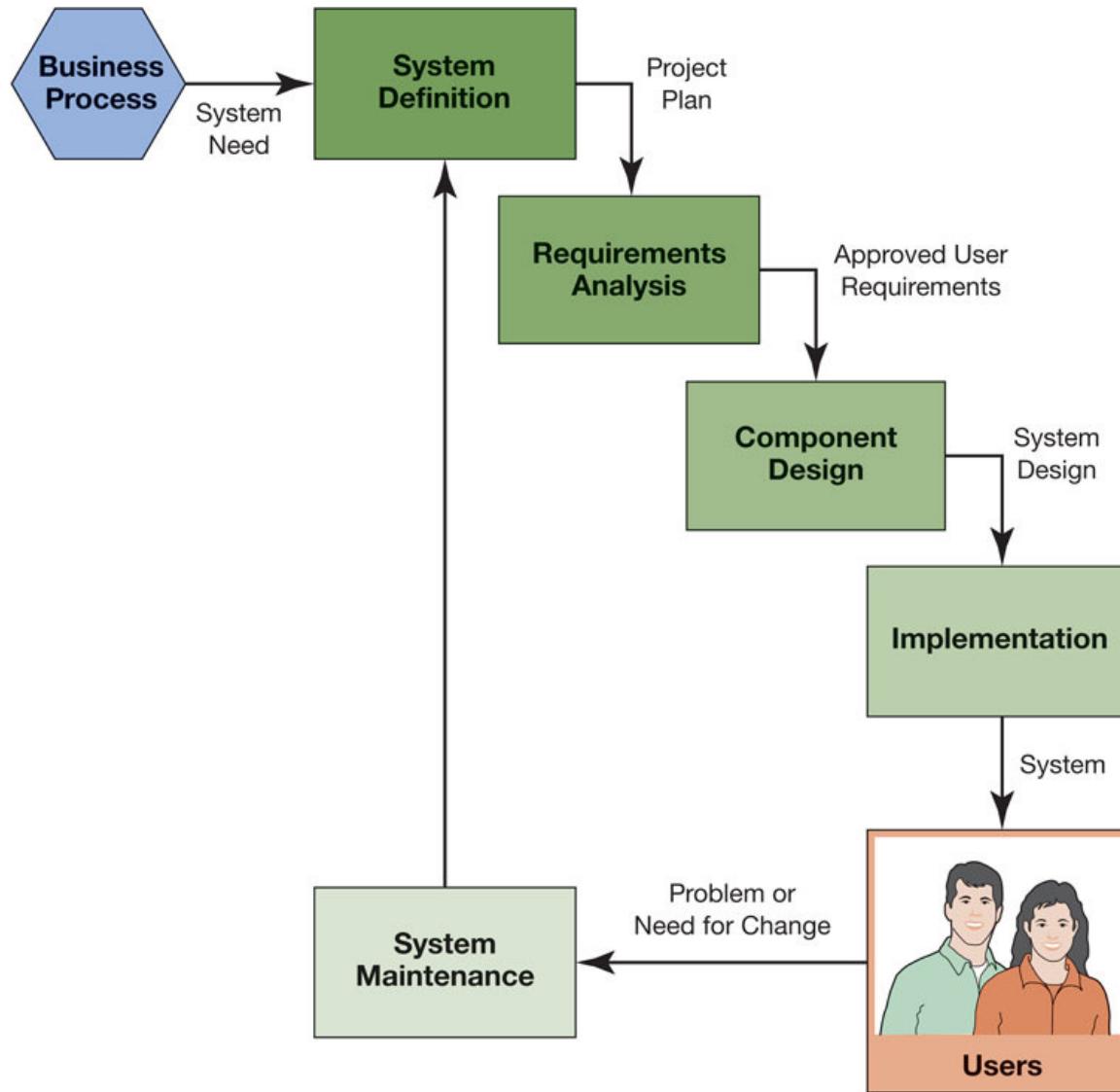


# Systems Analysis and Design

## Learn the basic steps of system analysis and design

- **Systems analysis and design** is the process of creating and maintaining information systems.
- The classic methodology used in systems analysis and design to developed information systems is called the **systems development life cycle (SDLC)** and is composed of the following steps:
  - Systems definition
  - Requirements analysis
  - Component design
  - Implementation
  - System maintenance

# Figure 4.5 The SDLC in Use



# The System Definition Step

## Learn the basic steps of system analysis and design

- The **system definition** step is a *process* that starts with the need for an information system to support a business process as its *input* and produces a plan as its *output*.
- This step includes:
  - Define the information system project goals and scope
  - Assess the feasibility of the project (cost, schedule, technical, organizational)
  - Form the project team
  - Plan the project (specify tasks, assign personnel, determine task dependencies, set schedules)
- The deliverable is the project plan

# The Requirements Analysis Step

## Learn the basic steps of system analysis and design

- The **requirements analysis** step is a *process* that starts with the project plan as its *input* and produces a set of approved user requirements as its *output*
- This step includes:
  - Conduct user interviews
  - Evaluate existing systems
  - Determine needed new forms/reports/queries
  - Identify needed new applications features and functions
  - Consider security
  - Consider the five components of an information system – hardware, software, data, procedures, people
- The deliverable is the approved user requirements

# The Component Design Step

## Learn the basic steps of system analysis and design

- The **component design** step is a *process* that starts with the approved user requirements as its *input* and produces a final system design as its *output*.
- This step includes:
  - Determine hardware specifications
  - Determine program (software) specifications
  - Create the database design
  - Design business procedures
  - Create job descriptions for business personnel
- The deliverable is the documented system design

# The Implementation Step

## Learn the basic steps of system analysis and design

- The **Implementation** step is a *process* that starts with the final system design as its *input* and produces a final system as its *output*.
- This step includes:
  - Build system components
  - Conduct component tests
  - Integrate the components
  - Conduct integrated component tests
  - Convert to the new system
- The deliverable is the installed and functioning information system.

# Figure 4.6 The SDLC Design and Implementation Steps for the Five Information System Components

	Hardware	Software	Data	Procedures	People
Component Design Step	Determine hardware specifications	Select off-the-shelf software if available. Design custom programs if necessary	Design database and related application components	Design user and operational procedures	Develop job descriptions
Implementation Step	Obtain, install, and test hardware	License and install off-the-shelf software. Create custom programs if necessary. Test programs	Create database. Populate with data. Test database and data.	Document procedures. Create training programs. Review and test procedures	Train personnel. Hire new personnel if necessary.
Integrated Testing and Startup					

# The System Maintenance Step

## Learn the basic steps of system analysis and design

- The **system maintenance** step is a *process* that starts with the implemented system as its *input* and produces an updated system or a request of system modification using the SDLC as its *output*.
- This step includes:
  - Update the system with patches, service packs, and new software releases
  - Record and prioritize requests for system changes or enhancements
- The deliverable is an updated system and the start of a new SDLC cycle to enhance the information system.

# What are the Steps in the Database Development Process?

## Learn the basic stages of database development

- The **database development process** is a subset of the SDLC that consists of three major stages:
  1. Requirements analysis
  2. Component design
  3. Implementation

# **Figure 4.7 Sources of Requirements for a Database Applications**

**User interviews**  
**Forms**  
**Reports**  
**Queries**  
**Application programs**  
**Web sites**  
**Use cases**  
**Business rules**

# The Entity-Relationship Data Model

## Know the principle components of the E-R model

- When you create a database, data requirements must first be documented in a data model
- A number of techniques can be used to create data models
  - The most popular is the **entity-relationship model** created by Peter Chen in 1976 (now interpreted as the extended entity-relationship model)
    - The most important elements of the E-R model are entities, attributes, identifiers, and relationships.

# Entities

## Know the principle components of the E-R model

- An **entity** is something that users want to track. Examples include customers, purchases, products, etc.
- Entities of a given type are grouped into an **entity class** such as EMPLOYEE (a collection of all EMPLOYEE entities) which are shown in all caps.
- An **entity instance** of an entity class is the occurrence of a particular entity, such as CUSTOMER 12345.

# Figure 4.8 The ITEM Entity and Two Entity Instances

ITEM
ItemNumber
Description
Cost
ListPrice
QuantityOnHand

Entity Class

1100
100 amp panel
\$127.50
\$170.00
14

2000
Door handle set
\$39.38
\$52.50
0

Two Entity Instances

# Attributes

## Know the principle components of the E-R model

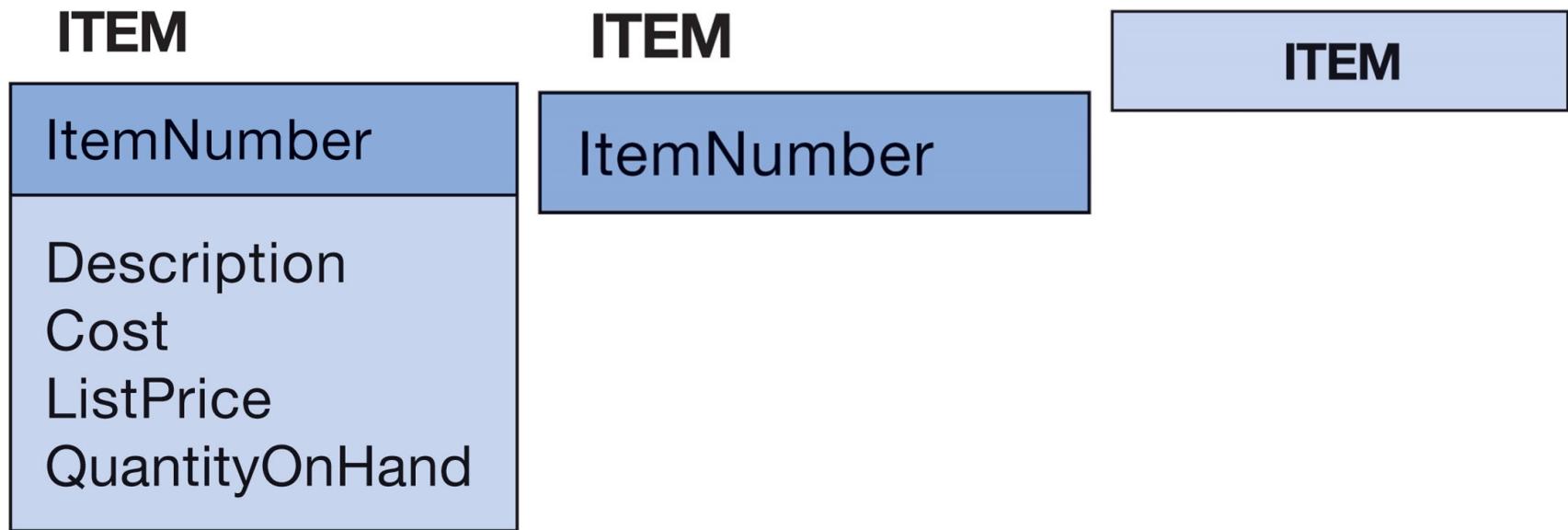
- Entities have **attributes**, which describe the entity's characteristics.
- Examples include EmployeeName, DateOfHire, and JobSkillCode.
- Attributes are shown with the first letter of each word capitalized as shown above.
- An attribute has a data type (character, numeric, date, currency, etc.)

# Identifiers

## Know the principle components of the E-R model

- Entity instances have **identifiers**, which are attributes that name, or identify, entity instances.
- Examples include ItemNumber identifying an instance of ITEM and SocialSecurityNumber identifying an instance of EMPLOYEE.
- Identifiers may be unique or nonunique
  - **Unique identifiers** identifies one, and only one, entity instance
  - **Nonunique identifiers** identifies a set in instances
    - An example is EmployeeName is an example as there may be more than one person with that name

# Figure 4.9 Levels of Entity Attribute Display

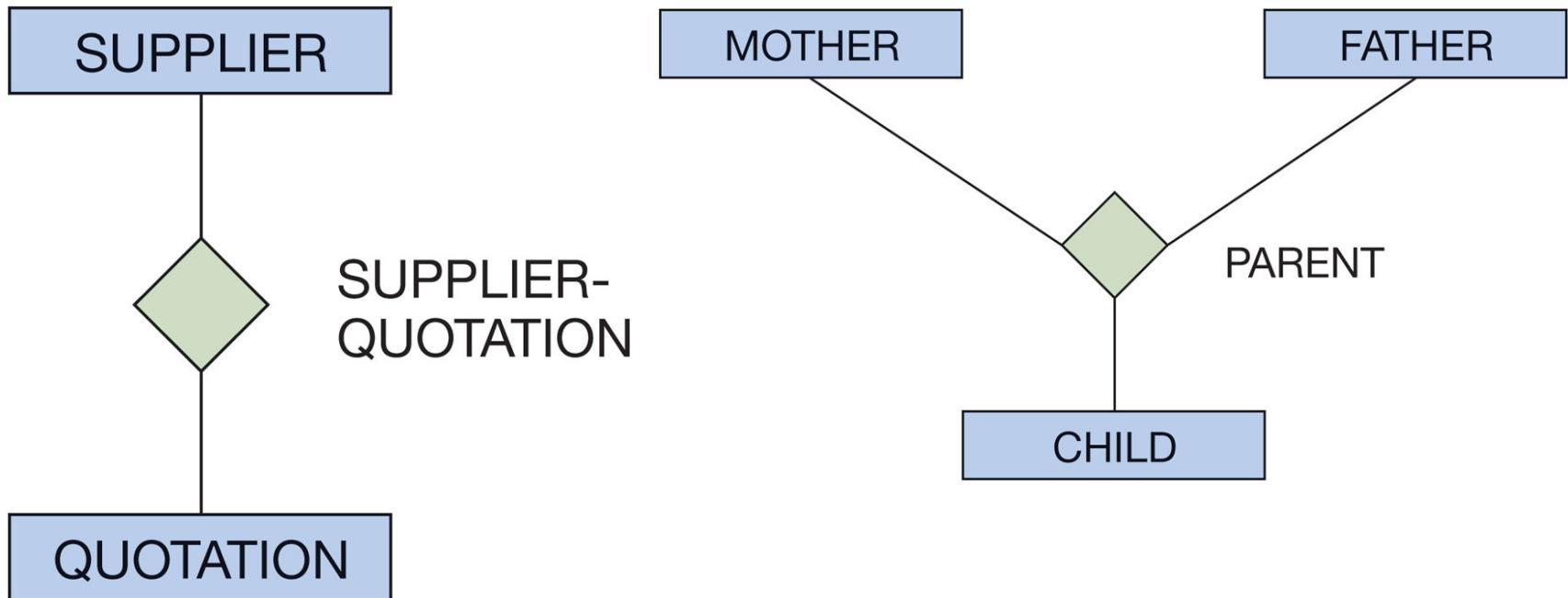


# Relationships

## Know the principle components of the E-R model

- Entities can be associated with one another in **relationships**.
- The number of entity classes in the relationship is known as the **degree** of the relationship as follows:
  - degree 2 is a **binary relationship**
  - degree 3 is a **ternary relationship**

# Figure 4.10 Example Relationships



# Figure 4.11 Three Types of Binary Relationships

LOCKER-ASSIGNMENT



ITEM-QUOTE



ITEM-SOURCE



# Maximum Cardinality (1 of 2)

## Know the principle components of the E-R model

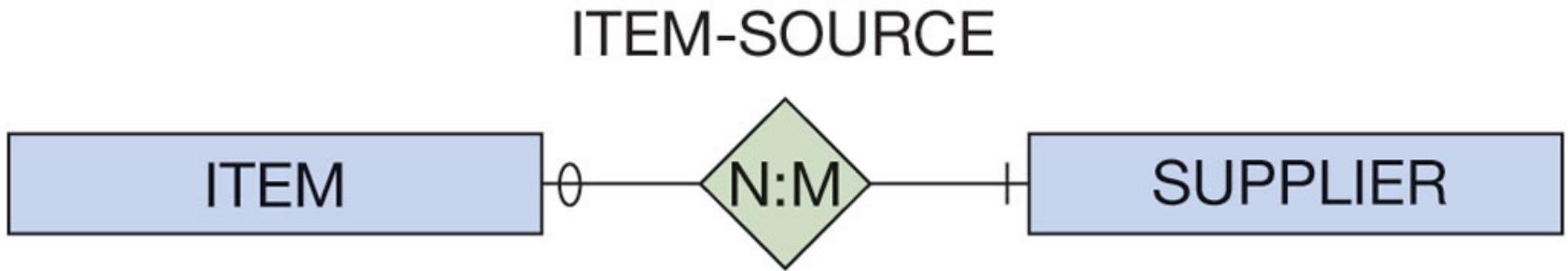
- Relationships are named and classified by their **cardinality**, which is a word that means *count*.
- Each of the three types of binary relationships shown in the previous slide have different *maximum cardinalities*.
- **Maximum cardinality** is the maximum number of entity instances that may participate in a relationship instance.

# Maximum Cardinality (2 of 2)

## Know the principle components of the E-R model

- **Minimum cardinality** is the minimum number of entity instances that *must* participate in a relationship instance.
- These values typically assume a value of zero (optional) or one (mandatory).

# Figure 4.12 A Relationship with Minimum Cardinalities



# HAS-A Relationships

## Understand how to interpret traditional E-R diagrams

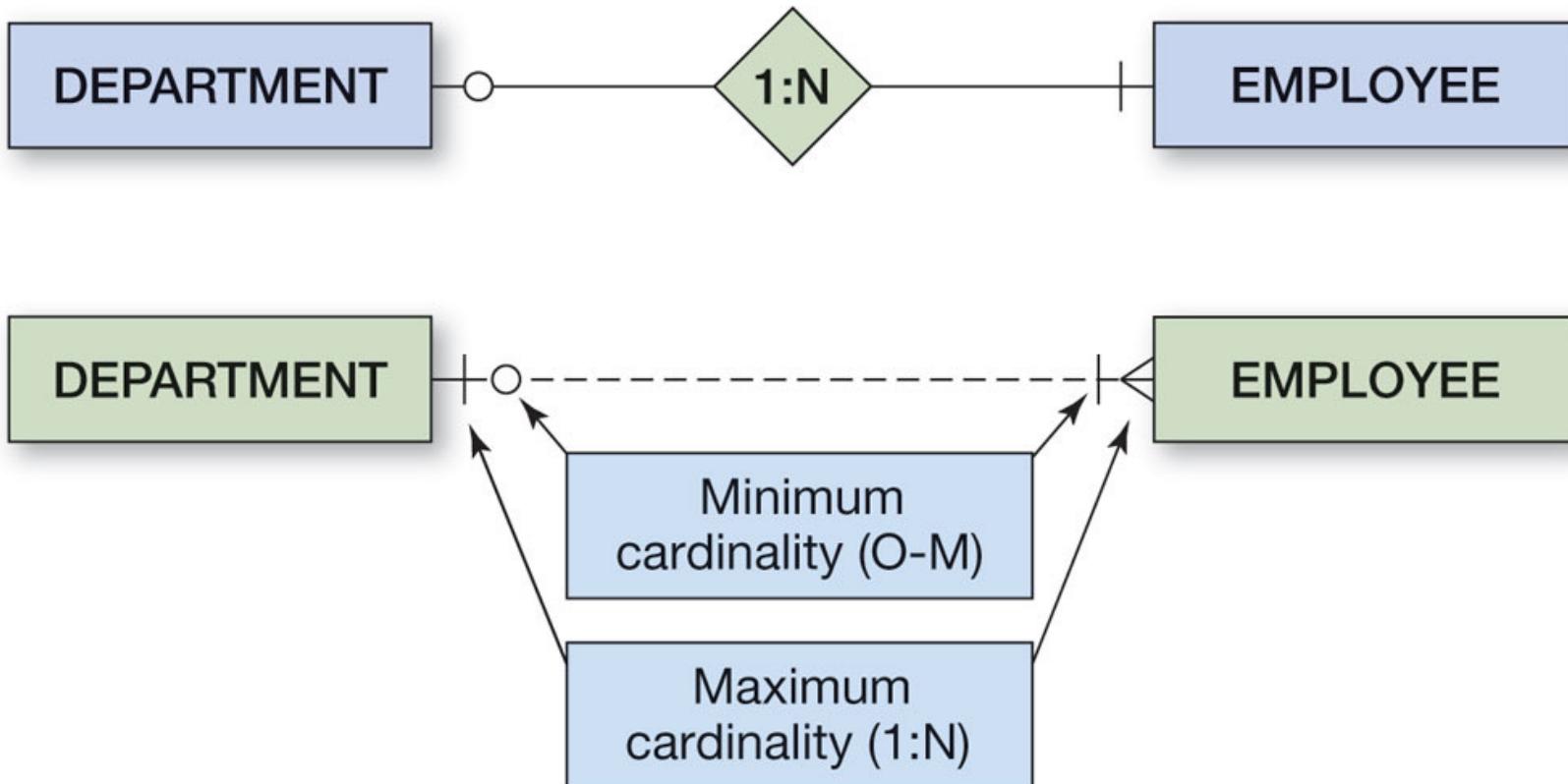
- The relationships in the previous slides are called **HAS-A relationships**.
- The term is used because each entity instance *has* a relationship to a second entity instance:
  - an employee has a badge
  - a badge has an employee

# Variations of the E-R Model

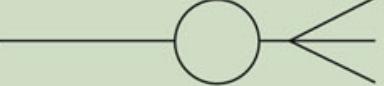
## Understand how to interpret the Information Engineering (IE) model's Crow's Foot E-R Diagrams

- **Information Engineering (IE)** [James Martin 1990] It uses “crow’s feet” to show the many sides of a relationship, and is sometimes called the **crow’s foot model**.
- **Integrated Definition 1, Extended (IDEFIX)** is a version of the E-R model that is a national standard.
- **Unified Modeling Language (UML)** is a set of structures and techniques for modeling and designing object-oriented programs (OOP) and applications.

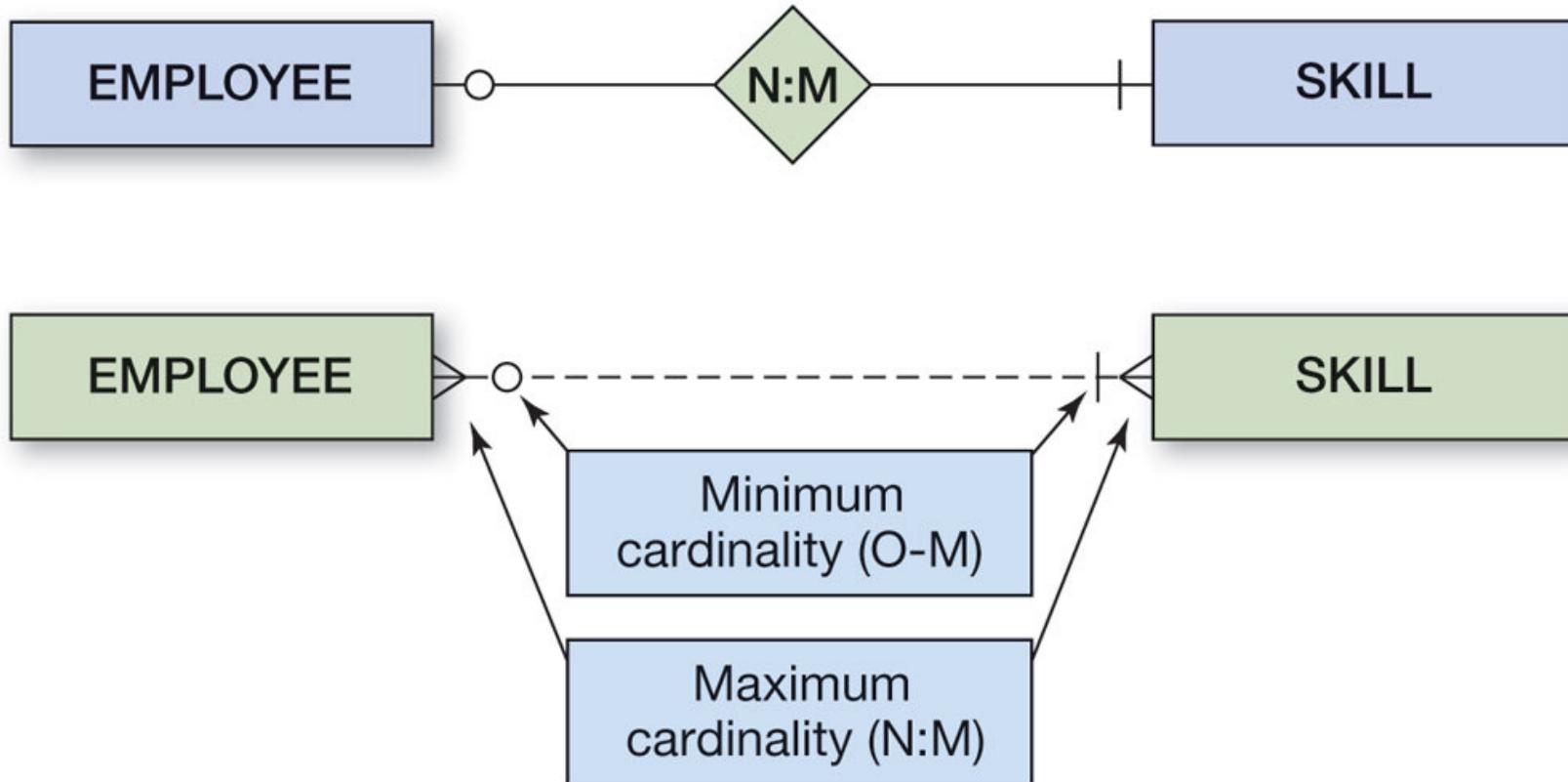
## Figure 4.13 Two Versions of a 1:N O-M Relationship



# Figure 4.14 Crow's Foot Notation

Symbol	Meaning	Numeric Meaning
	Mandatory—One	Exactly one
	Mandatory—Many	One or more
	Optional—One	Zero or one
	Optional—Many	Zero or more

## Figure 4.15 Two Versions of a N:M O-M Relationship



# Weak Entities

**Understand two types of weak entities and know how to use them**

- A **weak entity** is an entity that cannot exist in the database without the existence of another entity.
- Any entity that is *not* a weak entity is called a **strong entity**.

# ID-Dependent Weak Entities

**Understand two types of weak entities and know how to use them**

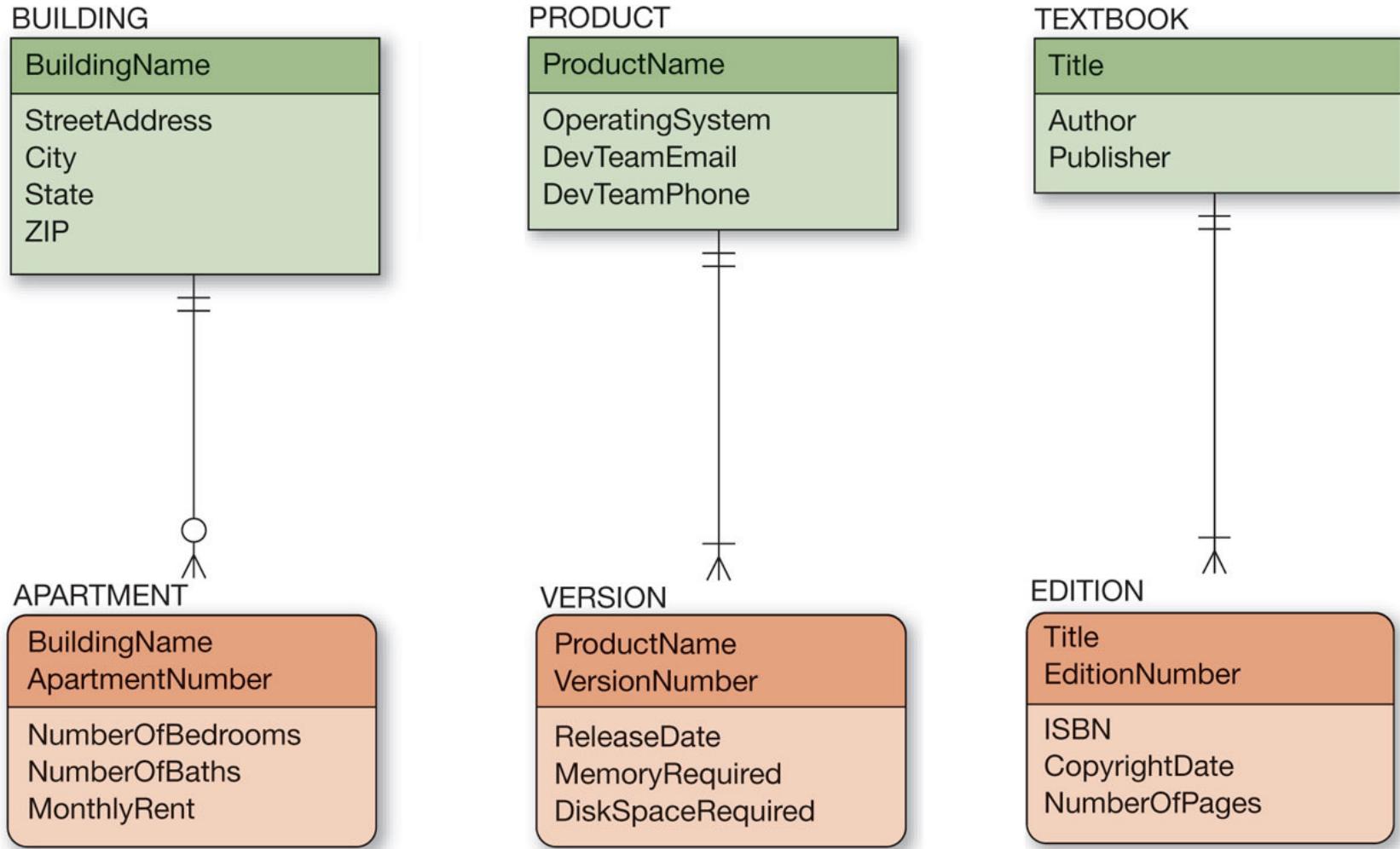
- An **ID-dependent weak entity** is a weak entity that cannot exist without its parent entity.
- An ID-dependent weak entity has a composite identifier:
  - the first part of the identifier is the identifier for the strong entity
  - the second part of the identifier is the identifier for the weak entity itself

# Weak Entity Relationships

## Understand nonidentifying and identifying relationships and how to use them

- The relationship between a strong and weak entity is termed an **identifying relationship** if the weak entity is ID-dependent:
  - represented by a solid line
- The relationship between a strong and weak entity is termed a **nonidentifying relationship** if the weak entity is non-ID-dependent:
  - represented by a dashed line
  - also used between strong entities

# Figure 4.16 Example ID-Dependent Entities

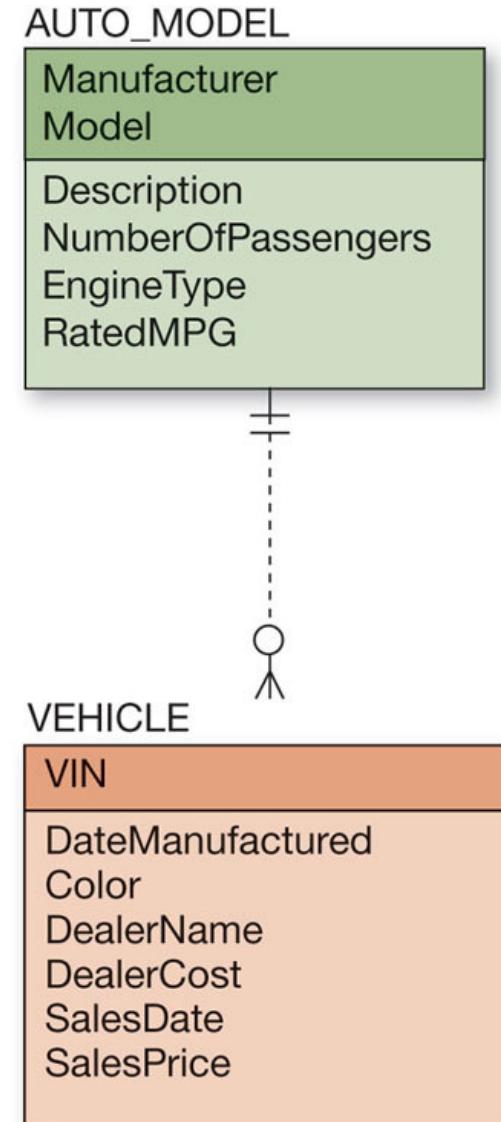
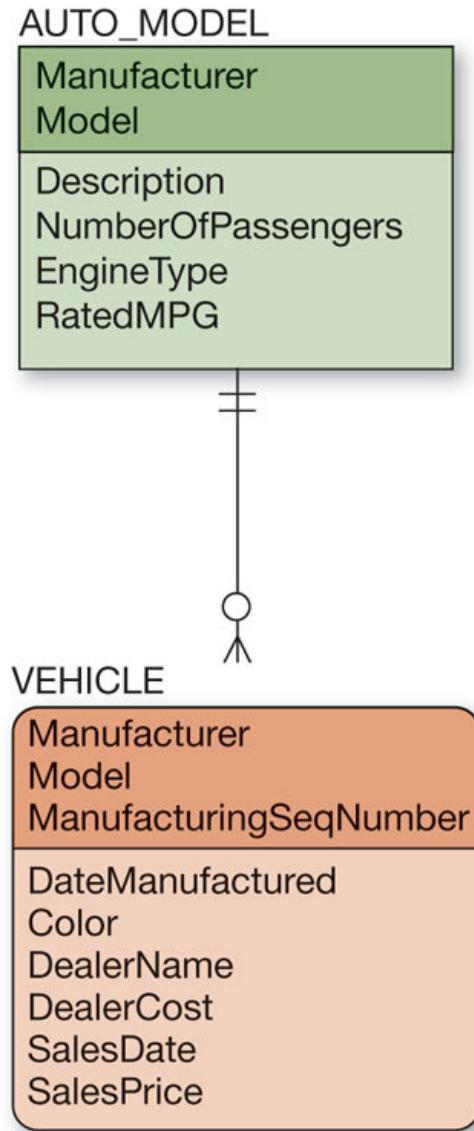


# Non-ID-Dependent Weak Entities

## Understand nonidentifying and identifying relationships and how to use them

- All ID-dependent entities are weak entities, but there are other entities that are weak but not ID-dependent.
- A non-ID-dependent weak entity may have a single or composite identifier, but the identifier of the parent entity will be a foreign key.

# Figure 4.17 Weak Entity Examples



# Figure 4.18 Examples of Required Entities

SALESPERSON
SalespersonID
SalespersonName
Phone
EmailAddress



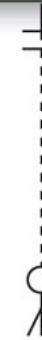
ORDER
OrderID
CustomerName
OrderDate
OrderSubtotal
OrderTax
OrderTotal

PROJECT
ProjectID
ProjectName
BudgetCode
Description



ASSIGNMENT
ProjectID
AssignmentID
StartDate
EndDate
BudgetAmount
ActualAmount

PATIENT
PatientID
PatientName
Address
City
State
ZIP



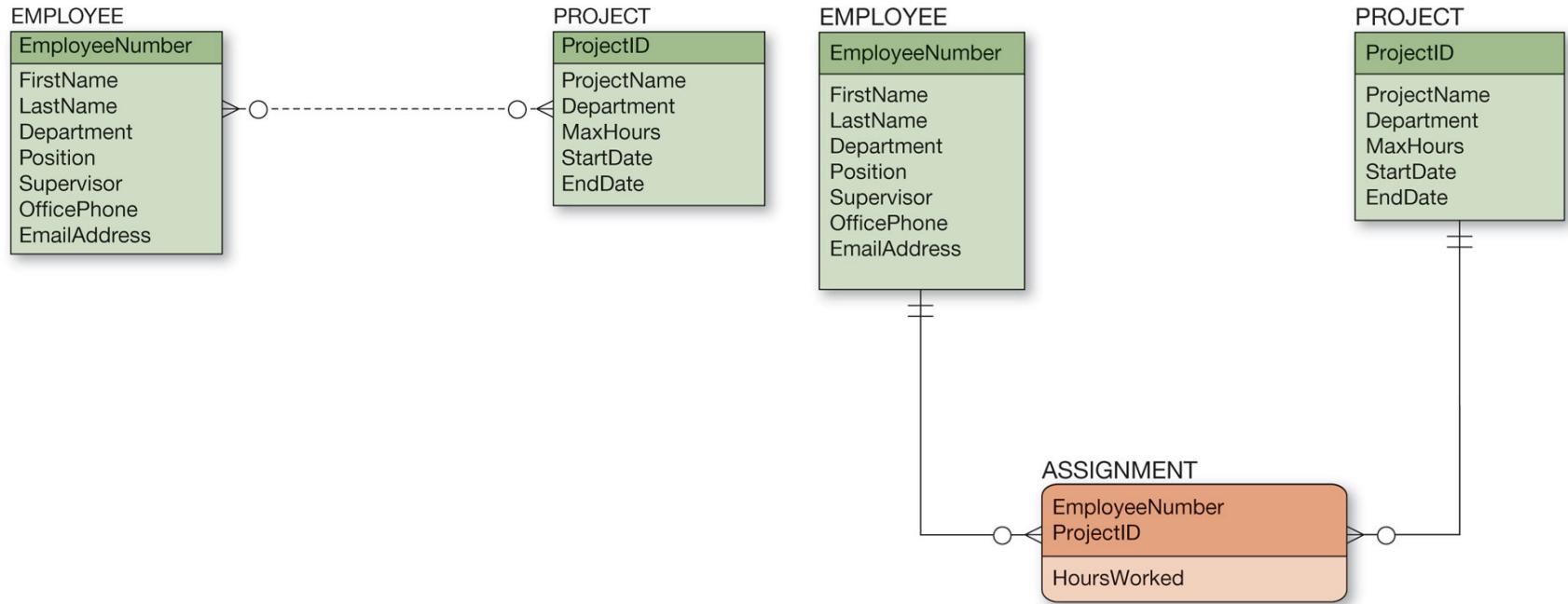
PRESCRIPTION
PrescriptionID
PrescriptionDate
PrescriptionText
isGenericDrugAllowed

# Associative Entities

## Understand nonidentifying and identifying relationships and how to use them

- An **associative entity** (also called an **association entity**) is used whenever a pure N:M relationship cannot properly hold attributes that are describing aspects of the relationship between two entities.
- A new entity is then created to:
  - link the two original entities
  - hold the attributes

# Figure 4.19 The Associative Entity

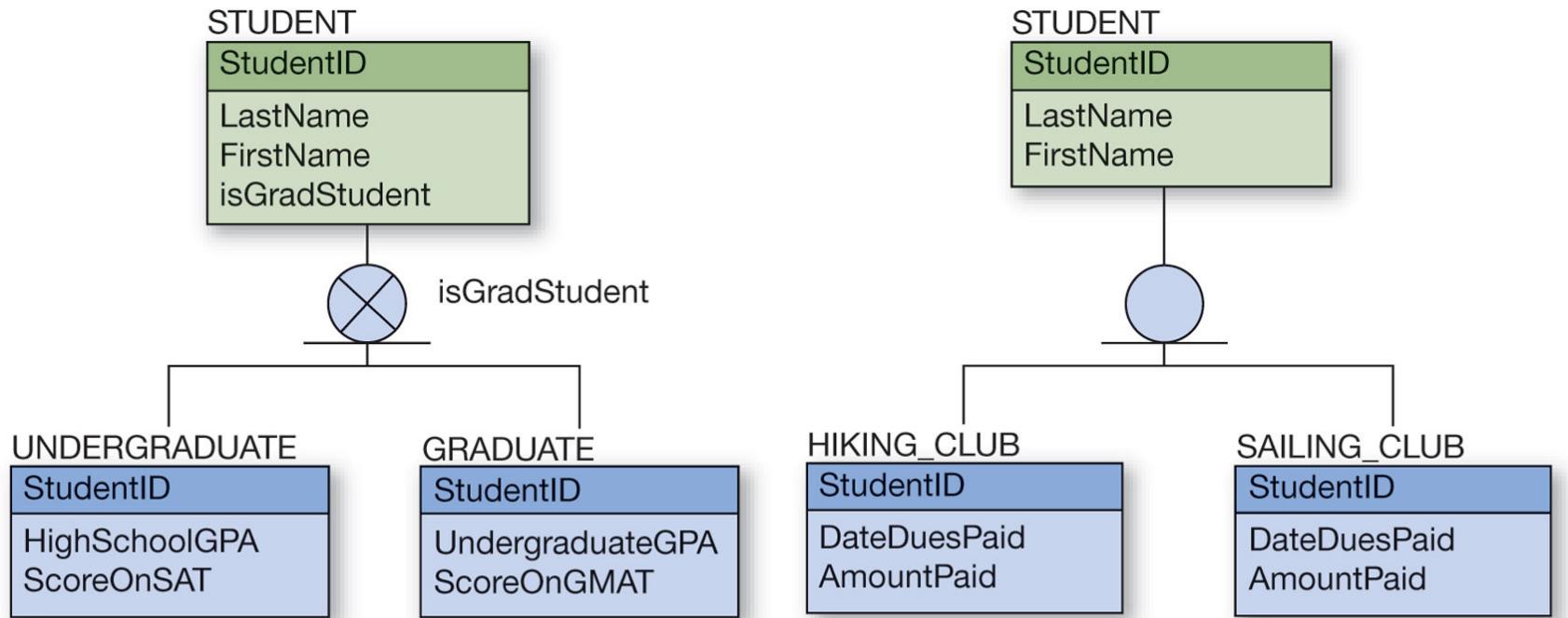


# Subtype Entities

## Know how to represent subtype entities with the E-R model

- A **subtype** entity is a special case of another entity called supertype.
- An attribute of the **supertype** indicates which of the subtypes is appropriate for a given instance and is called a **discriminator**.
- Subtypes can be exclusive or inclusive:
  - if **exclusive**, the supertype relates to at most one subtype
  - if **inclusive**, the supertype can relate to one or more subtypes

# Figure 4.20 Example Subtype Entities



# Subtype Entity Identifiers

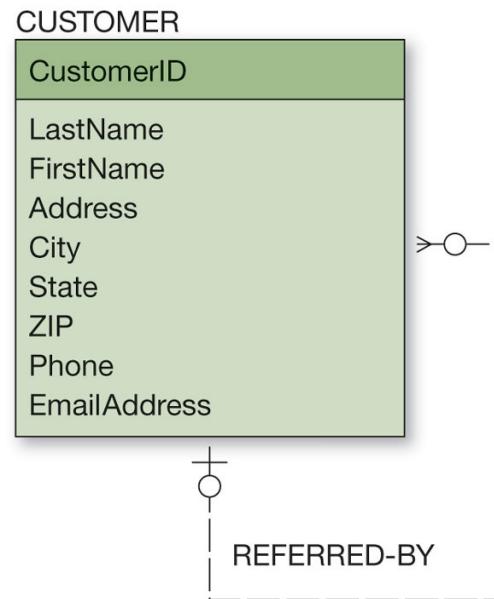
## Know how to represent subtype entities with the E-R model

- The relationships that connect supertypes and subtypes are called **IS-A relationship** because a subtype is the same entity as the supertype.
- The identifier of a supertype and all of its subtypes is the same attribute

# Figure 4.21 Example Recursive Relationship

Know how to represent recursive relationships with the E-R model

- It is possible for an entity to have a relationship to itself—this is called a **recursive relationship** (also known as a unary relationship)



# Developing an Example E-R Diagram

## Learn how to create an E-R Diagram from source documents

- Heather Sweeney Designs will be used as an on-going example throughout Chapters 4, 5, 6, and 7:
  - Heather Sweeney is an interior designer who specializes in home kitchen design
  - offers a variety of free seminars at home shows, kitchen and appliance stores, and other public locations
  - earns revenue by selling books and videos that instruct people on kitchen design
  - offers custom-design consulting services

# Figure 4.22 Example Seminar Customer List

*Heather Sweeney Designs  
Seminar Customer List*

Date: October 12, 2018

Location: San Antonio Convention Center

Time: 11 AM

Title: Kitchen on a Budget

Name	Phone	Email Address
------	-------	---------------

Nancy Jacobs	817-871-8123	Nancy.Jacobs@somewhere.com
--------------	--------------	----------------------------

Chantel Jacobs	817-871-8234	Chantel.Jacobs@somewhere.com
----------------	--------------	------------------------------

Ralph Able	210-281-7987	Ralph.Able@somewhere.com
------------	--------------	--------------------------

Etc.

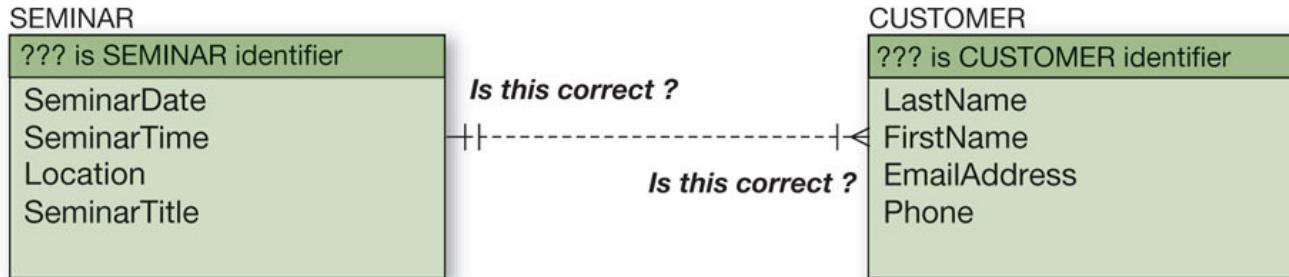
27 names in all

# Heather Sweeney Design Fact

## Learn how to create an E-R diagram from source documents

- Having missing facts is typical during the data modeling process as there are a number of undetermined facts from the Heather Sweeney design.
  - It is not certain about the 1:N cardinality
  - Neither is it know what to use for the identifier for each entity
- Once all information is know, the initial E-R Diagram can be created.

# Figure 4.23 Initial E-R Diagram for Heather Sweeney Designs



# Figure 4.24 Heather Sweeney Designs Customer Form Letter

**Heather Sweeney Designs**  
122450 Rockaway Road  
Dallas, Texas 75227  
972-233-6165

Ms. Nancy Jacobs  
1440 West Palm Drive  
Fort Worth, Texas 76110

Dear Ms. Jacobs:

Thank you for attending my seminar "Kitchen on a Budget" at the San Antonio Convention Center. I hope that you found the seminar topic interesting and helpful for your design projects.

As a seminar attendee, you are entitled to a 15 percent discount on all of my video and book products. I am enclosing a product catalog and I would also like to invite you to visit our Web site at [www.Sweeney.com](http://www.Sweeney.com).

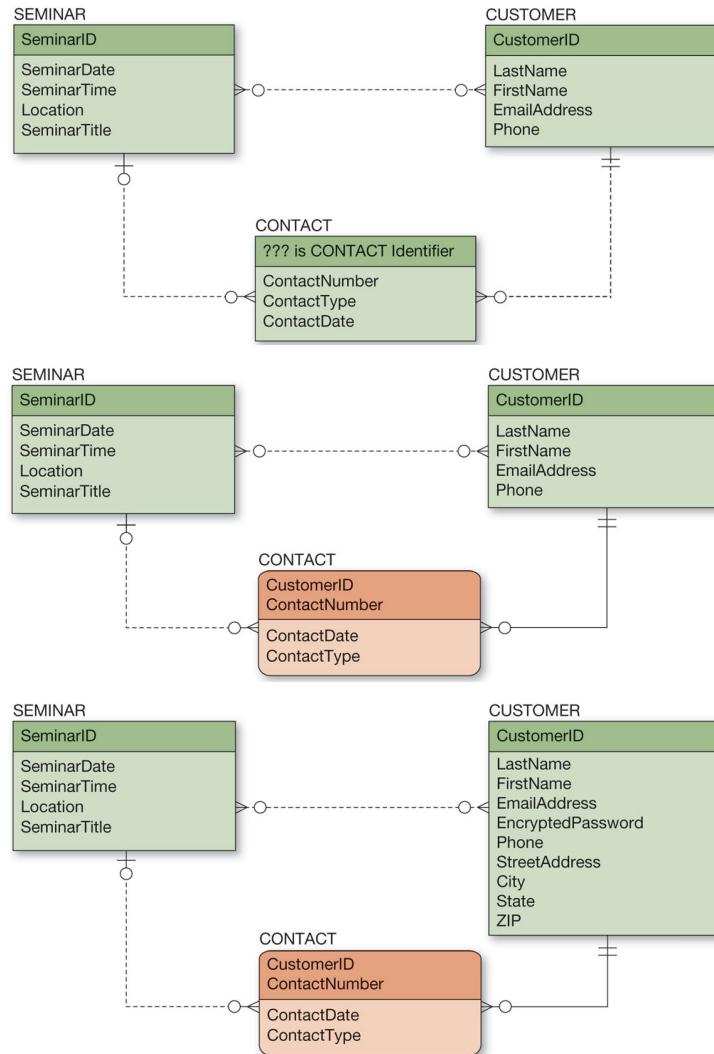
Also, as I mentioned at the seminar, I do provide customized design services to help you create that just-perfect kitchen. In fact, I have a number of clients in the Fort Worth area. Just give me a call at my personal phone number of 555-122-4873 if you'd like to schedule an appointment.

Thanks again and I look forward to hearing from you!

Best regards,

Heather Sweeney

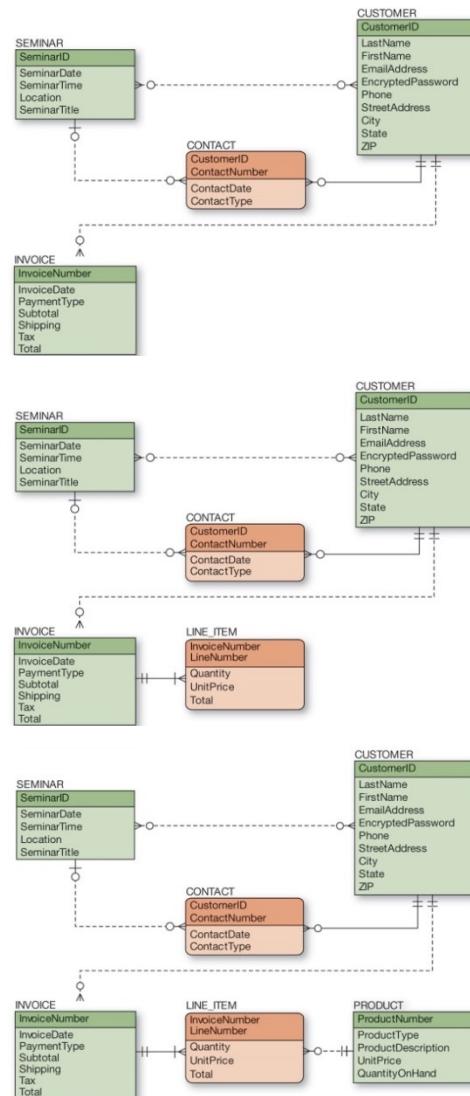
# Figure 4.25 Heather Sweeney Designs Data Model with CONTACT



# Figure 4.26 Heather Sweeney Designs Sales Invoice

Heather Sweeney Designs 122450 Rockaway Road Dallas, Texas 75227		Invoice No. 35000												
<b>INVOICE</b>														
<b>Customer</b>		<b>Misc</b>												
Name Address City Phone	Ralph Able 123 Elm Street San Antonio 210-281-7987	Date Order No. Rep FOB												
<table border="1"><thead><tr><th>Qty</th><th>Description</th><th>Unit Price</th><th>TOTAL</th></tr></thead><tbody><tr><td>1</td><td>Kitchen Remodeling Basics - Video</td><td>\$ 14.95</td><td>\$ 14.95</td></tr><tr><td>1</td><td>Kitchen Remodeling Basics - Video Companion</td><td>\$ 7.99</td><td>\$ 7.99</td></tr></tbody></table>			Qty	Description	Unit Price	TOTAL	1	Kitchen Remodeling Basics - Video	\$ 14.95	\$ 14.95	1	Kitchen Remodeling Basics - Video Companion	\$ 7.99	\$ 7.99
Qty	Description	Unit Price	TOTAL											
1	Kitchen Remodeling Basics - Video	\$ 14.95	\$ 14.95											
1	Kitchen Remodeling Basics - Video Companion	\$ 7.99	\$ 7.99											
<b>Payment</b>		<table border="1"><tr><td>Comments</td><td>Visa</td></tr><tr><td>Name</td><td>Ralph J. Able</td></tr><tr><td>CC #</td><td>xxxx xxxx xxxx xxxxxx</td></tr><tr><td>Expires</td><td>May-13</td></tr></table>	Comments	Visa	Name	Ralph J. Able	CC #	xxxx xxxx xxxx xxxxxx	Expires	May-13				
Comments	Visa													
Name	Ralph J. Able													
CC #	xxxx xxxx xxxx xxxxxx													
Expires	May-13													
		<table border="1"><tr><td>Tax Rate(s)</td><td>Subtotal</td><td>\$ 22.94</td></tr><tr><td></td><td>Shipping</td><td>\$ 5.95</td></tr><tr><td></td><td>5.70%</td><td>\$ 1.31</td></tr><tr><td></td><td><b>TOTAL</b></td><td>\$ 30.20</td></tr></table>	Tax Rate(s)	Subtotal	\$ 22.94		Shipping	\$ 5.95		5.70%	\$ 1.31		<b>TOTAL</b>	\$ 30.20
Tax Rate(s)	Subtotal	\$ 22.94												
	Shipping	\$ 5.95												
	5.70%	\$ 1.31												
	<b>TOTAL</b>	\$ 30.20												
<b>Office Use Only</b>														
<hr/> <hr/>														

# Figure 4.27 The Final Data Model for Heather Sweeney Designs

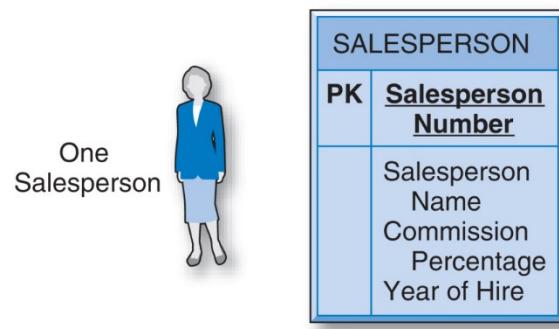


# Validating the Data Model

## Learn how to create an E-R diagram from source documents

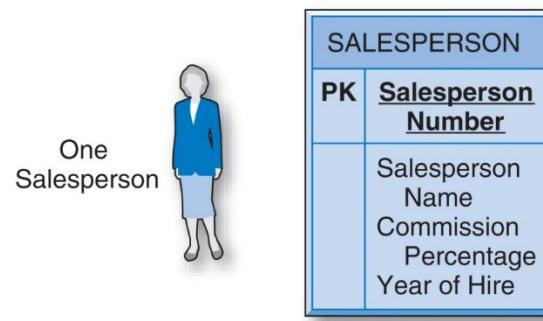
- After the data model has been completed, it needs to be validated:
  - The most common way is to show it to the users and obtain their feedback
  - **prototyping** is commonly used to validate forms and reports
  - A data model needs to be evaluated against all use cases

# E-R Model Entity (and its attributes)



- ◆ Rectangular shape
- ◆ Salesperson = a type of entity
- ◆ Name of entity is in caps above the separator line.

# E-R Model Entity (and its attributes)



- ◆ The attribute(s) which is the entity's **unique identifier** is shown with a **PK** ( Primary Key )
- ◆ All other Entity attributes are shown below the separator line.

# E-R Model (Relationship between Entities)

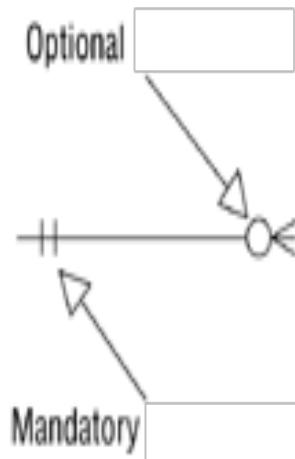
- Relationship – is the link between entities (relates to primary key-foreign key relationship in related tables)
- A relationship is always between entities, not attributes.

# E-R Model (Relationship between Entities)

- Cardinality - represents the maximum number of entities that can be involved in a particular relationship.
  - Many-to-Many Relationships
  - One-to-Many Relationships
  - One-to-One Relationships

# E-R Model (Relationship between Entities)

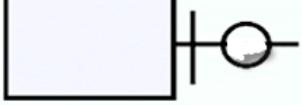
- Modality (Optionality)
  - The minimum number of entity occurrences that can be involved in a relationship.
  - “inner” symbol on E-R diagram (“outer” symbol is cardinality)



# Cardinality

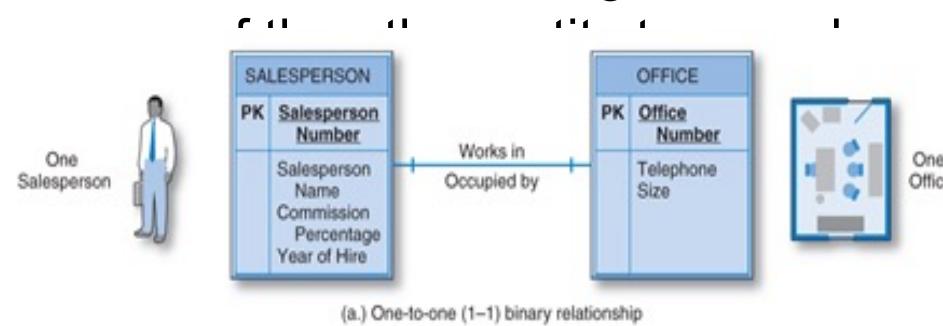
- ◆ Represents the maximum number of entities that can be involved in a particular relationship.
  - ♠ One-to-One Binary Relationship
  - ♠ One-to-Many Binary Relationship
  - ♠ Many-to-Many Binary Relationship

# Crow's Foot Notations (Drawing Symbols)

SYMBOL	MEANING
	One and only one
	One or many
	Zero, or one, or many
	Zero, or one

# One-to-One Binary Relationship

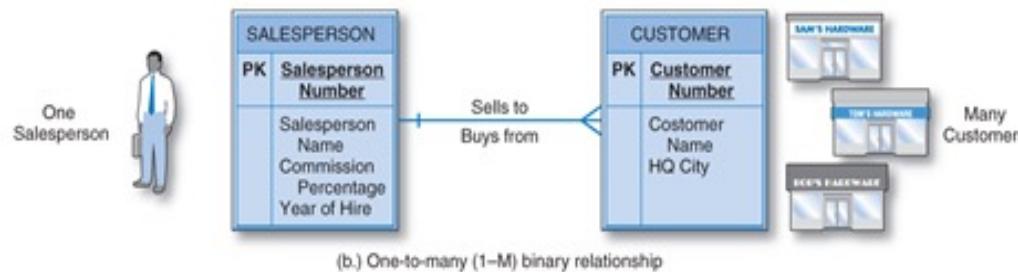
- ◆ 1-1
- ◆ A single occurrence of one entity type can be associated with a single occurrence of another entity type.



# One-to-Many Binary Relationship

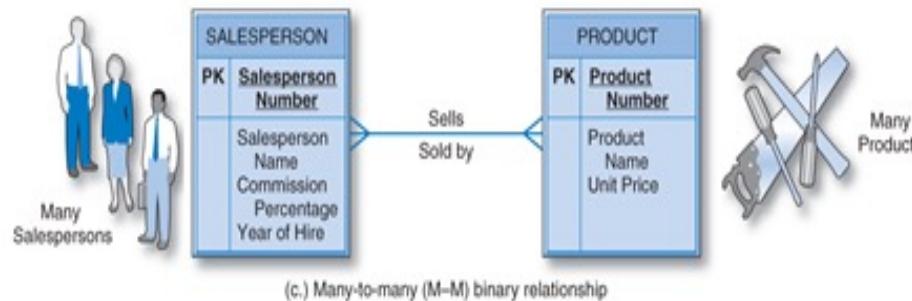
◆ 1-M

- ◆ Use “crow’s foot” to represent the multiple association.
- ◆ “many” = the maximum number of occurrences that can be involved, means a number that can be 1, 2, 3, ... n.



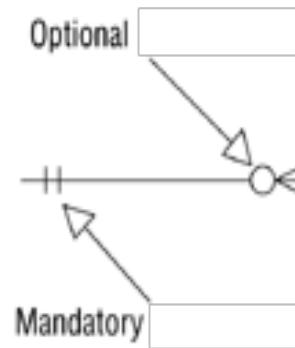
# Many-to-Many Binary Relationship

- ◆ M-M
- ◆ “many” can be either an exact number or have a known maximum

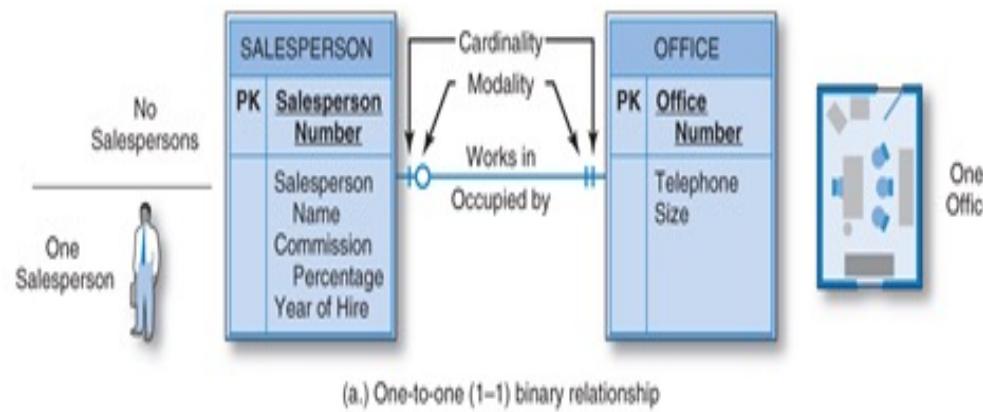


# Modality (Optionality)

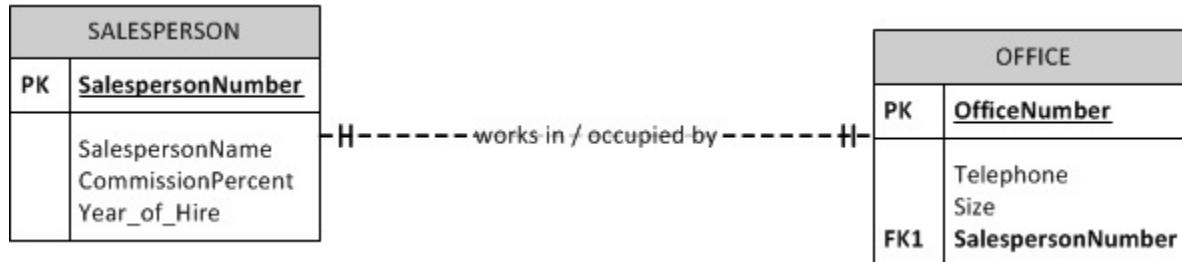
- ◆ The minimum number of entity occurrences that can be involved in a relationship.
- ◆ “inner” symbol on E-R diagram (“outer” symbol is cardinality)



# Cardinality & Modality



# Foreign Key



# Copyright



**This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.**



# Data Modeling

**Learning Outcome :**

**Utilize Data modeling techniques in the SDLC approach**

# LEARNING OBJECTIVES

You should be able to understand ...

- ❖ **Keys**
  - ❖ Primary Key
  - ❖ Unique Key
  - ❖ Artificial Key
  - ❖ Foreign Key
- ❖ **Mapping ERDs into Relational Schema**
  - ❖ Mapping E-R Diagrams to Relational Schema
  - ❖ Relational Schema Notation
  - ❖ Rules for Mapping E-R Diagrams

# Primary Key:

- Unique
  - e.g: FirstName & LastName cannot be always unique
- Always available
  - e.g: Visa residence numbers is not available for all
- Stable
  - e.g: Phone number is not stable
- Simple and easy to remember
  - Maximum two columns
- Never Null

# Unique Key

- A unique column that can not be a primary key because it can be null for some instances
  - Example: Passport numbers in one country are unique but some people do not have passports yet!

# Artificial Key

- It is a meaningless primary key introduced by the database designer to identify each row.
- It is usually an Integer
- Recommended to use it when:
  - There are many foreign keys
  - Natural key contains a null value
  - Natural key is not unique
  - Natural key is large and complex

# Foreign Key

It is an attribute that is referencing a primary key in another relation

EMP

EmpNo	Name	Address	Manager	DeptNo
100	Samir	Dubai		10
200	Younes	Sharjah	100	10
300	Ahmad	Al Ain	100	10
400	Noor	Dubai	100	10
500	John	Dubai	200	20
600	Khamis	RAK	200	20

DEPT

DeptNo	Dname
10	Bus
20	Bit

DeptNo in the Emp table is a foreign key referencing DeptNo in the Dept table.

The Manager column in the Emp table is a foreign key referencing EmpNo in the Emp table



# Hands-on Exercise I



- ❖ Think of Primary, Unique, Artificial, or Foreign keys that can exist inside each of the following Tables in a Database:
  - ❖ Employees Table that stores info about employees in a company
  - ❖ Cars Table that stores info about company cars
  - ❖ Sales Table that stores the relationship between Customers and Purchased Items
  - ❖ Addresses Table that stores info about student addresses

# Hands-on Exercise I





# Mapping ERDs into Relational Schema

# Mapping E-R Diagrams to Relational Schema

- Logical database design is about mapping the E-R Model to relational tables to produce an initial design.
- First, entities are mapped to simple tables.
- Second, attributes are mapped to columns.
- Third, unique identifiers are mapped to primary keys.
- Finally, relationships are mapped to foreign keys.
- The final product should be a relational “schema”, which will act as a template for the actual tables that will make up the database.

CUSTOMER (Cust-ID, Cust-Name, Address, City, Phone)

ORDER (Order-No, Date, Cust-ID\*)

ORDER\_LINE (Order-No\*, Prod-ID\*, Quantity)

PRODUCT (Prod-ID, Description, Unit-Price)

CUSTOMER

Cust ID	Cust Name	Address	City	Phone
2100	Al Futtaim Trading Ltd.	PO Box 50788	Dubai	04 243563
2785	Abdul Rahim Al Zarouni Stores	PO Box 1705 Oud Al Taoba St.	Al Ain	03 352887
3211	Al Falak Stores	Daira City Center 2nd Floor	Dubai	04 643447
4642	Al Buteen Furniture Est.	PO Box 1593 Oud Al Taoba r/a	Al Ain	03 665980
8972	Silent Night UAE LLC	PO Box 2604 Al Ittihad St.	Ajman	06 553112

ORDER

Order No	ODate	Cust ID
001	01/05/97	2100
002	09/05/97	2785
003	04/07/97	2100
004	01/08/97	2785
005	02/08/97	4642

ORDER\_LINE

Order No	Prod ID	Quantity
001	B128	5
001	C381	3
002	B128	6
003	C381	1
003	D036	2
003	T210	2
004	D036	15
005	B128	1
005	D036	5

PRODUCT

Prod ID	Description	Unit Price
B128	Bookcase	500
C381	Cabinet	375
D036	Desk	750
K193	Kitchen Trolley	395
K211	King Size Bed	1795
T210	Table	1250

# Relational Schema Notation

The following conventions are commonly used when representing a relational schema:

- Table name in capital letters
- Column names separated by commas and enclosed in brackets
- Primary key underlined
- Foreign key(s) denoted by an asterisk(\*)

CUSTOMER (*Cust-ID*, *Cust-Name*, *Address*, *City*, *Phone*)

ORDER (*Order-No*, *Date*, *Cust-ID*\*)

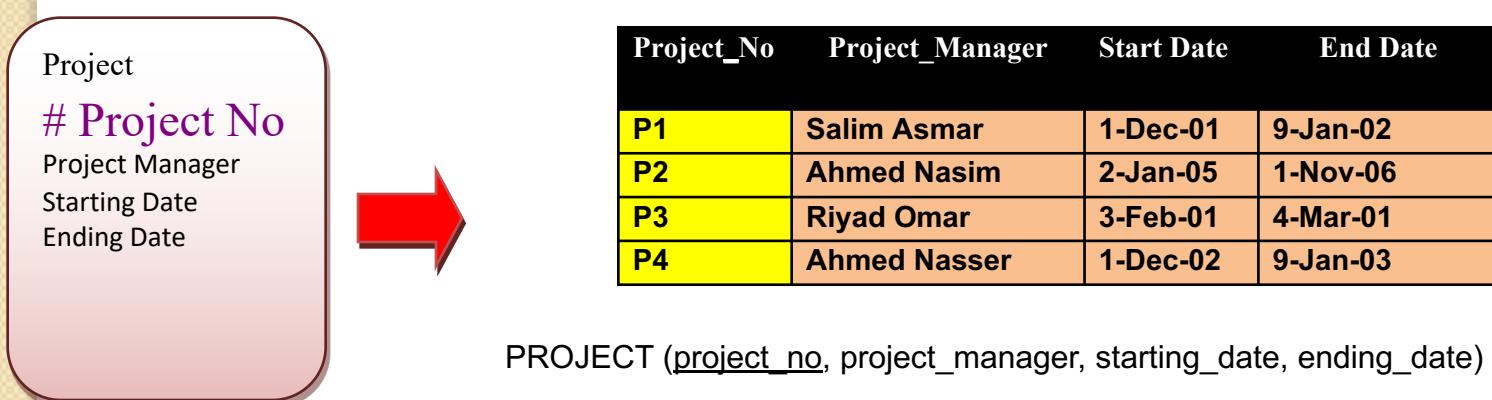
ORDER\_LINE (*Order-No*\*, *Prod-ID*\*, *Quantity*)

PRODUCT (*Prod-ID*, *Description*, *Unit-Price*)

# Rules for Mapping E-R Diagrams

## 1. Mapping Regular Entities

- Regular entities can be directly converted into tables.
- Each attribute of the entity becomes a column in the corresponding table.
- The primary key of the entity becomes the primary key of the table.



# Hands-on Exercise 2

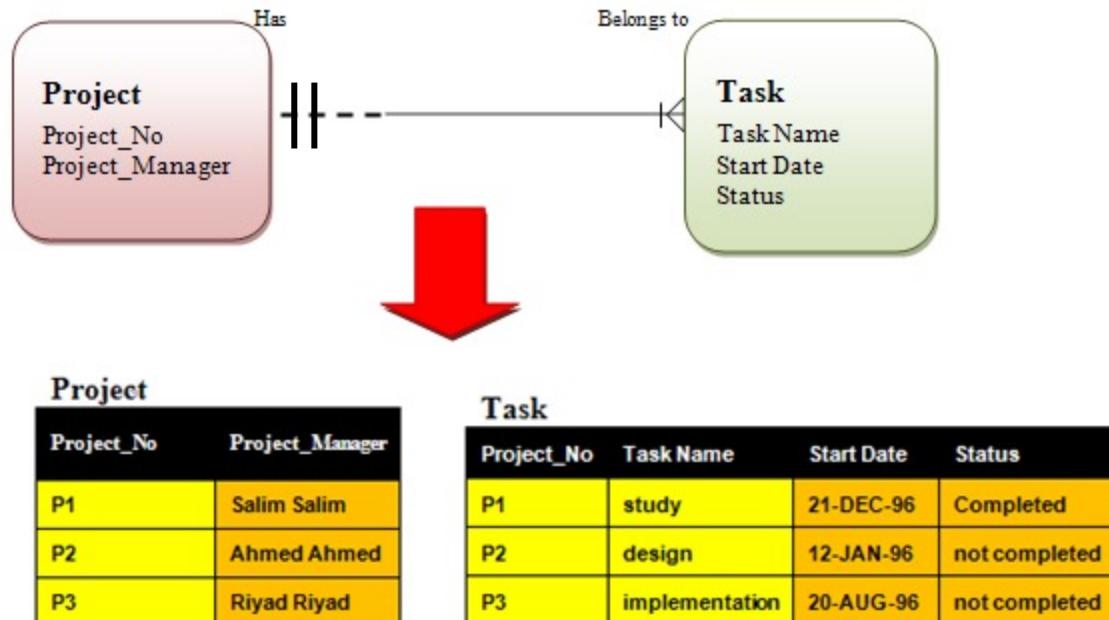


- ❖ Map the following ERD into a Relational Schema, showing the table names, the table attributes, and the primary key:

Student
Student_No
Student_Name
Student_DOB
Student_Address

## 2. Mapping Dependent (Weak) Entities

- Dependent entities become dependent tables.
- The primary key of the new table is a combination of the primary key of the parent table and the primary key of the new table.
- The foreign key refers to the parent table.
- The foreign key must be not null.



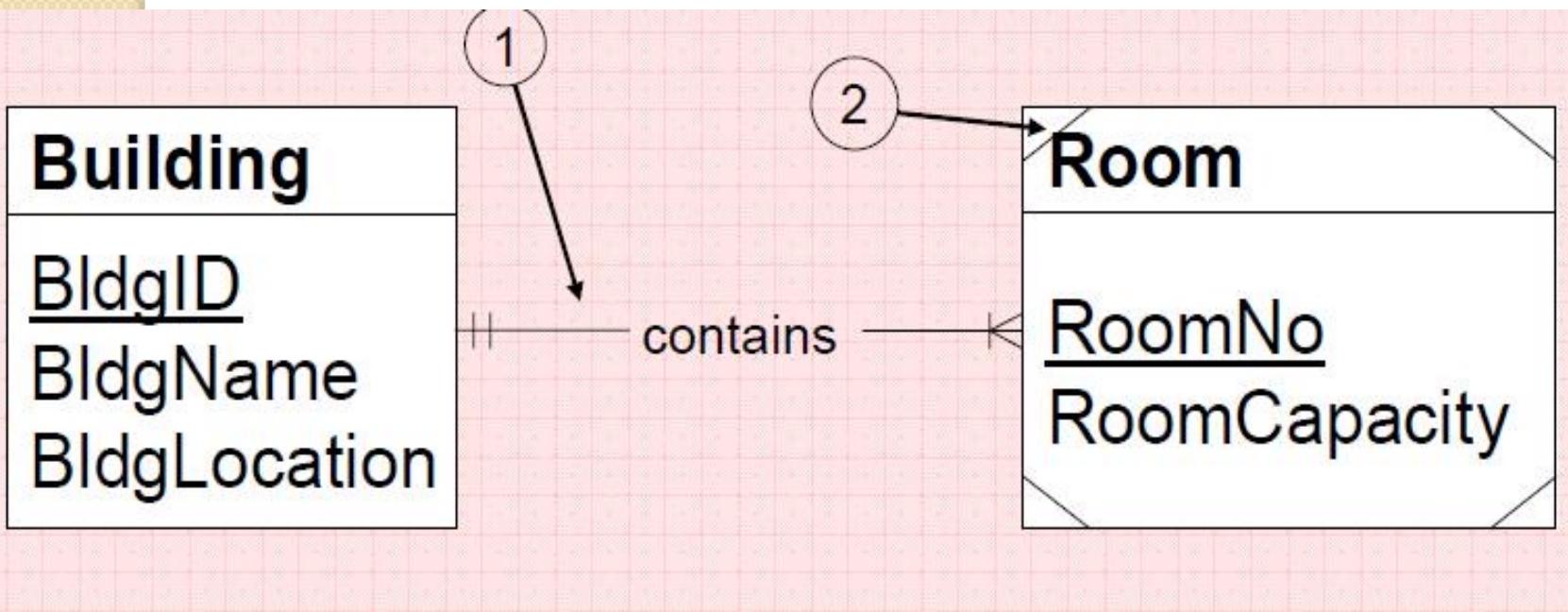
PROJECT(Project\_no, Project\_manager)

TASK (Project\_no\*, Task\_name, Start\_Date, Status)

# Hands-on Exercise 3



- ❖ Map the following ERD into a Relational Schema, showing the table names, the table attributes, and the primary and foreign keys:



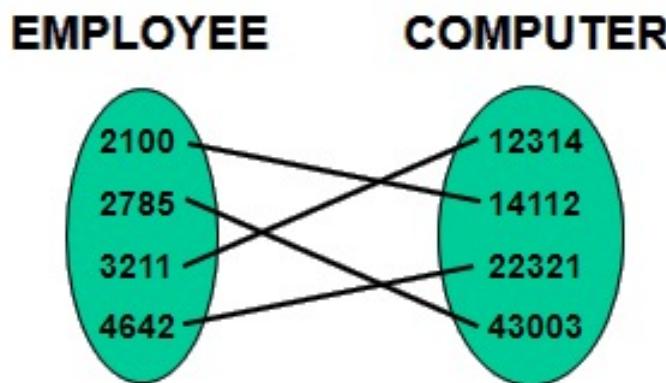
### 3. Mapping Binary Relationships

- After converting all of the entities into tables, the database designer must now focus on the various relationships between entities.
- Since binary relationships (ie, relationships between 2 entities) are the most common types of relationship encountered in data modeling, these are usually converted first.
- The procedure used to convert relationships depends on the maximum and minimum cardinality of the relationship.
- For binary relationships, there are six cases to consider:

Maximum Cardinality	Minimum Cardinality
a) 1:1	Both sides mandatory
b) 1:1	Either side optional
c) 1:1	Both sides optional
d) 1:M	M side mandatory
e) 1:M	M side optional
f) M:N	n/a

### 3a) 1:1 Both sides mandatory

- Merge the two entities into a single table.
- Choose one of the original primary keys as the primary key for the table.



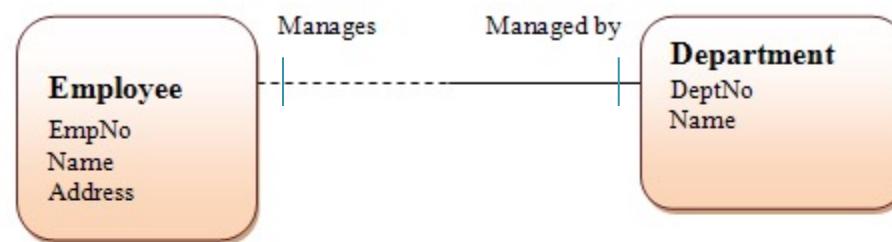
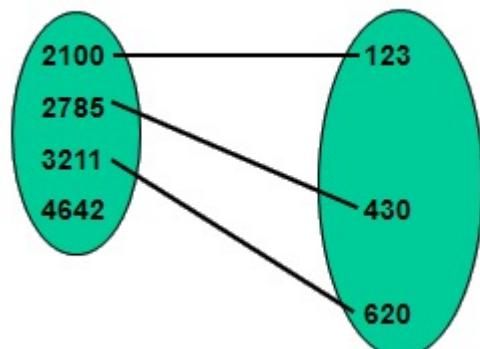
Emp ID	Emp Name	Phone	Serial No	Processor	RAM
2100	Eiman Al Ketheri	243 563	14112	P200	32Mb
2785	Salam a Al Muhairi	352 887	43003	P200	32Mb
3211	Tayeb Kam ali	665 980	12314	P100	16Mb
4642	Majed Al Qasseemi	553 112	22321	P120	16Mb

EMPLOYEE (Emp-ID, Emp-Name, Phone, Serial-No, Processor, RAM)

### 3b) 1:1 One Side Optional, Other side Mandatory

- In One-to-One relationship with a mandatory participation in one direction, the primary key of the optional side table will be the foreign key in the table at the mandatory end.

**EMPLOYEE      Department**



**Employee**

**Dept**

EmpNo	Ename	Address
2100	Samir	Dubai
2785	Younes	Sharjah
3211	Ahmad	Al Ain
4642	Noor	Dubai

**EMPLOYEE** (EmpNo, Ename, Address)

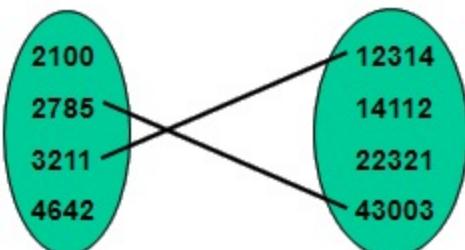
DeptNo	Dname	EmpNo
123	Bus	2100
430	Bit	2785
620	Eng	3211

**DEPT** (DeptNo, Dname, Empno\*)

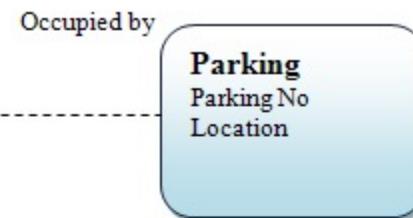
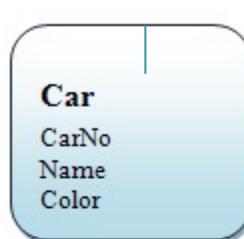
### 3c) 1:1 Both sides optional

- Place the primary keys from the two entities into a new table.
- Choose one of the columns as the primary key for the new table

**Car**



**Parking**



**Car**

CarNo	name	Color
2100	Honda	Red
2785	Mazda	White
3211	Nissan	Green
4642	Benz	Black

CAR (camo, name, color)

CarNo	ParkNo
2785	12314
3211	43003

PK ParkNO  
FK CarNo references Car  
FK ParkingNo references parking

**Parking**

ParkingNo	Location
12314	EW
14112	ES
22321	WE
43003	WN

PK ParkingNO

# Hands-on Exercise 4



- ❖ Draw the ERD for each of the following case requirements and then map the ERDs into a Relational Schema, showing the table names, the table attributes, and the primary and foreign keys:

## ❖ **CASE I:**

- ❖ Each Student must have only one laptop and each laptop must belong to only one student

# Hands-on Exercise 4



## ❖ CASE 2:

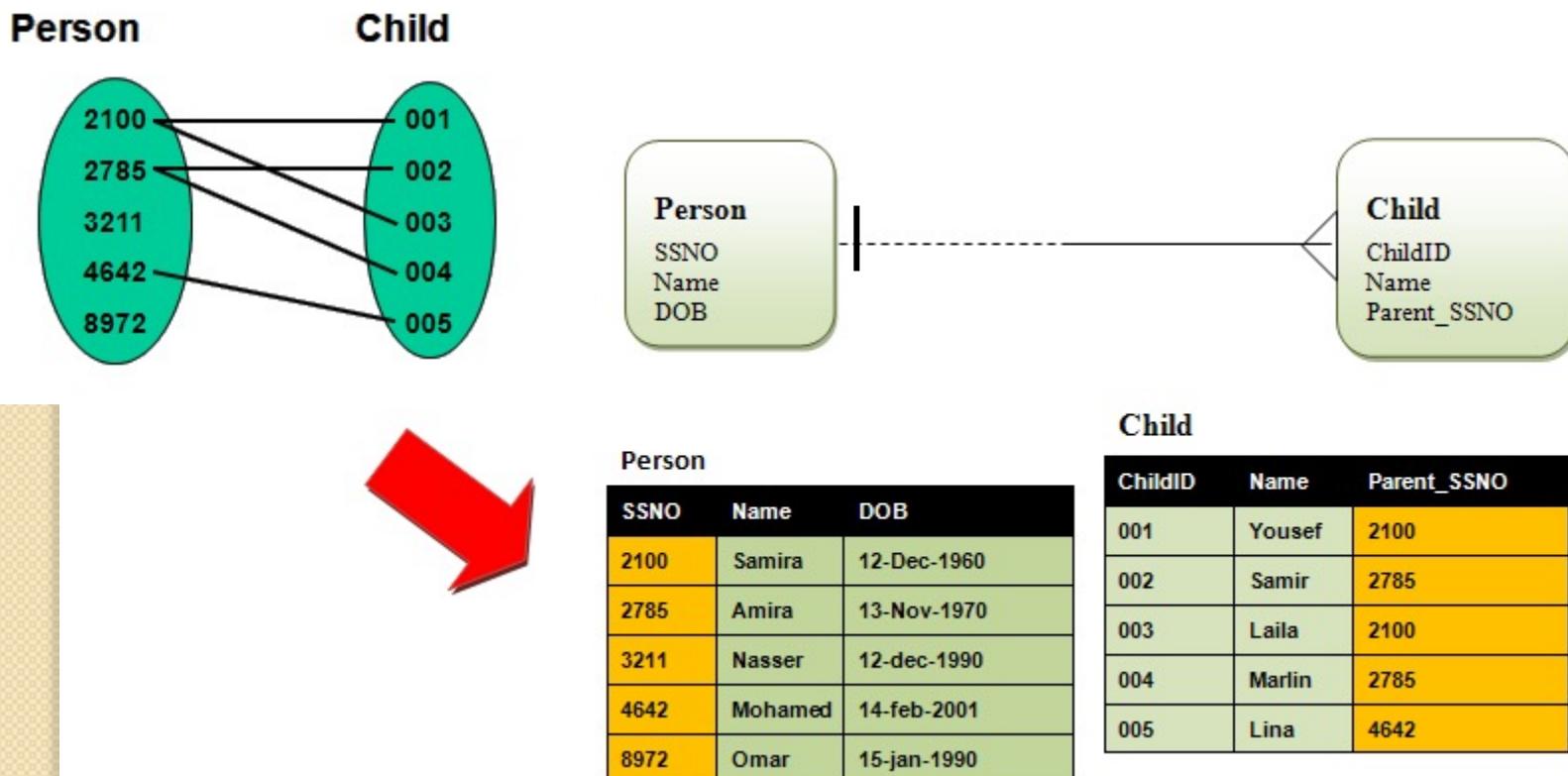
- ❖ Each Student may have only one laptop and each laptop must belong to only one student

## ❖ CASE 3:

- ❖ Each Student may have only one laptop and each laptop may belong to only one student

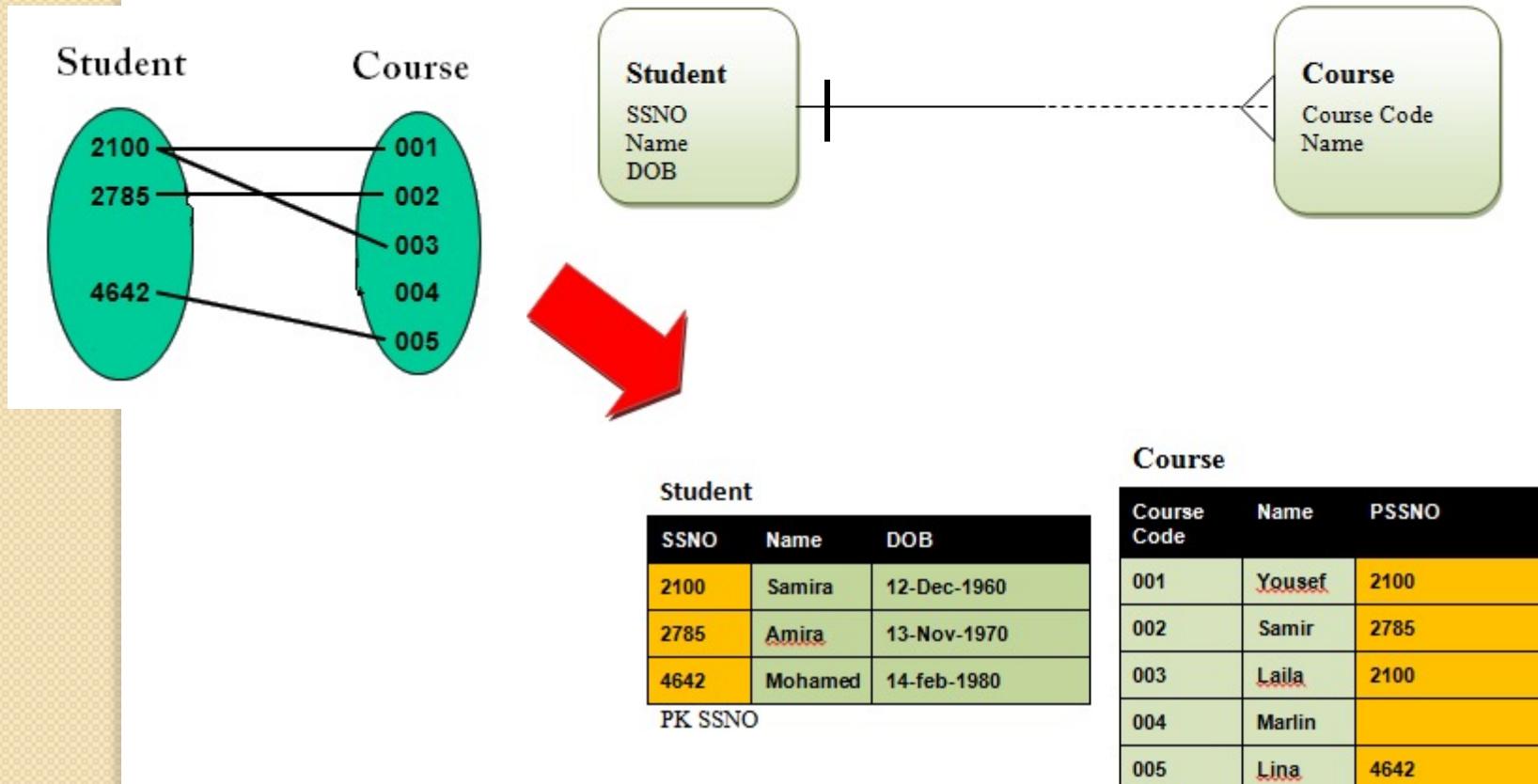
### 3d) 1:M M side mandatory

- Create a foreign key in the table representing the “child” entity (the M side) that references the primary key in the table representing the “parent” entity (the 1 side).
- The foreign key status is Not Null if the participation of the many side entity is mandatory



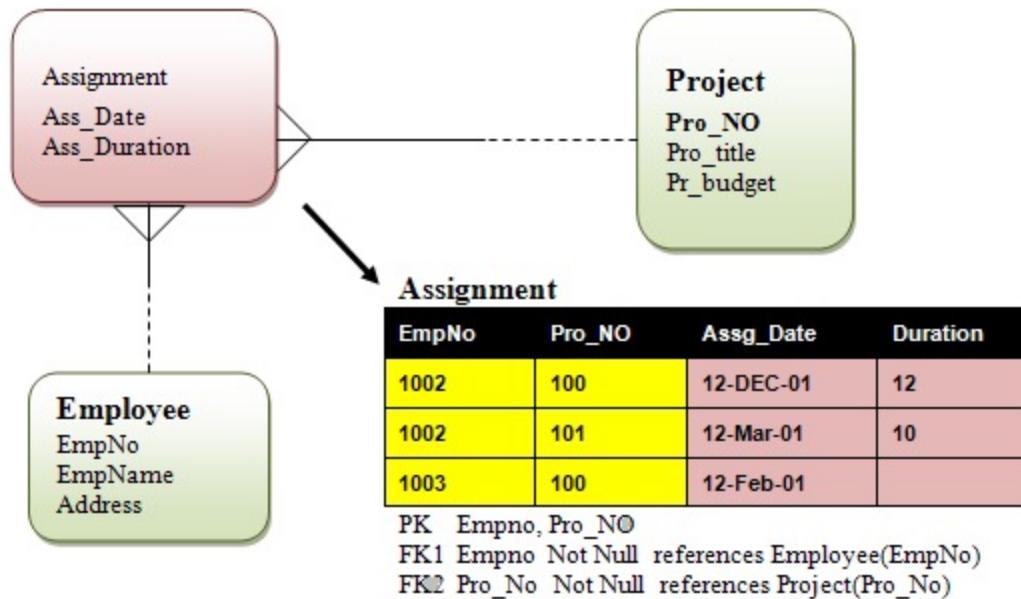
### 3e) 1:M M side optional

- Create a foreign key in the table representing the “child” entity (the M side) that references the primary key in the table representing the “parent” entity (the 1 side).
- The foreign key status is Null if the participation of the many side entity is optional



### 3f) M:N

- Place the primary keys from the two entities into a new table.
- Choose this two-key composite to be the primary key for the new table.
- Include any relationship attributes, if present.



**Employee**

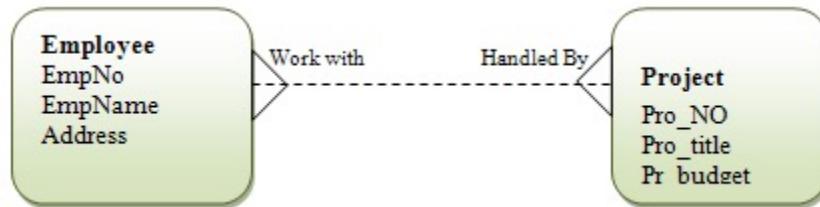
EmpNo	EmpName	Address
1002	Nasser	Dubai
1003	Adil	AXD

PK EmpNo

**Project**

Pro_NO	Pro_title	Pr_Budget
100	Building	12,000,000
101	Bridge	10,000,00

PK Pro No



# Hands-on Exercise 5



❖ Draw the ERD for each of the following case requirements and then map the ERDs into a Relational Schema, showing the table names, the table attributes, and the primary and foreign keys (State whether FK can have null values or not):

## ❖ **CASE I:**

❖ Each Student can have many laptops and each laptop must belong to one student

# Hands-on Exercise 5



## ❖ CASE 2:

- ❖ Each Student can have many laptops and each laptop may belong to one student

## ❖ CASE 3:

- ❖ Each Student can take many courses and each course may have many student

# Database Concepts

Ninth Edition



## Chapter 5

### Database Design

# Learning Objectives

- Learn how to transform E-R data models into relational designs
- Practice applying the normalization process
- Understand the need for denormalization
- Learn how to represent weak entities with the relational Model
- Know how to represent 1:1, 1:N, and N:M binary relationships
- Know how to represent 1:1, 1:N, and N:M recursive relationships
- Learn SQL statements for creating joins over binary and recursive relationships

# The Purpose of a Database Design

## Learn how to transform E-R data models into relational designs

- A **database design** is a set of database specifications that can actually be implemented as a database in a DBMS.
- The three designs are:
  - conceptual design (conceptual schema)
  - logical design (logical schema)
  - physical design (physical schema)
- The design studied in this chapter is equivalent to the logical design.

# Figure 5.1 The Steps for Transforming a Data Model into a Database Design

1. Create a table for each entity:
  - Specify primary key (consider surrogate keys as appropriate)
  - Specify properties for each column:
    - Data type
    - Null status
    - Default value (if any)
    - Specify data constraints (if any)
  - Verify normalization
2. Create relationships by placing foreign keys:
  - Strong entity relationships (1:1, 1:N, N:M)
  - ID-dependent and non-ID-dependent weak entity relationships
  - Subtypes
  - Recursive (1:1, 1:N, N:M)

# Figure 5.2 The ITEM Entity and Table

ITEM

ItemNumber
Description
Cost
ListPrice
QuantityOnHand

ITEM

 ItemNumber
Description
Cost
ListPrice
QuantityOnHand

# Figure 5.3 The Final ITEM Table

ITEM



ItemNumber: int IDENTITY(10000,1)

Description: varchar(100) NOT NULL

Cost: numeric(9,2) NOT NULL

ListPrice: numeric(9,2) NULL

QuantityOnHand: int NOT NULL

# Figure 5.4 The CUSTOMER Entity and Table

CUSTOMER

CustomerNumber
CustomerName
StreetAddress
City
State
ZIP
ContactName
Phone

CUSTOMER

 CustomerNumber
CustomerName
StreetAddress
City
State
ZIP
ContactName
Phone

CUSTOMER (CustomerNumber, CustomerName, StreetAddress, City, State, ZIP, ContactName, Phone)

# Figure 5.5 The Normalized CUSTOMER and Associated Tables



ZIP  
is a foreign key referencing  
ZIP in ZIP

ContactName  
is a foreign key referencing  
ContactName in CONTACT

CUSTOMER (CustomerNumber, CustomerName, StreetAddress, ZIP, ContactName)  
ZIP (ZIP, City, State)  
CONTACT (ContactName, Phone)

# Denormalization

## Understand the need for denormalization

- Normalizing relations (or breaking them apart into many component relations) may significantly increase the complexity of the data structure.
- The question is one of balance:
  - trading complexity for modification problems
- There are situations where denormalized relations are preferred.

# Figure 5.6 The Denormalized CUSTOMER and Associated CONTACT Tables

CUSTOMER

 CustomerNumber
CustomerName
StreetAddress
City
State
ZIP
ContactName

CONTACT

 ContactName
Phone

ContactName  
is a foreign key referencing  
ContactName in CONTACT

CUSTOMER (CustomerNumber, CustomerName, StreetAddress, City, State, ZIP,  
*ContactName*)  
CONTACT (ContactName, Phone)

# Represent Weak Entities

**Learn how to represent weak entities with the relational model**

- If not ID-dependent, use the same techniques as for strong entities
- If ID-dependent, then must add primary key of the parent entity

# Figure 5.7 The SALES\_COMMISSION Entity and Table

SALES\_COMMISSION

CheckNumber
SalespersonNumber
SalespersonLastName
SalespersonFirstName
Phone
CheckDate
CommissionPeriod
TotalCommissionSales
CommissionAmount
BudgetCategory

SALES\_COMMISSION

 CheckNumber
SalespersonNumber
SalespersonLastName
SalespersonFirstName
Phone
CheckDate
CommissionPeriod
TotalCommissionSales
CommissionAmount
BudgetCategory

# A Relational Design for the SALES\_COMMISSION Entity (1 of 2)

## Understand the need for denormalization

- Consider the original attributes of the SALES\_COMMISSION entity on the previous slide

SALES\_COMMISSION (SalespersonNumber,  
SalespersonLastName, SalespersonFirstName, Phone, CheckNumber,  
CheckDate, CommissionPeriod, TotalCommissionSales, CommissionAmount,  
BudgetCategory)

- Three additional functional dependencies are:

SalespersonNumber → (SalespersonLastName, SalespersonFirstName, Phone,  
BudgetCategory)

CheckNumber → CheckDate

(SalespersonNumber, CommissionPeriod) → (TotalCommissionSales,  
CommissionAmount, CheckNumber, CheckDate)

# A Relational Design for the SALES\_COMMISSION Entity (2 of 2)

## Understand the need for denormalization

- One look at the previous functional dependencies reveals that the original table, given the primary key CheckNumber, should actually be called COMMISSION\_CHECK.
- Once normalized properly you have...

SALESPERSON (SalespersonNumber, SalespersonLastName, SalespersonFirstName, Phone, BudgetCategory)

SALES\_COMMISSION (SalespersonNumber, CommissionPeriod, TotalCommissionSales, CommissionAmount, CheckNumber)

COMMISSION\_CHECK (CheckNumber, CheckDate)

- This is now shown in Figure 5.8 on the next slide.

# Figure 5.8 The Normalized SALES\_COMMISSION And Associated Tables

SALESPERSON

 SalespersonNumber
SalespersonLastName
SalespersonFirstName
Phone
BudgetCategory

SALES\_COMMISSION

 SalespersonNumber
 CommissionPeriod
TotalCommissionSales
CommissionAmount
CheckNumber

COMMISSION\_CHECK

 CheckNumber
CheckDate

SalespersonNumber is a foreign key referencing SalespersonNumber in SALESPERSON

SALES\_COMMISSION is ID-dependent on SALESPERSON

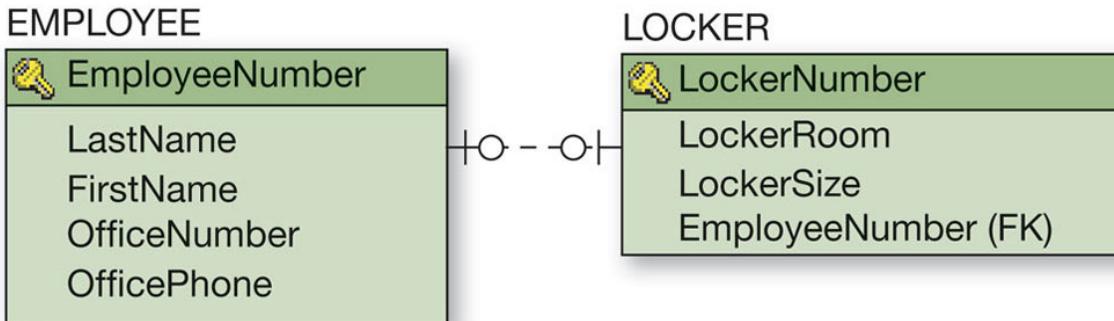
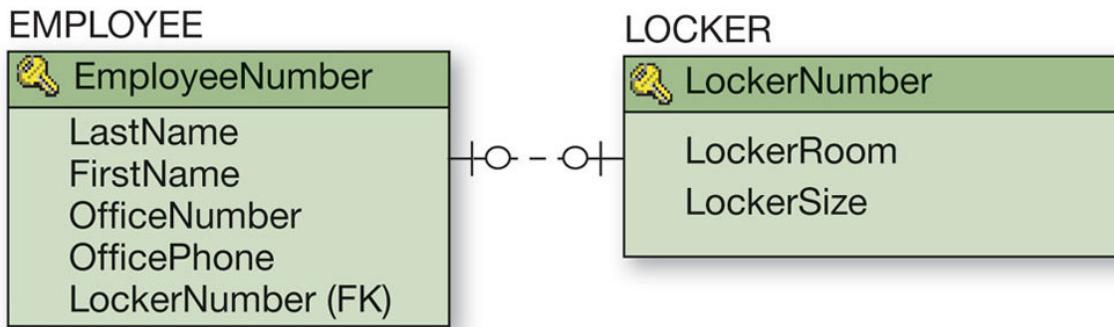
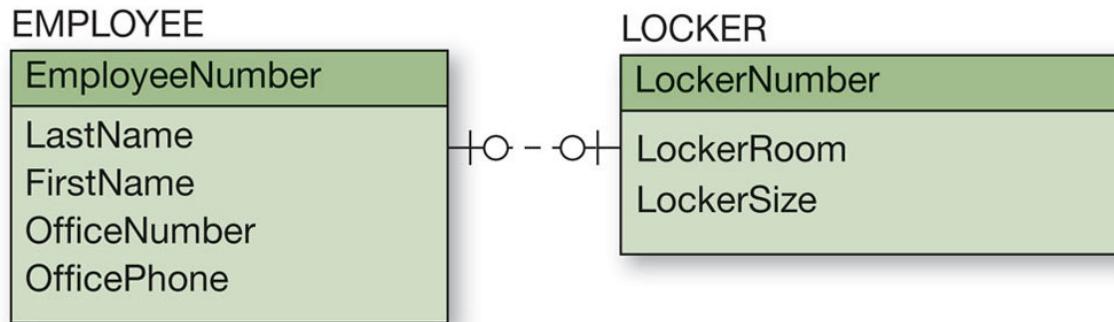
CheckNumber is a foreign key referencing CheckNumber in COMMISSION\_CHECK

# Representing 1:1 Relationships

## Know how to represent 1:1, 1:N, and N:M binary relationships

- The maximum cardinality determines how a relationship is represented.
- 1:1 Relationship:
  - the key from one relation is placed in the other as a *foreign key*
  - it does not matter which table receives the foreign key

# Figure 5.10 1:1 Strong Entity Relationships



# Results of a 1:1 Relationship

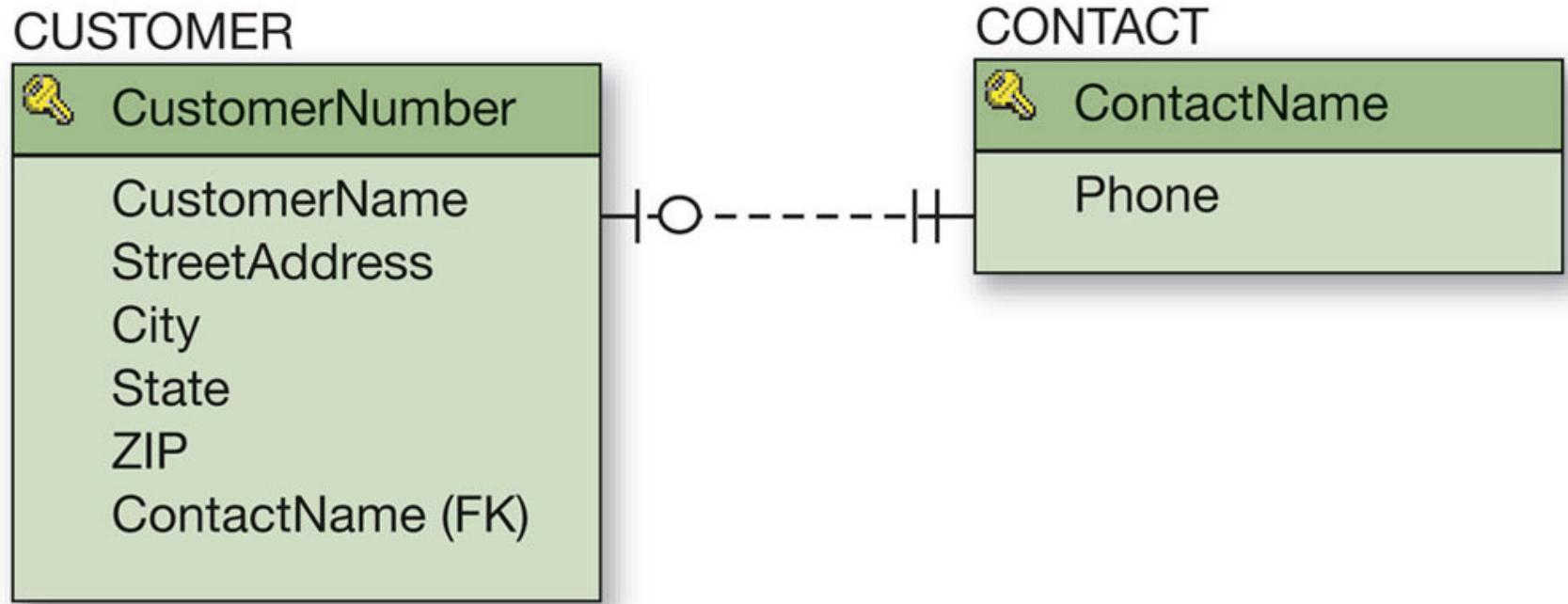
## Know how to represent 1:1, 1:N, and N:M binary relationships

- Because of a 1:1 relationship both of the following examples would not produce any unassigned employees.

```
SELECT *
FROM   EMPLOYEE JOIN LOCKER
      ON EMPLOYEE.LockerNumber=LOCKER.LockerNumber;
```

```
SELECT *
FROM   EMPLOYEE JOIN LOCKER
      ON EMPLOYEE.EmployeeNumber=LOCKER.EmployeeNumber;
```

# Figure 5.11 1:1 Strong Entity Relationship Between CUSTOMER and CONTACT

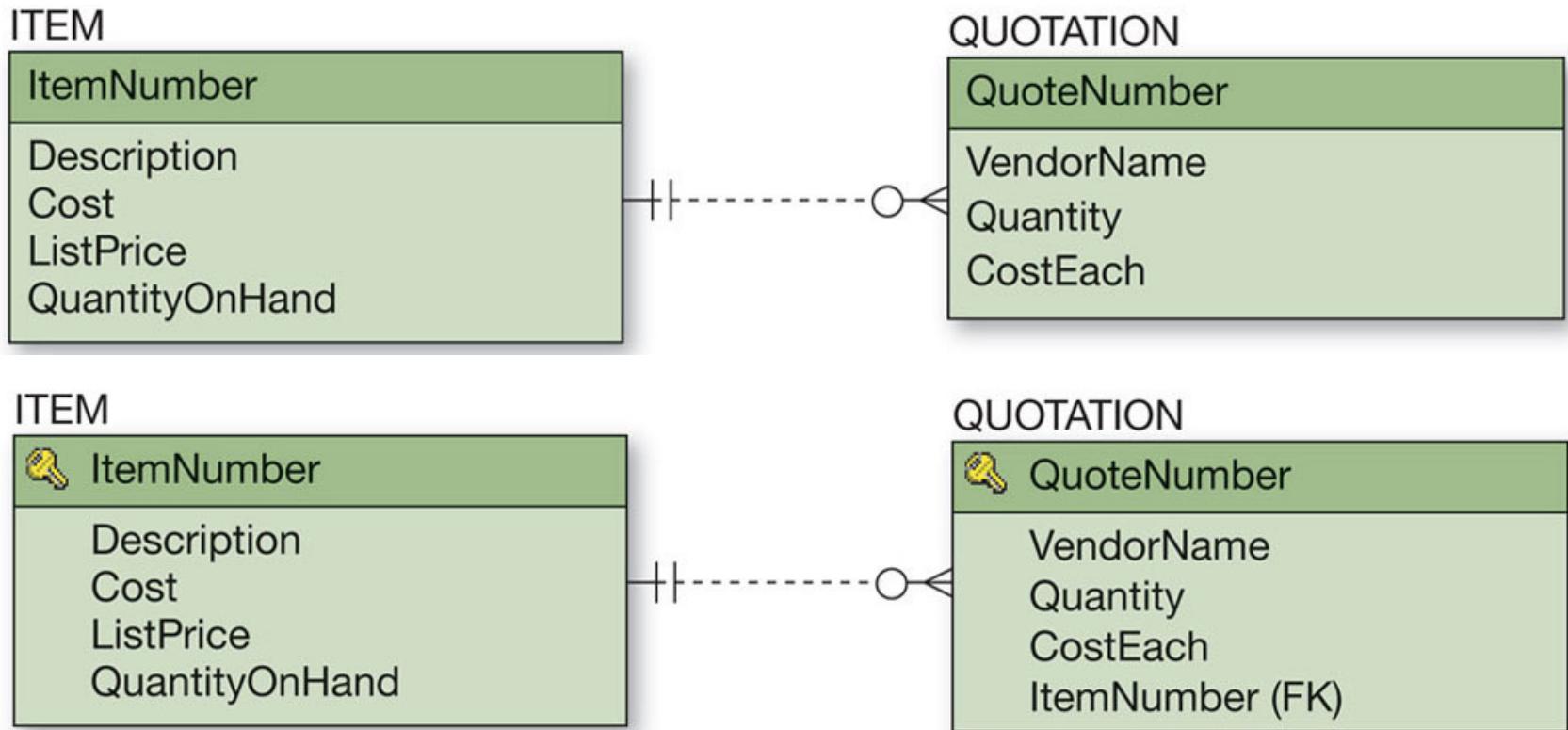


# Representing 1:N Strong Entity Relationships

## Know how to represent 1:1, 1:N, and N:M binary relationships

- Like a 1:1 relationship, a 1:N relationship is saved by placing the key from one table into another as a foreign key.
- However, in a 1:N the foreign key always goes into the many-side of the relationship:
  - the 1 side is called the **parent**
  - the N side is called the **child**

# Figure 5.12 1:N Strong Entity Relationships



# Results of a 1:N Relationship

## Know how to represent 1:1, 1:N, and N:M binary relationships

- In a 1:N relationship, the key of the parent must be placed in the child relation

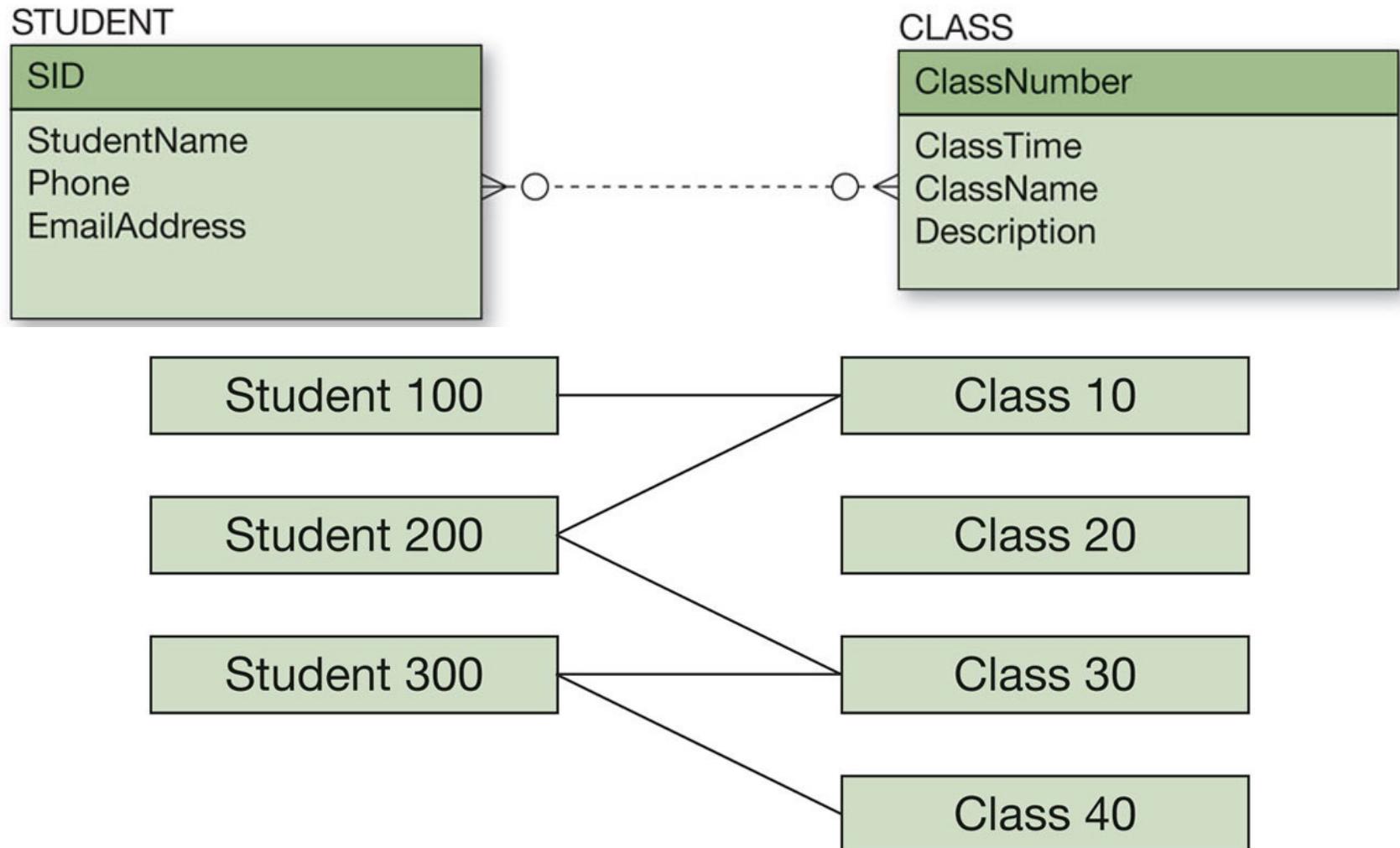
```
SELECT  *
FROM    ITEM JOIN QUOTATION
ON      ITEM.ItemNumber = QUOTATION.ItemNumber;
```

# Representing N:M Strong Entity Relationships

**Know how to represent 1:1, 1:N, and N:M binary relationships**

- To create an N:M relationship, a new table is created. This table is called an **intersection table**.
- An intersection table has a composite key consisting of the keys from each of the tables that it connects.

# Figure 5.13 N:M Strong Entity Relationships



# Figure 5.14 Incorrect Representation of an N:M Relationship

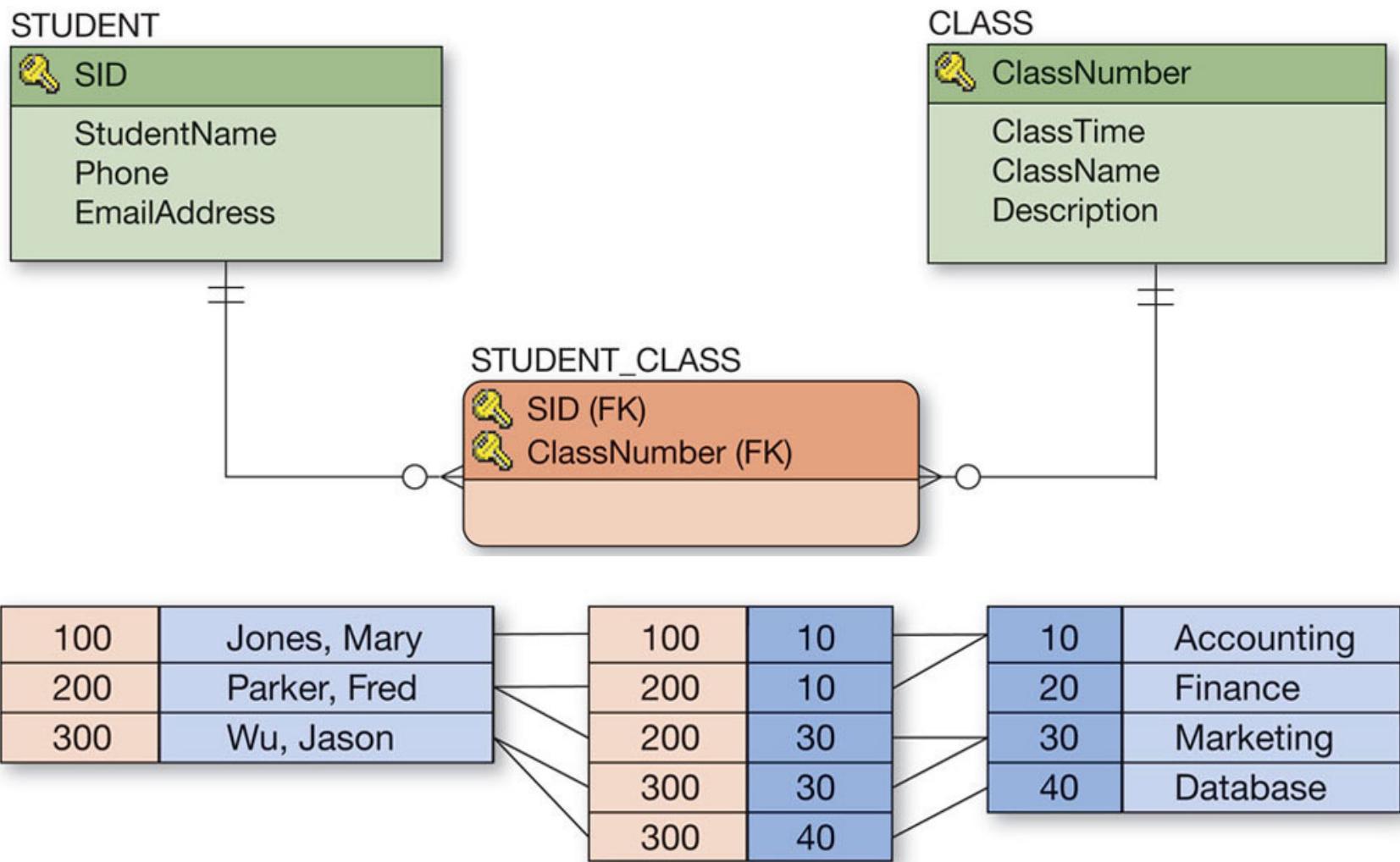
SID	Other STUDENT Data
100	...
200	...
300	...

STUDENT

ClassNumber	ClassTime	Other CLASS Data	SID
10	10:00 MWF	...	100
10	10:00 MWF	...	200
30	3:00 TH	...	200
30	3:00 TH	...	300
40	8:00 MWF	...	300

CLASS

# Figure 5.15 Representing a N:M Strong Entity Relationship



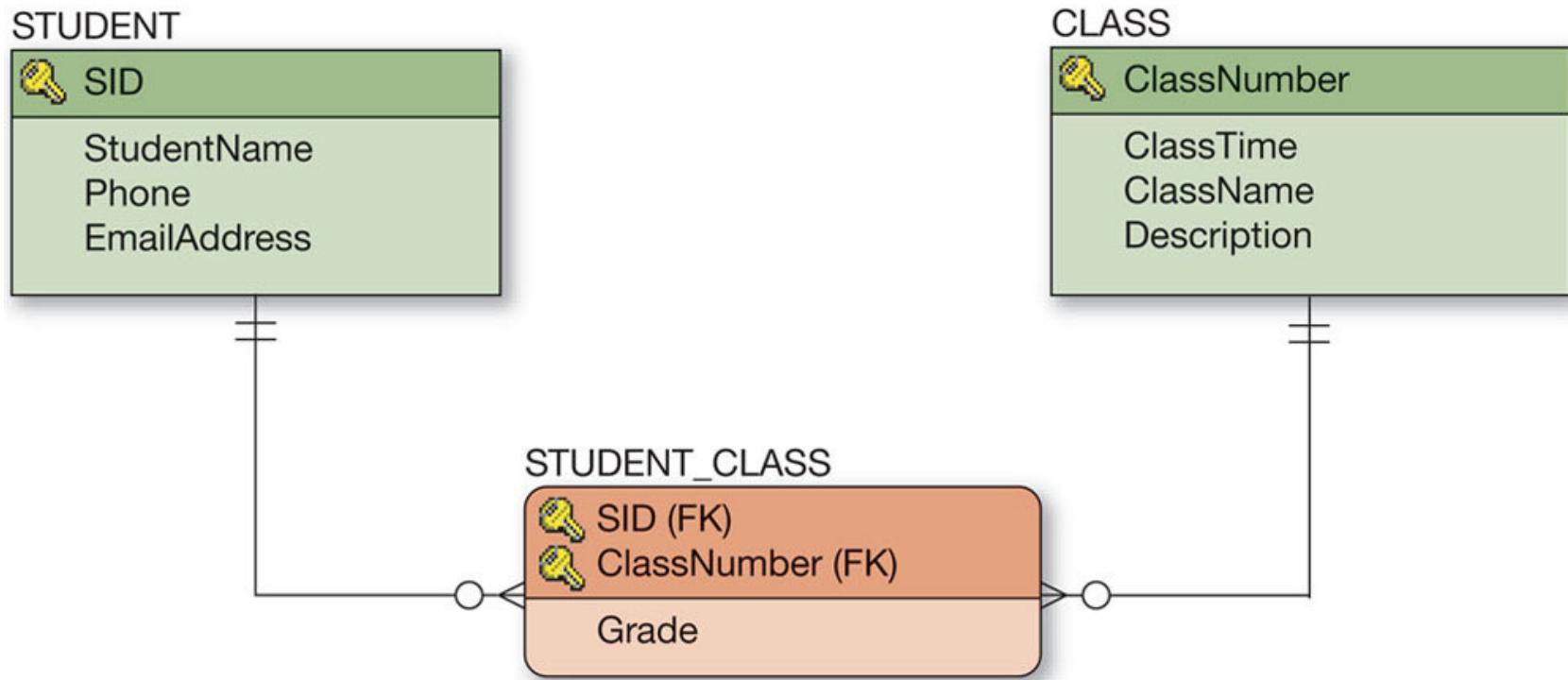
# Results of a N:M Relationships

## Learn SQL Statements for creating joins over binary and recursive relationships

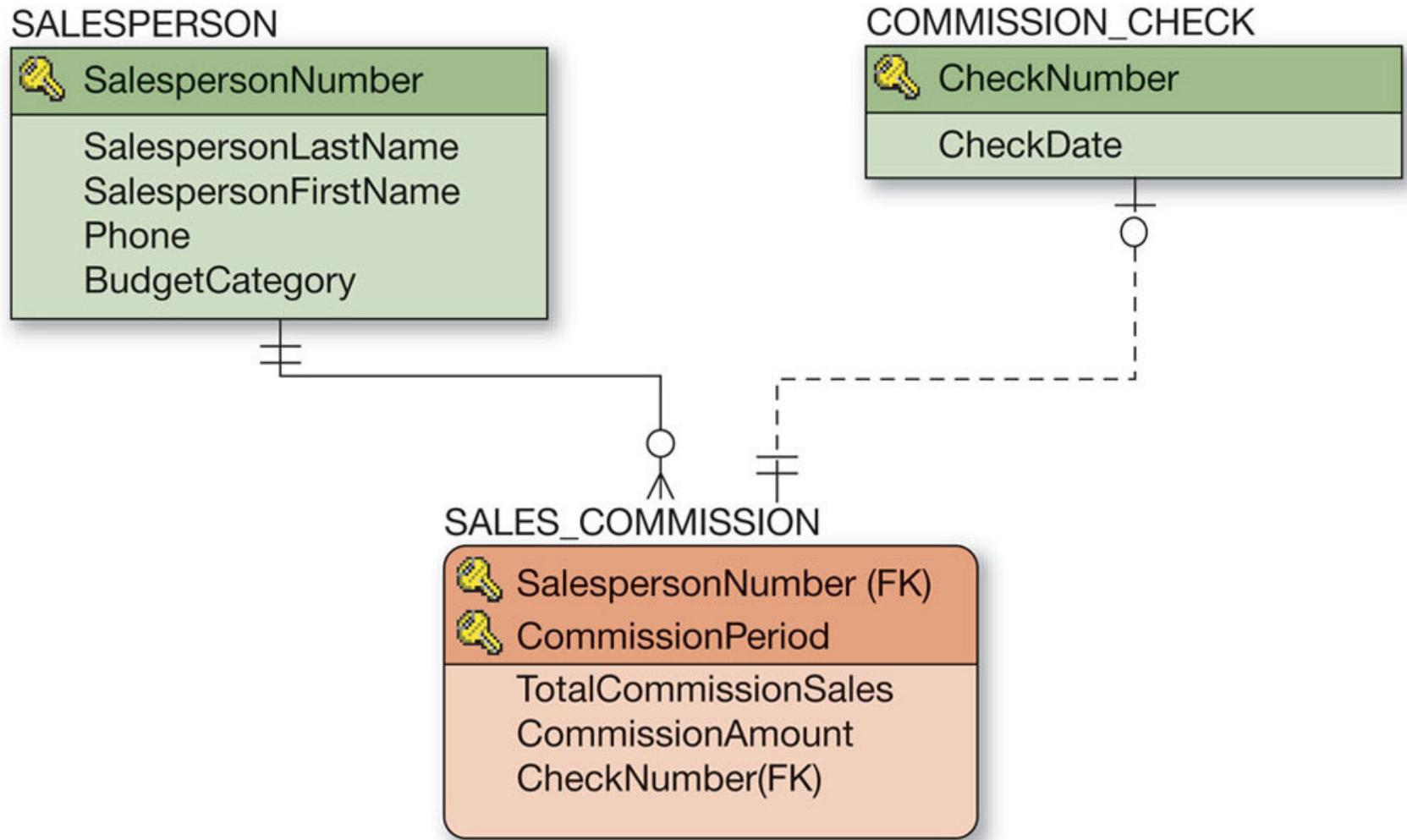
- In a N:M relationship, an intersection table must be created that will contain the primary keys of each of the other two tables as a composite primary key.

```
SELECT      *
FROM        STUDENT JOIN STUDENT_CLASS
          ON STUDENT.SID = STUDENT_CLASS.SID
          JOIN CLASS
          ON STUDENT_CLASS.ClassNumber= CLASS.ClassNumber;
```

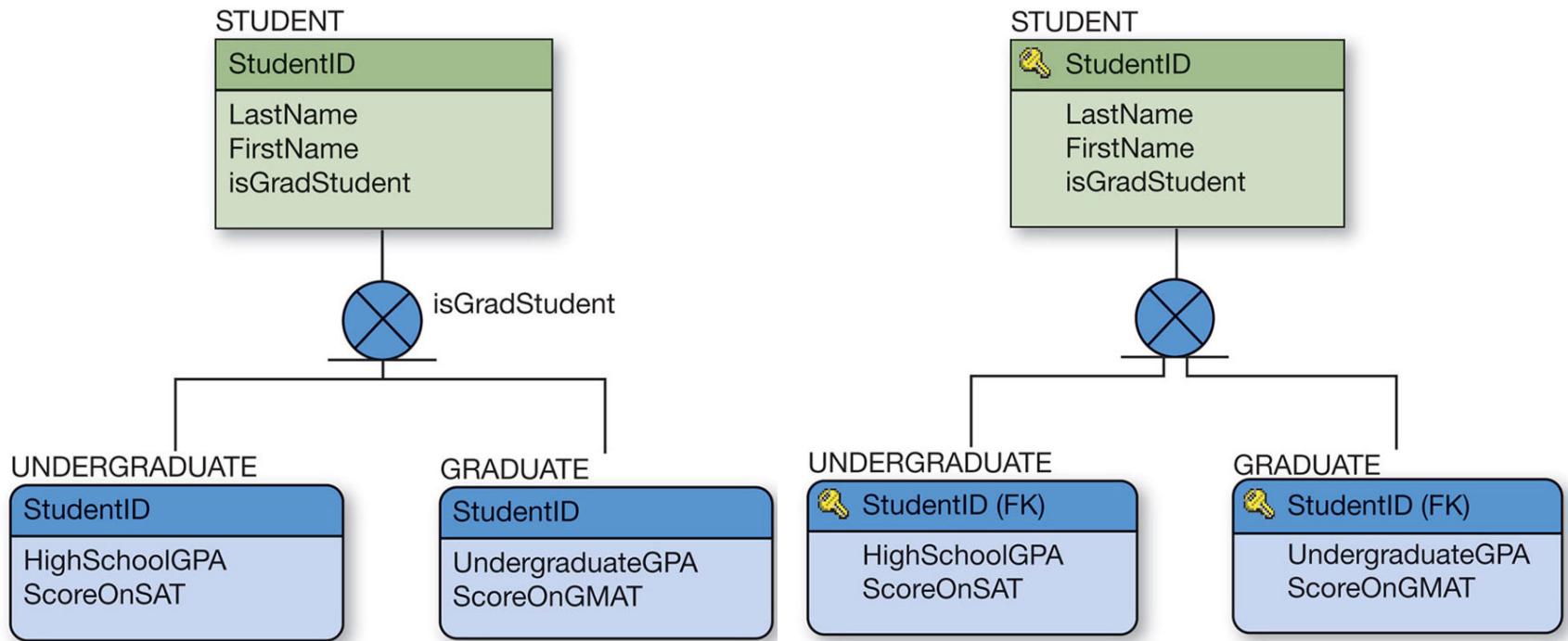
# Figure 5.18 The Association Relationship



# Figure 5.19 Mixed Entity Relationship Example



# Figure 5.20 Representing Subtypes

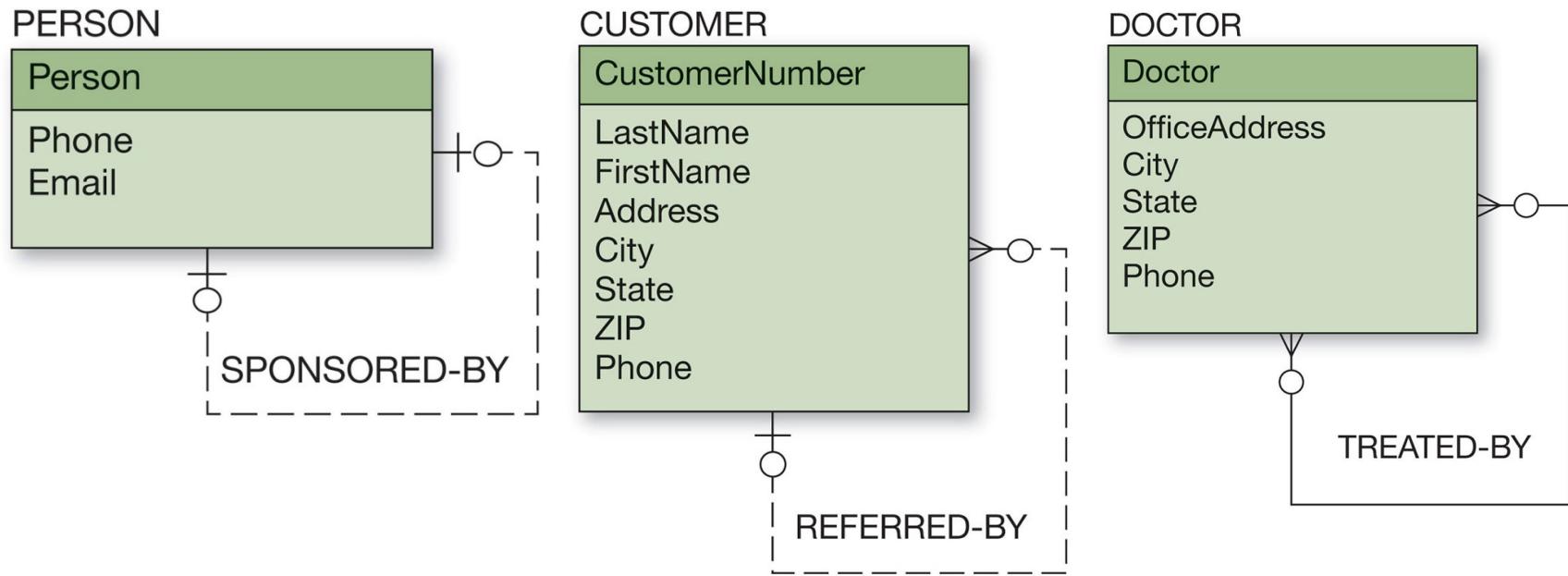


# Representing Recursive Relationships

## Know how to represent 1:1, 1:N, and N:M recursive relationships

- A recursive relationship is a relationship among entities of the same class (a relationship with itself).
- There are three types of recursive relationships:
  - 1:1 and 1:M relationships are saved using foreign keys
  - M:M relationships are saved by creating an intersecting relation

# Figure 5.21 Example Recursive Relationships



# Figure 5.22 Example 1:1 Recursive Relationship

Person

Jones  
Smith  
Parks  
Myrtle  
Pines

PERSON1 Relation		
Person	PersonSponsored	Other Attributes
Jones	Smith	...
Smith	Parks	...
Parks	null	...
Myrtle	Pines	...
Pines	null	...

Referential integrity constraint:

PersonSponsored in PERSON1  
must exist in Person in PERSON1

PERSON2 Relation

PERSON2 Relation		
Person	PersonSponsoredBy	Other Attributes
Jones	null	...
Smith	Jones	...
Parks	Smith	...
Myrtle	null	...
Pines	Myrtle	...

Referential integrity constraint:

PersonSponsoredBy in PERSON2  
must exist in Person in PERSON2

# Figure 5.23 Example 1:N Recursive Relationship

Customer Number

100	200, 400
300	500
400	600, 700

Referred These Customers

CUSTOMER Relation

CustomerNumber	CustomerData	ReferredBy
100	...	null
200	...	100
300	...	null
400	...	100
500	...	300
600	...	400
700	...	400

Referential integrity constraint:

ReferredBy in CUSTOMER must exist in CustomerNumber in CUSTOMER

```
SELECT *
FROM CUSTOMER A JOIN CUSTOMER B
ON A.CustomerNumber = B.ReferredBy;
```

# Figure 5.24 Example N:M Recursive Relationship

Provider

Jones  
Parks  
Smith  
Abernathy  
Franklin

Receiver

Smith  
Abernathy  
Jones  
Franklin

Name	DOCTOR Relation
	Other Attributes
Jones	...
Parks	...
Smith	...
Abernathy	...
O'Leary	...
Franklin	...

TREATMENT-INTERSECTION Relation

Physician      Patient

Jones	Smith
Parks	Smith
Smith	Abernathy
Abernathy	Jones
Parks	Franklin
Franklin	Abernathy
Jones	Abernathy

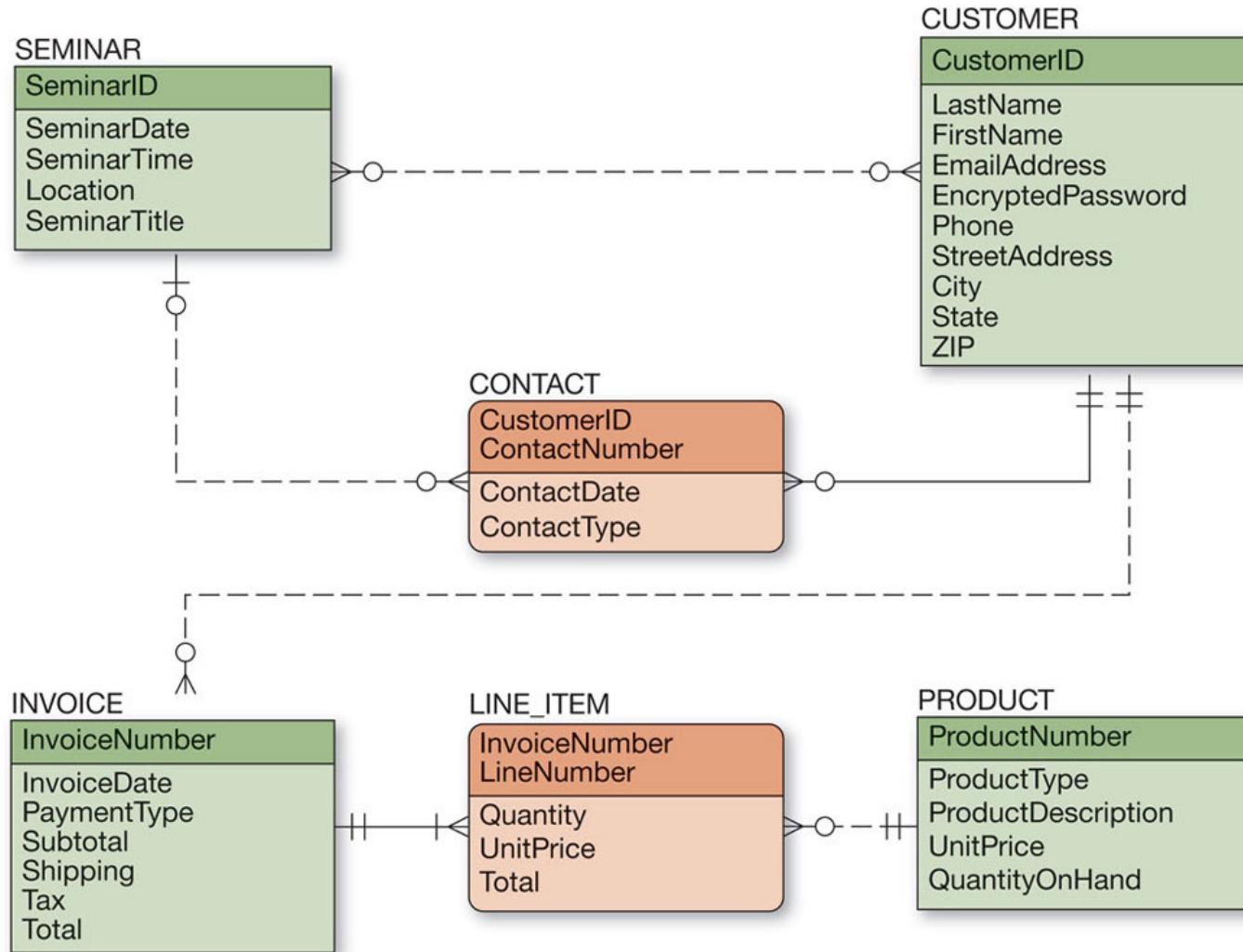
Referential integrity constraints:

Physician in TREATMENT-INTERSECTION must exist in Name in DOCTOR

Patient in TREATMENT-INTERSECTION must exist in Name in DOCTOR

```
SELECT *  
FROM DOCTOR A JOIN TREATMENT-INTERSECTION,  
DOCTOR B  
ON A.Name = TREATMENT-INTERSECTION.Physician  
JOIN DOCTOR B  
ON TREATMENT-INTERSECTION.Patient =  
B.Name;
```

# Figure 5.25 The Final Data Model for Heather Sweeney Designs



# Specifying Column Properties

**Learn how to transform E-R data models into relational designs**

- Column properties must be specified for each table.
- The finalized column properties for the HSD tables are on the next set of slides – these are the column characteristics after additional needed foreign keys have been added. This includes any new intersection tables.

# Figure 5.26 Heather Sweeney Designs HSD Database Column Specifications (1 of 3)

Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
SeminarID	Integer	Primary Key	Yes	DBMS supplied	Surrogate Key: Initial value=1 Increment=1
SeminarDate	Date	No	Yes	None	Format: yyyy-mm- dd
SeminarTime	Time	No	Yes	None	Format: 00:00:00.000
Location	VarChar (100)	No	Yes	None	
SeminarTitle	VarChar (100)	No	Yes	None	

Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
CustomerID	Integer	Primary Key	Yes		DBMS Supplied Surrogate Key: Initial Value=1 Increment=1
LastName	Char (25)	No	Yes	None	
FirstName	Char (25)	No	Yes	None	
EmailAddress	VarChar (100)	Primary Key	Yes	None	
EncryptedPassword	VarChar(50)	No	No	None	
Phone	Char (12)	No	Yes	None	Format: ###-###-####
StreetAddress	Char (35)	No	No	None	
City	Char (35)	No	No	Dallas	
State	Char (2)	No	No	TX	Format: AA
ZIP	Char (10)	No	No	75201	Format: #####-####

# Figure 5.26 Heather Sweeney Designs HSD Database Column Specifications (2 of 3)

Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
CustomerID	Integer	Primary Key, Foreign Key	Yes	None	REF: CUSTOMER
ContactNumber	Integer	Primary Key	Yes	None	
ContactDate	Date	No	Yes	None	Format: yyyy-mm-dd
ContactType	Char (15)	No	Yes	None	

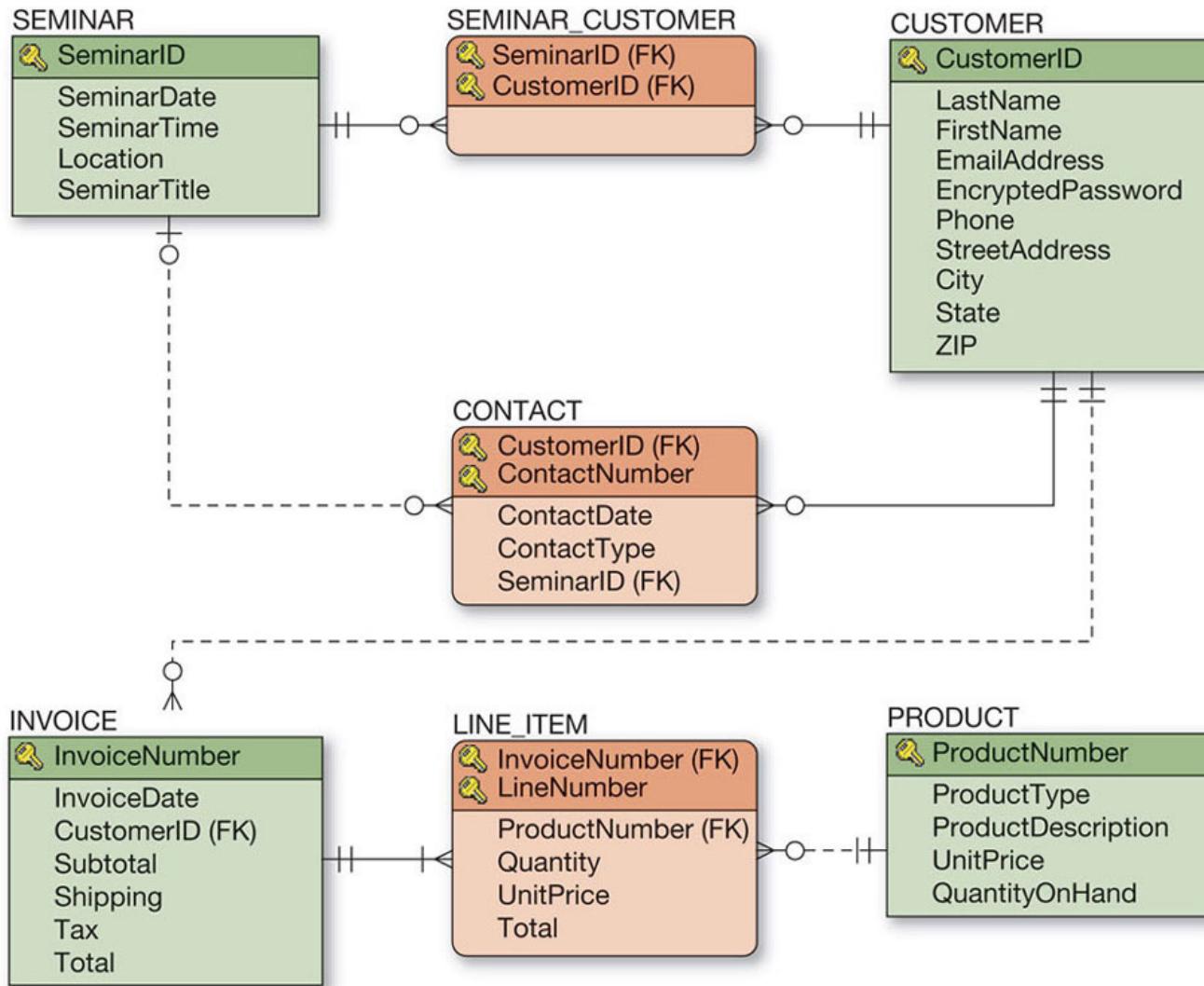
Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
InvoiceNumber	Integer	Primary Key	Yes	DBMS supplied	Surrogate Key: Initial value=35000 Increment=1
InvoiceDate	Date	No	Yes	None	Format: yyyy- mm-dd
PaymentType	Char (25)	No	Yes	Cash	
Subtotal	Numeric (9,2)	No	No	None	
Shipping	Numeric (9,2)	No	No	None	
Tax	Numeric (9,2)	No	No	None	
Total	Numeric (9,2)	No	No	None	

# Figure 5.26 Heather Sweeney Designs HSD Database Column Specifications (3 of 3)

Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
InvoiceNumber	Integer	Primary Key, Foreign Key	Yes	None	REF: INVOICE
LineNumber	Integer	Primary Key	Yes	None	This is not quite a Surrogate Key—for each InvoiceNumber: Increment=1 Application logic will be needed to supply the correct value
Quantity	Integer	No	No	None	
UnitPrice	Numeric (9,2)	No	No	None	
Total	Numeric (9,2)	No	No	None	

Column Name	Data Type (Length)	Key	Required	Default Value	Remarks
ProductNumber	Char (35)	Primary Key	Yes	None	
ProductType	Char (24)	No	Yes	None	
ProductDescription	VarChar (100)	No	Yes	None	
UnitPrice	Numeric (9, 2)	No	Yes	None	
QuantityOnHand	Integer	No	Yes	0	

# Figure 5.27 Database Design for Heather Sweeney



# Heather Sweeney's Database Design Schema

Learn how to transform E-R data models into relational designs

SEMINAR (SeminarID, SeminarDate, SeminarTime, Location, SeminarTitle)

CUSTOMER (CustomerID, LastName, FirstName, EmailAddress,\  
EncryptedPassword, Phone, StreetAddress, City, State, ZIP)

SEMINAR\_CUSTOMER (SeminarID, CustomerID)

CONTACT (CustomerID, ContactDate, ContactDate, ContactType, SeminarID)

PRODUCT (ProductNumber, ProductType, ProductDescription, UnitPrice,  
QuantityOnHand)

INVOICE (InvoiceNumber, InvoiceDate, CustomerID, PaymentType, SubTotal,  
Shipping, Tax, Total)

LINE\_ITEM (InvoiceNumber, LineNumber, ProductNumber, Quantity, UnitPrice,  
Total)

# Figure 5.29 Referential Integrity Constraint Enforcement for Heather Sweeney Designs

Referential Integrity Constraint Enforcement for Heather Sweeney Designs

Parent	Child	Referential Integrity Constraint	Cascading Behavior	
			On Update	On Delete
SEMINAR	SEMINAR_CUSTOMER	SeminarID in SEMINAR_CUSTOMER must exist in SeminarID in SEMINAR	No	No
CUSTOMER	SEMINAR_CUSTOMER	CustomerID in SEMINAR_CUSTOMER must exist in CustomerID in CUSTOMER	No	No
SEMINAR	CONTACT	SeminarID in CONTACT must exist in SeminarID in SEMINAR	No	No
CUSTOMER	CONTACT	CustomerID in CONTACT must exist in CustomerID in CUSTOMER	No	No
CUSTOMER	INVOICE	CustomerID in INVOICE must exist in CustomerID in CUSTOMER	No	No
INVOICE	LINE_ITEM	InvoiceNumber in LINE_ITEM must exist in InvoiceNumber in INVOICE	No	Yes
PRODUCT	LINE_ITEM	ProductNumber in LINE_ITEM must exist in ProductNumber in PRODUCT	Yes	No

# Copyright



**This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.**

# Database Concepts

Ninth Edition



## Chapter 6

### Database Administration

# Learning Objectives (1 of 2)

- Understand the need for and importance of database administration
- Know basic administrative and managerial DBA functions
- Understand the need for concurrency control, security, and backup and recovery
- Learn about typical problems that can occur when multiple users process a database concurrently
- Understand the use of locking and the problem of deadlock
- Learn the difference between optimistic and pessimistic locking
- Know the meaning of ACID transaction
- Learn the four 1992 ANSI standard isolation levels
- Learn different ways of processing a database using cursors

# Learning Objectives (2 of 2)

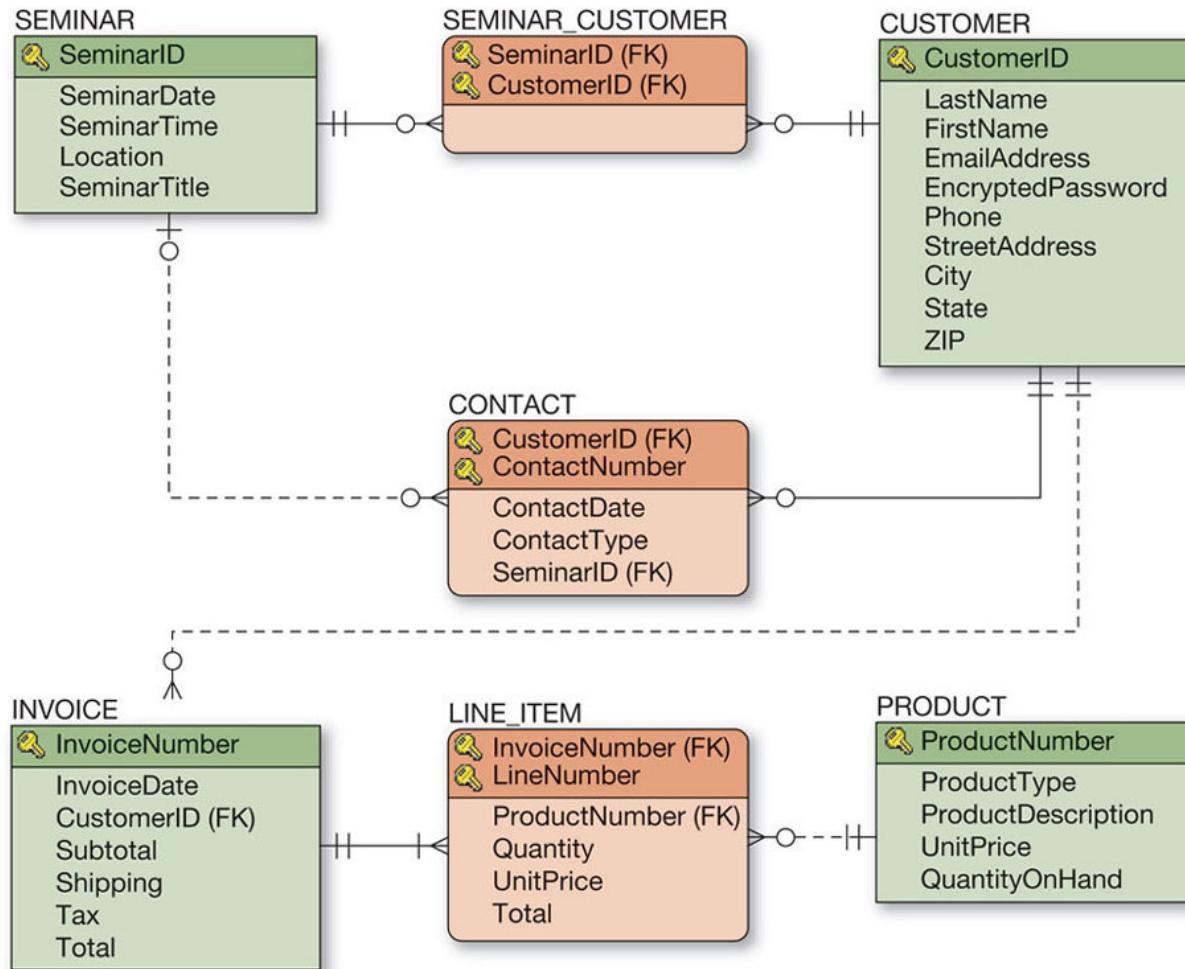
- Understand the need for security and specific tasks for improving database security
- Know the difference between recovery via reprocessing and recovery via rollback/rollforward
- Understand the nature of the tasks required for recovery using rollback/rollforward

# Data Administration Terms

## Understand the need for and importance of database Administration

- **Data administration** refers to a function that applies to an entire organization concerning corporate data privacy and security issues.
- **Database administration** refers to a more technical function that is specific to a particular database, including applications associated with it.
- **Database administrator (DBA)** refers to the person in charge of a database and facilitates the development of use of it.

# Figure 5.27 Heather Sweeney Designs: Database Design



Note: From the previous chapter showing the final database design for Heather Sweeney

# Figure 6.1 The HSD Database in MySQL Workbench

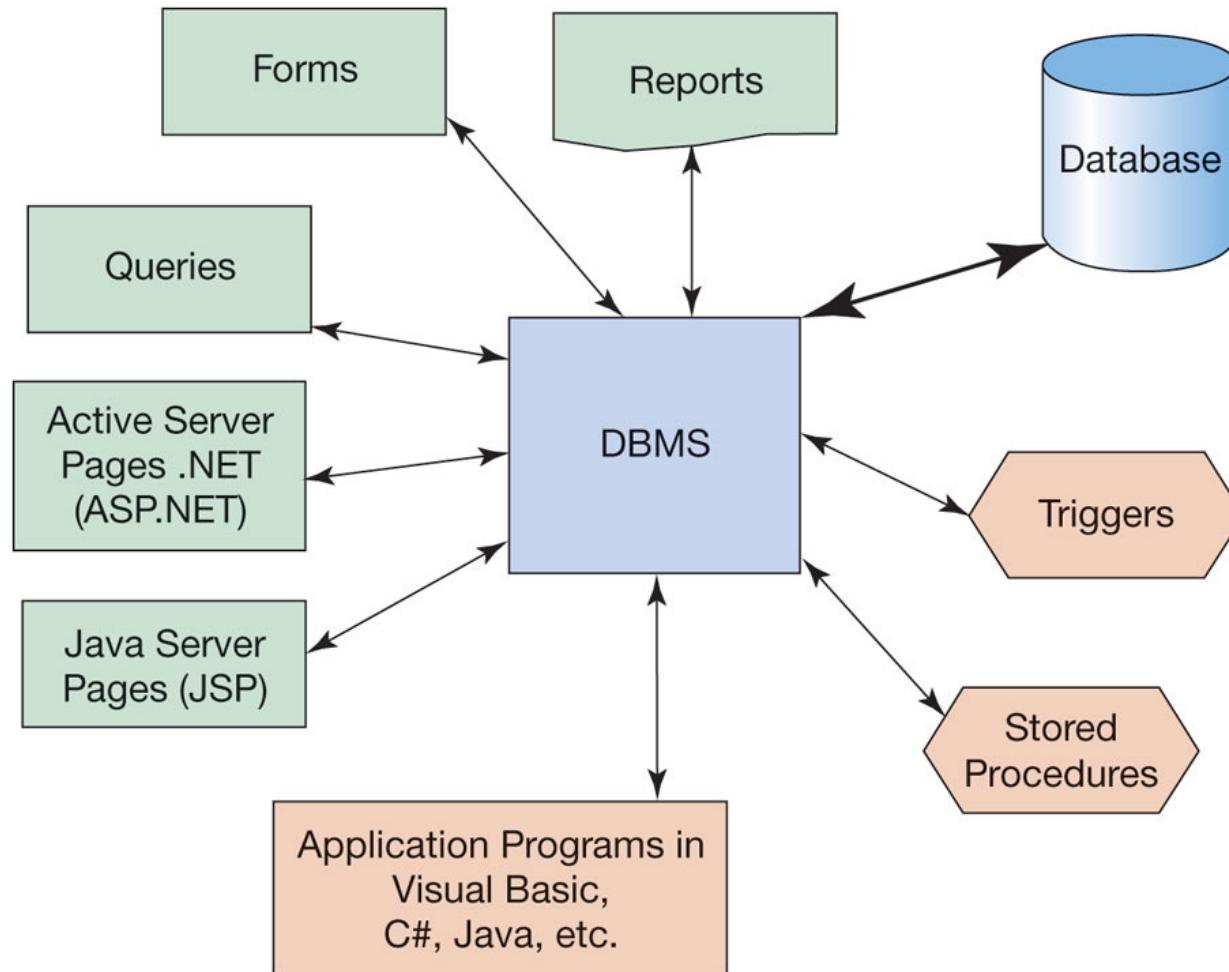
The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the 'Tables' section of the 'hsd' schema, the 'customer' table is selected. A callout box labeled 'The HSD database and table objects' points to this selection. In the central Results Grid, the data from the 'customer' table is displayed in a tabular format. A callout box labeled 'The data in the CUSTOMER table' points to the first few rows of the grid. The grid columns are: CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword, Phone, StreetAddress, City, State, and ZIP. The data rows are as follows:

CustomerID	LastName	FirstName	EmailAddress	EncryptedPassword	Phone	StreetAddress	City	State	ZIP
1	Jacobs	Nancy	Nancy.Jacobs@somewhere.com	nf46tG9E	817-871-8123	1440 West Palm Drive	Fort Worth	TX	76110
2	Jacobs	Chantel	Chantel.Jacobs@somewhere.com	b6STG03f	817-871-8234	1550 East Palm Drive	Fort Worth	TX	76112
3	Able	Ralph	Ralph.Able@somewhere.com	m56fGH08	210-281-7987	123 Elm Street	San Antonio	TX	78214
4	Baker	Susan	Susan.Baker@somewhere.com	PC93fEk9	210-281-7876	456 Oak Street	San Antonio	TX	78216
5	Eagleton	Sam	Sam.Eagleton@elsewhere.com	bvnR44W8	210-281-7765	789 Pine Street	San Antonio	TX	78218
6	Foxtrot	Kathy	Kathy.Foxtrot@somewhere.com	aa8tY4GL	972-233-6234	11023 Elm Street	Dallas	TX	75220
7	George	Sally	Sally.George@somewhere.com	LK8G2tyF	972-233-6345	12034 San Jacinto	Dallas	TX	75223
8	Hullett	Shawn	Shawn.Hullett@elsewhere.com	bu78WW3t	972-233-6456	13045 Flora	Dallas	TX	75224
9	Pearson	Bobbi	Bobbi.Pearson@elsewhere.com	kg6N2O0p	512-974-3344	43 West 23rd Street	Austin	TX	78710

The bottom Output pane shows the executed query: 'select \* from customer LIMIT 0, 50000' and the result '12 row(s) returned'.

Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 6.2 The Database Processing Environment



# Concurrency Control

## Understand the need for concurrency control, security, and backup and recovery

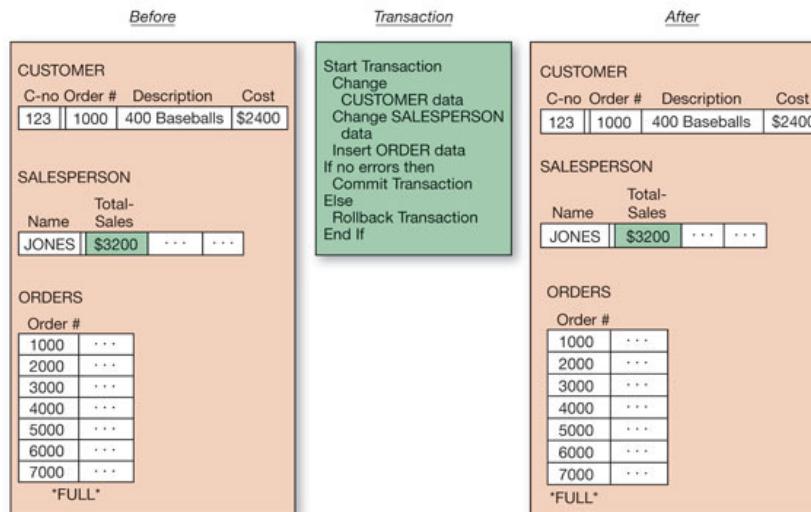
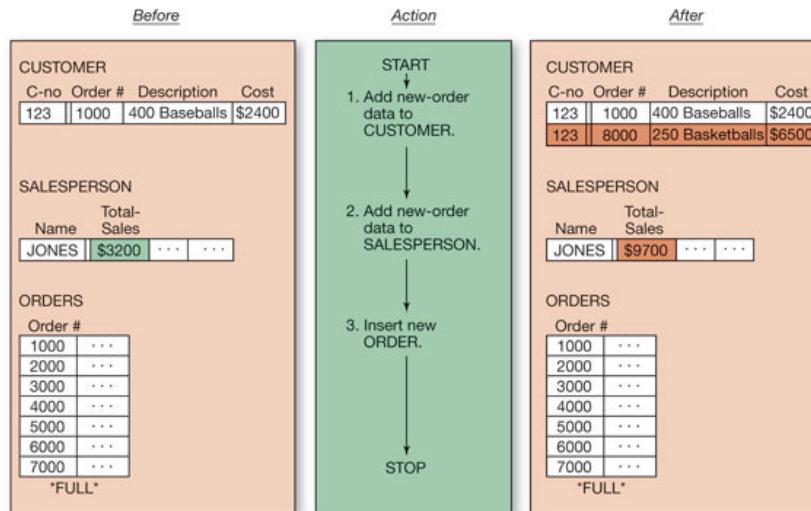
- **Concurrency control** ensures that one user's work does not inappropriately influence another user's work.
- Users should be able to enter an order and get the same result whether there are no other users, or hundreds of other users.
- No concurrency control technique is ideal for all circumstances, they all involve trade-offs.
  - Strict concurrency requires locking the database, thus not allowing any other user (high price to pay)
  - Other measures allow more **throughput** (maximum rate of processing) but has a lower concurrency control

# The Need for Atomic Transactions

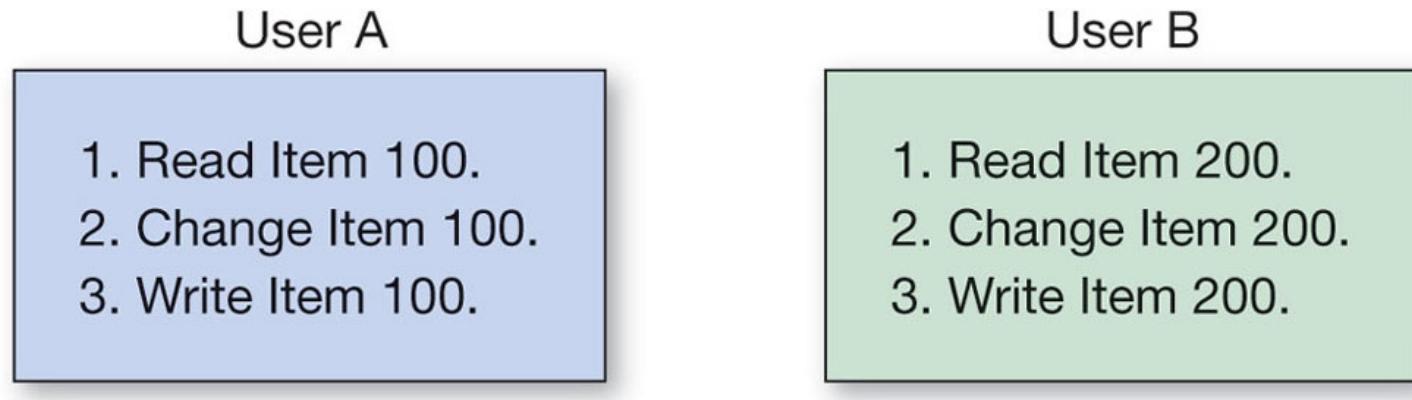
## Understand the need for concurrency control, security, and backup and recovery

- Users submit work in the form of **transactions**, also known as **logical units of work (LUWs)**.
- An **atomic transaction** is one where a series of actions are taken on a database such that all of them are performed successfully or none of them are performed at all.
- An example of an atomic transaction is:
  - Change the customer transaction, increasing the value of Amount Owed.
  - Change the salesperson record, increasing the value of Commission Due.
  - Insert the new-order record into the database.

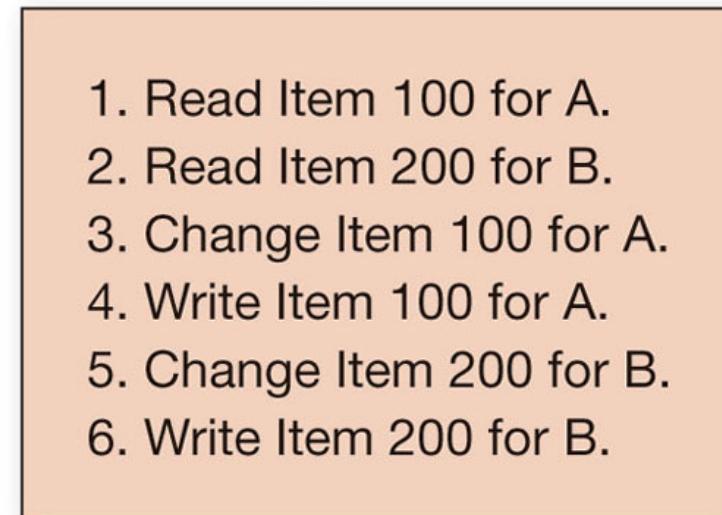
# Figure 6.3 Comparison of the Results of Applying Serial Actions Versus a Multiple-Step Transaction



## Figure 6.4 Example of Concurrent Processing of Two Users' Tasks



One possible order of processing at database server



# Lost Update Problem

Learn about typical problems that can occur when multiple users process a database concurrently

- If two users are attempting to update the same piece of data at the same time, it is possible that one update may overwrite the other update.
- If one user's update overwrites the other's, this is called a **lost update problem** (concurrent problem).
- If one user reads data that have been processed by only a portion of another user's transaction, this is called a **inconsistent read problem**.

# Lost Updates

TABLE 9.3 LOST UPDATES

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	$\text{PROD\_QOH} = 35 + 100$	
4	T2	$\text{PROD\_QOH} = 35 - 30$	
5	T1	Write PROD_QOH <b>(Lost update)</b>	135
6	T2	Write PROD_QOH	5

# Figure 6.5 Example of the Lost Update Problem

User A

1. Read Item 100  
(assume item count is 10).
2. Reduce count of items by 5.
3. Write Item 100.

User B

1. Read Item 100  
(assume item count is 10).
2. Reduce count of items by 3.
3. Write Item 100.

Order of processing at database server

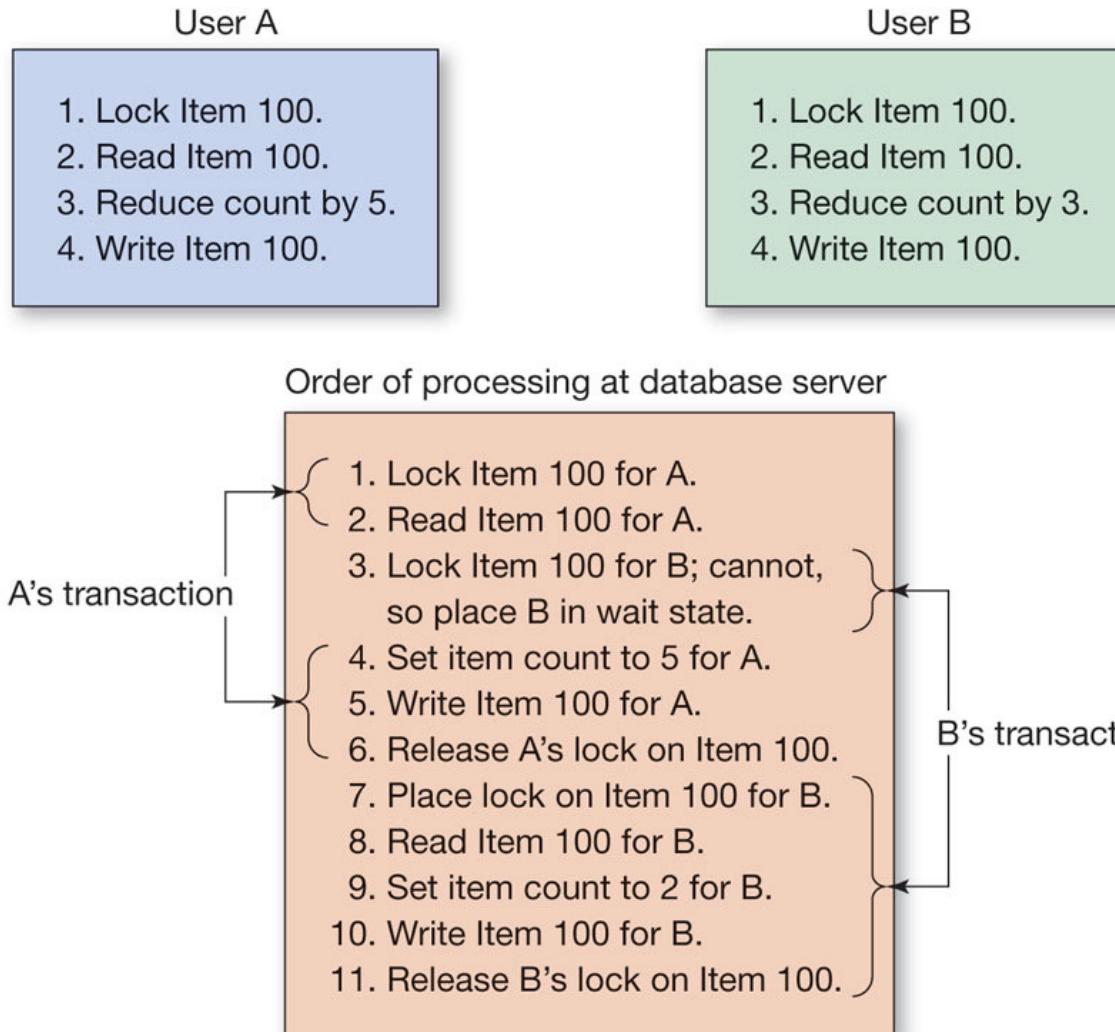
1. Read Item 100 (for A).
2. Read Item 100 (for B).
3. Set item count to 5 (for A).
4. Write Item 100 for A.
5. Set item count to 7 (for B).
6. Write Item 100 for B.

# Resource Locking

## Understand the use of locking and the problem of deadlock

- **Resource locking** prevents concurrent processing problems by disallowing sharing by locking data that are retrieved for update.
- Locks placed by the DBMS are called **implicit locks**, while those placed by command are called **explicit locks**.
- **Exclusive lock** locks an item from access of any type.
  - No other transaction can read or change the data
- A **shared lock** locks an item from being changed but not from being read.
  - Other transactions can read the data but not alter it

# Figure 6.6 Example of Concurrent Processing with Explicit Locks



# Serializable Transactions

## Understand the use of locking and the problem of deadlock

- When two or more transactions are processed concurrently, the results in the database should be logically consistent with the results that would have been achieved had the transactions been processed in an arbitrary serial fashion.
- A scheme for processing concurrent transactions in this way is said to be **serializable**

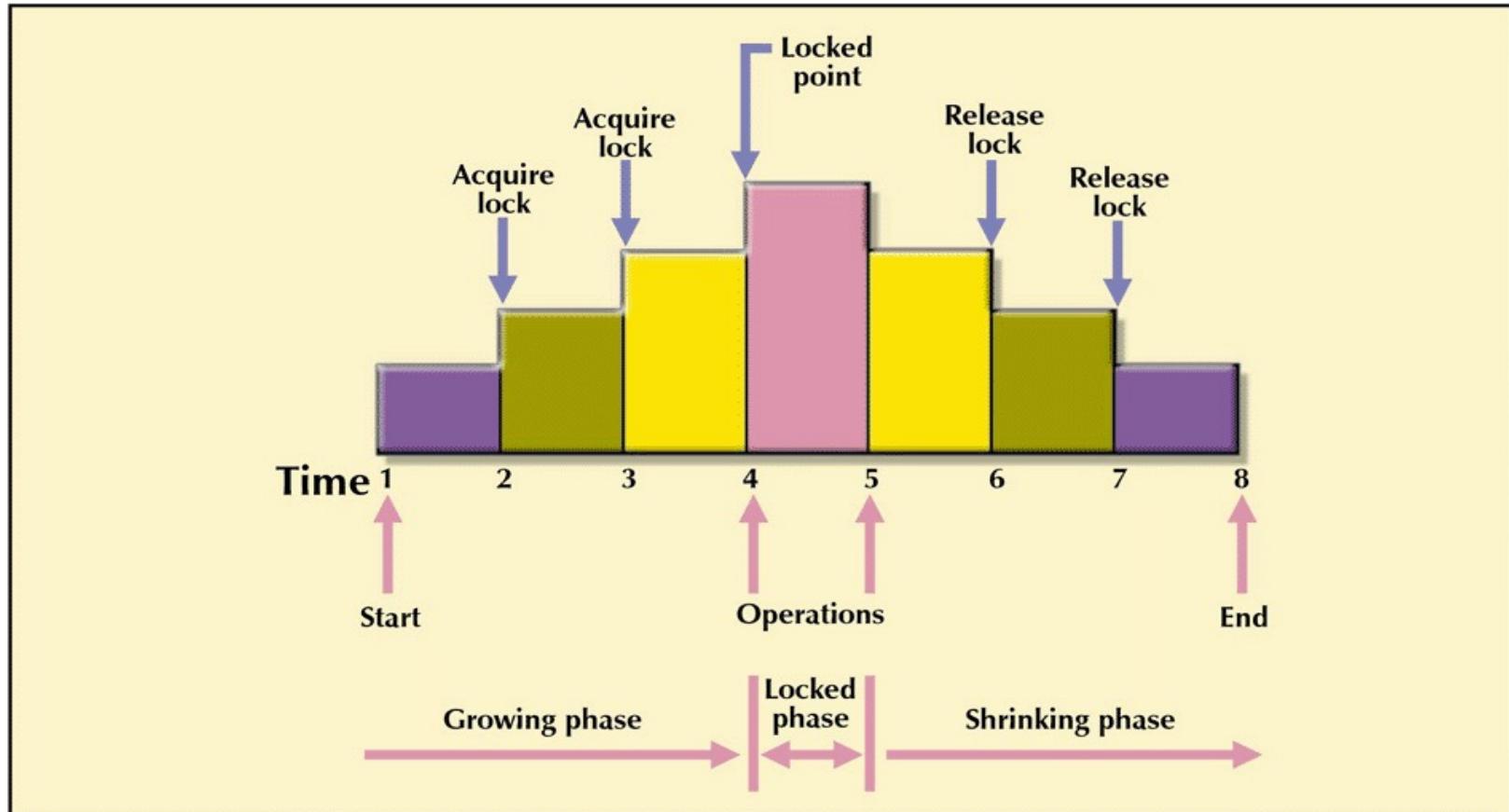
# Two-Phased Locking

## Understand the use of locking and the problem of deadlock

- One way to achieve serializable transactions is by using two-phased locking.
- **Two-phased locking** lets locks be obtained and released as they are needed:
  - a **growing phase**, when the transaction continues to request additional locks
  - a **shrinking phase**, when the transaction begins to release the locks

# Two-Phase Locking Protocol

FIGURE 9.7 TWO-PHASE LOCKING PROTOCOL



# Deadlock

## Understand the use of locking and the problem of deadlock

- As a transaction begins to lock resources, it may have to wait for a particular resource to be released by another transaction.
- Sometimes two transactions may indefinitely wait on each other to release resources. This condition is known as a **deadlock** or the **deadly embrace**.
- This condition can lock up a computer preventing use by any user until it is fixed.

# Figure 6.7 Examples of Deadlock

User A

1. Lock paper.
2. Take paper.
3. Lock pencils.

User B

1. Lock pencils.
2. Take pencils.
3. Lock paper.

Order of processing at database server

1. Lock paper for User A.
2. Lock pencils for User B.
3. Process A's request; write paper record.
4. Process B's request; write pencil record.
5. Put A in wait state for pencils.
6. Put B in wait state for paper.

\*\* Locked \*\*

# Optimistic Versus Pessimistic Locking

Learn the difference between optimistic and pessimistic locking

- **Optimistic locking** assumes that no conflict will occur. Data are read, the transaction processed, updates are issued, and then a check is made to see if conflict occurred. If a conflict occurred it is rolled back and repeated until successful.
- **Pessimist locking** assumes that conflict will occur, thus locks are issued, the transaction completed, and then the locks are released.

# Figure 6.8 Example of Optimistic Locking

```
SELECT    PRODUCT.Name, PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.Name = 'Pencil'

OldQuantity = PRODUCT.Quantity

Set NewQuantity = PRODUCT.Quantity - 5

{process transaction – take exception action if NewQuantity < 0, etc.}

{If no errors have occurred:}

LOCK PRODUCT {at some level of granularity}

UPDATE    PRODUCT
SET       PRODUCT.Quantity = NewQuantity
WHERE     PRODUCT.Name = 'Pencil'
        AND PRODUCT.Quantity = OldQuantity

UNLOCK   PRODUCT

{check to see if update was successful;
if not, repeat transaction}
```

# Figure 6.9 Example of Pessimistic Locking

```
LOCK      PRODUCT {at some level of granularity}  
  
SELECT    PRODUCT.Name, PRODUCT.Quantity  
FROM      PRODUCT  
WHERE     PRODUCT.Name = 'Pencil'  
  
Set NewQuantity = PRODUCT.Quantity - 5  
  
{process transaction – take exception action if NewQuantity < 0, etc.}  
  
{If no errors have occurred:}  
  
UPDATE    PRODUCT  
SET       PRODUCT.Quantity = NewQuantity  
WHERE     PRODUCT.Name = 'Pencil'  
  
UNLOCK    PRODUCT  
  
{no need to check if update was successful}
```

# SQL Transaction Control Language (TLC)

**Learn the difference between optimistic and pessimistic locking**

- The **SQL BEGIN TRANSACTION** statement
- The **SQL COMMIT TRANSACTION** statement
- The **SQL ROLLBACK TRANSACTION** statement

Note: Exact SQL syntax varies between various DBMS products.

# Figure 6.10 Example of Marking Transaction Boundaries

START TRANSACTION:

```
SELECT      PRODUCT.Name, PRODUCT.Quantity  
FROM        PRODUCT  
WHERE       PRODUCT.Name = 'Pencil'
```

Set NewQuantity = PRODUCT.Quantity – 5

{process part of transaction – take exception action if NewQuantity < 0, etc.}

```
UPDATE      PRODUCT  
SET         PRODUCT.Quantity = NewQuantity  
WHERE       PRODUCT.Name = 'Pencil'
```

IF transaction has completed normally      THEN

    COMMIT

ELSE

    ROLLBACK

END IF

Continue processing other actions not part of this transaction . . .

# ACID Transaction (1 of 2)

## Know the meaning of ACID transaction

- Sometimes the acronym *ACID* is applied to transactions.
- An **ACID transaction** is one that is ***atomic*, *consistent*, *isolated*, and *durable***.
  - Atomic:
    - an **atomic** transaction is one in which all of the database actions occur, or none of them do
    - a transaction consists of a series of steps, each of which must be successful to be saved
  - Consistent
    - a **consistent** transaction means that no other transactions are permitted on the records until the current transaction finishes; referred to as **statement level consistency** among all records

# ACID Transaction (2 of 2)

## Know the meaning of ACID transaction

- Isolation
  - because of multiuser environments, different transactions may be operating on the same data which can result in continuously changing data content
  - 1992 ANSI SQL standard defines four **isolation levels** that specify which of the concurrency control problems are allowed
- Durable
  - a **durable** transaction is one in which all committed changes are permanent

# Figure 6.11 Summary of Data Read Problems

Data Read Problem Type	Definition
Dirty Read	The transaction reads a row that has been changed, but the change has <i>not</i> been committed. If the change is rolled back, the transaction has incorrect data.
Nonrepeatable Read	The transaction rereads data that has been changed, and finds changes due to committed transactions.
Phantom Read	The transaction rereads data and finds new rows inserted by a committed transaction.

# An Uncommitted Data Problem

TABLE 9.5 AN UNCOMMITTED DATA PROBLEM

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 <b>(Read uncommitted data)</b>	135
5	T2	$\text{PROD\_QOH} = 135 - 30$	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

# Retrieval During Update

TABLE 9.6 RETRIEVAL DURING UPDATE

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;

# None-Repeatable Read

Transaction 1	Transaction 2
<code>BEGIN;</code>	<code>BEGIN;</code>
<code>SELECT * FROM users WHERE id = 1;</code>	<code>UPDATE users SET age = 21 WHERE id = 1;</code>
	<code>COMMIT;</code>
<code>SELECT * FROM users WHERE id = 1; /* will read 21 */</code>	

# Figure 6.12 Summary of Isolation Levels

		Isolation Level			
		Read Uncommitted	Read Committed	Repeatable Read	Serializable
Problem Type	Dirty Read	Possible	Not possible	Not possible	Not possible
	Nonrepeatable Read	Possible	Possible	Not possible	Not possible
	Phantom Read	Possible	Possible	Possible	Not possible

# Cursor

## Learn different ways of processing a database using cursors

- A **cursor** is a pointer into a set of rows that is the result set from an SQL SELECT statement.
- Cursors are usually defined using SELECT statements.

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-DECLARE-CURSOR-CH06-01 *** */
DECLARE CURSOR TransCursor FOR
    SELECT      *
    FROM        [TRANSACTION]
    WHERE       PurchasePrice > 10000;
```

# Cursor Types

## Learn different ways of processing a database using cursors

- Cursors can be **forward only** or **scrollable**.
- In SQL Server, these cursors can be one of three different types:
  - **static cursor** (takes a snapshot and processes it)
  - **dynamic cursor** (a fully featured cursor)
  - **keyset cursor** (combines some features of static and some of dynamic cursors)
- Other DBMS products may define a different set of cursors.

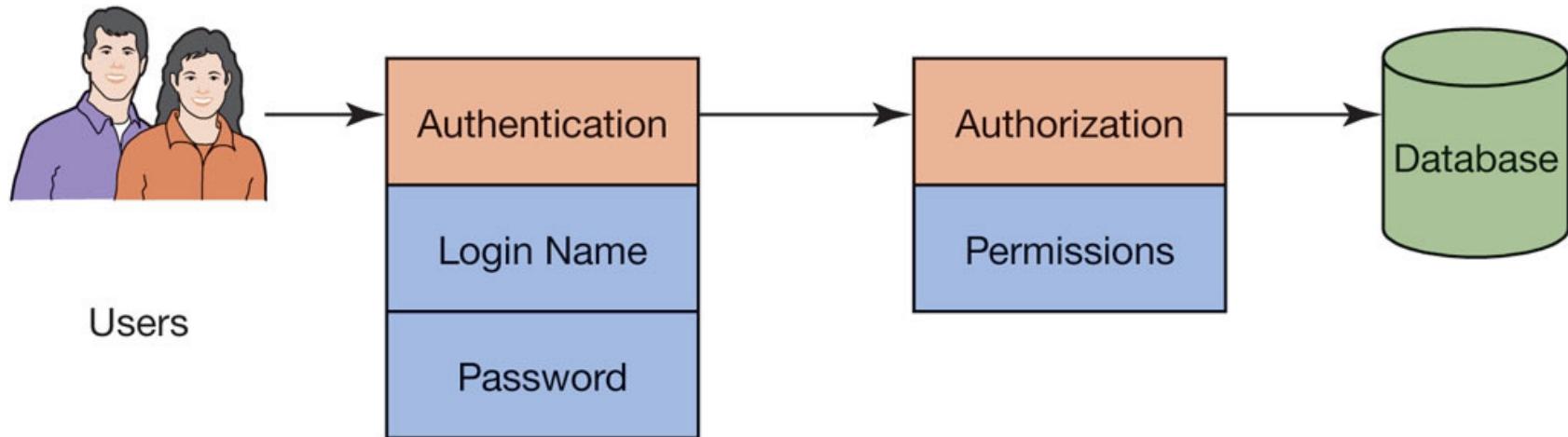
# Figure 6.13 Summary of Cursor Types

Cursor Type	Description	Comments
Static	Application sees the data as they were at the time the cursor was opened.	Changes made by this cursor are visible. Changes from other sources are not visible. Backward and forward scrolling are allowed.
Keyset	When the cursor is opened, a primary key value is saved for each row in the recordset. When the application accesses a row, the key is used to fetch the current values for the row.	Updates from any source are visible. Inserts from sources outside this cursor are not visible (there is no key for them in the keyset). Inserts from this cursor appear at the bottom of the recordset. Deletions from any source are visible. Changes in row order are not visible. If the isolation level is dirty read, then committed updates and deletions are visible; otherwise, only committed updates and deletions are visible.
Dynamic	Changes of any type and from any source are visible.	All inserts, updates, deletions, and changes in recordset order are visible. If the isolation level is dirty read, then uncommitted changes are visible. Otherwise, only committed changes are visible.

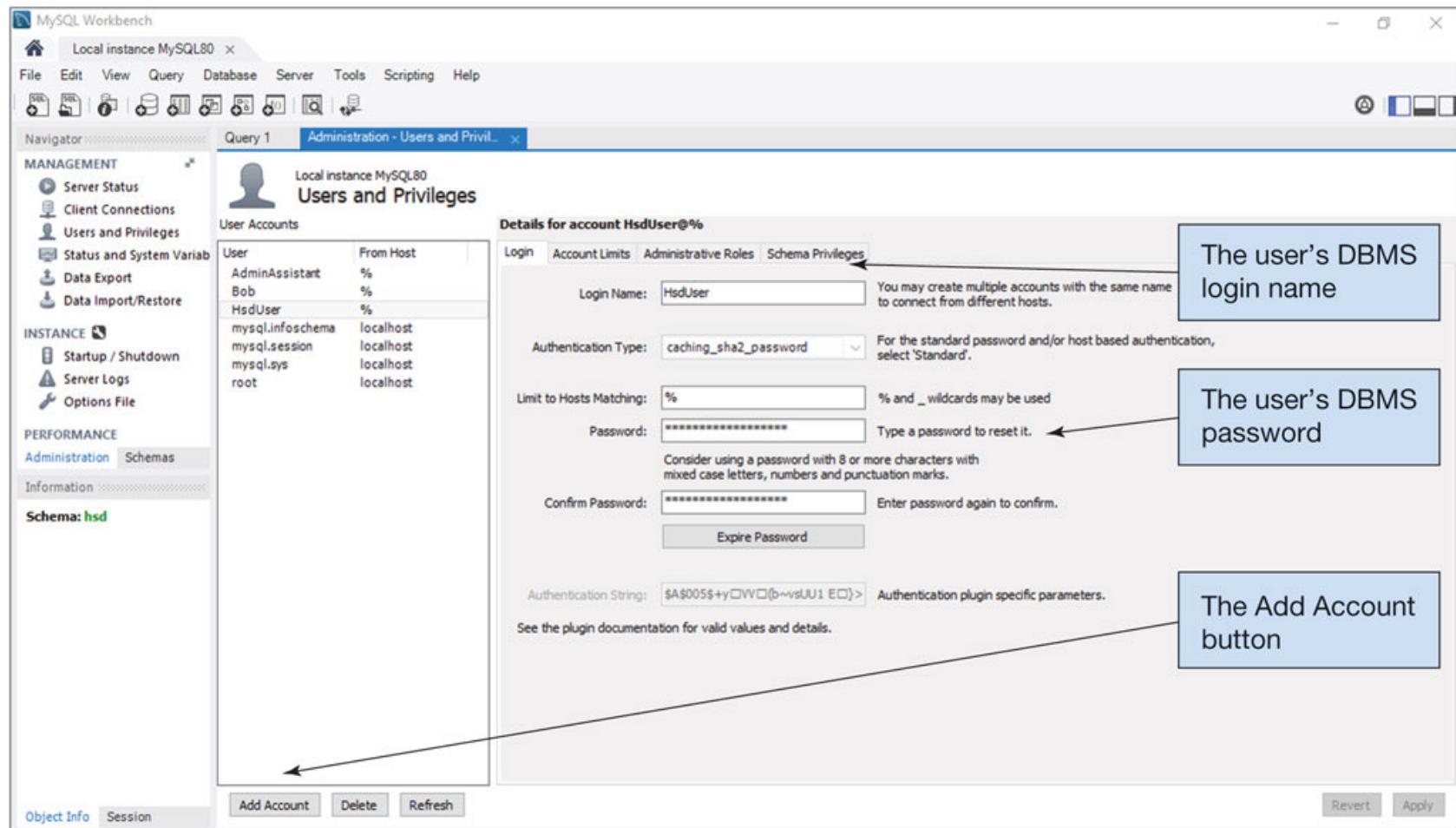
# Figure 6.14 Database Security Authentication and Authorization

**Database security** strives to ensure that:

- only **authorized** users
- perform **authorized** activities

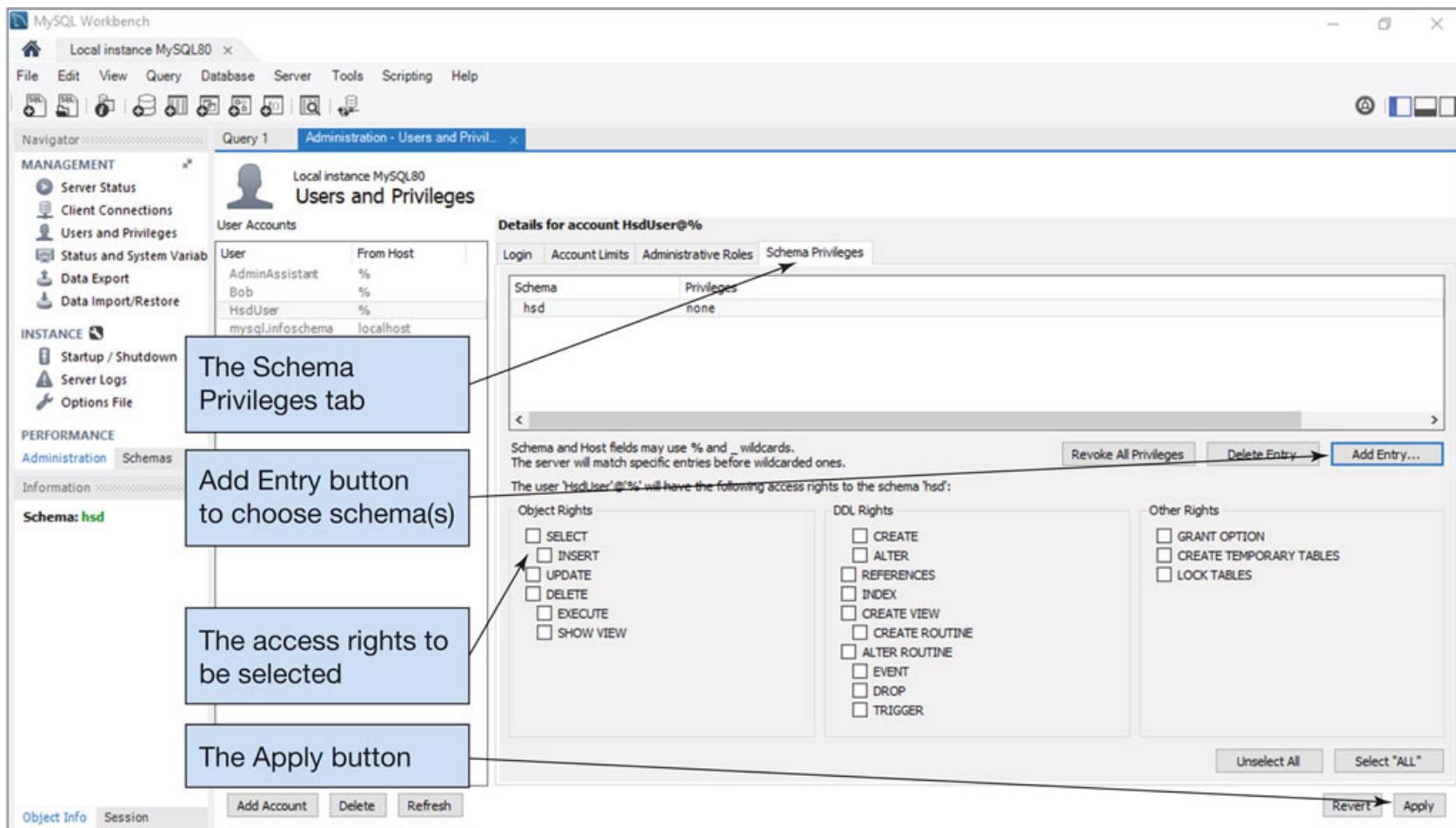


# Figure 6.15 Creating the MySQL User Login



Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 6.16 Using the Workbench to Grant Privileges to a Database Schema



Oracle MySQL Community Server 8.0, Oracle Corporation

# User Processing Rights and Responsibilities

## Understand the need for security and specific tasks for improving database security

- Processing rights define who is permitted to do what, and when they can do it.
- The individuals performing these activities have full responsibility for the implications of their actions.
- An important principle of database security administration is that administrative permissions are given to user *groups* (also known as user *roles*) and not to individual users unless necessary.
- Individuals are identified by a username and a password.

# Figure 6.17 A Model of DBMS Security

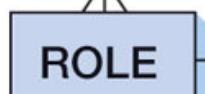
Eleanore Wu

James Johnson

Richard Ent



Eleanore Wu can execute the MonthEnd stored procedure.  
James Johnson can alter all tables.



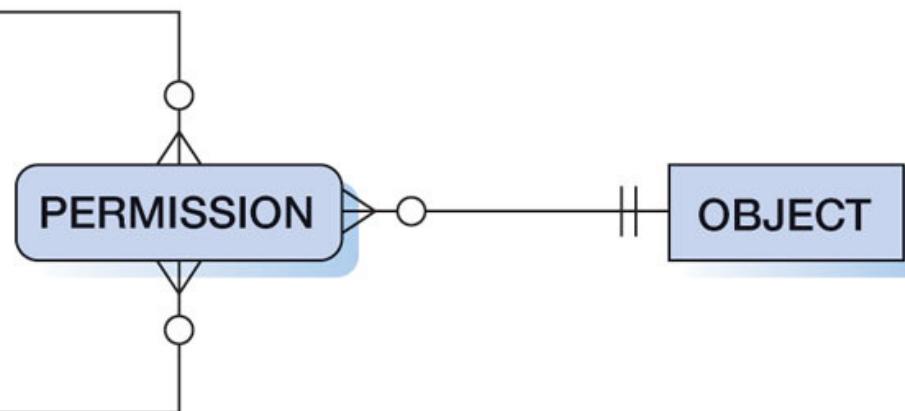
Accounting can update the CUSTOMER table.

Accounting

Tellers

Shop Managers

Unknown Public



# Figure 6.18 MySQL Administrative Roles

The screenshot shows the MySQL Workbench interface for managing users and privileges. The left sidebar has sections for MANAGEMENT (Server Status, Client Connections, Users and Privileges, etc.), INSTANCE (Startup / Shutdown, Server Logs, Options File), and PERFORMANCE (Administration, Schemas). The 'Schemas' tab is selected under PERFORMANCE, showing 'Schema: hsd'. The main window title is 'Administration - Users and Privileges' for 'Local instance MySQL80'. It displays 'User Accounts' with columns 'User' and 'From Host'. The 'Administrative Roles' tab is selected, showing a list of roles with their descriptions:

Role	Description
DBA	grants the rights to perform all tasks
MaintenanceAdmin	grants rights needed to maintain server
ProcessAdmin	rights needed to assess, monitor, and kill any user proce...
UserAdmin	grants rights to create users logins and reset passwords
SecurityAdmin	rights to manage logins and grant and revoke server an...
MonitorAdmin	minimum set of rights needed to monitor server
DBManager	grants full rights on all databases
DBDesigner	rights to create and reverse engineer any database sche...
ReplicationAdmin	rights needed to setup and manage replication
BackupAdmin	minimal rights needed to backup any database

To the right, a vertical list of 'Global Privileges' is shown with checkboxes:

- ALTER
- ALTER ROUTINE
- CREATE
- CREATE ROUTINE
- CREATE TABLESPACE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- EVENT
- EXECUTE
- FILE
- GRANT OPTION
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- SELECT

At the bottom, buttons include 'Revoke All Privileges', 'Revert', and 'Apply'.

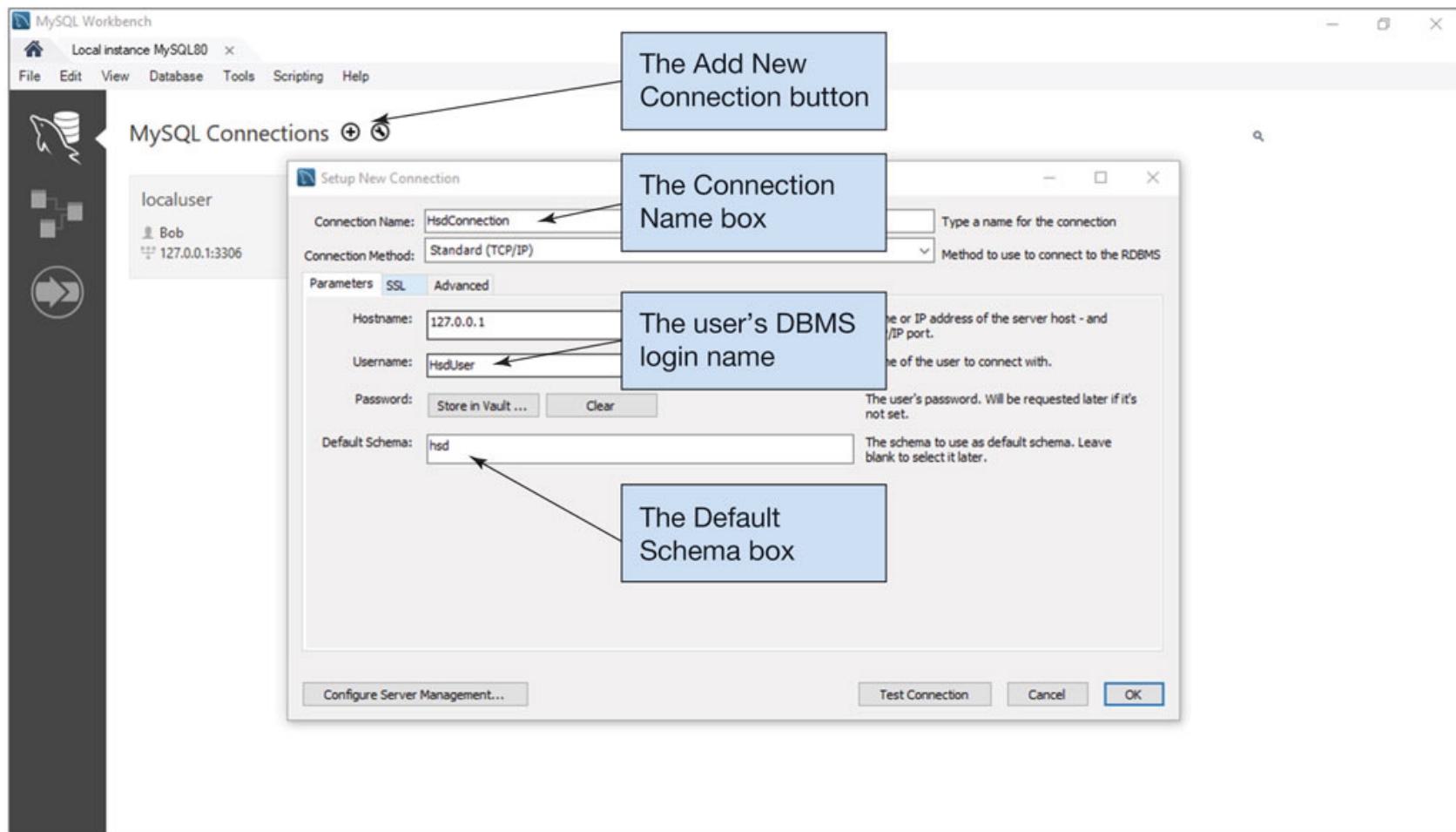
Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 6.19 Processing Rights at Heather Sweeney Designs

## DATABASE RIGHTS GRANTED

Table	Administrative Assistants	Management	System Administrator
SEMINAR	Read, Insert, Change	Read, Insert, Change, Delete	Grant Rights, Modify Structure
CUSTOMER	Read, Insert, Change	Read, Insert, Change	Grant Rights, Modify Structure
SEMINAR_CUSTOMER	Read, Insert, Change, Delete	Read, Insert, Change, Delete	Grant Rights, Modify Structure
CONTACT	Read, Insert, Change	Read, Insert, Change, Delete	Grant Rights, Modify Structure
INVOICE	Read, Insert, Change	Read, Insert, Change, Delete	Grant Rights, Modify Structure
LINE_ITEM	Read, Insert, Change, Delete	Read, Insert, Change, Delete	Grant Rights, Modify Structure
PRODUCT	Read, Insert, Change	Read, Insert, Change, Delete	Grant Rights, Modify Structure

# Figure 6.20 Creating a New Connection for HsdUser



Oracle MySQL Community Server 8.0, Oracle Corporat

# Figure 6.21 DBMS Security Guidelines

- Run the DBMS behind a firewall
- Apply the latest operating system and DBMS service packs and fixes
- Limit DBMS functionality to needed features
- Protect the computer that runs the DBMS
- Manage accounts and passwords
- Encryption of sensitive data transmitted across the network
- Encryption of sensitive data stored in databases

# Application-Level Security

## Understand the need for security and specific tasks for improving database security

- Application level security is often provided on the Web server computer.
- When application security is executed on this server, sensitive security data do not need to be transmitted over the network.
- For example, when users (who are logged in) click a particular button on a browser page, the following query is sent to the Web server and then to the DBMS

```
/* *** EXAMPLE CODE - DO NOT RUN *** */  
/* *** SQL-QUERY-CH06-03 *** */  
SELECT      *  
FROM        EMPLOYEE  
WHERE       EMPLOYEE.Name = 'Benjamin Franklin';
```

# Database Backup and Recovery

## Know the difference between recovery via reprocessing and recovery via rollback/rollforward

- Common causes of database failures:
  - hardware failures
  - programming bugs
  - human errors/mistakes
  - malicious actions
- As these issues are impossible to completely avoid, recovery procedures are essential.

# Recovery via Reprocessing

**Know the difference between recovery via reprocessing and recovery via rollback/rollforward**

- In **reprocessing**, all activities since the backup was performed are redone.
- The time it takes to do this is the same amount of time it took when first entered.
- One downside is that if the system is heavily scheduled, it might never catch up.

# Recovery via Rollback & Rollforward

## Know the difference between recovery via reprocessing and recovery via rollback/rollforward

- Most database management systems provide a mechanism to record activities into a **log file**.
  - to undo a transaction the log must contain a copy of every data-base record before it was changed
    - such records are called **before-images**
    - a transaction is undone by applying before-images of all its changes to the database
  - to redo a transaction the log must contain a copy of every database record (or page) after it was changed
    - these records are called **after-images**
    - a transaction is redone by applying after-images of all its changes to the database
- The log file is then used for recovery via rollback or rollforward.

# Rollforward

## Know the difference between recovery via reprocessing and recovery via rollback/rollforward

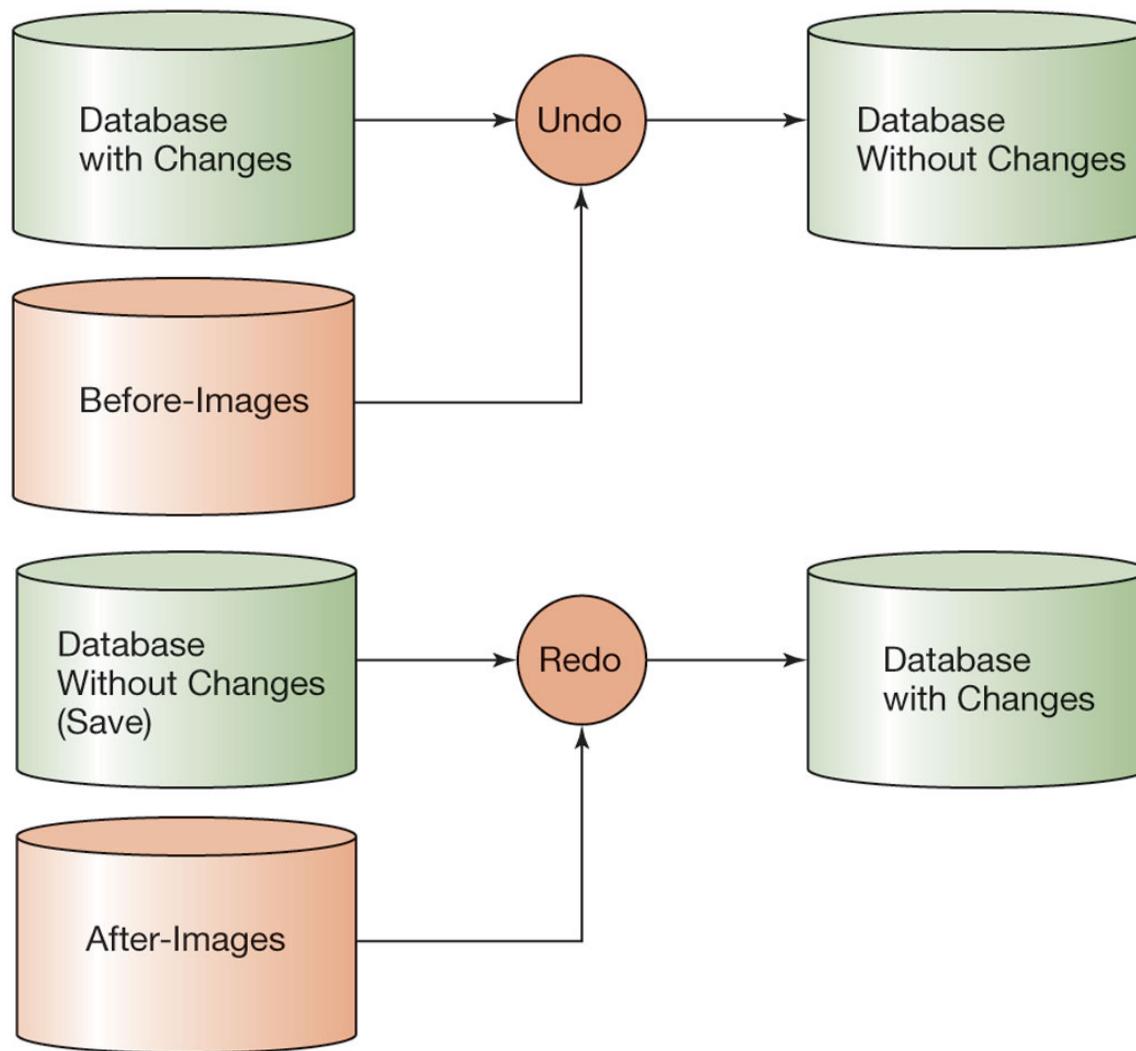
- Activities recorded in the log files may be replayed.
- In doing so, all activities are reapplied to the database.
- This procedure is used to resynchronize restored database data by adding transactions to the last full backup.

# Rollback

## Know the difference between recovery via reprocessing and recovery via rollback/rollforward

- Log files save activities in sequence order.
- It is possible to undo activities in reverse order that they were originally executed.
- This is performed to correct or undo erroneous or malicious transaction(s) after a database is recovered from a full backup.

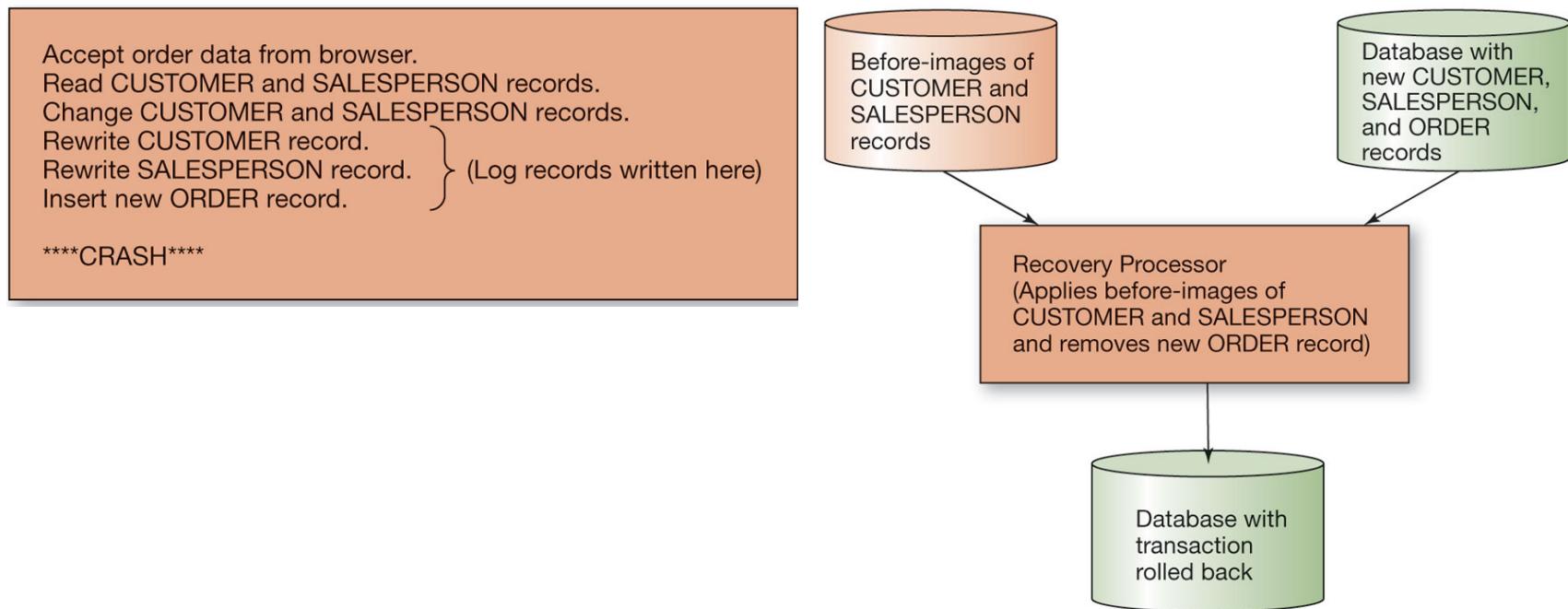
# Figure 6.22 Undo and Redo Transactions



# Figure 6.23 Transaction Log Example

Relative Record Number	Transaction ID	Reverse Pointer	Forward Pointer	Time	Type of Operation	Object	Before-Image	After-Image
1	OT1	0	2	11:42	START			
2	OT1	1	4	11:43	MODIFY	CUST 100	(old value)	(new value)
3	OT2	0	8	11:46	START			
4	OT1	2	5	11:47	MODIFY	SP AA	(old value)	(new value)
5	OT1	4	7	11:47	INSERT	ORDER 11		(value)
6	CT1	0	9	11:48	START			
7	OT1	5	0	11:49	COMMIT			
8	OT2	3	0	11:50	COMMIT			
9	CT1	6	10	11:51	MODIFY	SP BB	(old value)	(new value)
10	CT1	9	0	11:51	COMMIT			

# Figure 6.24 Recovery Example



# A Transaction Log

TABLE 9.1 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				

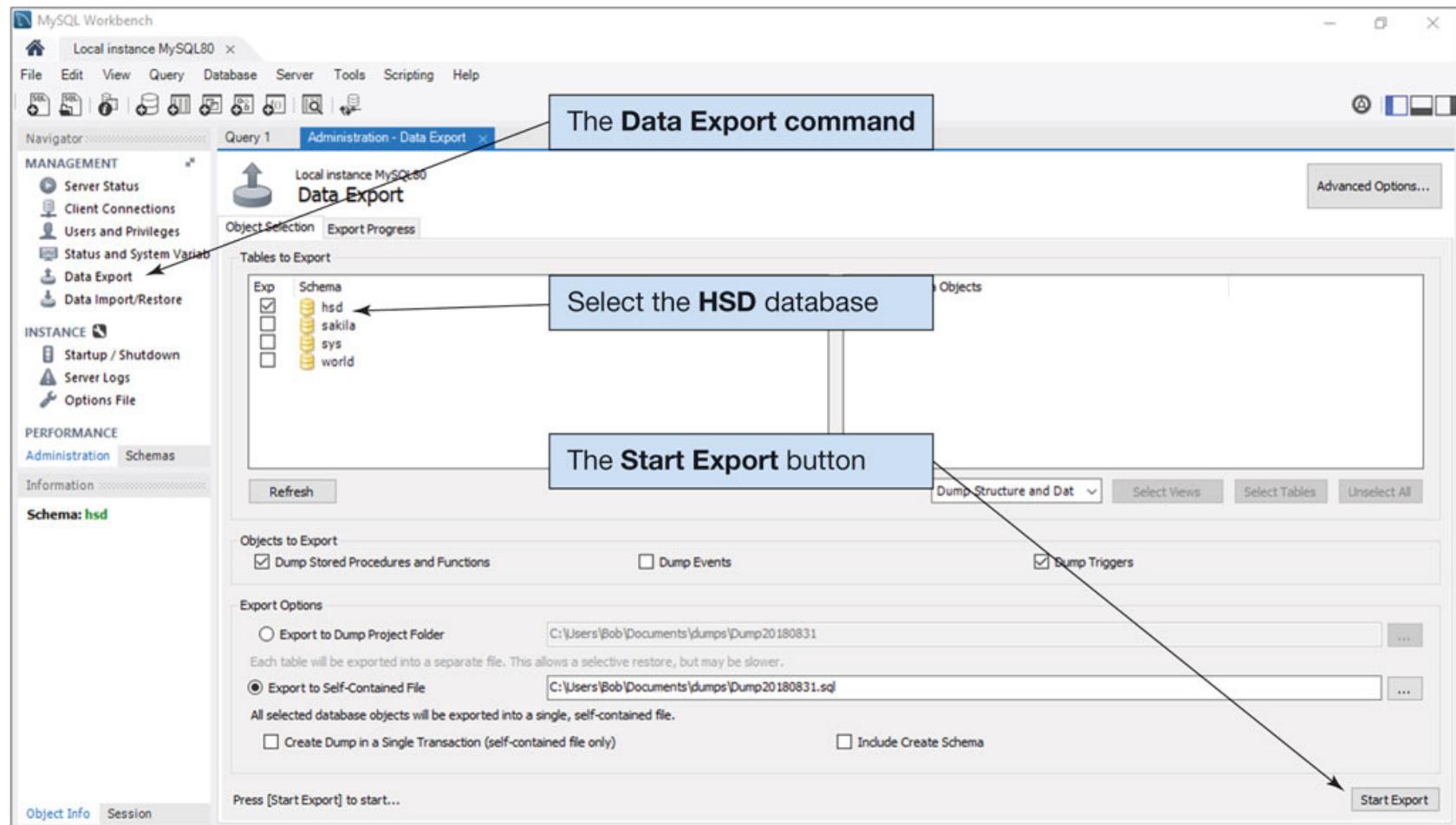
**TRL\_ID** = Transaction log record ID

**PTR** = Pointer to a transaction log record ID

**TRX\_NUM** = Transaction number

(Note: The transaction number is automatically assigned by the DBMS.)

# Figure 6.25 Backing Up the HSD Database



Oracle MySQL Community Server 8.0, Oracle Corporation

# **Additional DBA Responsibilities (1 of 2)**

## **Know basic administrative and managerial DBA functions**

- The DBA needs to ensure that a system exists to gather and record user-reported errors and other problems:
  - a method must be devised to prioritize those errors and problems and to ensure that they are corrected accordingly
- The DBA needs to create and manage a process for controlling the database configuration:
  - procedures for recording change requests
  - conducting user and developer reviews of such requests
  - creating projects and tasks

# Additional DBA Responsibilities (2 of 2)

## Know basic administrative and managerial DBA functions

- The DBA is responsible for ensuring that appropriate documentation is maintained:
  - database structure
  - concurrency control
  - security and backup and recovery
  - applications used
- Every organization should have a **Service Level Agreement** with the cloud provider that covers backups, application response time, and error reporting.

# Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

# Database Concepts

Ninth Edition



## Chapter 7

Data Warehouses, Business  
Intelligence Systems, and Big Data

# Learning Objectives

- Learn the basic concepts of data warehouses and data marts
- Learn the basic concepts of dimensional databases
- Learn the basic concepts of business intelligence (BI) systems
- Learn the basic concepts of online analytical processing (OLAP)
- Learn the basic concepts of virtualization and virtual machines
- Learn the basic concepts of cloud computing
- Learn the basic concepts of Big Data, structured storage, and the MapReduce process
- Understand the limitations and tradeoffs of replicated, partitioned stores as indicated by the CAP (Consistency, Availability, Partition Tolerance) theorem
- Learn the basic concepts of JavaScript Object Notation (JSON) as a way of structuring nonrelational data

# Big Data

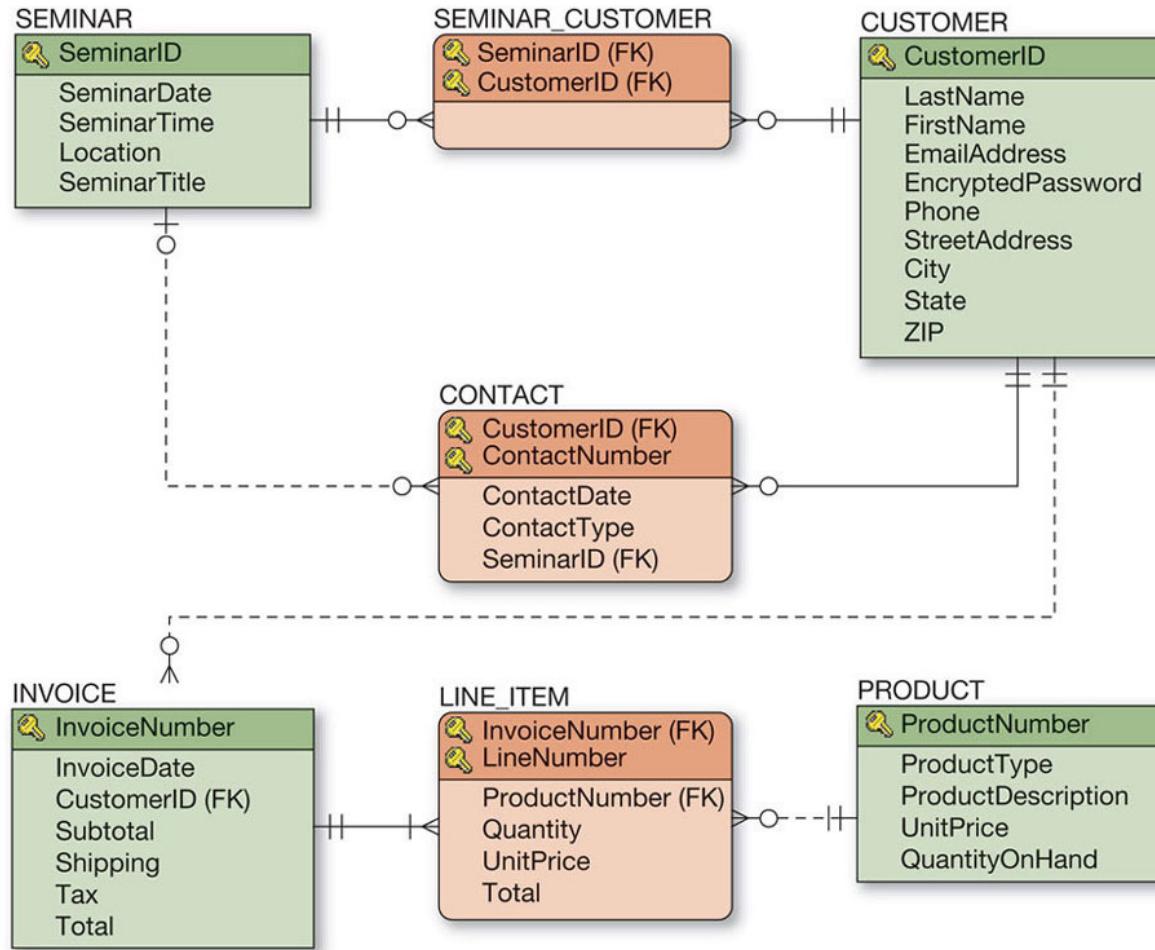
## Learn the basic concepts of data warehouses and data marts

- **Big Data** is the current term for the enormous datasets generated by applications such as search tools like Google and Bing
- Web 2.0 social networks include:
  - Facebook
  - LinkedIn
  - Twitter

# Figure 7.1 Storage Capacity Terms

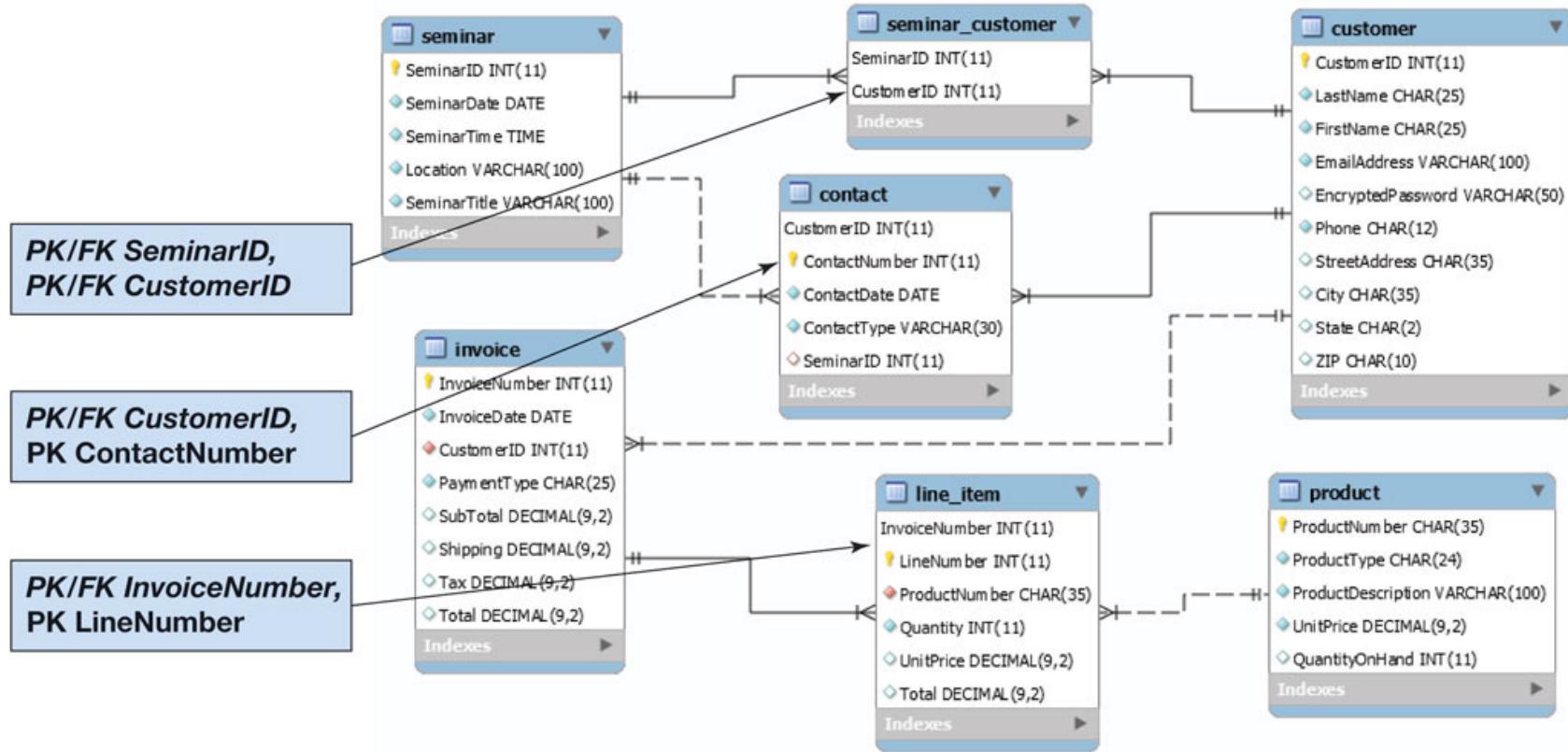
Name	Symbol	Approximate Value for Reference	Actual Value
Byte			8 bits [Store one character]
Kilobyte	KB	About $10^3$	$2^{10} = 1,024$ bytes
Megabyte	MB	About $10^6$	$2^{20} = 1,024$ KB
Gigabyte	GB	About $10^9$	$2^{30} = 1,024$ MB
Terabyte	TB	About $10^{12}$	$2^{40} = 1,024$ GB
Petabyte	PB	About $10^{15}$	$2^{50} = 1,024$ TB
Exabyte	EB	About $10^{18}$	$2^{60} = 1,024$ PB
Zettabyte	ZB	About $10^{21}$	$2^{70} = 1,024$ EB
Yottabyte	YB	About $10^{24}$	$2^{80} = 1,024$ ZB

# Figure 5.27 Heather Sweeney Designs: Database Design



Note: From Chapter 5 showing the final database design for Heather Sweeney

# Figure 7.2 The HSD Database Diagram



Oracle MySQL Community Server 8.0, Oracle Corporation

# Business Intelligence Systems

## Learn the basic concepts of business intelligence (BI) systems

- **Business intelligence (BI)** systems are information systems that:
  - assist managers and other professionals in the analysis of current and past activities and in the prediction of future events
  - do not support operational activities, such as the recording and processing of orders
    - these are supported by transaction processing systems
  - support management assessment, analysis, planning and control
- BI systems fall into two broad categories:
  - **reporting systems** that sort, filter, group, and make elementary calculations on operational data
  - **data mining applications** that perform sophisticated analyses on data; analyses that usually involve complex statistical and mathematical processing

# Figure 7.4 Characteristics of Business Intelligence Applications

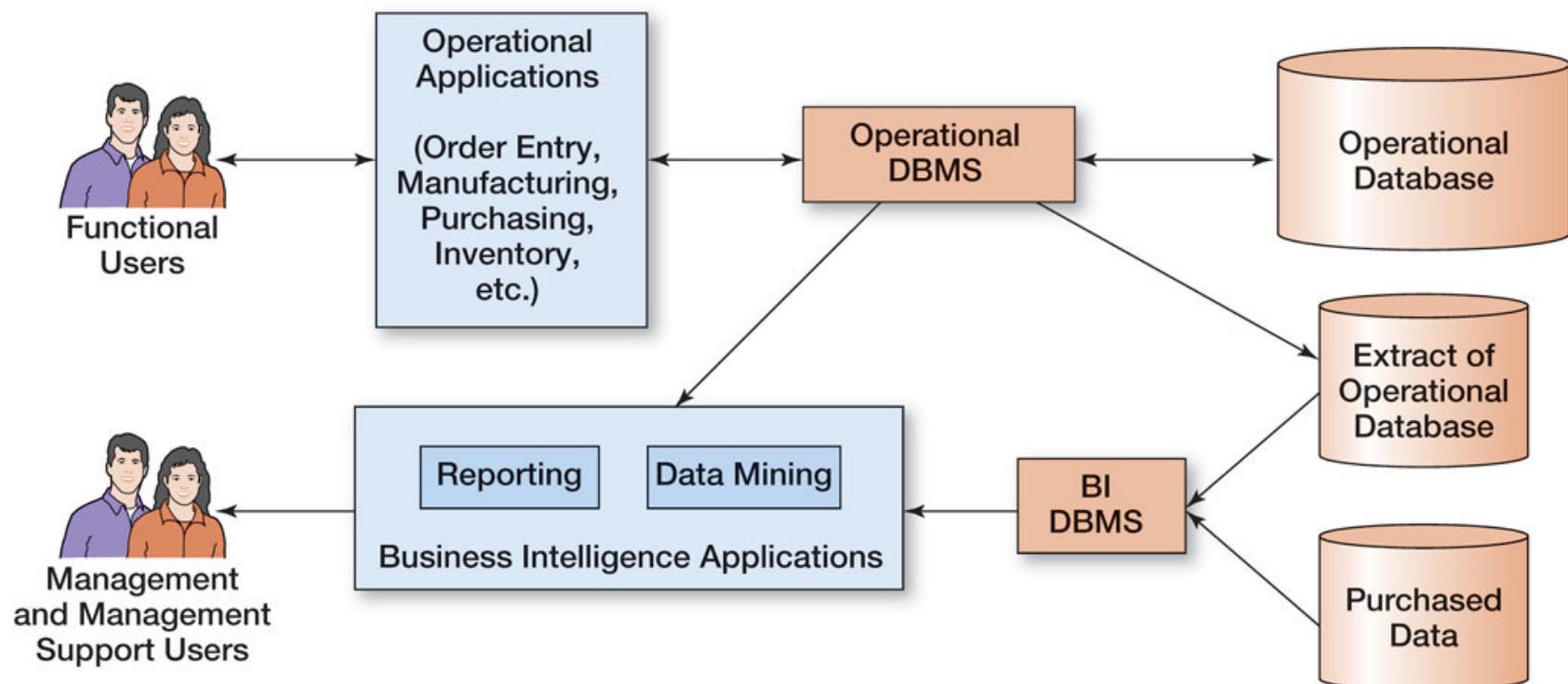
- Reporting
  - Filter, sort, group, and make simple calculations
  - Summarize current status
  - Compare current status to past or predicted status
  - Classify entities (customers, products, employees, etc.)
  - Report delivery crucial
- Data Mining
  - Often employ sophisticated statistical and mathematical techniques
  - Used for:
    - What-if analyses
    - Predictions
    - Decisions
  - Results often incorporated into some other report or system

# BI Terms

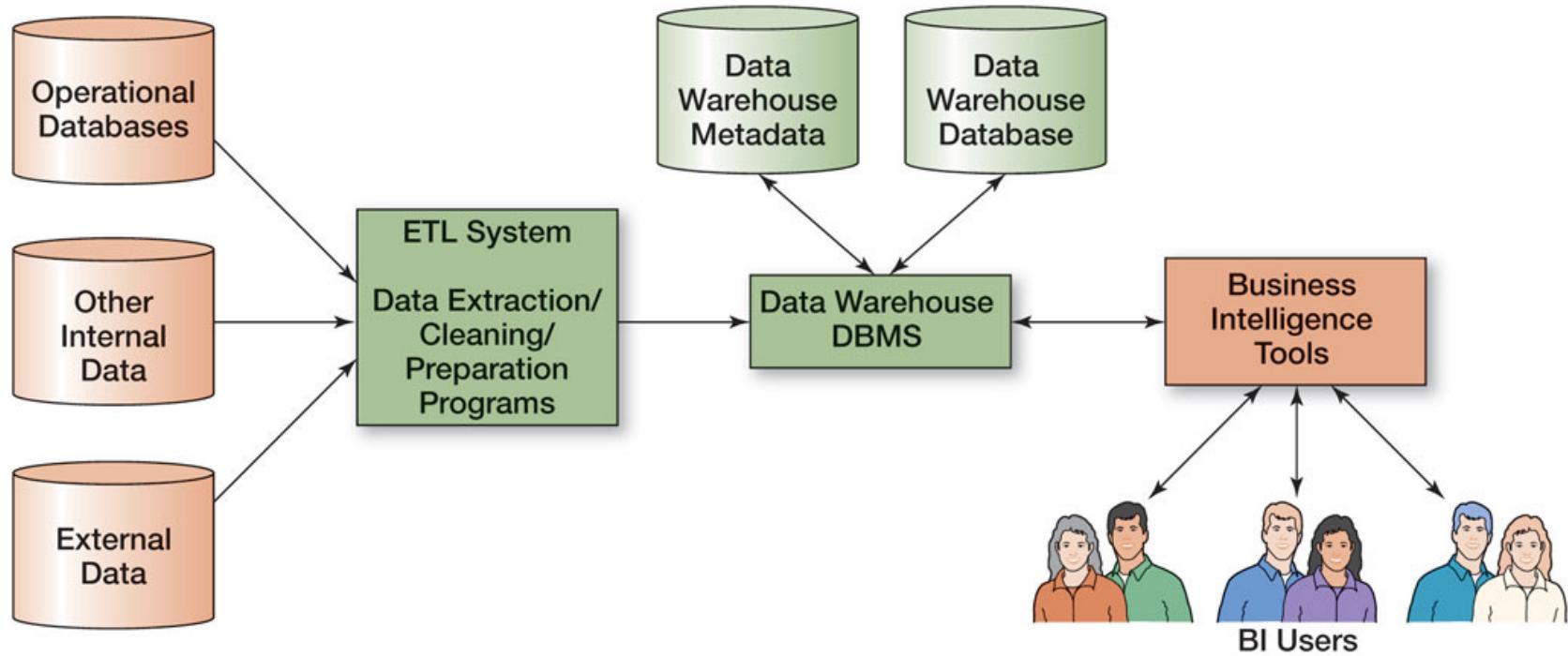
## Learn the basic concepts of business intelligence (BI) systems

- **Online analytical processing (OLAP)** is the processing of non-operational data used in data mining and other applications
- **Data lake** is a repository that includes all data relevant to a business (re: photos, documents, other files of any type)
- A **data warehouse** is a database system that has data, programs, and personnel that specialize in the preparation of data for BI processing.

# Figure 7.3 The Relationship Between Operational and BI Applications



# Figure 7.5 Components of a Data Warehouse



# Problems with Operational Data

## Learn the basic concepts of business intelligence (BI) systems

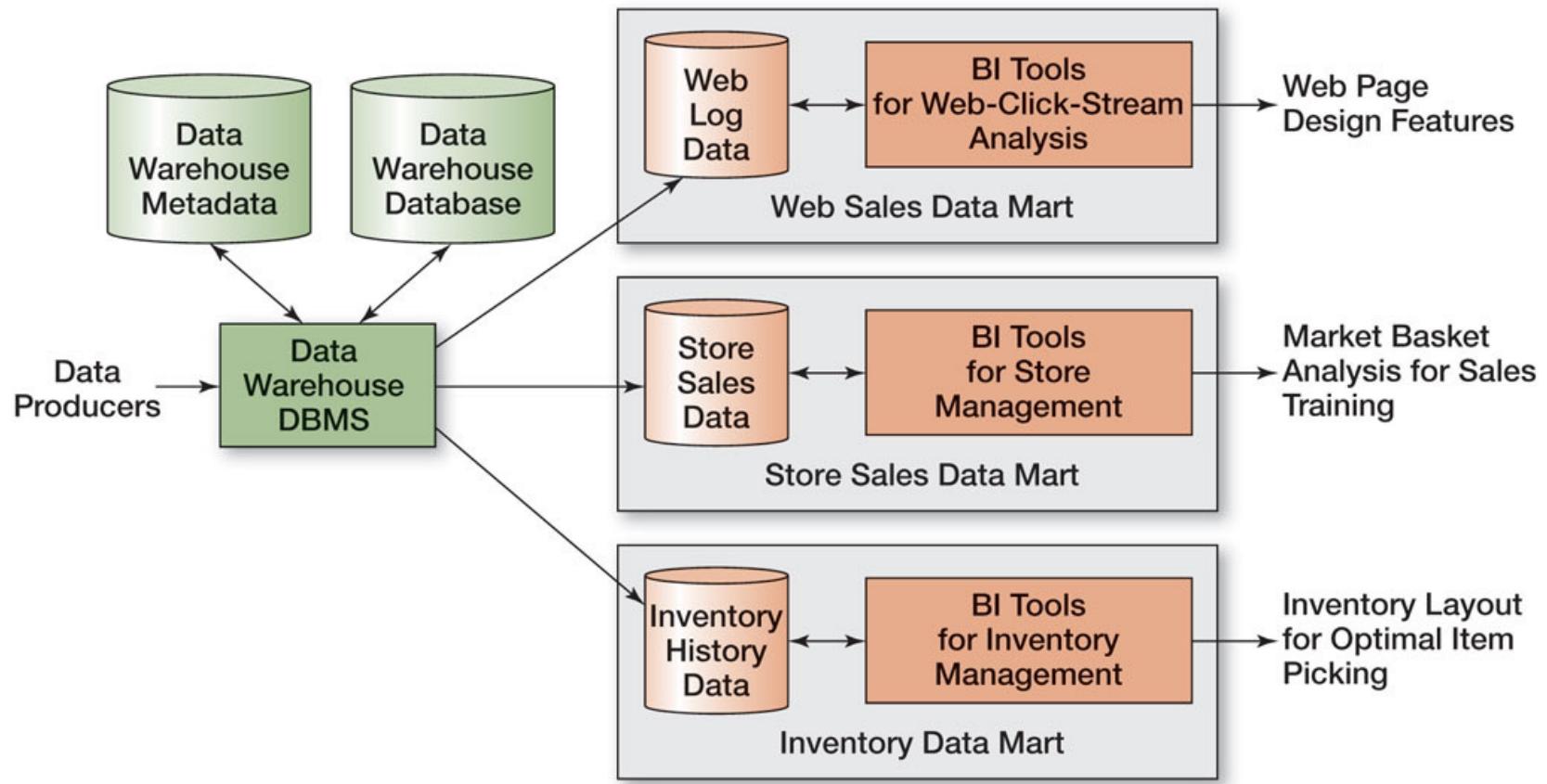
- “Dirty data,” examples include:
  - “G” for gender
  - “213” for age
- Missing values
- Inconsistent data
  - data that has changed (ex: customer’s phone number)
- Nonintegrated data (data from multiple sources)
- Incorrect format (ex: too many or not enough digits)
- Too much data (ex: an excess number of columns)

# ETL Data Transformation

## Learn the basic concepts of business intelligence (BI) systems

- Data may need to be transformed for use in a data warehouse.  
For example:
  - {CountryCode → CountryName}
  - “US” → “United States”
  - Email address to Email domain
    - joe@somewhere.com → “somewhere.com”

# Figure 7.6 Data Warehouses and Data Marts



# Data Warehouses Versus Data Marts

## Learn the basic concepts of data warehouses and data marts

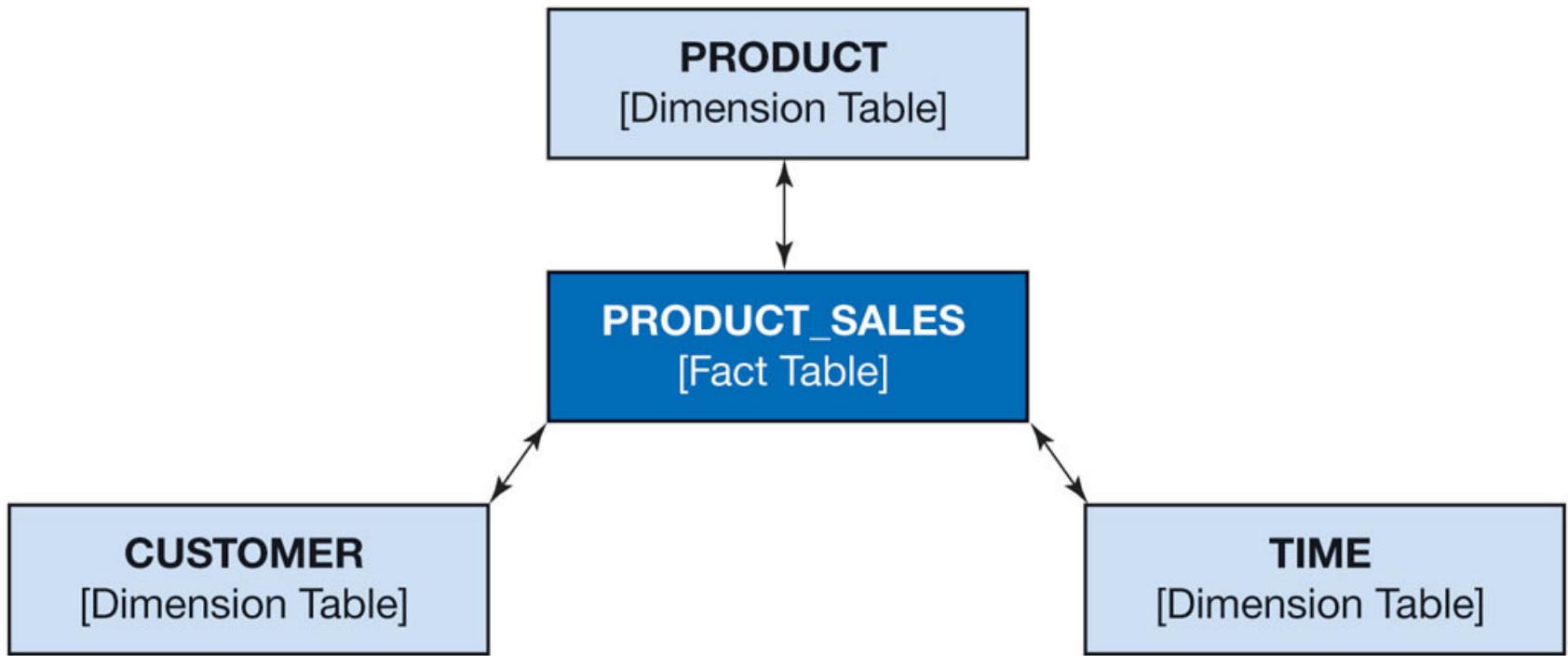
- A **data mart** is a collection of data that is smaller than the data warehouse that addresses a specific component
- An **enterprise data warehouse (EDW) architecture** is when a data mart is combined with the data warehouse architecture.

# Figure 7.7 Characteristics of Operational and Dimensional Databases

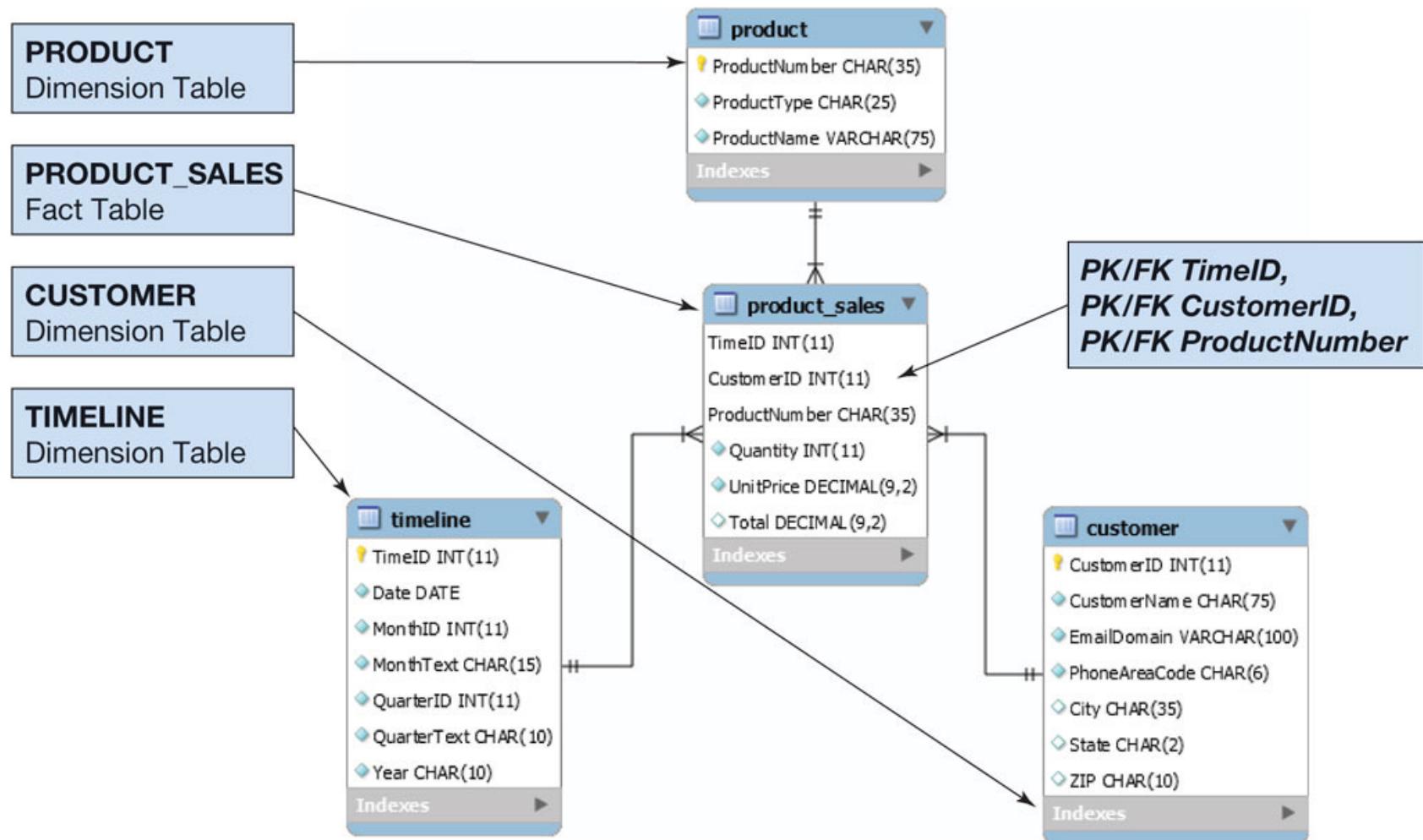
- A **dimensional database** is a design used for efficient analysis and efficient queries for data warehouses.
- Uses a star-schema with a fact table at the center (fully normalized) and a dimensional table radiating out from the center (may be non-normalized)
- May use slowly changing dimensions to track data such as a date or time dimension.

Operational Database	Dimensional Database
Used for structured transaction data processing	Used for unstructured analytical data processing
Current data are used	Current and historical data are used
Data are inserted, updated, and deleted by users	Data are loaded and updated systematically, not by users

# Figure 7.8 A Star Schema



# Figure 7.9 The HSD–DW Star Schema



Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 7.10 The HSD–DW SQL Create Table Statements

```
CREATE TABLE TIMELINE(
    TimeID      Int          NOT NULL,
    Date        Date         NOT NULL,
    MonthID    Int          NOT NULL,
    MonthText   Char(15)    NOT NULL,
    QuarterID  Int          NOT NULL,
    QuarterText Char(10)    NOT NULL,
    Year        Char(10)    NOT NULL,
    CONSTRAINT  TIMELINE_PK PRIMARY KEY(TimeID)
);

CREATE TABLE CUSTOMER(
    CustomerID  Int          NOT NULL,
    CustomerName Char(75)    NOT NULL,
    EmailDomain  VarChar(100) NOT NULL,
    PhoneAreaCode Char(6)     NOT NULL,
    City          Char(35)    NULL,
    State         Char(2)     NULL,
    ZIP           Char(10)    NULL,
    CONSTRAINT   CUSTOMER_PK PRIMARY KEY(CustomerID)
);

CREATE TABLE PRODUCT(
    ProductNumber Char(35)    NOT NULL,
    ProductType   Char(25)    NOT NULL,
    ProductName   VarChar(75) NOT NULL,
    CONSTRAINT   PRODUCT_PK PRIMARY KEY(ProductNumber)
);

CREATE TABLE PRODUCT_SALES(
    TimeID      Int          NOT NULL,
    CustomerID  Int          NOT NULL,
    ProductNumber Char(35)    NOT NULL,
    Quantity     Int          NOT NULL,
    UnitPrice    Numeric(9,2) NOT NULL,
    Total        Numeric(9,2) NULL,
    CONSTRAINT   PRODUCT_SALES_PK PRIMARY KEY (TimeID, CustomerID, ProductNumber),
    CONSTRAINT   PS_TIMELINE_FK FOREIGN KEY(TimeID)
                  REFERENCES TIMELINE(TimeID)
                  ON UPDATE NO ACTION
                  ON DELETE NO ACTION,
    CONSTRAINT   PS_CUSTOMER_FK FOREIGN KEY(CustomerID)
                  REFERENCES CUSTOMER(CustomerID)
                  ON UPDATE NO ACTION
                  ON DELETE NO ACTION,
    CONSTRAINT   PS_PRODUCT_FK FOREIGN KEY(ProductNumber)
                  REFERENCES PRODUCT(ProductNumber)
                  ON UPDATE NO ACTION
                  ON DELETE NO ACTION
);
;
```

# Figure 7.11 The HSD–DW Table Data

	TimeID	Date	MonthID	MonthText	QuarterID	QuarterText	Year
▶	43388	2018-10-15	10	October	3	Qtr3	2018
	43398	2018-10-25	10	October	3	Qtr3	2018
	43454	2018-12-20	12	December	3	Qtr3	2018
	43549	2019-03-25	3	March	1	Qtr1	2019
	43551	2019-03-27	3	March	1	Qtr1	2019
	43555	2019-03-31	3	March	1	Qtr1	2019
	43558	2019-04-03	4	April	2	Qtr2	2019
	43563	2019-04-08	4	April	2	Qtr2	2019
	43578	2019-04-23	4	April	2	Qtr2	2019
	43592	2019-05-07	5	May	2	Qtr2	2019
	43606	2019-05-21	5	May	2	Qtr2	2019
	43621	2019-06-05	6	June	2	Qtr2	2019

	CustomerID	CustomerName	EmailDomain	PhoneAreaCode	City	State	ZIP
▶	1	Jacobs, Nancy	somewhere.com	817	Fort Worth	TX	76110
	2	Jacobs, Chantel	somewhere.com	817	Fort Worth	TX	76112
	3	Able, Ralph	somewhere.com	210	San Antonio	TX	78214
	4	Baker, Susan	elsewhere.com	210	San Antonio	TX	78216
	5	Eagleton, Sam	elsewhere.com	210	San Antonio	TX	78218
	6	Foxtrot, Kathy	somewhere.com	972	Dallas	TX	75220
	7	George, Sally	somewhere.com	972	Dallas	TX	75223
	8	Hullett, Shawn	elsewhere.com	972	Dallas	TX	75224
	9	Pearson, Bobbi	elsewhere.com	512	Austin	TX	78710
	10	Ranger, Terry	somewhere.com	512	Austin	TX	78712
	11	Tyler, Jenny	somewhere.com	972	Dallas	TX	75225
	12	Wayne, Joan	elsewhere.com	817	Fort Worth	TX	76115

	ProductNumber	ProductType	ProductName
▶	BK001	Book	Kitchen Remodeling Basics For Everyone
	BK002	Book	Advanced Kitchen Remodeling For Everyone
	BK003	Book	Kitchen Remodeling Dallas Style For Everyone
	VB001	Video Companion	Kitchen Remodeling Basics Video Companion
	VB002	Video Companion	Advanced Kitchen Remodeling Video Companion
	VB003	Video Companion	Kitchen Remodeling Dallas Style Video Companion
	VK001	Video	Kitchen Remodeling Basics
	VK002	Video	Advanced Kitchen Remodeling
	VK003	Video	Kitchen Remodeling Dallas Style
	VK004	Video	Heather Sweeney Seminar Live in Dallas on 25-OCT-17

	TimeID	CustomerID	ProductNumber	Quantity	UnitPrice	Total
▶	43388	3	VB001	1	7.99	7.99
	43388	3	VK001	1	14.95	14.95
	43398	4	BK001	1	24.95	24.95
	43398	4	VB001	1	7.99	7.99
	43398	4	VK001	1	14.95	14.95
	43454	7	VK004	1	24.95	24.95
	43454	7	BK002	1	24.95	24.95
	43454	7	VK002	1	14.95	14.95
	43454	7	VK004	1	24.95	24.95
	43551	6	BK002	1	24.95	24.95
	43551	6	VB003	1	9.99	9.99
	43551	6	VK002	1	14.95	14.95
	43551	6	VK003	1	19.95	19.95
	43551	6	VK004	1	24.95	24.95
	43551	7	BK001	1	24.95	24.95
	43551	7	VB004	1	7.99	7.99
	43551	7	VK001	1	14.95	14.95
	43558	11	VB003	2	9.99	19.98
	43558	11	VK003	2	19.95	39.90
	43558	11	VK004	2	24.95	49.90
	43563	1	BK001	1	24.95	24.95
	43563	1	VB001	1	7.99	7.99
	43563	1	VK001	1	14.95	14.95
	43563	5	BK001	1	24.95	24.95
	43563	5	VB001	1	7.99	7.99
	43563	5	VK001	1	14.95	14.95
	43578	3	BK001	1	24.95	24.95
	43592	9	VB002	1	7.99	7.99
	43592	9	VK002	1	14.95	14.95
	43606	8	VB003	1	9.99	9.99
	43606	8	VK003	1	19.95	19.95
	43606	8	VK004	1	24.95	24.95
	43621	3	BK002	1	24.95	24.95
	43621	3	VB001	1	7.99	7.99
	43621	3	VB002	2	7.99	15.98
	43621	3	VK001	1	14.95	14.95
	43621	3	VK002	2	14.95	29.90
	43621	11	VB002	2	7.99	15.98
	43621	11	VK002	2	14.95	29.90
	43621	12	BK002	1	24.95	24.95
	43621	12	VB003	1	9.99	9.99
	43621	12	VK002	1	14.95	14.95
	43621	12	VK003	1	19.95	19.95
	43621	12	VK004	1	24.95	24.95

Oracle MySQL Community Server 8.0, Oracle Corporation

# A Query to Summarize Products Sold by Customer and Product

## Learn the basic concepts of business intelligence (BI) systems

- The following SQL code is used to summarize products sold by Customer and Product

```
/* *** SQL-QUERY-CH07-02 *** */  
SELECT      C.CustomerID, CustomerName, C.City, P.ProductNumber,  
            P.ProductName, T.Year, T.QuarterText, SUM(PS.Quantity)  
            AS TotalQuantity  
FROM        CUSTOMER C, PRODUCT_SALES PS, PRODUCT P, TIMELINE T  
WHERE       C.CustomerID = PS.CustomerID  
            AND P.ProductNumber = PS.ProductNumber AND T.TimeID =  
                    PS.TimeID  
GROUP BY    C.CustomerID, C.CustomerName, C.City, P.ProductNumber,  
            P.ProductName, T.QuarterText, T.Year  
ORDER BY    C.CustomerName, T.Year, T.QuarterText;
```

## Figure 7.12 The HSD–DW Query Results: Summarize Product Units Sold by Customer and Product

	CustomerID	CustomerName	ProductNumber	ProductName	TotalQuantity
▶	1	Jacobs, Nancy	BK001	Kitchen Remodeling Basics For Everyone	1
	1	Jacobs, Nancy	VB001	Kitchen Remodeling Basics Video Companion	1
	1	Jacobs, Nancy	VK001	Kitchen Remodeling Basics	1
	3	Able, Ralph	BK001	Kitchen Remodeling Basics For Everyone	1
	3	Able, Ralph	BK002	Advanced Kitchen Remodeling For Everyone	1
	3	Able, Ralph	VB001	Kitchen Remodeling Basics Video Companion	2
	3	Able, Ralph	VB002	Advanced Kitchen Remodeling Video Companion	2
	3	Able, Ralph	VK001	Kitchen Remodeling Basics	2
	3	Able, Ralph	VK002	Advanced Kitchen Remodeling	2
	4	Baker, Susan	BK001	Kitchen Remodeling Basics For Everyone	1
	4	Baker, Susan	BK002	Advanced Kitchen Remodeling For Everyone	1
	4	Baker, Susan	VB001	Kitchen Remodeling Basics Video Companion	1
	4	Baker, Susan	VK001	Kitchen Remodeling Basics	1
	4	Baker, Susan	VK002	Advanced Kitchen Remodeling	1
	4	Baker, Susan	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	1
	5	Eagleton, Sam	BK001	Kitchen Remodeling Basics For Everyone	1
	5	Eagleton, Sam	VB001	Kitchen Remodeling Basics Video Companion	1
	5	Eagleton, Sam	VK001	Kitchen Remodeling Basics	1
	6	Foxtrot, Kathy	BK002	Advanced Kitchen Remodeling For Everyone	1
	6	Foxtrot, Kathy	VB003	Kitchen Remodeling Dallas Style Video Companion	1
	6	Foxtrot, Kathy	VK002	Advanced Kitchen Remodeling	1
	6	Foxtrot, Kathy	VK003	Kitchen Remodeling Dallas Style	1
	6	Foxtrot, Kathy	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	1
	7	George, Sally	BK001	Kitchen Remodeling Basics For Everyone	1
	7	George, Sally	BK002	Advanced Kitchen Remodeling For Everyone	1
	7	George, Sally	VK003	Kitchen Remodeling Dallas Style	1
	7	George, Sally	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2
	8	Hullett, Shawn	VB003	Kitchen Remodeling Dallas Style Video Companion	1
	8	Hullett, Shawn	VK003	Kitchen Remodeling Dallas Style	1
	8	Hullett, Shawn	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	1
	9	Pearson, Bobbi	BK001	Kitchen Remodeling Basics For Everyone	1
	9	Pearson, Bobbi	VB001	Kitchen Remodeling Basics Video Companion	1
	9	Pearson, Bobbi	VB002	Advanced Kitchen Remodeling Video Companion	1
	9	Pearson, Bobbi	VK001	Kitchen Remodeling Basics	1
	9	Pearson, Bobbi	VK002	Advanced Kitchen Remodeling	1
	11	Tyler, Jenny	VB002	Advanced Kitchen Remodeling Video Companion	2
	11	Tyler, Jenny	VB003	Kitchen Remodeling Dallas Style Video Companion	2
	11	Tyler, Jenny	VK002	Advanced Kitchen Remodeling	2
	11	Tyler, Jenny	VK003	Kitchen Remodeling Dallas Style	2
	11	Tyler, Jenny	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2
	12	Wayne, Joan	BK002	Advanced Kitchen Remodeling For Everyone	1
	12	Wayne, Joan	VB003	Kitchen Remodeling Dallas Style Video Companion	1
	12	Wayne, Joan	VK002	Advanced Kitchen Remodeling	1
	12	Wayne, Joan	VK003	Kitchen Remodeling Dallas Style	1
	12	Wayne, Joan	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	1

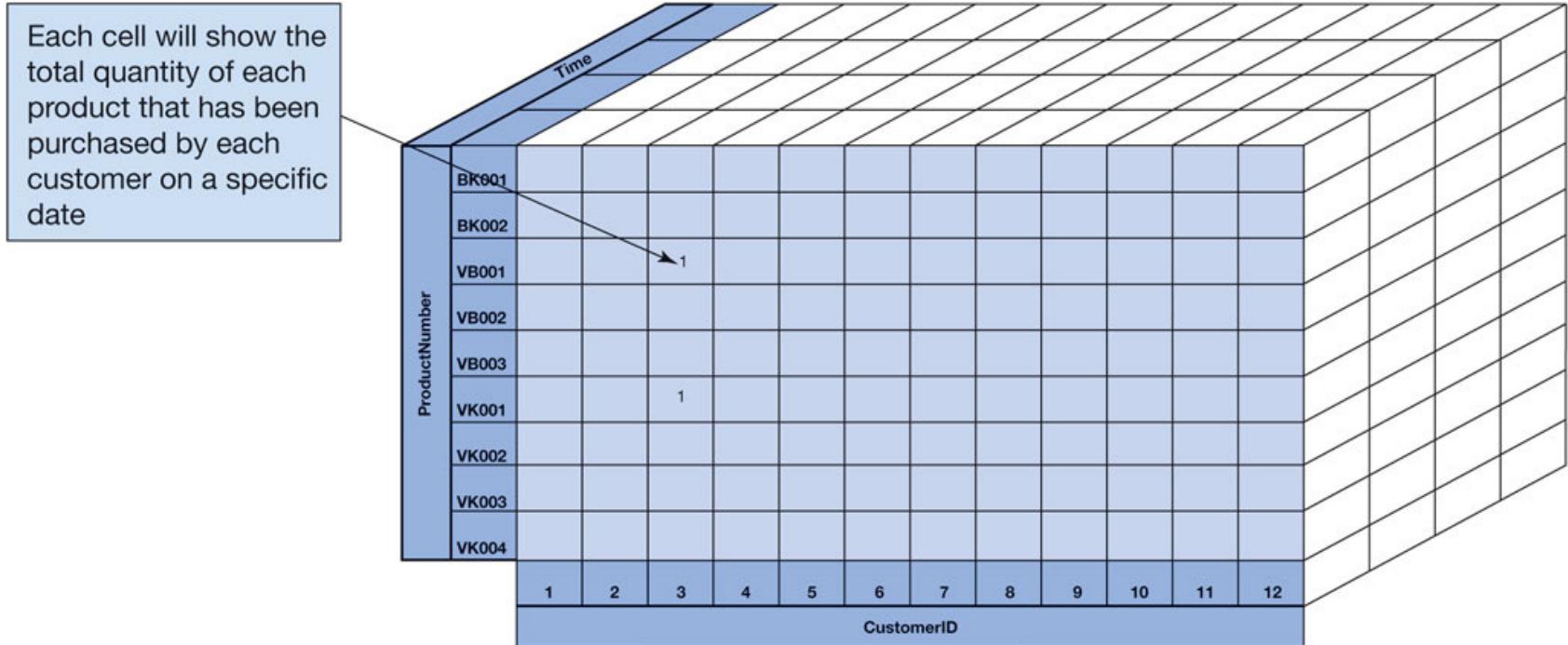
Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 7.13 The Two-Dimensional ProductNumber-CustomerID Matrix

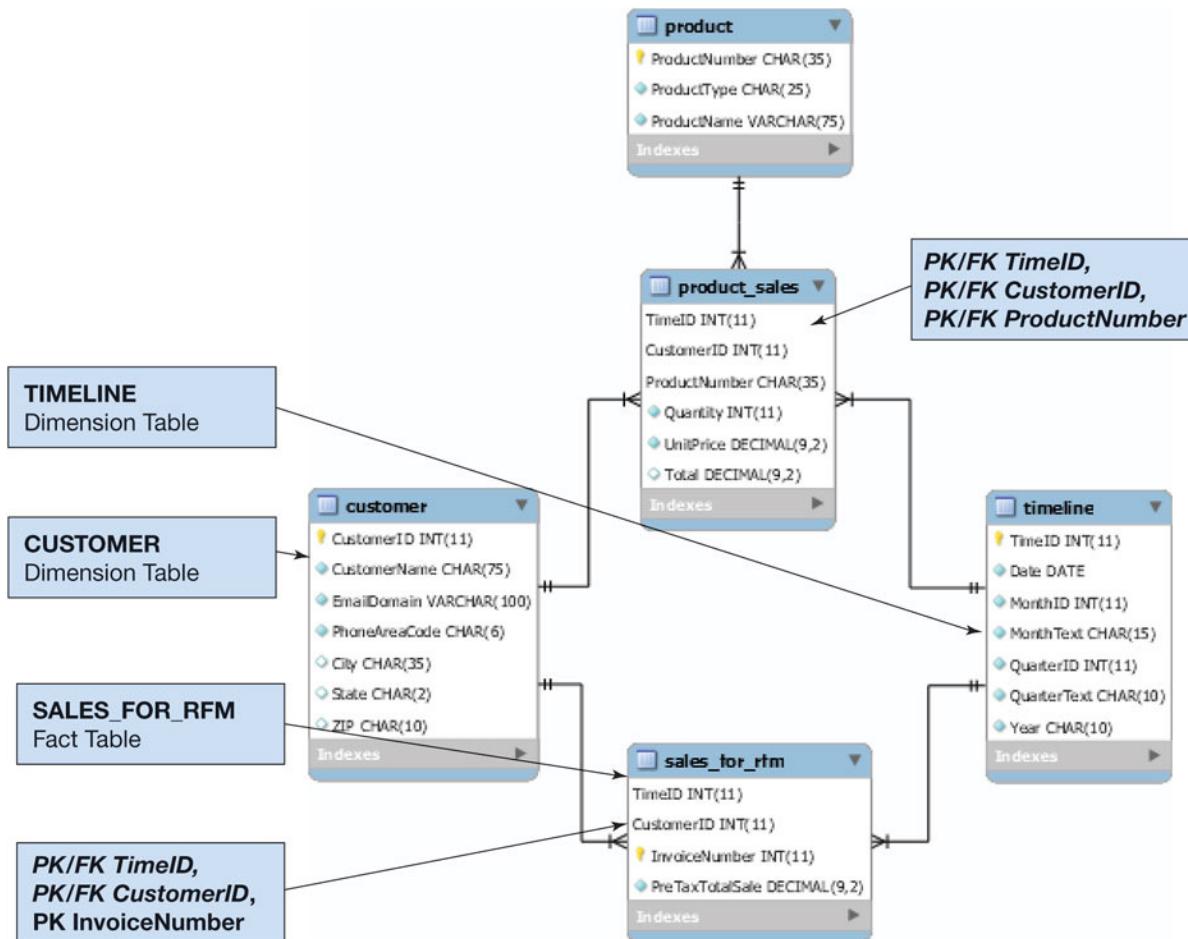
Each cell shows the total quantity of each product that has been purchased by each customer

ProductNumber	CustomerID											
	1	2	3	4	5	6	7	8	9	10	11	12
BK001	1		1	1			1		1			
BK002			1	1		1	1					1
VB001	1		2	1	1				1			
VB002			2						1		2	
VB003						1		1			2	1
VK001	1		2	1	1				1			
VK002			2	1		1			1		2	1
VK003						1	1	1			2	1
VK004				1		1	2	1			2	1

# Figure 7.14 The Three-Dimensional Time-ProductNumber-CustomerID Cube



# Figure 7.15 The HSD–DW Star Schema Extended for RFM Analysis



Oracle MySQL Community Server 8.0, Oracle Corporation

# Online Analytical Processing (OLAP)

## Learn the basic concepts of online analytical processing (OLAP)

- **Online Analytical Processing (OLAP)** is a technique for dynamically examining database data:
  - OLAP uses arithmetic functions such as Sum and Average
  - OLAP uses the dimensional database model
- OLAP systems produce **OLAP reports** (also called OLAP cube)
  - An OLAP report uses inputs called *dimensions*
  - An OLAP report calculates outputs called *measures*
  - *Excel PivotTables* can be used to create OLAP reports

# SQL Query for OLAP Data

## Learn the basic concepts of online analytical processing (OLAP)

```
/* *** SQL-QUERY-CH07-02 *** */
SELECT      C.CustomerID, CustomerName, C.City, P.ProductNumber,
            P.ProductName, T.Year, T.QuarterText, SUM(PS.Quantity)
            AS TotalQuantity
FROM        CUSTOMER C, PRODUCT_SALES PS, PRODUCT P, TIMELINE T
WHERE       C.CustomerID = PS.CustomerID
AND         P.ProductNumber = PS.ProductNumber AND T.TimeID =
            PS.TimeID
GROUP BY    C.CustomerID, C.CustomerName, C.City, P.ProductNumber,
            P.ProductName, T.QuarterText, T.Year
ORDER BY    C.CustomerName, T.Year, T.QuarterText;
```

# SQL View for OLAP Data

## Learn the basic concepts of online analytical processing (OLAP)

```
/* *** SQL-CREATE-VIEW-CH07-01 *** */
CREATE VIEW HSDDWProductSalesView AS
    SELECT      C.CustomerID, C.CustomerName, C.City,
                P.ProductNumber, P.ProductName, T.Year,
                T.QuarterText, SUM(PS.Quantity) AS TotalQuantity
    FROM        CUSTOMER C, PRODUCT_SALES PS, PRODUCT P, TIMELINE T
    WHERE       C.CustomerID = PS.CustomerID
    AND         P.ProductNumber = PS.ProductNumber
    AND         T.TimeID = PS.TimeID
    GROUP BY   C.CustomerID, C.CustomerName, C.City,
                P.ProductNumber, P.ProductName,
                T.QuarterText, T.Year;
```

# Figure 7.16 The HSD-DW Query for OLAP Results: Time-Product-Customer Cube

	CustomerID	CustomerName	City	ProductNumber	ProductName	Year	QuarterText	TotalQuantity
▶	3	Able, Ralph	San Antonio	VK001	Kitchen Remodeling Basics	2018	Qtr3	1
	3	Able, Ralph	San Antonio	VB001	Kitchen Remodeling Basics Video Companion	2018	Qtr3	1
	3	Able, Ralph	San Antonio	VB001	Kitchen Remodeling Basics Video Companion	2019	Qtr2	1
	3	Able, Ralph	San Antonio	BK002	Advanced Kitchen Remodeling For Everyone	2019	Qtr2	1
	3	Able, Ralph	San Antonio	VK002	Advanced Kitchen Remodeling	2019	Qtr2	2
	3	Able, Ralph	San Antonio	VB002	Advanced Kitchen Remodeling Video Companion	2019	Qtr2	2
	3	Able, Ralph	San Antonio	BK001	Kitchen Remodeling Basics For Everyone	2019	Qtr2	1
	3	Able, Ralph	San Antonio	VK001	Kitchen Remodeling Basics	2019	Qtr2	1
	4	Baker, Susan	San Antonio	BK001	Kitchen Remodeling Basics For Everyone	2018	Qtr3	1
	4	Baker, Susan	San Antonio	VK001	Kitchen Remodeling Basics	2018	Qtr3	1
	4	Baker, Susan	San Antonio	VB001	Kitchen Remodeling Basics Video Companion	2018	Qtr3	1
	4	Baker, Susan	San Antonio	BK002	Advanced Kitchen Remodeling For Everyone	2019	Qtr1	1
	4	Baker, Susan	San Antonio	VB002	Advanced Kitchen Remodeling	2019	Qtr1	1
	4	Baker, Susan	San Antonio	VK002	Advanced Kitchen Remodeling Video Companion	2019	Qtr1	1
	4	Baker, Susan	San Antonio	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr1	1
	5	Eagleton, Sam	San Antonio	VB001	Kitchen Remodeling Basics Video Companion	2019	Qtr2	1
	5	Eagleton, Sam	San Antonio	BK001	Kitchen Remodeling Basics For Everyone	2019	Qtr2	1
	5	Eagleton, Sam	San Antonio	VK001	Kitchen Remodeling Basics	2019	Qtr2	1
	6	Foxtrot, Kathy	Dallas	BK002	Advanced Kitchen Remodeling For Everyone	2019	Qtr1	1
	6	Foxtrot, Kathy	Dallas	VB002	Advanced Kitchen Remodeling	2019	Qtr1	1
	6	Foxtrot, Kathy	Dallas	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr1	1
	6	Foxtrot, Kathy	Dallas	VB003	Kitchen Remodeling Dallas Style Video Companion	2019	Qtr1	1
	6	Foxtrot, Kathy	Dallas	VK003	Kitchen Remodeling Dallas Style	2019	Qtr1	1
	7	George, Sally	Dallas	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2018	Qtr3	1
	7	George, Sally	Dallas	BK002	Advanced Kitchen Remodeling For Everyone	2019	Qtr1	1
	7	George, Sally	Dallas	BK001	Kitchen Remodeling Basics For Everyone	2019	Qtr1	1
	7	George, Sally	Dallas	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr1	1
	7	George, Sally	Dallas	VK003	Kitchen Remodeling Dallas Style	2019	Qtr1	1
	8	Hullett, Shawn	Dallas	VB003	Kitchen Remodeling Dallas Style Video Companion	2019	Qtr2	1
	8	Hullett, Shawn	Dallas	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr2	1
	8	Hullett, Shawn	Dallas	VK003	Kitchen Remodeling Dallas Style	2019	Qtr2	1
	1	Jacobs, Nancy	Fort Worth	VB001	Kitchen Remodeling Basics Video Companion	2019	Qtr2	1
	1	Jacobs, Nancy	Fort Worth	BK001	Kitchen Remodeling Basics For Everyone	2019	Qtr2	1
	1	Jacobs, Nancy	Fort Worth	VK001	Kitchen Remodeling Basics	2019	Qtr2	1
	9	Pearson, Bobbi	Austin	BK001	Kitchen Remodeling Basics For Everyone	2019	Qtr1	1
	9	Pearson, Bobbi	Austin	VK001	Kitchen Remodeling Basics	2019	Qtr1	1
	9	Pearson, Bobbi	Austin	VB001	Kitchen Remodeling Basics Video Companion	2019	Qtr1	1
	9	Pearson, Bobbi	Austin	VK002	Advanced Kitchen Remodeling	2019	Qtr2	1
	9	Pearson, Bobbi	Austin	VB002	Advanced Kitchen Remodeling Video Companion	2019	Qtr2	1
	11	Tyler, Jenny	Dallas	VK003	Kitchen Remodeling Dallas Style	2019	Qtr2	2
	11	Tyler, Jenny	Dallas	VK002	Advanced Kitchen Remodeling	2019	Qtr2	2
	11	Tyler, Jenny	Dallas	VB002	Advanced Kitchen Remodeling Video Companion	2019	Qtr2	2
	11	Tyler, Jenny	Dallas	VB003	Kitchen Remodeling Dallas Style Video Companion	2019	Qtr2	2
	11	Tyler, Jenny	Dallas	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr2	2
	12	Wayne, Joan	Fort Worth	VB003	Kitchen Remodeling Dallas Style Video Companion	2019	Qtr2	1
	12	Wayne, Joan	Fort Worth	VK004	Heather Sweeney Seminar Live in Dallas on 25-OCT-17	2019	Qtr2	1
	12	Wayne, Joan	Fort Worth	VK003	Kitchen Remodeling Dallas Style	2019	Qtr2	1
	12	Wayne, Joan	Fort Worth	BK002	Advanced Kitchen Remodeling For Everyone	2019	Qtr2	1
	12	Wayne, Joan	Fort Worth	VK002	Advanced Kitchen Remodeling	2019	Qtr2	1

Oracle MySQL Community Server 8.0, Oracle Corporation

# Figure 7.17 The HSD-DW OLAP ProductNumber by City Report

The screenshot shows an Excel spreadsheet titled "DBC-e09-HSD-DW-B.xlsx - Excel". The main area displays a PivotTable report titled "Sum of TotalQuantity" with data for various cities (Austin, Dallas, Fort Worth, San Antonio) across different product numbers (BK001 through VK004). The PivotTable Fields pane on the right side lists fields such as CustomerID, CustomerName, City, ProductNumber, Year, QuarterText, and TotalQuantity, with ProductNumber selected as the column field. The report is identified as being in the "HSD-DW PivotTable" worksheet, while the data source is the "HSD-DW Query Results" worksheet.

	BK001	BK002	VB001	VB002	VB003	VK001	VK002	VK003	VK004	Grand Total
Austin	1	1	1	1	1	1	1	1	1	5
Dallas	1	2		2	4		3	5	6	23
Fort Worth	1	1	1		1	1	1	1	1	8
San Antonio	3	2	4	2		4	3		1	19
Grand Total	6	5	6	5	5	6	8	6	8	55

The PivotTable Fields pane—select the report elements to be displayed here

The PivotTable report

The PivotTable is in the HSD-DW PivotTable worksheet

The data table is in the HSD-DW Query Results worksheet

Excel 2019, Windows 10, Microsoft Corporation

# Figure 7.18 The HSD–DW OLAP ProductNumber by City Report: CustomerName and Year Dimensions Added

	A	B	C	D	E	F	G	H	I	J	K
3	Sum of TotalQuantity	Column Labels									
4	Row Labels	BK001	BK002	VB001	VB002	VB003	VK001	VK002	VK003	VK004	Grand Total
5	✉ Austin		1	1	1		1	1			5
6	✉ Dallas		1	2		2	4		3	5	23
7	✉ Fort Worth		1	1	1		1	1	1	1	8
8	✉ San Antonio		3	2	4	2		4	3	1	19
9	✉ Able, Ralph		1	1	2	2		2	2		10
10	2018				1			1			2
11	2019		1	1	1	2		1	2		8
12	✉ Baker, Susan		1	1	1		1	1		1	6
13	✉ Eagleton, Sam		1		1		1				3
14	Grand Total		6	5	6	5	5	6	8	6	55

The City = San Antonio data are also showing customer data

The Customer = Able, Ralph data are also showing year data

Excel 2019, Windows 10, Microsoft Corporation

# Figure 7.19 The HSD–DW OLAP City by ProductNumber, CustomerName, and Year Report

	A	B	C	D	E	F
3	Sum of TotalQuantity	Column Labels				
4	Row Labels	Austin	Dallas	Fort Worth	San Antonio	Grand Total
5	+ BK001		1	1	1	3
6	+ BK002		2	1	2	5
7	+ VB001		1	1	4	6
8	+ Able, Ralph				2	2
9	2018				1	1
10	2019				1	1
11	+ Baker, Susan				1	1
12	+ Eagleton, Sam				1	1
13	+ Jacobs, Nancy			1		1
14	+ Pearson, Bobbi	1				1
15	+ VB002	1	2		2	5
16	+ VB003		4	1		5
17	+ VK001	1		1	4	6
18	+ VK002	1	3	1	3	8
19	+ VK003		5	1		6
20	+ VK004		6	1	1	8
21	Grand Total	5	23	8	19	55

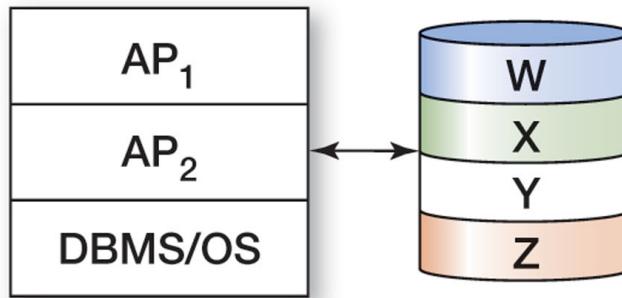
Excel 2019, Windows 10, Microsoft Corporation

# Distributed Database Processing

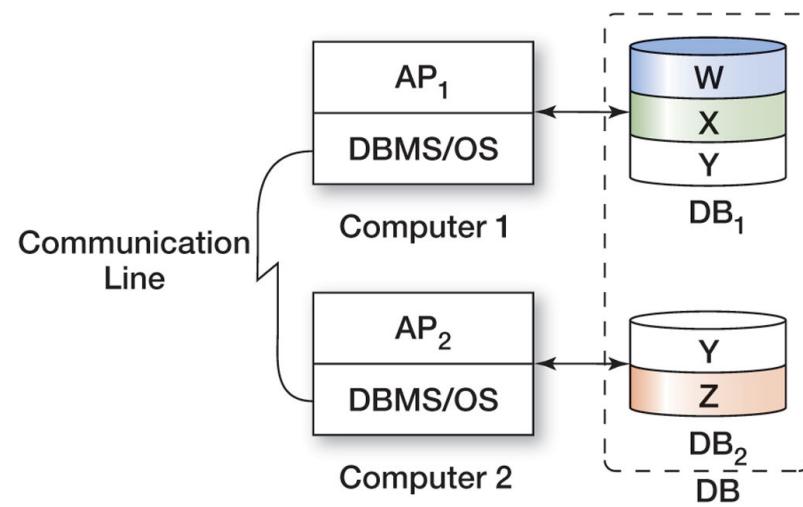
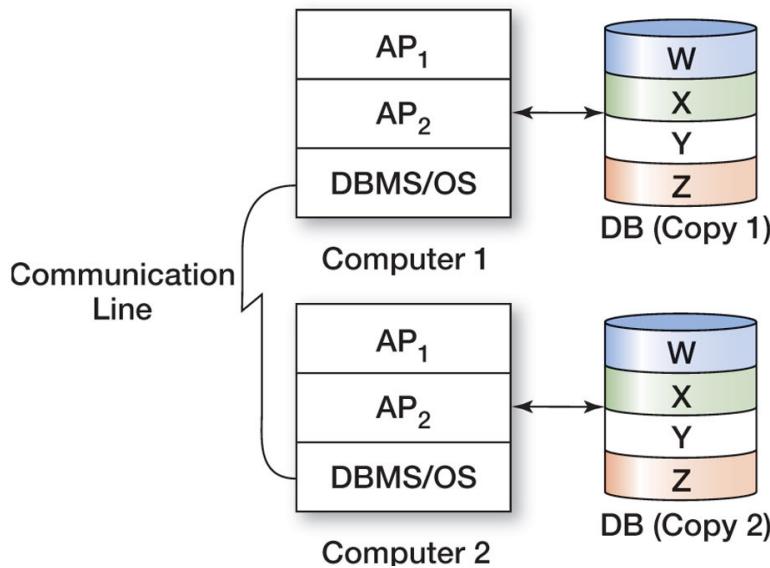
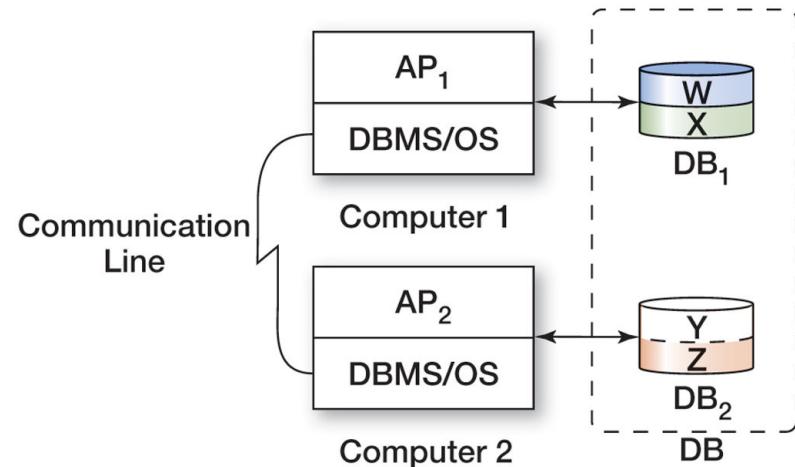
## Learn the basic concepts of virtualization and virtual machines

- A **distributed database** is a database that is stored and processed on more than one computer.
- A **service cluster** is a group of associated servers where the database shared between them is called a distributed database.
- A database is distributed when it is:
  - partitioned
  - replicated
  - both partitioned and replicated
- Databases are distributed for two major reasons – performance and control.

# Figure 7.20 Types of Distributed Databases



Single Processing Computer



# Object-Relational Databases

## Learn the basic concepts of virtualization and virtual machines

- **Object-oriented programming (OOP)** is a methodology for designing and writing computer programs.
- Some object-oriented programming languages are Java, Python, C++, C#, and Visual Basic.NET
- **Objects** are data structures that have both **methods** (programs that perform some task with the object) and **properties** (data items particular to an object).
- **Object persistence** is the permanent storing of values of properties of an object in secondary memory (usually disk)

# OODBMS

## Learn the basic concepts of virtualization and virtual machines

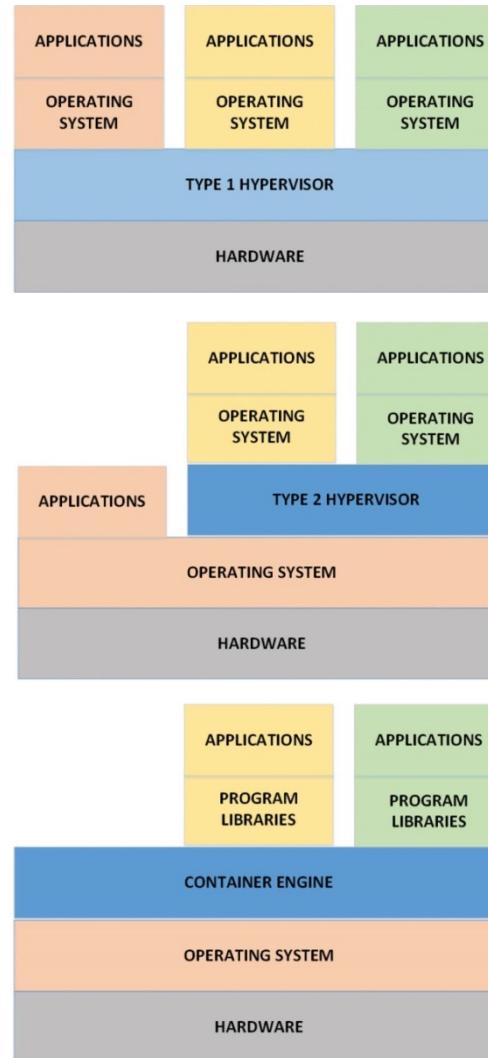
- **Object-oriented DBMSs (OODBMSs)** are special-purpose DBMS products for storing data developed by several vendors in the early 1990s. These never achieved commercial success.
  - too expensive to transfer existing data from traditional legacy databases
  - not cost justifiable

# Virtualization

## Learn the basic concepts of virtualization and virtual machines

- **Virtualization** is using hardware and software to simulate another hardware resource.
- Virtualization is done by having one physical computer host one or more virtual computers, also known as **virtual machines**.
- The **host machine** runs a special program known as a **virtual machine manager** or **hypervisor**.
- There are two ways to implement hypervisors:
  - “bare metal” or type 1 (used in large data centers)
  - “hosted” or type 2 (used by students and others)

# Figure 7.21 Type 1 and Type 2 Hypervisors versus Containers



# Cloud Computing

## Learn the basic concepts of cloud computing

- **Cloud computing** services are ultimately provided by large data centers.
- **Storage area networks (SANs)** have dedicated network paths from servers to disk arrays, where several physical disks are combined together to act as a single, larger disk.
- **Redundant arrays of independent disks (RAID)** can be configured for maximum access speed or for reliability.
- Three basic ways to lease cloud services:
  - **Software as a service (SaaS)**, Ex: Salesforce.com
  - **Platform as a service (PaaS)**, Ex: operating systems, software development tools & system programs provided
  - **Infrastructure as a service (IaaS)**, Ex: Only hardware provided and users manage their own software

# Not Only SQL

## Learn the basic concepts of Big Data, structured storage, and the MapReduce process

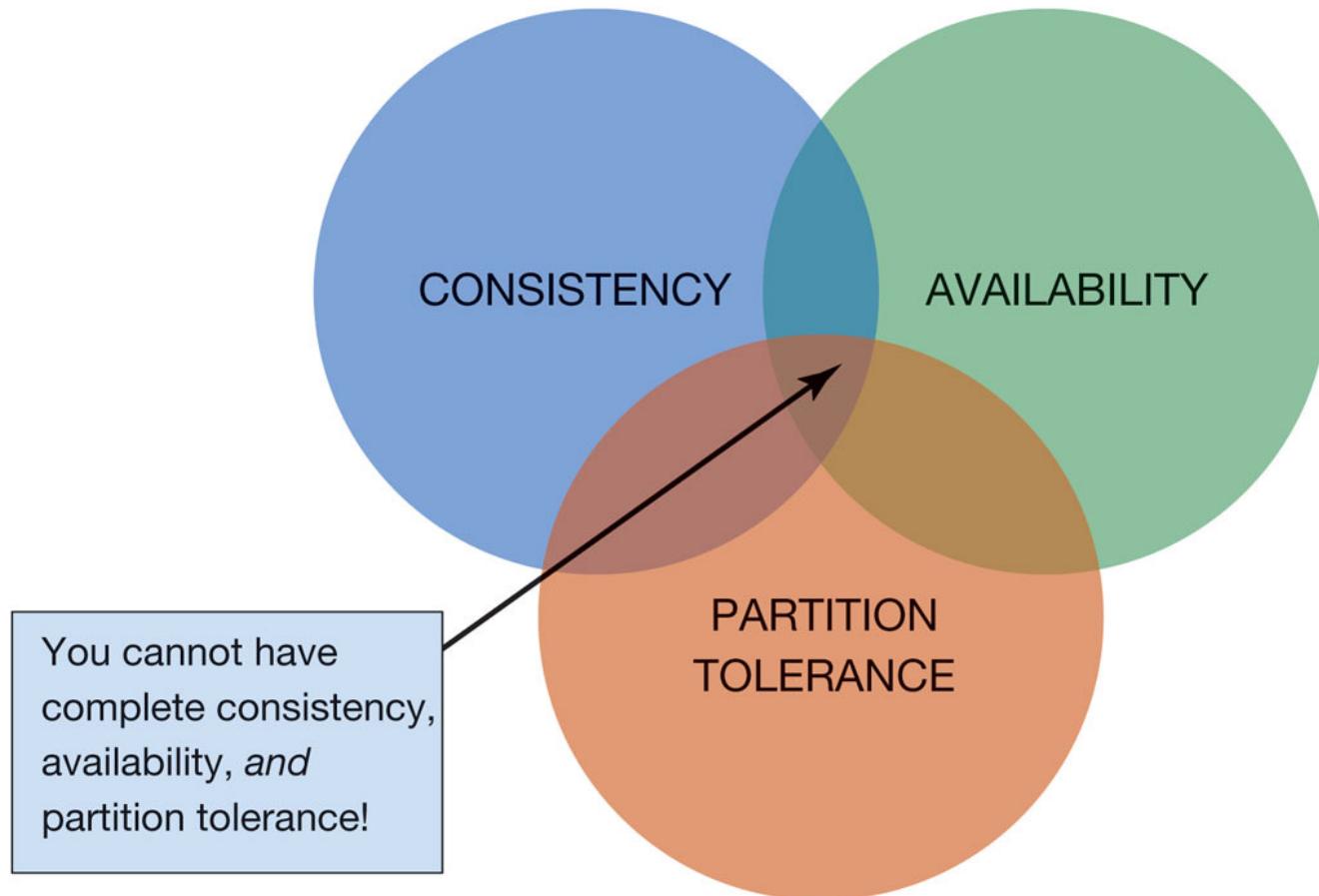
- The **Not only SQL** (previously **NoSQL**) movement is a movement to use non-relational databases.
- These databases are often a distributed, replicated database to support large datasets.

# The CAP Theorem

**Understand the limitations and tradeoffs of replicated, partitioned stores as indicated by the CAP (Consistency, Availability, Partition Tolerance) theorem**

- The **CAP theorem** defines three properties of distributed database systems:
  - **Consistency** means that all database replicas see the same data at any given point in time
  - **Availability** means that every request received by a server will result in a response, as long as the network is available
  - **Partition tolerance** means that the distributed database can continue to operate even when the cluster is partitioned by network failures into two or more disconnected sections (partitions)

# Figure 7.22 The CAP Theorem—You Can't Have All Three at the Same Time!



# Categories of NoSQL Database Management Systems

**Learn the basic concepts of Big Data, structured storage, and the MapReduce process**

- Four categories of NoSQL databases include:
  - **Key-Value** database
    - Dynamo, MemcacheDB, Redis
  - **Document** database
    - Couchbase, ArangoDB, Mongo DB
  - **Column family** database
    - Vertica, Apache Cassandra, Hbase
  - **Graph** database
    - Neo4j, AllegroGraph, Titan

# Figure 7.23 An ArangoDB AQL Query and JSON Results

The screenshot shows the ArangoDB Web Interface. On the left is a sidebar with navigation links: ArangoDB Community Edition, Collections, Queries (selected), Graphs, Services, Logs, Support, Help Us, and Get Enterprise. The main area has tabs for Queries, New, and Save as. A status bar at the bottom shows 3:314. The URL in the address bar is 127.0.0.1:8529/\_db/ArtCourseDB/\_admin/aardvark/index.html#queries.

**The AQL query:** The code is:

```
1 /* *** AQL-QUERY-CH07-01 *** */
2 FOR C in Courses
3   FILTER C.Fee > 400
4   RETURN {CourseName: C.Course,
5         CourseDate: C.CourseDate,
6         StudentLastNames: C.Enrollments[*].CustomerLastName}
```

**The query results in a tabular format:** The results are displayed in a table:

CourseName	CourseDate	StudentLastNames
Adv Pastels	11/15/2019	["Kyle"]
Adv Pastels	10/1/2019	["Johnson","Jackson","Pearson"]

**Button to toggle between tabular and JSON syntax:** The button is located at the bottom right of the results table, labeled "Table".

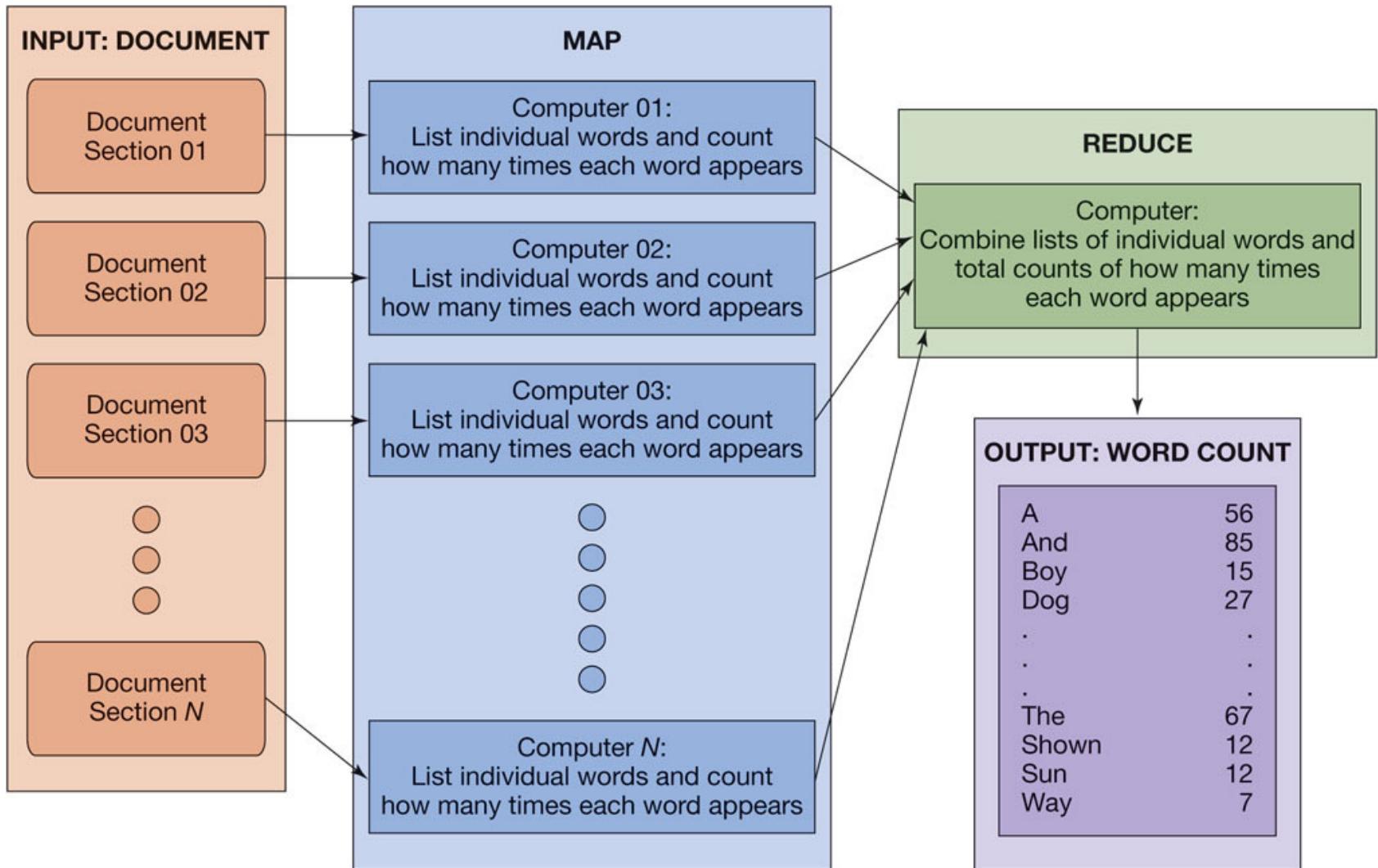
Courtesy of ArangoDB

# MapReduce and Hadoop

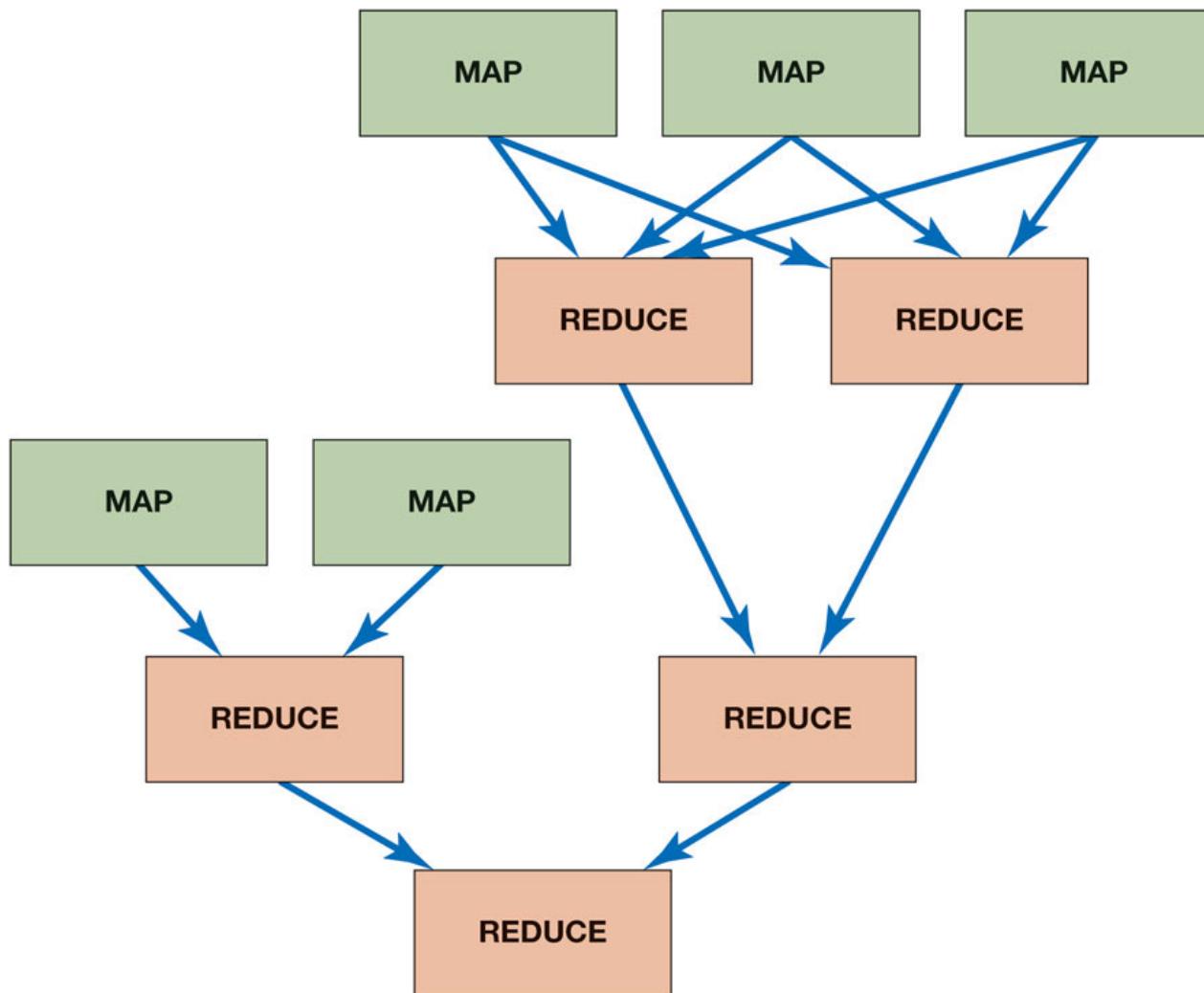
**Learn the basic concepts of Big Data, structured storage, and the MapReduce process**

- **MapReduce** is a technique to analyze extremely large datasets by breaking down tasks and assigning them to a cluster of computers and then combine them into the final product of the original task.
- **Hadoop Distributed File System (HDFS)** provides standard file services to clustered servers so their file systems can function as one distributed, replicated file system that can support large-scale MapReduce processing.

# Figure 7.24 MapReduce



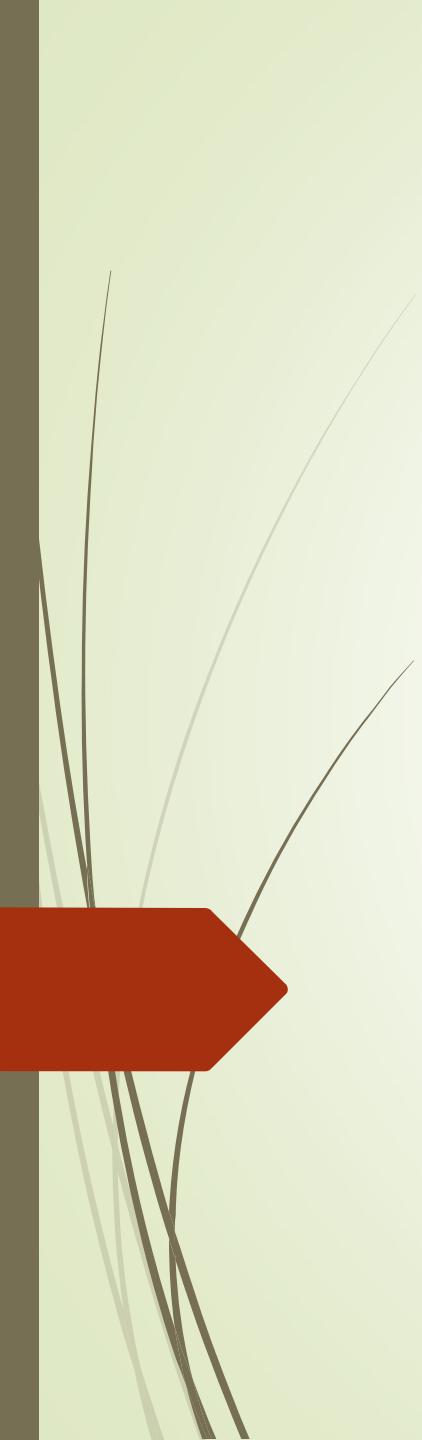
# Figure 7.25 Example of Apache Tez Directed Acyclic Graph



# Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.



# Oracle PL/SQL

# PL/SQL

- ▶ Originally modeled after ADA
  - ▶ Created for Dept. of Defense
- ▶ Allows expanded functionality of database applications
- ▶ Continues to improve with each new database release

# PL/SQL

## ► Features

- Tight integration with SQL
  - Supports data types, functions, pseudo-columns, etc.
- Increased performance
  - A block of statements sent as a single statement
- Increased productivity
  - Same techniques can be used with most Oracle products
- Portability
  - Works on any Oracle platform
- Tighter security
  - Users may not access database objects without granted privileges

# PL/SQL Programs

- ▶ Declaration section (optional)
  - ▶ Any needed variables declared here
- ▶ Executable or begin section
  - ▶ Program code such as statements to retrieve or manipulate data in a table
- ▶ Exception section (optional)
  - ▶ Error traps can catch situations which might ordinarily crash the program

# PL/SQL Block Structure

```
DECLARE
    variables used in this program unit are declared here
BEGIN
    executable sections containing PL/SQL and SQL statements
EXCEPTION
    statements for dealing with errors
END;
/
```

# PL/SQL Variables

- ▶ Variables are local to the code block
- ▶ Names can be up to 30 characters long and must begin with a character
- ▶ Declaration is like that in a table
  - ▶ Name then data type the semi-colon
  - ▶ Can be initialized using := operator in the declaration
  - ▶ Can be changed with := in the begin section
  - ▶ Can use constraints
- ▶ Variables can be composite or collection types
  - ▶ Multiple values of different or same type

# Common PL/SQL Data Types

- ▶ CHAR ( max\_length )
- ▶ VARCHAR2 ( max\_length )
- ▶ NUMBER ( precision, scale )
- ▶ BINARY\_INTEGER – more efficient than number
- ▶ RAW ( max\_length )
- ▶ DATE
- ▶ BOOLEAN (true, false, null)
- ▶ Also LONG, LONG RAW and LOB types but the capacity is usually less in PL/SQL than SQL

# PL/SQL Variable Constraints

- ▶ NOT NULL
  - ▶ Can not be empty
- ▶ CONSTANT
  - ▶ Can not be changed

# PL/SQL Variables Examples

Age number;

Last char ( 10 );

DVal Date := Sysdate;

SID number not null;

Adjust constant number := 1;

CanLoop boolean := true

# Block Types

- ▶ Anonymous

```
[DECLARE]  
  
BEGIN  
  --statements  
  
[EXCEPTION]  
  
END;
```

## Procedure

```
PROCEDURE name  
IS  
  
BEGIN  
  --statements  
  
[EXCEPTION]  
  
END;
```

## Function

```
FUNCTION name  
RETURN datatype  
IS  
  
BEGIN  
  --statements  
  RETURN value;  
[EXCEPTION]  
  
END;
```

# Create your first anonymous block



```
DECLARE
  message  varchar2(20):= 'Hello, World!';
BEGIN
  dbms_output.put_line(message);
END;
/ |
```

Results Explain Describe Saved SQL History

Hello, World!  
Statement processed.  
0.00 seconds

# Conditional Structures

- ▶ IF-THEN
- ▶ IF-THEN-ELSE
- ▶ IF-THEN-ELSIF
  - ▶ An alternative to nested IF-THEN\_ELSE

# IF-THEN Structure

```
IF [T/F condition] THEN  
    statements to perform when condition is true;  
END IF;
```

# Example

```
1 DECLARE n_sales NUMBER := 2000000;
2 BEGIN
3   IF n_sales > 100000 THEN
4     DBMS_OUTPUT.PUT_LINE( 'Sales revenue is greater than 100K' );
5   END IF;
6 END;
```

Results Explain Describe Saved SQL History

Sales revenue is greater than 100K

Statement processed.

0.01 seconds

# IF-THEN-ELSE Structure

```
IF [T/F condition] THEN  
    statements to perform when condition is true;  
ELSE  
    statements to perform when condition is false;  
END IF;
```

# IF-THEN-ELSIF Structure

```
IF [first T/F condition] THEN
    statements to perform when first condition is true;
ELSIF [second T/F condition] THEN
    statements to perform when second condition is true;
ELSIF [third T/F condition] THEN
    statements to perform when third condition is true;
ELSE
    statements to perform when all conditions are false;
END IF;
```

# Example

```
1 DECLARE
2   n_sales NUMBER := 300000;
3   n_commission NUMBER( 10, 2 ) := 0;
4
5 BEGIN
6   IF n_sales > 200000 THEN
7     n_commission := n_sales * 0.1;
8   ELSIF n_sales <= 200000 AND n_sales > 100000 THEN
9     n_commission := n_sales * 0.05;
10  ELSIF n_sales <= 100000 AND n_sales > 50000 THEN
11    n_commission := n_sales * 0.03;
12  ELSE
13    n_commission := n_sales * 0.02;
14  END IF;
15
16  DBMS_OUTPUT.PUT_LINE( 'This sales deal is profitable' || ' ' || ' ' || n_commission );
17 END;
```

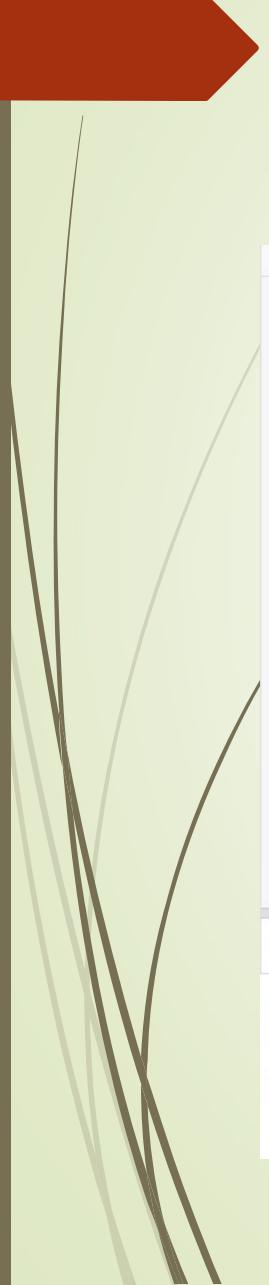
Results Explain Describe Saved SQL History

This sales deal is profitable 30000

Statement processed.

0.00 seconds

# Let us try to access an existing record



```
1 DECLARE
2   sales NUMBER(8,2) := 10100;
3   quota NUMBER(8,2) := 10000;
4   bonus NUMBER(6,2);
5   emp_id NUMBER(6) := 7839;
6 BEGIN
7   IF sales > 5000 THEN
8     bonus := 100;
9
10    UPDATE emp SET sal = (sal + bonus)
11      WHERE empno = emp_id;
12
13
14
15  END IF;
16
17
18
19
20 END;
```

The code editor window shows a PL/SQL block. Lines 1 through 18 are part of the block, while lines 19 and 20 are currently selected. Below the code editor is a results panel with tabs for 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is active, displaying the output: '1 row(s) updated.' and '0.01 seconds'.

► Check if it is updated

# Let us try to show the results based in the block itself



```
DECLARE
  sales  NUMBER(8,2) := 10100;
  quota  NUMBER(8,2) := 10000;
  bonus  NUMBER(6,2);
  emp_id NUMBER(6) := 7839;
  r_empno emp%ROWTYPE;
BEGIN
  IF sales > 5000 THEN
    bonus := 100;
  UPDATE emp SET sal = (sal + bonus)
  WHERE empno = emp_id;
  END IF;
  SELECT * INTO r_empno
  FROM emp
  WHERE empno = emp_id;
  -- show the employee info
  dbms_output.put_line( r_empno.empno || ' ' || 'New Salary ' || r_empno.sal );
END;
```

Results Explain Describe Saved SQL History

7839 New Salary 6300  
1 row(s) updated.  
0.05 seconds

# Iterative Structures

- ▶ LOOP ... EXIT ... END LOOP
  - ▶ EXIT with an If Avoids Infinite Loop
- ▶ LOOP ... EXIT WHEN ... END LOOP
  - ▶ Do Not Need An If to Control EXIT
- ▶ WHILE ... LOOP ... END LOOP
  - ▶ Eliminates Need for EXIT
- ▶ FOR ... IN ... END LOOP
  - ▶ Eliminates Need for Initialization of Counter

# Loop Example

```
1 DECLARE
2   l_counter NUMBER := 0;
3 BEGIN
4   LOOP
5     l_counter := l_counter + 1;
6     IF l_counter > 3 THEN
7       EXIT;
8     END IF;
9     dbms_output.put_line( 'Inside loop: ' || l_counter ) ;
10    END LOOP;
11  -- control resumes here after EXIT
12  dbms_output.put_line( 'After loop: ' || l_counter );
13 END;
```

Results   Explain   Describe   Saved SQL   History

Inside loop: 1  
Inside loop: 2  
Inside loop: 3  
After loop: 4

Statement processed.

0.01 seconds



# Class Exercise

- ▶ Create an anonymous block that will accept the hours and the hourly wage and compute the total wage.
- ▶ Create an anonymous block that will create a table, get data from another table and insert these data to the new table created.

# Exercise 1

- ▶ Create an anonymous block that will do the following:
  - ▶ Accept votes for two candidates
  - ▶ Compute the total voting population
  - ▶ Determine the percentage each vote got based on the total voting population
  - ▶ Display the TVP and the percentages
  - ▶ Determine who is the winner
  - ▶ Detect if there is a tie and display it is a tie

```
The total voting population is 35.00  
The percentage of candidate 1 is 42.86  
The percentage of candidate 2 is 57.14  
Candidate 2 won
```

Statement processed.

0.01 seconds

# The into statement

- SELECT INTO statement is the simplest and fastest way to fetch a single row from a table into variables.

```
DECLARE
    l_customer_name customers.name%TYPE;
BEGIN
    -- get name of the customer 100 and assign it to l_customer_name
    SELECT name INTO l_customer_name
    FROM customers
    WHERE customer_id = 100;
    -- show the customer name
    dbms_output.put_line( v_customer_name );
END;
```

```
DECLARE
    l_customer_name customers.name%TYPE;
    l_contact_first_name contacts.first_name%TYPE;
    l_contact_last_name contacts.last_name%TYPE;
BEGIN
    -- get customer and contact names
    SELECT
        name,
        first_name,
        last_name
    INTO
        l_customer_name,
        l_contact_first_name,
        l_contact_last_name
    FROM
        customers
    INNER JOIN contacts USING( customer_id )
    WHERE
        customer_id = 100;
    -- show the information
    dbms_output.put_line(
        l_customer_name || ', Contact Person: ' ||
        l_contact_first_name || ' ' || l_contact_last_name );
END;
```

```
DECLARE
    r_customer customers%ROWTYPE;
BEGIN
    -- get the information of the customer 100
    SELECT * INTO r_customer
    FROM customers
    WHERE customer_id = 100;
    -- show the customer info
    dbms_output.put_line( r_customer.name || ', website: ' || r_customer.website );
END;
```

# Cursors

Result Set

**MIS380    DATABASE DESIGN    4**

**MIS202    INFORMATION SYSTEMS    3 <Cursor**

**MIS485    MANAGING TECHNOLOGY    4**

**MIS480    ADVANCED DATABASE    4**

# Cursors

- ▶ Declaring an Explicit Cursor

**CURSOR CursorName IS  
SelectStatement;**

- ▶ Opening an Explicit Cursor

**OPEN CursorName;**

- ▶ Accessing Rows from an Explicit Cursor

**FETCH CursorName INTO  
RowVariables;**

# Cursors

- ▶ Cursors Hold Result of an SQL Statement
- ▶ Two Types of Cursors in PL/SQL
  - ▶ Implicit – Automatically Created When a Query or Manipulation is for a Single Row
  - ▶ Explicit – Must Be Declared by the User
    - ▶ Creates a Unit of Storage Called a Result Set

# Cursors

- ▶ Declaring Variables of the Proper Type with %TYPE  
`VarName  
TableName.FieldName%TYPE;`
- ▶ Declaring Variables to Hold An Entire Row  
`VarName CursorName%ROWTYPE;`
- ▶ Releasing the Storage Area Used by an Explicit Cursor  
`CLOSE CursorName;`

# Cursor Control With Loops

- ▶ Need a Way to Fetch Repetitively
- ▶ Need a Way to Determine How Many Rows to Process With a Cursor
  - ▶ Cursor Attributes
    - ▶ **CursorName%ROWCOUNT** – Number of Rows in a Result Set
    - ▶ **CursorName%FOUND** – True if a Fetch Returns a Row
    - ▶ **CursorName%NOTFOUND** – True if Fetch Goes Past Last Row

# Let us try to access data using loop



```
1 DECLARE
2   v_emp_rec emp%ROWTYPE;
3   CURSOR cur_emp_name IS
4     SELECT *
5       FROM emp;
6 BEGIN
7   OPEN cur_emp_name;
8   LOOP
9     FETCH cur_emp_name INTO v_emp_rec;
10    EXIT WHEN cur_emp_name%NOTFOUND;
11    dbms_output.put_line('Name: ' || v_emp_rec.ename || ' :: Salary: ' || v_emp_rec.sal);
12  END LOOP;
13  CLOSE cur_emp_name;
14 END;
```

The screenshot shows a SQL developer interface. The top half displays a PL/SQL block that iterates through the 'emp' table using a cursor and prints each employee's name and salary to the screen. The bottom half shows the results of this execution, listing all employees from the 'emp' table.

Results	
Result	Text
1	Name: KING :: Salary: 6300
2	Name: BLAKE :: Salary: 2850
3	Name: CLARK :: Salary: 2450
4	Name: JONES :: Salary: 2975
5	Name: SCOTT :: Salary: 3000
6	Name: FORD :: Salary: 3000
7	Name: SMITH :: Salary: 800
8	Name: ALLEN :: Salary: 1600
9	Name: WARD :: Salary: 1250
10	Name: MARTIN :: Salary: 1250