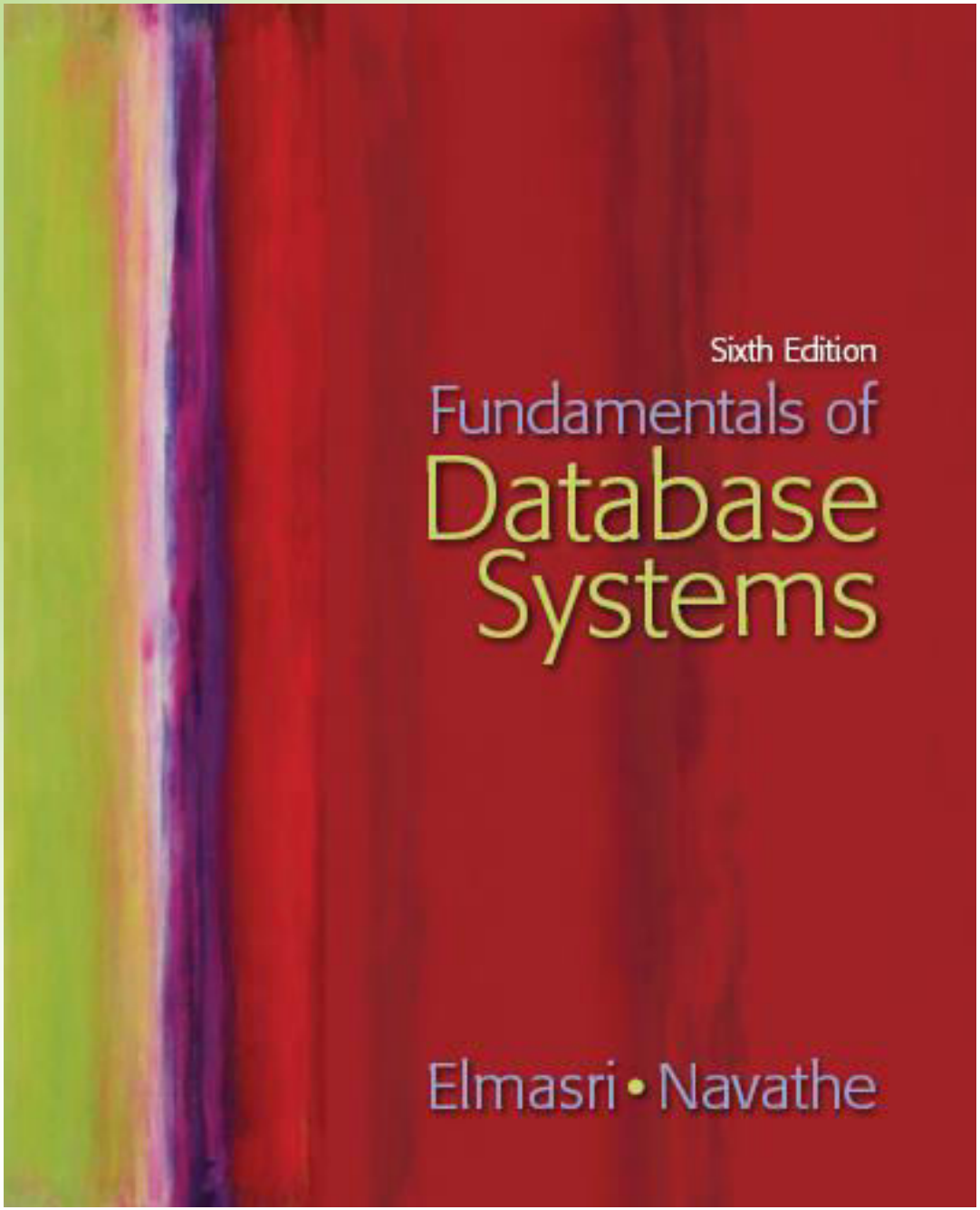
Chapter 1

Databases and Database Users



Addison-Wesley
is an imprint of
PEARSON

Chapter 1 Outline

- Introduction
- An Example
- Characteristics of the Database Approach
- Actors on the Scene
- Workers behind the Scene
- Advantages of Using the DBMS Approach
- A Brief History of Database Applications
- When Not to Use a DBMS

Overview

- **Traditional database applications**
  - Store textual or numeric information
- **Multimedia databases**
  - Store images, audio clips, and video streams digitally
- **Geographic information systems (GIS)**
  - Store and analyze maps, weather data, and satellite images

Overview (cont'd.)

- **Data warehouses and online analytical processing (OLAP) systems**
  - Extract and analyze useful business information from very large databases
  - Support decision making
- **Real-time and active database technology**
  - Control industrial and manufacturing processes

Introduction

- **Database**
  - Collection of related data
  - Known facts that can be recorded and that have implicit meaning
  - **Miniworld** or **universe of discourse (UoD)**
  - Represents some aspect of the real world
  - Logically coherent collection of data with inherent meaning
  - Built for a specific purpose

Introduction (cont'd.)

- Example of a large commercial database
  - Amazon.com
- **Database management system (DBMS)**
  - Collection of programs
  - Enables users to create and maintain a database
- **Defining** a database
  - Specify the data types, structures, and constraints of the data to be stored

Introduction (cont'd.)

- **Meta-data**
  - Database definition or descriptive information
  - Stored by the DBMS in the form of a database catalog or dictionary
- **Manipulating** a database
  - Query and update the database miniworld
  - Generate reports

Introduction (cont'd.)

- **Sharing** a database
  - Allow multiple users and programs to access the database simultaneously
- **Application program**
  - Accesses database by sending queries to DBMS
- **Query**
  - Causes some data to be retrieved

Introduction (cont'd.)

- **Transaction**
  - May cause some data to be read and some data to be written into the database
- **Protection** includes:
  - System protection
  - Security protection
- **Maintain** the database system
  - Allow the system to evolve as requirements change over time

An Example

- UNIVERSITY database
  - Information concerning students, courses, and grades in a university environment
- **Data records**
  - STUDENT
  - COURSE
  - SECTION
  - GRADE_REPORT
  - PREREQUISITE

An Example (cont'd.)

- Specify structure of records of each file by specifying **data type** for each **data element**
  - String of alphabetic characters
  - Integer
  - Etc.

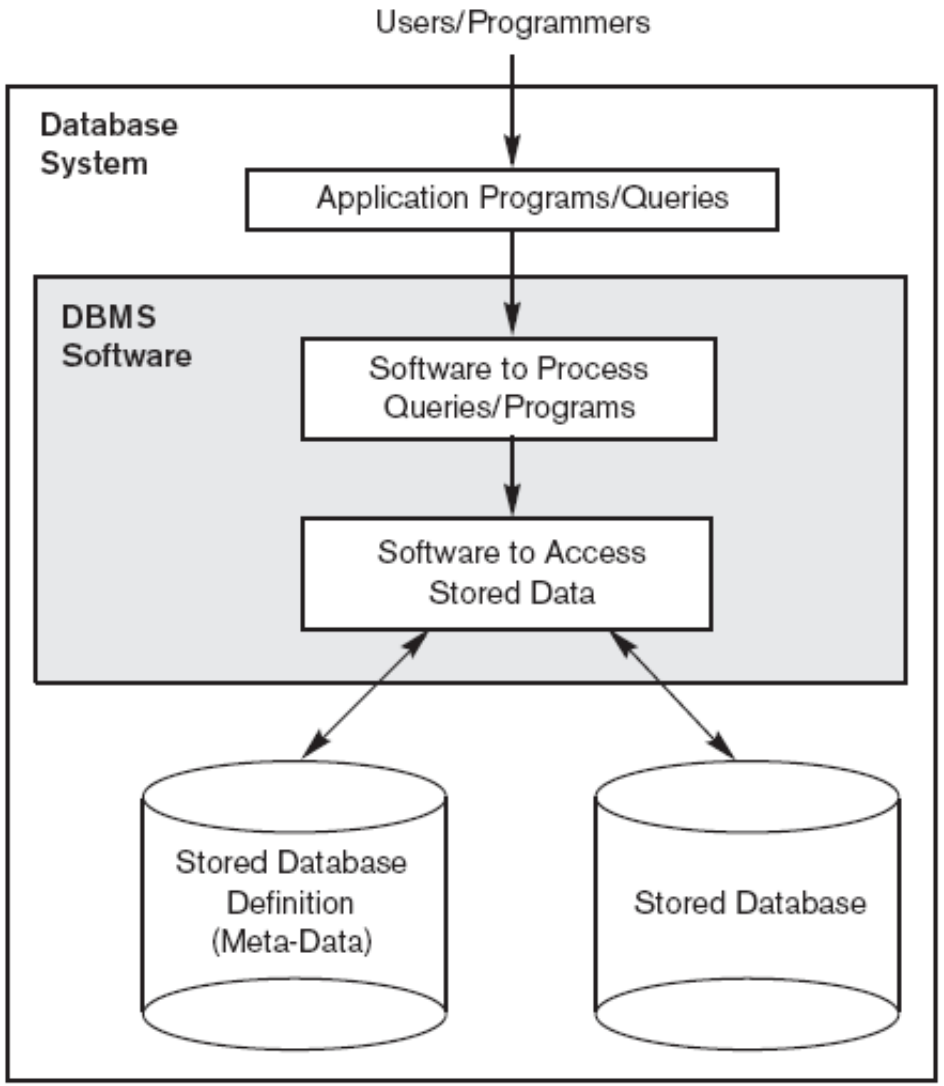


Figure 1.1
A simplified database system environment.

An Example (cont'd.)

- Construct UNIVERSITY database
  - Store data to represent each student, course, section, grade report, and prerequisite as a record in appropriate file
- Relationships among the records
- Manipulation involves querying and updating

An Example (cont'd.)

- Examples of queries:
  - Retrieve the transcript
  - List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
  - List the prerequisites of the 'Database' course

An Example (cont'd.)

- Examples of updates:
  - Change the class of ‘Smith’ to sophomore
  - Create a new section for the ‘Database’ course for this semester
  - Enter a grade of ‘A’ for ‘Smith’ in the ‘Database’ section of last semester

An Example (cont'd.)

- Phases for designing a database:
  - **Requirements specification and analysis**
  - **Conceptual design**
  - **Logical design**
  - **Physical design**

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Characteristics of the Database Approach

- Traditional **file processing**
  - Each user defines and implements the files needed for a specific software application
- Database approach
  - Single repository maintains data that is defined once and then accessed by various users

Characteristics of the Database Approach (cont'd.)

- Main characteristics of database approach
  - Self-describing nature of a database system
  - Insulation between programs and data, and data abstraction
  - Support of multiple views of the data
  - Sharing of data and multiuser transaction processing

Self-Describing Nature of a Database System

- Database system contains complete definition of structure and constraints
- **Meta-data**
  - Describes structure of the database
- Database catalog used by:
  - DBMS software
  - Database users who need information about database structure

Insulation Between Programs and Data

- **Program-data independence**
  - Structure of data files is stored in DBMS catalog separately from access programs
- Program-operation independence
  - **Operations** specified in two parts:
    - Interface includes operation name and data types of its arguments
    - Implementation can be changed without affecting the interface

Data Abstraction

- **Data abstraction**
  - Allows program-data independence and program-operation independence
- **Conceptual representation** of data
  - Does not include details of how data is stored or how operations are implemented
- **Data model**
  - Type of data abstraction used to provide conceptual representation

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
….	….	…..
….	….	…..
….	….	…..
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

Support of Multiple Views of the Data

- **View**
  - Subset of the database
  - Contains **virtual data** derived from the database files but is not explicitly stored
- Multiuser DBMS
  - Users have a variety of distinct applications
  - Must provide facilities for defining multiple views

Sharing of Data and Multiuser Transaction Processing

- Allow multiple users to access the database at the same time
- **Concurrency control software**
  - Ensure that several users trying to update the same data do so in a controlled manner
    - Result of the updates is correct
- **Online transaction processing (OLTP) application**

Sharing of Data and Multiuser Transaction Processing (cont'd.)

- **Transaction**
  - Central to many database applications
  - Executing program or process that includes one or more database
  - **Isolation** property
    - Each transaction appears to execute in isolation from other transactions
  - **Atomicity** property
    - Either all the database operations in a transaction are executed or none are

Actors on the Scene

- **Database administrators (DBA)** are responsible for:
  - Authorizing access to the database
  - Coordinating and monitoring its use
  - Acquiring software and hardware resources
- **Database designers** are responsible for:
  - Identifying the data to be stored
  - Choosing appropriate structures to represent and store this data

Actors on the Scene (cont'd.)

- **End users**
  - People whose jobs require access to the database
  - Types
    - **Casual end users**
    - **Naive** or **parametric end users**
    - **Sophisticated end users**
    - **Standalone users**

Actors on the Scene (cont'd.)

- **System analysts**
  - Determine requirements of end users
- **Application programmers**
  - Implement these specifications as programs

Workers behind the Scene

- **DBMS system designers and implementers**
  - Design and implement the DBMS modules and interfaces as a software package
- **Tool developers**
  - Design and implement **tools**
- **Operators and maintenance personnel**
  - Responsible for running and maintenance of hardware and software environment for database system

Advantages of Using the DBMS Approach

- Controlling redundancy
  - **Data normalization**
  - **Denormalization**
    - Sometimes necessary to use **controlled redundancy** to improve the performance of queries
- Restricting unauthorized access
  - Security and authorization subsystem
  - Privileged software

Advantages of Using the DBMS Approach (cont'd.)

- Providing **persistent** storage for program objects
  - Complex object in C++ can be stored permanently in an object-oriented DBMS
  - **Impedance mismatch problem**
    - Object-oriented database systems typically offer data structure **compatibility**

Advantages of Using the DBMS Approach (cont'd.)

- Providing storage structures and search techniques for efficient query processing
  - **Indexes**
  - **Buffering** and **caching**
  - **Query processing and optimization**

Advantages of Using the DBMS Approach (cont'd.)

- Providing backup and recovery
  - **Backup and recovery subsystem** of the DBMS is responsible for recovery
- Providing multiple user interfaces
  - **Graphical user interfaces (GUIs)**
- Representing complex relationships among data
  - May include numerous varieties of data that are interrelated in many ways

Advantages of Using the DBMS Approach (cont'd.)

- Enforcing **integrity constraints**
  - **Referential integrity** constraint
    - Every section record must be related to a course record
  - **Key** or **uniqueness** constraint
    - Every course record must have a unique value for Course_number
  - **Business rules**
  - **Inherent rules** of the data model

Advantages of Using the DBMS Approach (cont'd.)

- Permitting inferencing and actions using rules
  - **Deductive database systems**
    - Provide capabilities for defining deduction **rules**
    - Inferencing new information from the stored database facts
  - **Trigger**
    - Rule activated by updates to the table
  - **Stored procedures**
    - More involved procedures to enforce rules

Advantages of Using the DBMS Approach (cont'd.)

- Additional implications of using the database approach
  - Reduced application development time
  - Flexibility
  - Availability of up-to-date information
  - Economies of scale

A Brief History of Database Applications

- Early database applications using hierarchical and network systems
  - Large numbers of records of similar structure
- Providing data abstraction and application flexibility with relational databases
  - Separates physical storage of data from its conceptual representation
  - Provides a mathematical foundation for data representation and querying

A Brief History of Database Applications (cont'd.)

- Object-oriented applications and the need for more complex databases
  - Used in specialized applications: engineering design, multimedia publishing, and manufacturing systems
- Interchanging data on the Web for e-commerce using XML
  - Extended markup language (XML) primary standard for interchanging data among various types of databases and Web pages

A Brief History of Database Applications (cont'd.)

- Extending database capabilities for new applications
  - Extensions to better support specialized requirements for applications
  - **Enterprise resource planning (ERP)**
  - **Customer relationship management (CRM)**
- Databases versus information retrieval
  - **Information retrieval (IR)**
    - Deals with books, manuscripts, and various forms of library-based articles

When Not to Use a DBMS

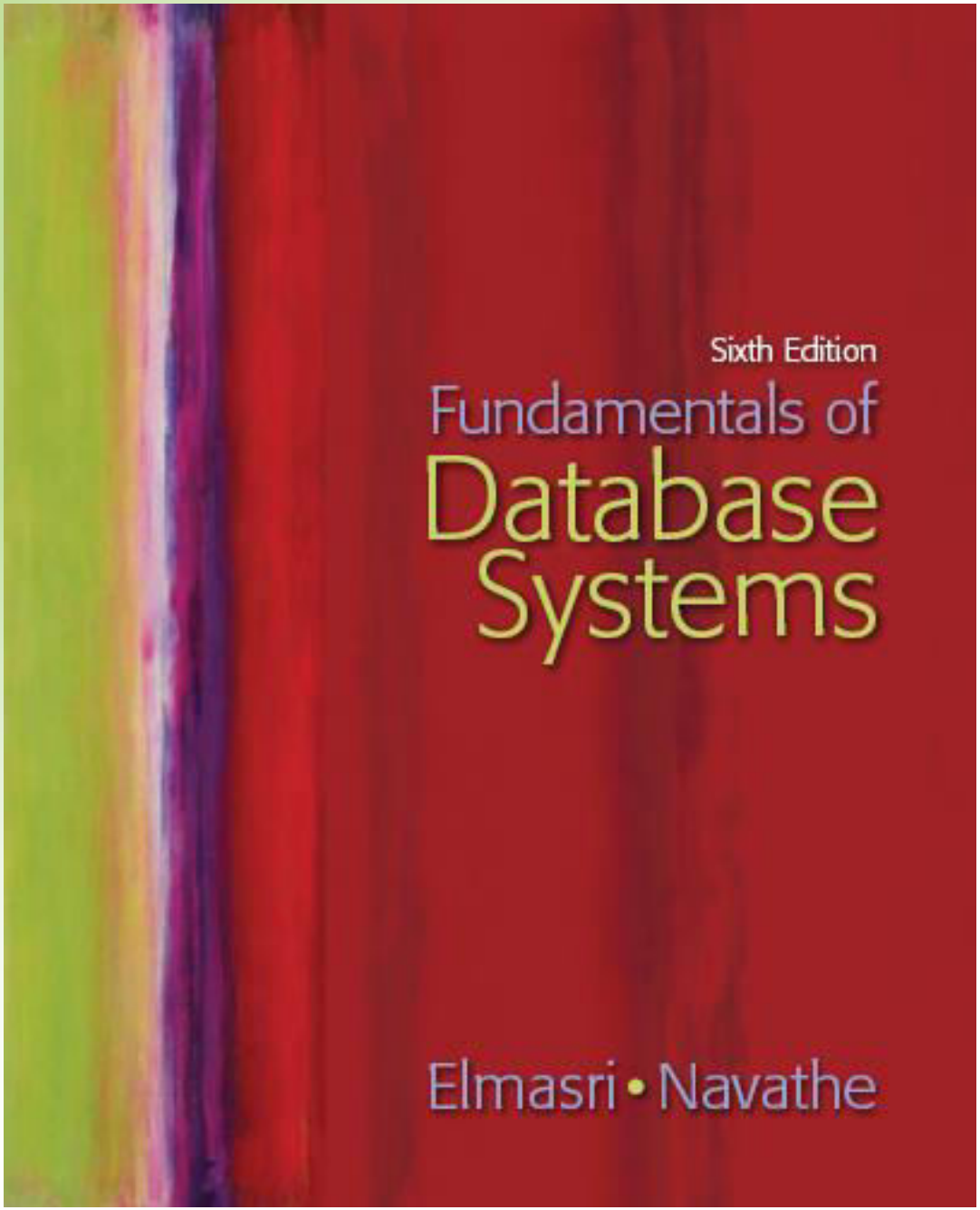
- More desirable to use regular files for:
  - Simple, well-defined database applications not expected to change at all
  - Stringent, real-time requirements that may not be met because of DBMS overhead
  - Embedded systems with limited storage capacity
  - No multiple-user access to data

Summary

- Database
  - Collection of related data (recorded facts)
- DBMS
  - Generalized software package for implementing and maintaining a computerized database
- Several categories of database users
- Database applications have evolved
  - Current trends: IR, Web

Chapter 2

Database System Concepts and Architecture



Addison-Wesley
is an imprint of
PEARSON

Chapter 2 Outline

- Data Models, Schemas, and Instances
- Three-Schema Architecture and Data Independence
- Database Languages and Interfaces
- The Database System Environment
- Centralized and Client/Server Architectures for DBMSs
- Classification of Database Management Systems

Database System Concepts and Architecture

- Basic client/server DBMS architecture
  - **Client module**
  - **Server module**

Data Models, Schemas, and Instances

- **Data abstraction**
  - Suppression of details of data organization and storage
  - Highlighting of the essential features for an improved understanding of data

Data Models, Schemas, and Instances (cont'd.)

- **Data model**
  - Collection of concepts that describe the structure of a database
  - Provides means to achieve data abstraction
  - **Basic operations**
    - Specify retrievals and updates on the database
  - **Dynamic aspect** or **behavior** of a database application
    - Allows the database designer to specify a set of valid operations allowed on database objects

Categories of Data Models

- **High-level** or **conceptual data models**
  - Close to the way many users perceive data
- **Low-level** or **physical data models**
  - Describe the details of how data is stored on computer storage media
- **Representational data models**
  - Easily understood by end users
  - Also similar to how data organized in computer storage

Categories of Data Models (cont'd.)

- **Entity**
  - Represents a real-world object or concept
- **Attribute**
  - Represents some property of interest
  - Further describes an entity
- **Relationship** among two or more entities
  - Represents an association among the entities
  - **Entity-Relationship model**

Categories of Data Models (cont'd.)

- **Relational data model**
  - Used most frequently in traditional commercial DBMSs
- **Object data model**
  - New family of higher-level implementation data models
  - Closer to conceptual data models

Categories of Data Models (cont'd.)

- **Physical data models**
  - Describe how data is stored as files in the computer
  - **Access path**
    - Structure that makes the search for particular database records efficient
  - **Index**
    - Example of an access path
    - Allows direct access to data using an index term or a keyword

Schemas, Instances, and Database State

- **Database schema**
  - Description of a database
- **Schema diagram**
  - Displays selected aspects of schema
- **Schema construct**
  - Each object in the schema
- **Database state** or **snapshot**
  - Data in database at a particular moment in time

Schemas, Instances, and Database State (cont'd.)

Figure 2.1
Schema diagram for the database in Figure 1.2.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

[6]Schema changes are usually needed as the requirements of the database applications change. Newer database systems include operations for allowing schema changes, although the schema change process is more involved than simple database updates.

[7]It is customary in database parlance to use *schemas* as the plural for *schema*, even though *schemata* is the proper plural form. The word *scheme* is also sometimes used to refer to a schema.

Schemas, Instances, and Database State (cont'd.)

- **Define** a new database
  - Specify database schema to the DBMS
- **Initial state**
  - **Populated** or **loaded** with the initial data
- **Valid state**
  - Satisfies the structure and constraints specified in the schema

Schemas, Instances, and Database State (cont'd.)

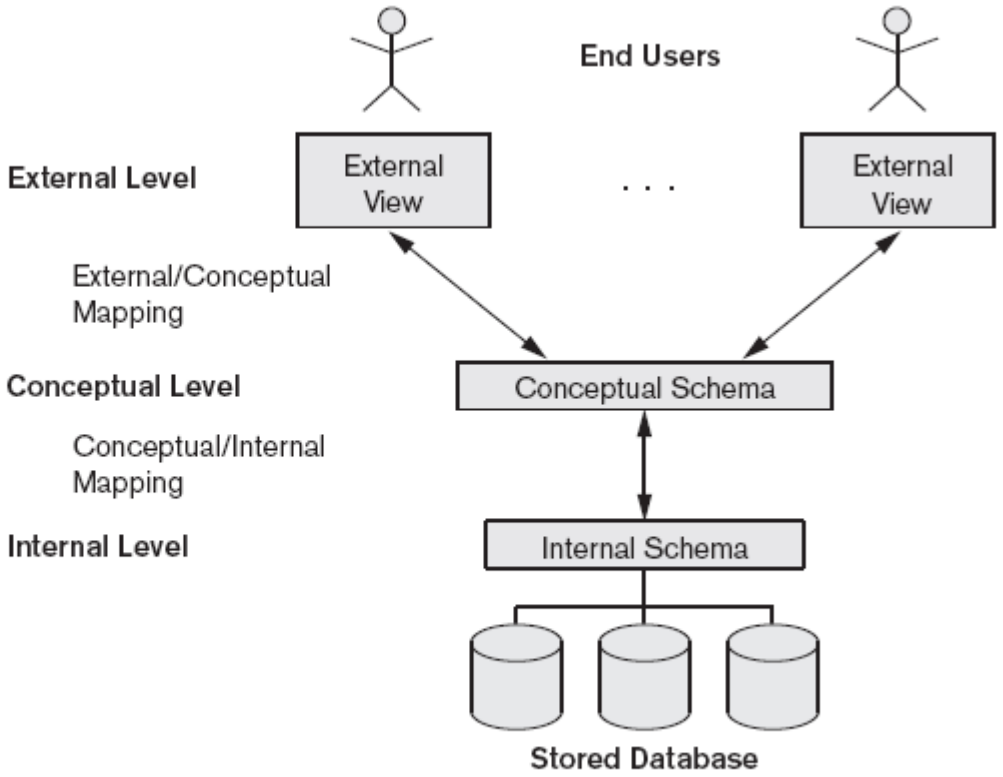
- **Schema evolution**
  - Changes applied to schema as application requirements change

Three-Schema Architecture and Data Independence

- **Internal level**
  - Describes physical storage structure of the database
- **Conceptual level**
  - Describes structure of the whole database for a community of users
- **External** or **view level**
  - Describes part of the database that a particular user group is interested in

Three-Schema Architecture and Data Independence (cont'd.)

Figure 2.2
The three-schema architecture.



Data Independence

- Capacity to change the schema at one level of a database system
  - Without having to change the schema at the next higher level
- Types:
  - **Logical**
  - **Physical**

DBMS Languages

- **Data definition language (DDL)**
  - Defines both schemas
- **Storage definition language (SDL)**
  - Specifies the internal schema
- **View definition language (VDL)**
  - Specifies user views/mappings to conceptual schema
- **Data manipulation language (DML)**
  - Allows retrieval, insertion, deletion, modification

DBMS Languages (cont'd.)

- **High-level** or **nonprocedural** DML
  - Can be used on its own to specify complex database operations concisely
  - **Set-at-a-time** or **set-oriented**
- **Low-level** or **procedural** DML
  - Must be embedded in a general-purpose programming language
  - **Record-at-a-time**

DBMS Interfaces

- Menu-based interfaces for Web clients or browsing
- Forms-based interfaces
- Graphical user interfaces
- Natural language interfaces
- Speech input and output
- Interfaces for parametric users
- Interfaces for the DBA

The Database System Environment

- DBMS component modules
  - **Buffer management**
  - **Stored data manager**
  - **DDL compiler**
  - **Interactive query** interface
    - **Query compiler**
    - **Query optimizer**
  - **Precompiler**

The Database System Environment (cont'd.)

- DBMS component modules
  - **Runtime database processor**
  - **System catalog**
  - **Concurrency control system**
  - **Backup and recovery system**

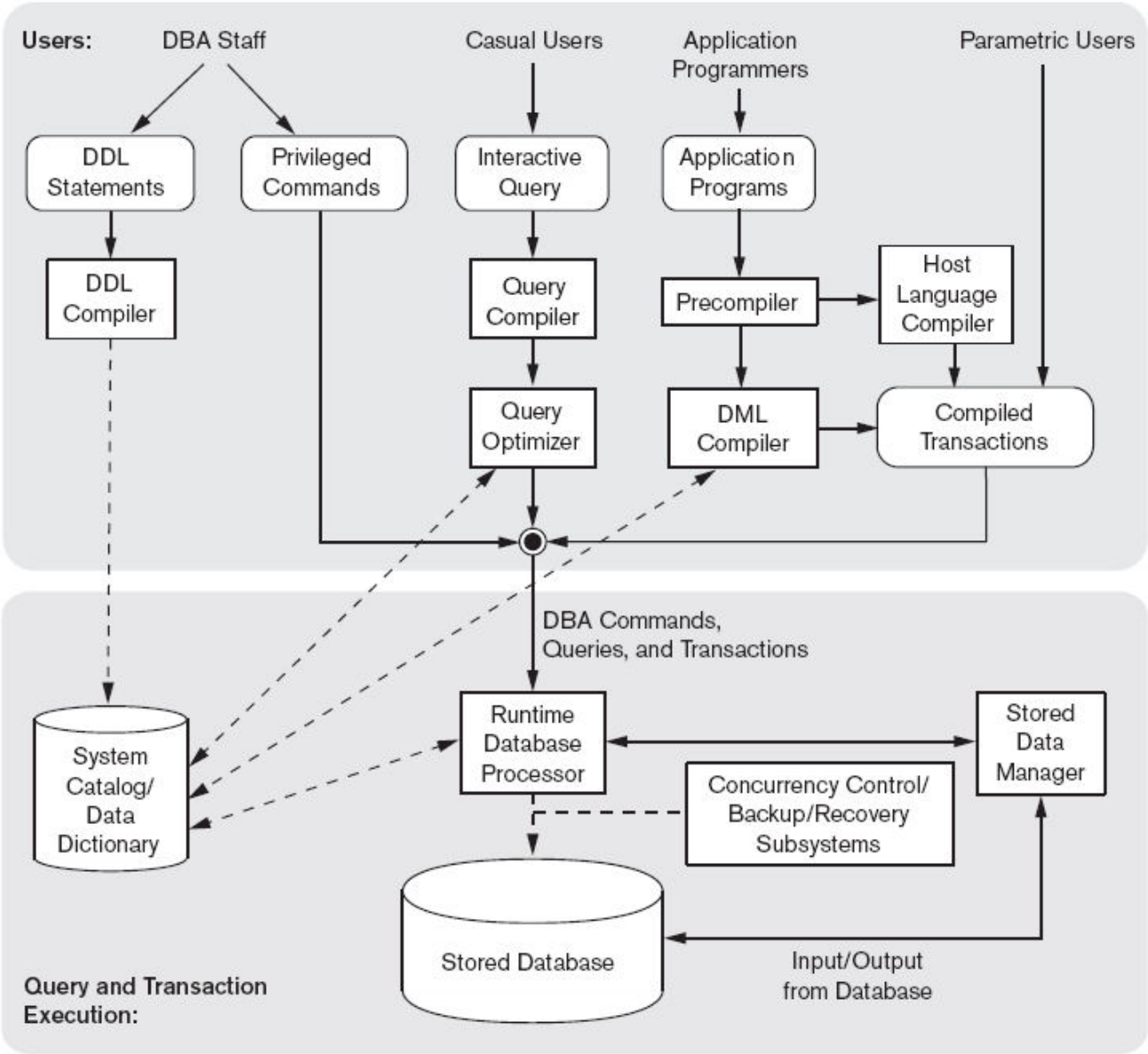


Figure 2.3
Component modules of a DBMS and their interactions.

Database System Utilities

- **Loading**
  - Load existing data files
- **Backup**
  - Creates a backup copy of the database

Database System Utilities (cont'd.)

- **Database storage reorganization**
  - Reorganize a set of database files into different file organizations
- **Performance monitoring**
  - Monitors database usage and provides statistics to the DBA

Tools, Application Environments, and Communications Facilities

- CASE Tools
- **Data dictionary (data repository) system**
  - Stores design decisions, usage standards, application program descriptions, and user information
- **Application development environments**
- **Communications software**

Centralized and Client/Server Architectures for DBMSs

- **Centralized DBMSs Architecture**
  - All DBMS functionality, application program execution, and user interface processing carried out on one machine

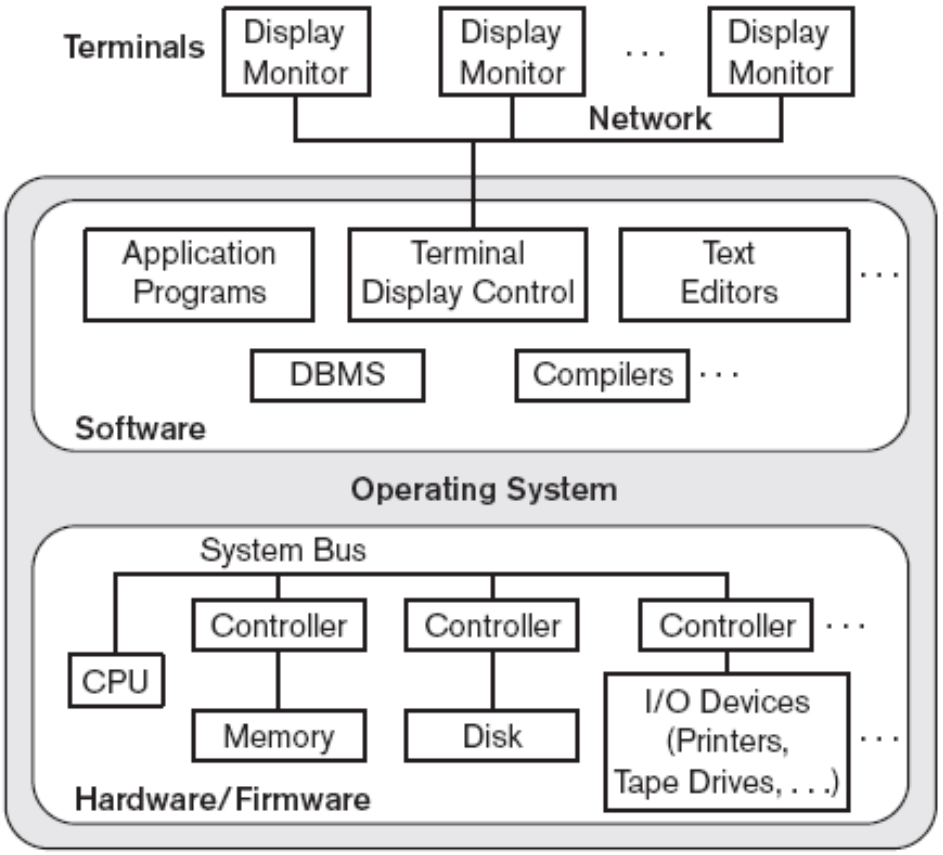


Figure 2.4
A physical centralized architecture.

Basic Client/Server Architectures

- **Servers** with specific functionalities
  - **File server**
    - Maintains the files of the client machines.
  - **Printer server**
    - Connected to various printers; all print requests by the clients are forwarded to this machine
  - **Web servers** or **e-mail servers**

Basic Client/Server Architectures (cont'd.)

- **Client machines**
  - Provide user with:
    - Appropriate interfaces to utilize these servers
    - Local processing power to run local applications

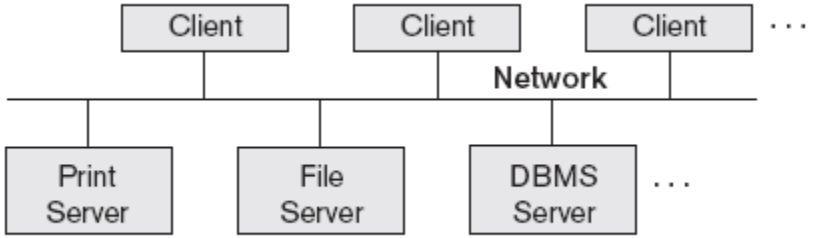


Figure 2.5
Logical two-tier client/server architecture.

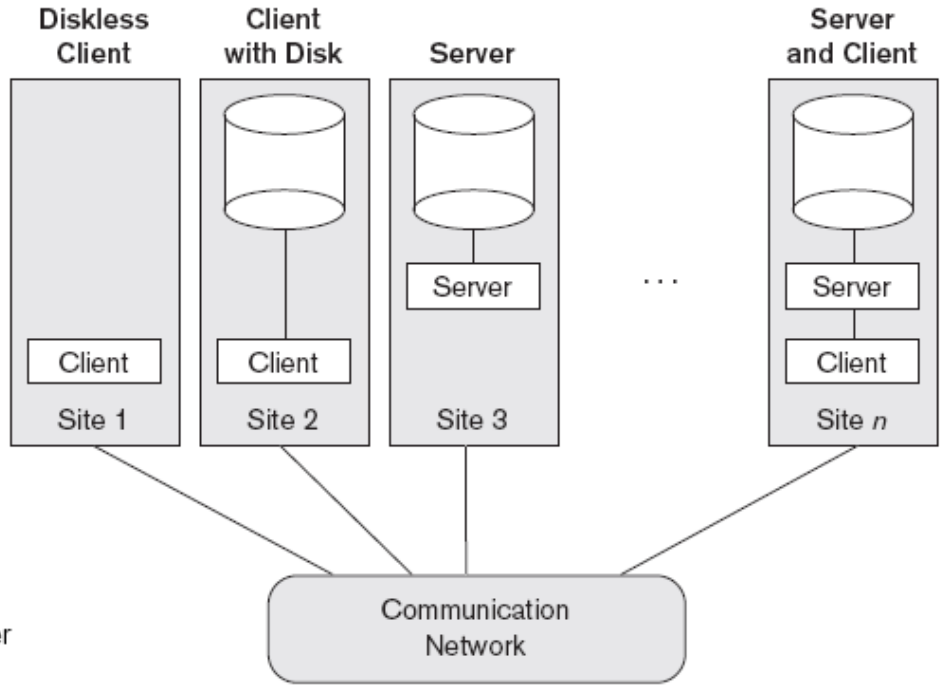


Figure 2.6
Physical two-tier client/server architecture.

Basic Client/Server Architectures (cont'd.)

- **Client**
  - User machine that provides user interface capabilities and local processing
- **Server**
  - System containing both hardware and software
  - Provides services to the client machines
    - Such as file access, printing, archiving, or database access

Two-Tier Client/Server Architectures for DBMSs

- Server handles
  - Query and transaction functionality related to SQL processing
- Client handles
  - User interface programs and application programs

Two-Tier Client/Server Architectures (cont'd.)

- Open Database Connectivity (ODBC)
  - Provides application programming interface (API)
  - Allows client-side programs to call the DBMS
    - Both client and server machines must have the necessary software installed
- JDBC
  - Allows Java client programs to access one or more DBMSs through a standard interface

Three-Tier and n-Tier Architectures for Web Applications

- **Application server** or **Web server**
  - Adds intermediate layer between client and the database server
  - Runs application programs and stores business rules
- N-tier
  - Divide the layers between the user and the stored data further into finer components

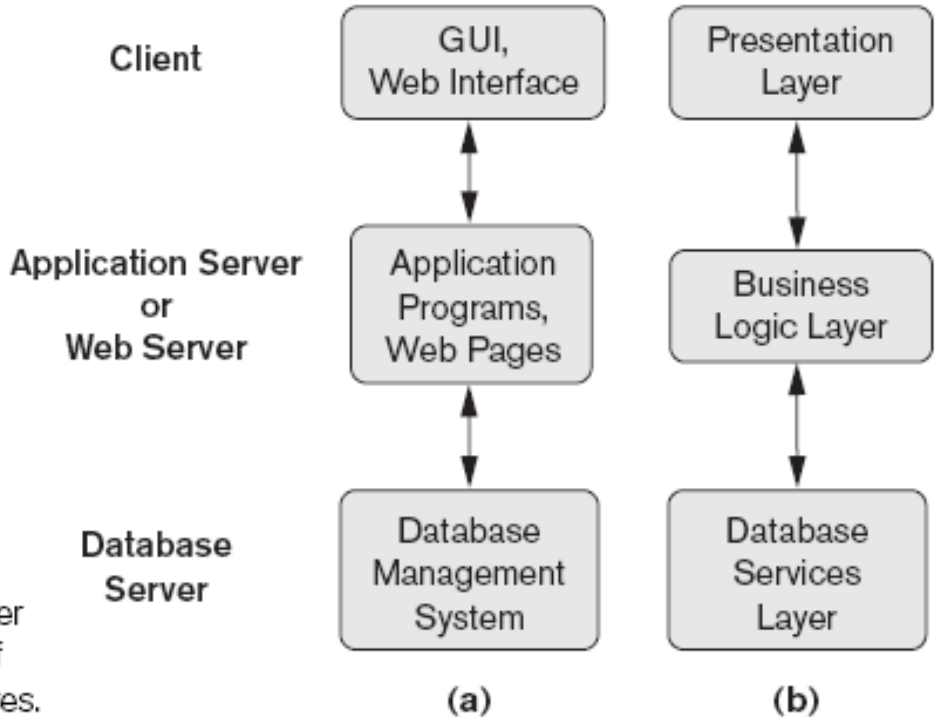


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

Classification of Database Management Systems

- **Data model**
  - **Relational**
  - **Object**
  - **Hierarchical** and **network** (legacy)
  - **Native XML DBMS**
- **Number of users**
  - **Single-user**
  - **Multiuser**

Classification of Database Management Systems (cont'd.)

- **Number of sites**
  - **Centralized**
  - **Distributed**
    - **Homogeneous**
    - **Heterogeneous**
- **Cost**
  - Open source
  - Different types of licensing

Classification of Database Management Systems (cont'd.)

- **Types of access path options**
- **General** or **special-purpose**

Classification of Database Management Systems (cont'd.)

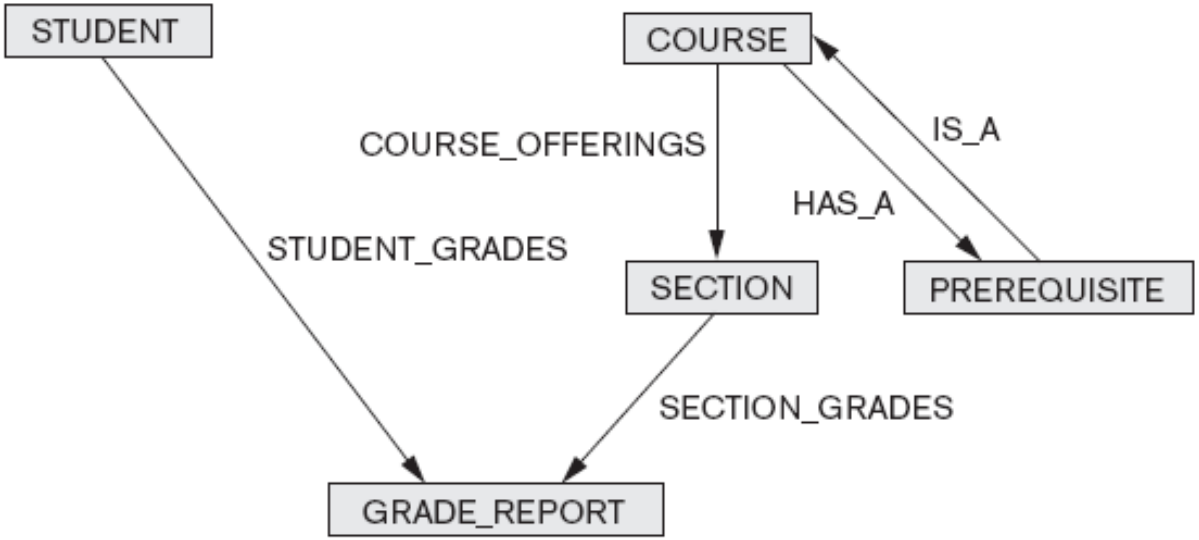


Figure 2.8
The schema of Figure 2.1 in network model notation.

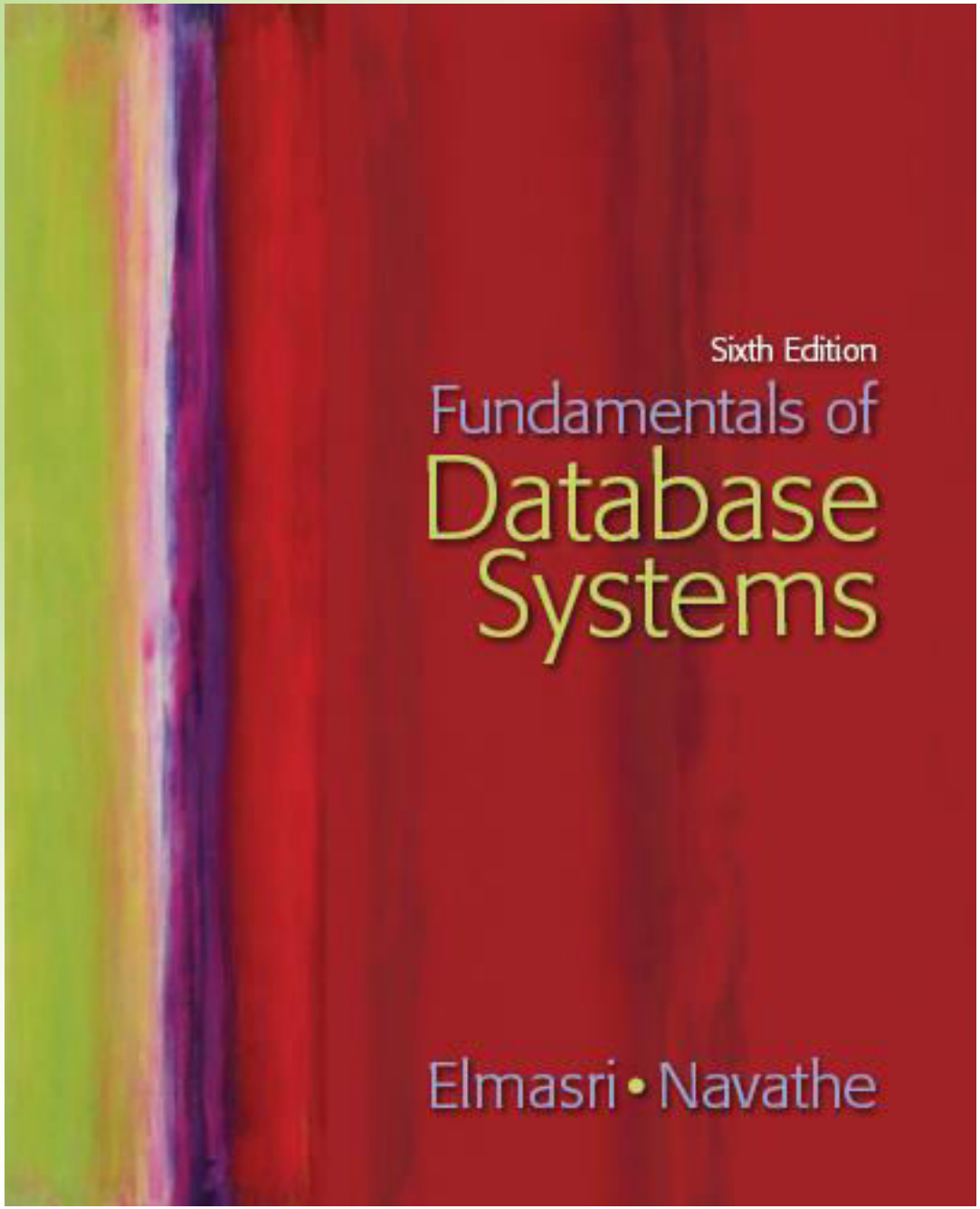
[14]CODASYL DBTG stands for Conference on Data Systems Languages Database Task Group, which is the committee that specified the network model and its language.

Summary

- Concepts used in database systems
- Main categories of data models
- Types of languages supported by DMBSs
- Interfaces provided by the DBMS
- DBMS classification criteria:
  - Data model, number of users, number of sties, access paths, cost

Chapter 3

The Relational Data Model and Relational Database Constraints



Addison-Wesley
is an imprint of
PEARSON

Chapter 3 Outline

- The Relational Data Model and Relational Database Constraints
- Relational Model Constraints and Relational Database Schemas
- Update Operations, Transactions, and Dealing with Constraint Violations

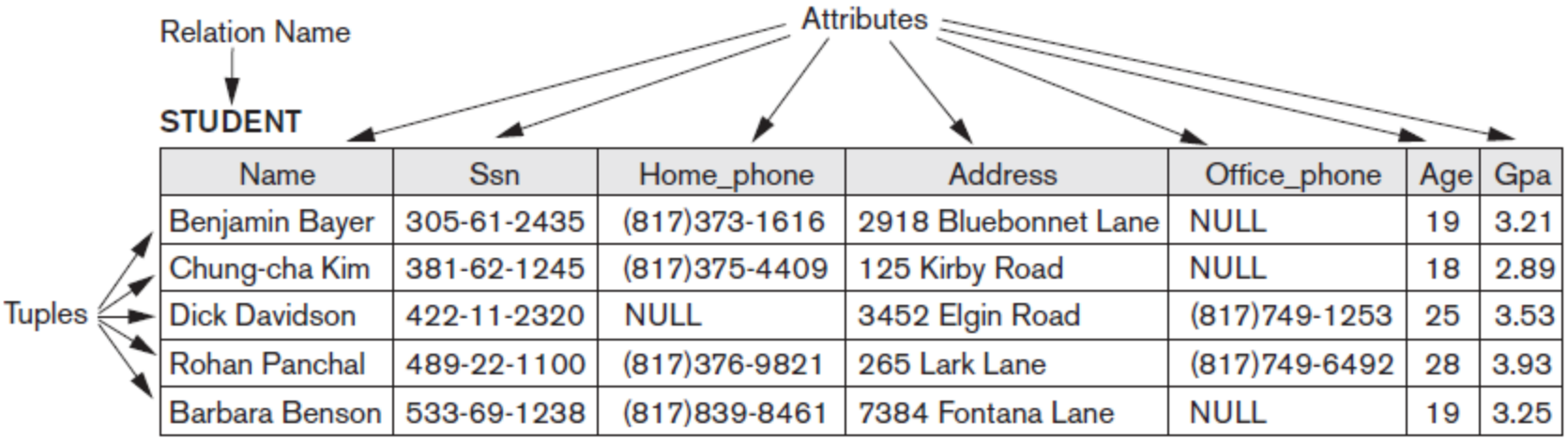
The Relational Data Model and Relational Database Constraints

- Relational model
  - First commercial implementations available in early 1980s
  - Has been implemented in a large number of commercial system
- Hierarchical and network models
  - Preceded the relational model

Relational Model Concepts

- Represents data as a collection of relations
- **Table** of values
  - Row
    - Represents a collection of related data values
    - Fact that typically corresponds to a real-world entity or relationship
    - *Tuple*
  - Table name and column names
    - Interpret the meaning of the values in each row *attribute*

Relational Model Concepts (cont'd.)



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 3.1
The attributes and tuples of a relation STUDENT.

Domains, Attributes, Tuples, and Relations

- **Domain** D
  - Set of atomic values
- **Atomic**
  - Each value indivisible
- Specifying a domain
  - **Data type** specified for each domain

Domains, Attributes, Tuples, and Relations (cont'd.)

- **Relation schema** R
  - Denoted by $R(A_1, A_2, ..., A_n)$
  - Made up of a relation name R and a list of attributes, $A_1, A_2, ..., A_n$
- Attribute A_i
  - Name of a role played by some domain D in the relation schema R
- **Degree** (or **arity**) of a relation
  - Number of attributes n of its relation schema

Domains, Attributes, Tuples, and Relations (cont'd.)

- **Relation** (or **relation state**)
  - Set of ***n*-tuples** $r = \{t_1, t_2, ..., t_m\}$
  - Each n-tuple t
    - Ordered list of n values $t = <v_1, v_2, ..., v_n>$
    - Each value v_i, $1 \leq i \leq n$, is an element of dom(A_i) or is a special NULL value

Domains, Attributes, Tuples, and Relations (cont'd.)

- Relation (or relation state) $r(R)$
  - **Mathematical relation** of degree n on the domains $\text{dom}(A_1)$, $\text{dom}(A_2)$, ..., $\text{dom}(A_n)$
  - **Subset** of the **Cartesian product** of the domains that define R:
    - $r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times ... \times \text{dom}(A_n))$

Domains, Attributes, Tuples, and Relations (cont'd.)

- **Cardinality**
  - Total number of values in domain
- **Current relation state**
  - Relation state at a given time
  - Reflects only the valid tuples that represent a particular state of the real world
- Attribute names
  - Indicate different **roles**, or interpretations, for the domain

Characteristics of Relations

- Ordering of tuples in a relation
  - Relation defined as a set of tuples
  - Elements have no order among them
- Ordering of values within a tuple and an alternative definition of a relation
  - Order of attributes and values is not that important
  - As long as correspondence between attributes and values maintained

Characteristics of Relations (cont'd.)

- Alternative definition of a relation
  - Tuple considered as a set of (<attribute>, <value>) pairs
  - Each pair gives the value of the mapping from an attribute A_i to a value v_i from dom(A_i)
- Use the first definition of relation
  - Attributes and the values within tuples are ordered
  - Simpler notation

Characteristics of Relations (cont'd.)

Figure 3.2
The relation STUDENT from Figure 3.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

Characteristics of Relations (cont'd.)

- Values and NULLs in tuples
  - Each value in a tuple is atomic
  - **Flat relational model**
    - Composite and multivalued attributes not allowed
    - **First normal form** assumption
  - Multivalued attributes
    - Must be represented by separate relations
  - Composite attributes
    - Represented only by simple component attributes in basic relational model

Characteristics of Relations (cont'd.)

- NULL values
  - Represent the values of attributes that may be unknown or may not apply to a tuple
  - Meanings for NULL values
    - *Value unknown*
    - *Value exists but is not available*
    - *Attribute does not apply to this tuple (also known as value undefined)*

Characteristics of Relations (cont'd.)

- Interpretation (meaning) of a relation
  - **Assertion**
    - Each tuple in the relation is a **fact** or a particular instance of the assertion
  - **Predicate**
    - Values in each tuple interpreted as values that satisfy predicate

Relational Model Notation

- Relation schema R of degree n
  - Denoted by $R(A_1, A_2, ..., A_n)$
- Uppercase letters Q, R, S
  - Denote relation names
- Lowercase letters q, r, s
  - Denote relation states
- Letters t, u, v
  - Denote tuples

Relational Model Notation

- Name of a relation schema: STUDENT
  - Indicates the current set of tuples in that relation
- Notation: STUDENT(Name, Ssn, ...)
  - Refers only to relation schema
- Attribute *A* can be qualified with the relation name *R* to which it belongs
  - Using the dot notation *R.A*

Relational Model Notation

- *n-tuple t* in a relation *r*(*R*)
  - Denoted by $t = \langle v_1, v_2, ..., v_n \rangle$
  - v_i is the value corresponding to attribute A_i
- Component values of tuples:
  - $t[A_i]$ and $t.A_i$ refer to the value v_i in t for attribute A_i
  - $t[A_u, A_w, ..., A_z]$ and $t.(A_u, A_w, ..., A_z)$ refer to the subtuple of values $\langle v_u, v_w, ..., v_z \rangle$ from t corresponding to the attributes specified in the list

Relational Model Constraints

- Constraints
  - Restrictions on the actual values in a database state
  - Derived from the rules in the miniworld that the database represents
- **Inherent model-based constraints** or **implicit constraints**
  - Inherent in the data model

Relational Model Constraints (cont'd.)

- **Schema-based constraints** or **explicit constraints**
  - Can be directly expressed in schemas of the data model
- **Application-based** or **semantic constraints** or **business rules**
  - Cannot be directly expressed in schemas
  - Expressed and enforced by application program

Domain Constraints

- Typically include:
  - Numeric data types for integers and real numbers
  - Characters
  - Booleans
  - Fixed-length strings
  - Variable-length strings
  - Date, time, timestamp
  - Money
  - Other special data types

Key Constraints and Constraints on NULL Values

- No two tuples can have the same combination of values for all their attributes.
- **Superkey**
  - No two distinct tuples in any state *r* of *R* can have the same value for SK
- **Key**
  - Superkey of R
  - Removing any attribute *A* from *K* leaves a set of attributes *K* that is not a superkey of *R* any more

Key Constraints and Constraints on NULL Values (cont'd.)

- Key satisfies two properties:
  - Two distinct tuples in any state of relation cannot have identical values for (all) attributes in key
  - Minimal superkey
    - Cannot remove any attributes and still have uniqueness constraint in above condition hold

Key Constraints and Constraints on NULL Values (cont'd.)

- **Candidate key**
  - Relation schema may have more than one key
- **Primary key** of the relation
  - Designated among candidate keys
  - Underline attribute
- Other candidate keys are designated as **unique keys**

Key Constraints and Constraints on NULL Values (cont'd.)

CAR

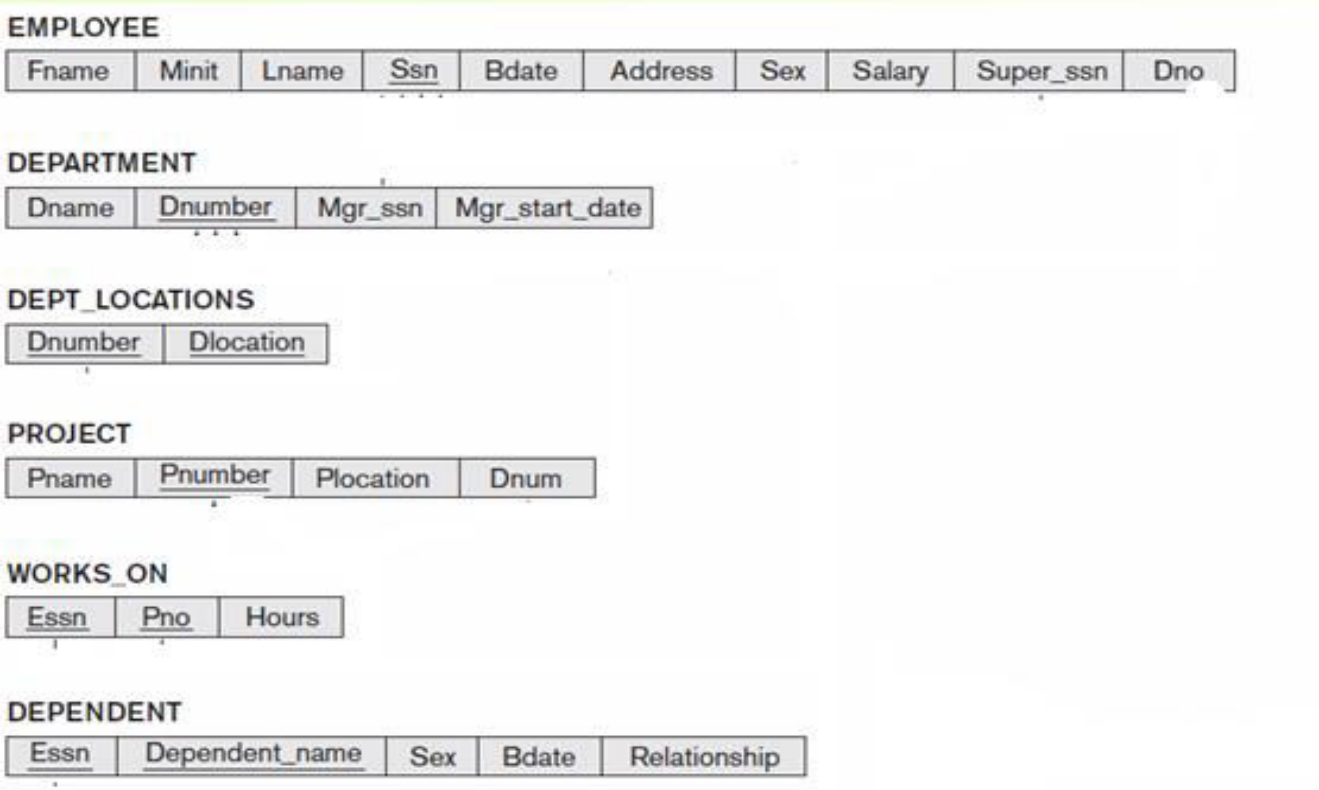
License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 3.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

Relational Databases and Relational Database Schemas

- **Relational database schema** S
  - Set of relation schemas $S = \{R_1, R_2, ..., R_m\}$
  - Set of integrity constraints IC
- **Relational database state**
  - Set of relation states $DB = \{r_1, r_2, ..., r_m\}$
  - Each r_i is a state of R_i and such that the r_i relation states satisfy integrity constraints specified in IC

Relational database schema



Relational database state

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Relational Databases and Relational Database Schemas (cont'd.)

- **Invalid state**
  - Does not obey all the integrity constraints
- **Valid state**
  - Satisfies all the constraints in the defined set of integrity constraints IC

Integrity, Referential Integrity, and Foreign Keys

- **Entity integrity constraint**
  - No primary key value can be NULL
- **Referential integrity constraint**
  - Specified between two relations
  - Maintains consistency among tuples in two relations

Integrity, Referential Integrity, and Foreign Keys (cont'd.)

- **Foreign key** rules:
  - The attributes in FK have the same domain(s) as the primary key attributes PK
  - Value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL

Integrity, Referential Integrity, and Foreign Keys (cont'd.)

- Diagrammatically display referential integrity constraints
  - Directed arc from each foreign key to the relation it references
- All integrity constraints should be specified on relational database schema

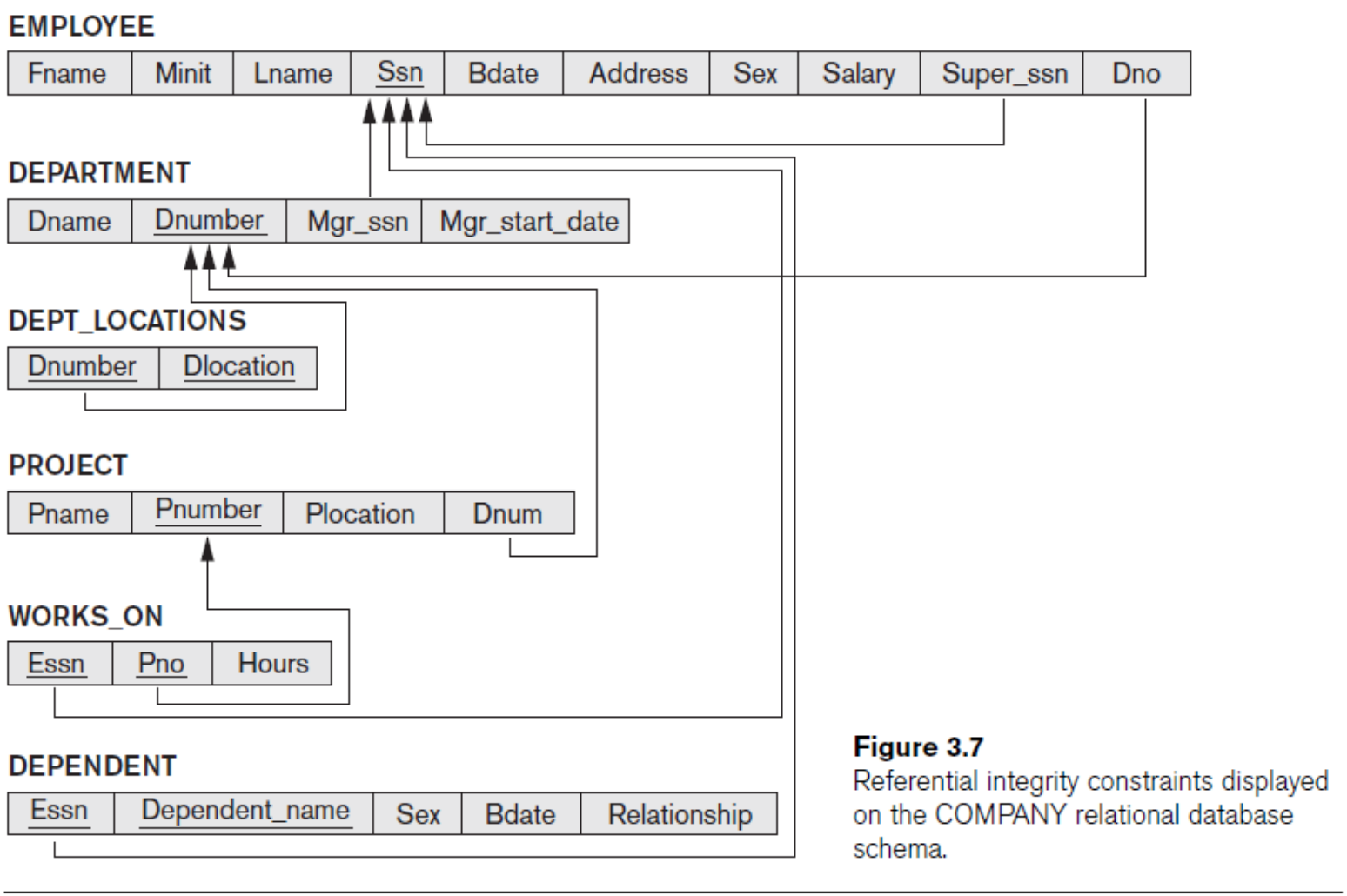


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Other Types of Constraints

- Semantic integrity constraints
  - May have to be specified and enforced on a relational database
  - Use **triggers** and **assertions**
  - More common to check for these types of constraints within the application programs

Other Types of Constraints (cont'd.)

- Functional dependency constraint
  - Establishes a functional relationship among two sets of attributes *X* and *Y*
  - Value of *X* determines a unique value of *Y*
- **State constraints**
  - Define the constraints that a valid state of the database must satisfy
- **Transition constraints**
  - Define to deal with state changes in the database

Update Operations, Transactions, and Dealing with Constraint Violations

- Operations of the relational model can be categorized into retrievals and updates
- Basic operations that change the states of relations in the database:
  - Insert
  - Delete
  - Update (or Modify)

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Figure 3.6
One possible database state for the COMPANY relational database schema.

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

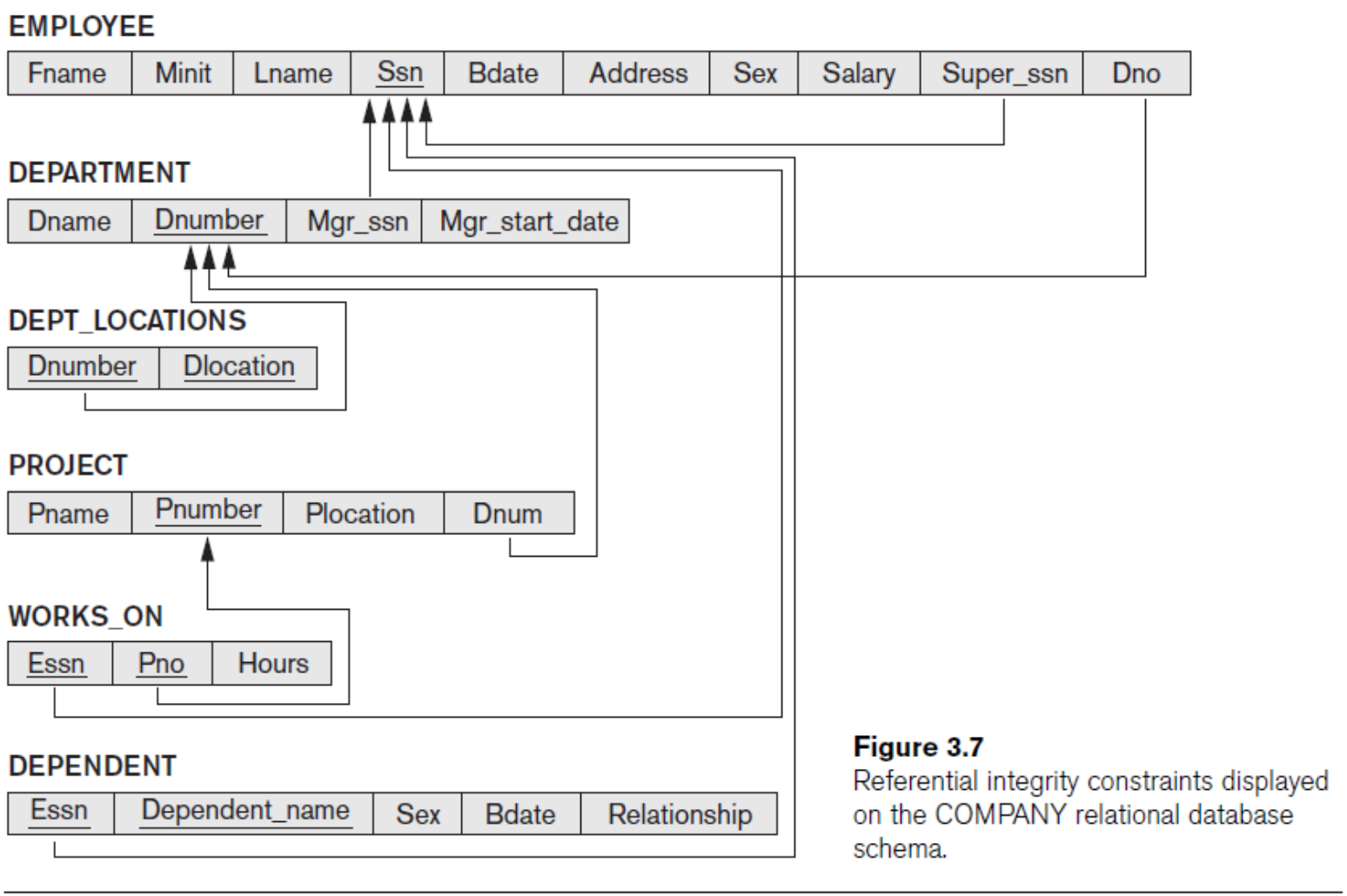
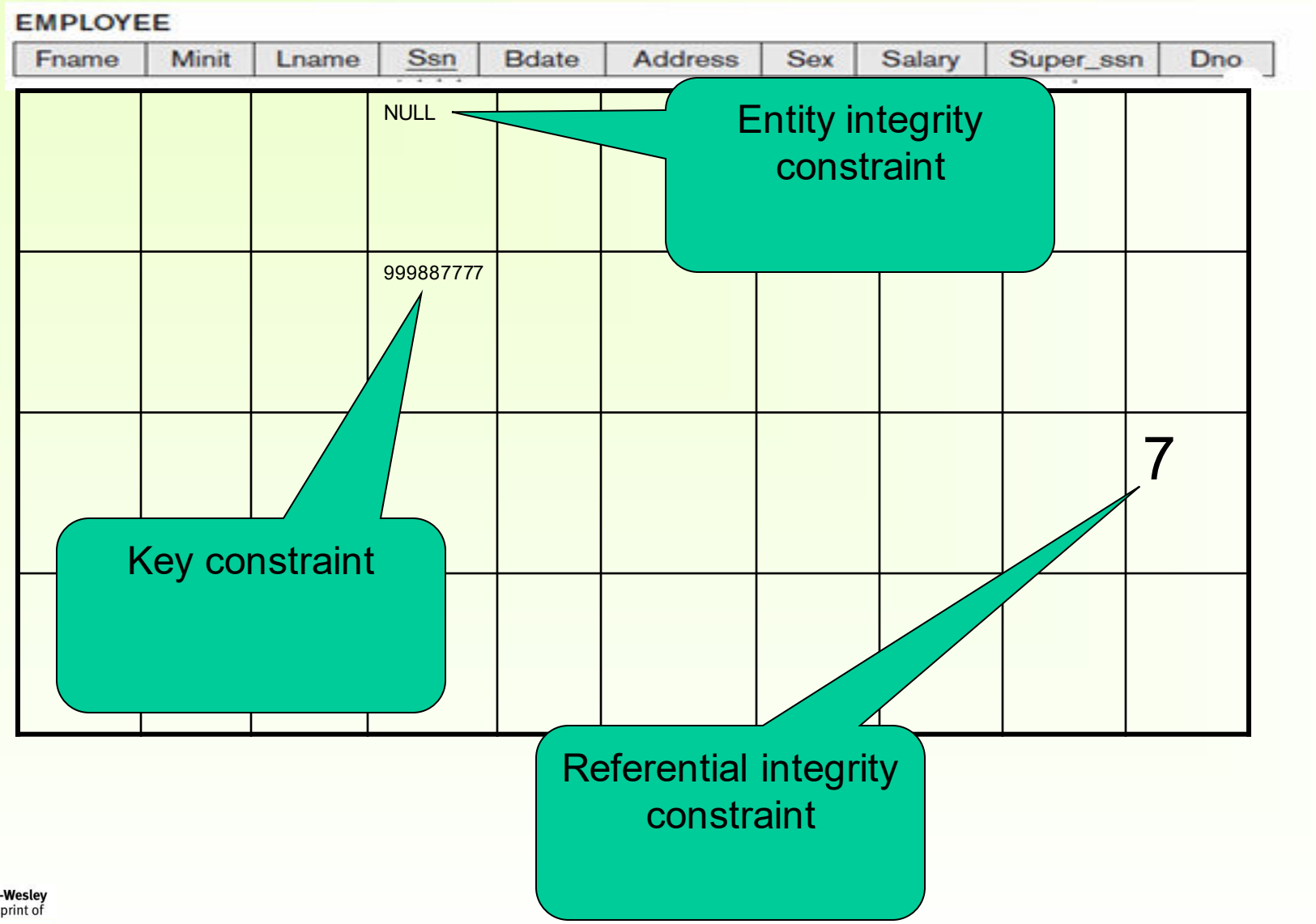


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

The Insert Operation

- Provides a list of attribute values for a new tuple t that is to be inserted into a relation R
- Can violate any of the four types of constraints
- If an insertion violates one or more constraints
  - Default option is to reject the insertion

The Insert Operation



The Delete Operation

- Can violate only referential integrity
  - If tuple being deleted is referenced by foreign keys from other tuples
  - **Restrict**
    - Reject the deletion
  - **Cascade**
    - Propagate the deletion by deleting tuples that reference the tuple that is being deleted
  - **Set null** or **set default**
    - Modify the referencing attribute values that cause the violation

The Update Operation

- Necessary to specify a condition on attributes of relation
  - Select the tuple (or tuples) to be modified
- If attribute not part of a primary key nor of a foreign key
  - Usually causes no problems
- Updating a primary/foreign key
  - Similar issues as with Insert/Delete

The Transaction Concept

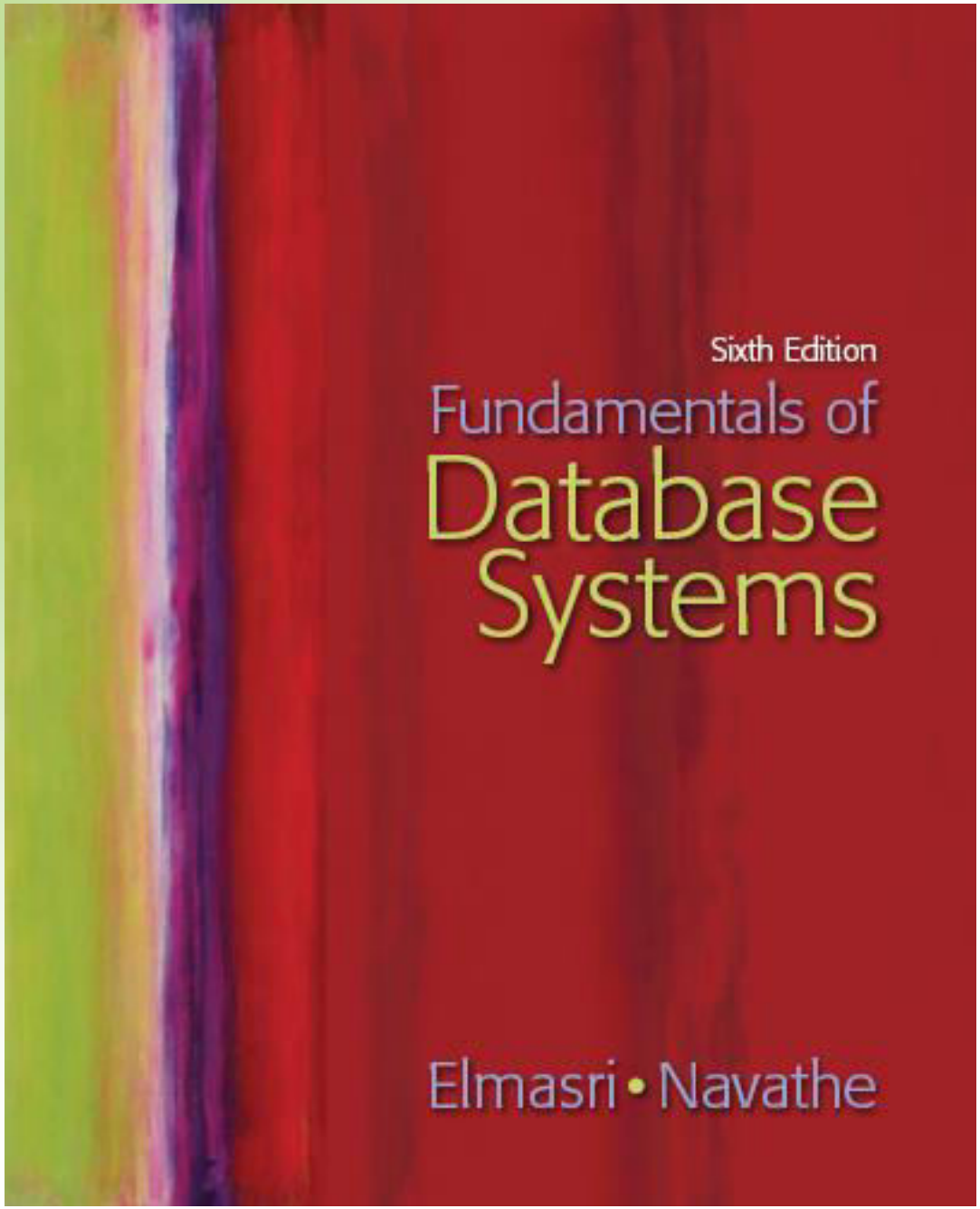
- **Transaction**
  - Executing program
  - Includes some database operations
  - Must leave the database in a valid or consistent state
- **Online transaction processing (OLTP) systems**
  - Execute transactions at rates that reach several hundred per second

Summary

- Characteristics differentiate relations from ordinary tables or files
- Classify database constraints into:
  - Inherent model-based constraints, explicit schema-based constraints, and application-based constraints
- Modification operations on the relational model:
  - Insert, Delete, and Update

Chapter 4

Basic SQL



Addison-Wesley
is an imprint of
PEARSON

Chapter 4 Outline

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- `INSERT`, `DELETE`, and `UPDATE` Statements in SQL
- Additional Features of SQL

Basic SQL

- SQL language
  - Considered one of the major reasons for the commercial success of relational databases
- **SQL**
  - **Structured Query Language**
  - Statements for data definitions, queries, and updates (both DDL and DML)
  - **Core specification**
  - Plus specialized **extensions**

SQL Data Definition and Data Types

- Terminology:
  - **Table**, **row**, and **column** used for relational model terms relation, tuple, and attribute
- `CREATE` statement
  - Main SQL command for data definition

Schema and Catalog Concepts in SQL

- **SQL schema**
  - Identified by a **schema name**
  - Includes an **authorization identifier** and **descriptors** for each element
- Schema **elements** include
  - Tables, constraints, views, domains, and other constructs
- Each statement in SQL ends with a semicolon

Schema and Catalog Concepts in SQL (cont'd.)

- `CREATE SCHEMA` statement
  - `CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';`
- **Catalog**
  - Named collection of schemas in an SQL environment
- SQL **environment**
  - Installation of an SQL-compliant RDBMS on a computer system

The CREATE TABLE Command in SQL

- Specify a new relation
  - Provide name
  - Specify attributes and initial constraints
- Can optionally specify schema:
  - `CREATE TABLE COMPANY.EMPLOYEE ...`

    or
  - `CREATE TABLE EMPLOYEE ...`

The CREATE TABLE Command in SQL (cont'd.)

- **Base tables** (**base relations**)
  - Relation and its tuples are actually created and stored as a file by the DBMS
- **Virtual relations**
  - Created through the `CREATE VIEW` statement

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)        NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)        NOT NULL,
          Ssn            CHAR(9)            NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE DEPARTMENT
        ( Dname          VARCHAR(15)        NOT NULL,
          Dnumber        INT                NOT NULL,
          Mgr_ssn        CHAR(9)            NOT NULL,
          Mgr_start_date DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber           INT             NOT NULL,
          Dlocation         VARCHAR(15)     NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname             VARCHAR(15)     NOT NULL,
          Pnumber           INT             NOT NULL,
          Plocation         VARCHAR(15),
          Dnum              INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn              CHAR(9)         NOT NULL,
          Pno               INT             NOT NULL,
          Hours             DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn              CHAR(9)         NOT NULL,
          Dependent_name    VARCHAR(15)     NOT NULL,
          Sex               CHAR,
          Bdate             DATE,
          Relationship      VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

The CREATE TABLE Command in SQL (cont'd.)

- Some foreign keys may cause errors
  - Specified either via:
    - Circular references
    - Or because they refer to a table that has not yet been created

Attribute Data Types and Domains in SQL

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(`*n*`)`, `CHARACTER(`*n*`)`
    - Varying length: `VARCHAR(`*n*`)`, `CHAR VARYING(`*n*`)`, `CHARACTER VARYING(`*n*`)`

Attribute Data Types and Domains in SQL (cont'd.)

- **Bit-string** data types
  - Fixed length: `BIT(n)`
  - Varying length: `BIT VARYING(n)`
- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`
- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD

Attribute Data Types and Domains in SQL (cont'd.)

- Additional data types
  - **Timestamp** data type (`TIMESTAMP`)
    - Includes the `DATE` and `TIME` fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional `WITH TIME ZONE` qualifier
  - **`INTERVAL`** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

Attribute Data Types and Domains in SQL (cont'd.)

- Domain
  - Name used with the attribute specification
  - Makes it easier to change the data type for a domain that is used by numerous attributes
  - Improves schema readability
  - Example:
    - `CREATE DOMAIN SSN_TYPE AS CHAR(9);`

Specifying Constraints in SQL

- Basic constraints:
  - Key and referential integrity constraints
  - Restrictions on attribute domains and NULLs
  - Constraints on individual tuples within a relation

Specifying Attribute Constraints and Attribute Defaults

- `NOT NULL`
  - `NULL` is not permitted for a particular attribute
- Default value
  - **`DEFAULT`** `<value>`
- **`CHECK`** clause
  - `Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);`

```
CREATE TABLE EMPLOYEE
    ( . . . ,
      Dno        INT          NOT NULL        DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    ( . . . ,
      Mgr_ssn    CHAR(9)      NOT NULL        DEFAULT '888665555',
      . . . ,
    CONSTRAINT DEPTPK
      PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
      UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
      FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET DEFAULT   ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( . . . ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE CASCADE           ON UPDATE CASCADE);
```

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

Specifying Key and Referential Integrity Constraints

- **`PRIMARY KEY`** clause
  - Specifies one or more attributes that make up the primary key of a relation
  - `Dnumber INT PRIMARY KEY;`
- **`UNIQUE`** clause
  - Specifies alternate (secondary) keys
  - `Dname VARCHAR(15) UNIQUE;`

Specifying Key and Referential Integrity Constraints (cont'd.)

- **`FOREIGN KEY`** clause
  - Default operation: reject update on violation
  - Attach **referential triggered action** clause
    - Options include `SET NULL`, `CASCADE`, and `SET DEFAULT`
    - Action taken by the DBMS for `SET NULL` or `SET DEFAULT` is the same for both `ON DELETE` and `ON UPDATE`
    - `CASCADE` option suitable for "relationship" relations

Giving Names to Constraints

- Keyword **`CONSTRAINT`**
  - Name a constraint
  - Useful for later altering

Specifying Constraints on Tuples Using CHECK

- `CHECK` clauses at the end of a `CREATE TABLE` statement
  - Apply to each tuple individually
  - `CHECK (Dept_create_date <= Mgr_start_date);`

Basic Retrieval Queries in SQL

- `SELECT` statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Unlike relational model
  - Multiset or bag behavior

The SELECT-FROM-WHERE Structure of Basic SQL Queries

- Basic form of the `SELECT` statement:

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

The SELECT-FROM-WHERE Structure of Basic SQL Queries (cont'd.)

- Logical comparison operators
  - =, <, <=, >, >=, and <>
- **Projection attributes**
  - Attributes whose values are to be retrieved
- **Selection condition**
  - Boolean condition that must be true for any retrieved tuple

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Figure 3.6
One possible database state for the COMPANY relational database schema.

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

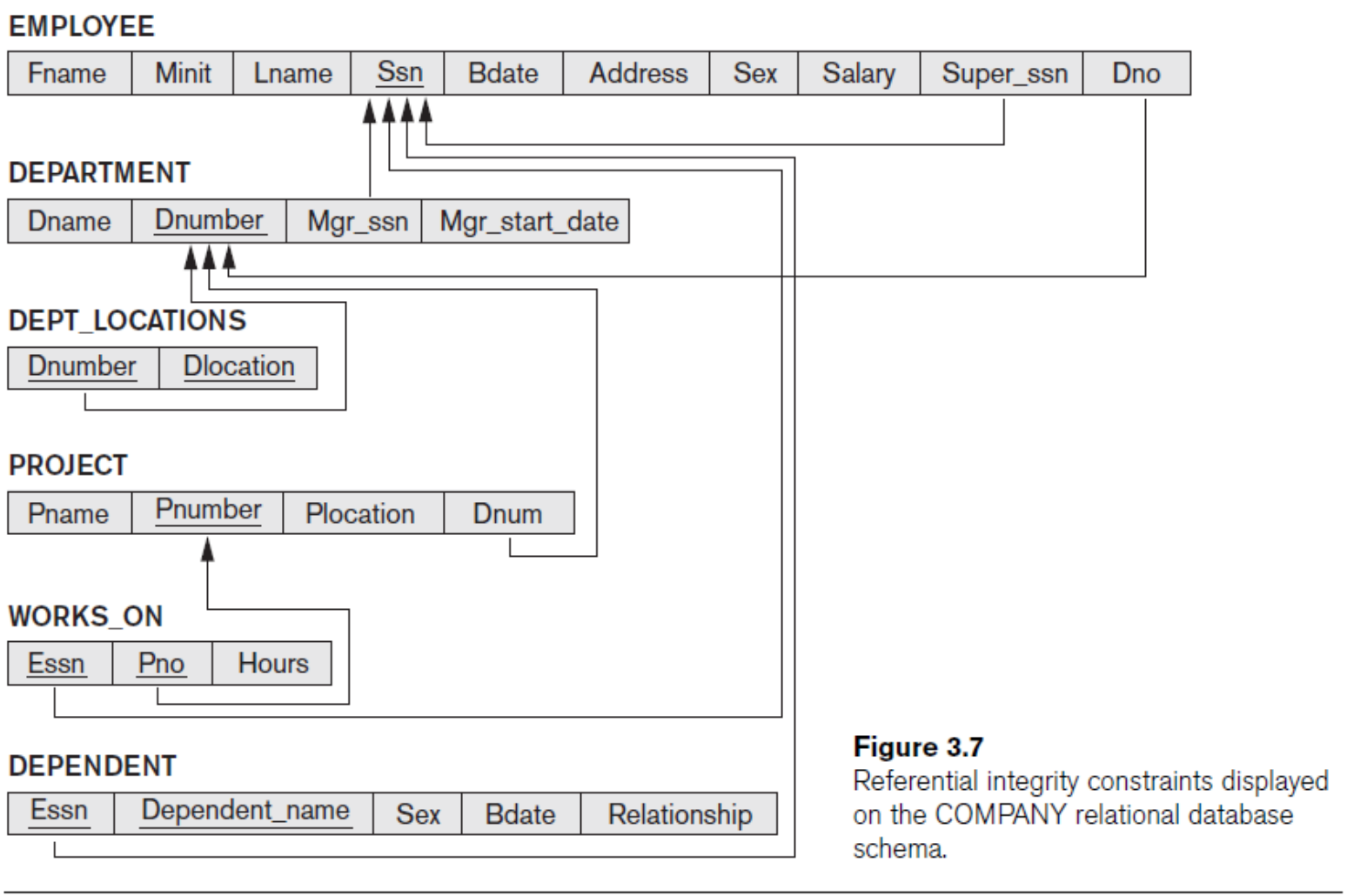


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Figure 4.3
Results of SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q0. (b) Q1. (c) Q2. (d) Q8. (e) Q9. (f) Q10. (g) Q1C.

(a)

Bdate	Address
1965-01-09	731Fondren, Houston, TX

(b)

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

Figure 4.3
Results of SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q0. (b) Q1. (c) Q2. (d) Q8. (e) Q9. (f) Q10. (g) Q1C.

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes
  - As long as the attributes are in different relations
  - Must **qualify** the attribute name with the relation name to prevent ambiguity

```
Q1A:    SELECT    Fname, EMPLOYEE.Name, Address
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DEPARTMENT.Name='Research' AND
                  DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, Renaming, and Tuple Variables

- **Aliases** or **tuple variables**
  - Declare alternative relation names E and S
  - `EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)`

Unspecified WHERE Clause and Use of the Asterisk

- Missing `WHERE` clause
  - Indicates no condition on tuple selection
- `CROSS PRODUCT`
  - All possible tuple combinations

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

```
Q9:   SELECT   Ssn
      FROM     EMPLOYEE;

Q10:  SELECT   Ssn, Dname
      FROM     EMPLOYEE, DEPARTMENT;
```

Unspecified WHERE Clause and Use of the Asterisk (cont'd.)

- Specify an asterisk (*)
  - Retrieve all the attribute values of the selected tuples

```
Q1C:    SELECT    *
        FROM      EMPLOYEE
        WHERE     Dno=5;

Q1D:    SELECT    *
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     Dname='Research' AND Dno=Dnumber;

Q10A:   SELECT    *
        FROM      EMPLOYEE, DEPARTMENT;
```

Tables as Sets in SQL

- SQL does not automatically eliminate duplicate tuples in query results
- Use the keyword **`DISTINCT`** in the `SELECT` clause
  - Only distinct tuples should remain in the result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;

Q11A:   SELECT    DISTINCT Salary
        FROM      EMPLOYEE;
```

Tables as Sets in SQL (cont'd.)

- Set operations
  - UNION, **EXCEPT** (difference), **INTERSECT**
  - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL)

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A: (SELECT    DISTINCT Pnumber
      FROM      PROJECT, DEPARTMENT, EMPLOYEE
      WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                AND Lname='Smith' )
      UNION
     ( SELECT   DISTINCT Pnumber
      FROM      PROJECT, WORKS_ON, EMPLOYEE
      WHERE     Pnumber=Pno AND Essn=Ssn
                AND Lname='Smith' );
```

Substring Pattern Matching and Arithmetic Operators

- **`LIKE`** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
- Standard arithmetic operators:
  - Addition (+), subtraction (–), multiplication (*), and division (/)
- **`BETWEEN`** comparison operator

Ordering of Query Results

- Use **`ORDER BY`** clause
  - Keyword **`DESC`** to see result in a descending order of values
  - Keyword **`ASC`** to specify ascending order explicitly
  - `ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC`

Discussion and Summary of Basic SQL Retrieval Queries

```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

INSERT, DELETE, and UPDATE Statements in SQL

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`

The INSERT Command

- Specify the relation name and a list of values for the tuple

```
U1:   INSERT INTO   EMPLOYEE
      VALUES        ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                      Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

```
U3B:  INSERT INTO   WORKS_ON_INFO ( Emp_name, Proj_name,
                    Hours_per_week )
      SELECT        E.Lname, P.Pname, W.Hours
      FROM          PROJECT P, WORKS_ON W, EMPLOYEE E
      WHERE         P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

The DELETE Command

- Removes tuples from a relation
  - Includes a `WHERE` clause to select the tuples to be deleted

```
U4A:  DELETE FROM   EMPLOYEE
      WHERE         Lname='Brown';

U4B:  DELETE FROM   EMPLOYEE
      WHERE         Ssn='123456789';

U4C:  DELETE FROM   EMPLOYEE
      WHERE         Dno=5;

U4D:  DELETE FROM   EMPLOYEE;
```

The UPDATE Command

- Modify attribute values of one or more selected tuples
- Additional **`SET`** clause in the `UPDATE` command
  - Specifies attributes to be modified and new values

```
U5:   UPDATE   PROJECT
      SET      Plocation = 'Bellaire', Dnum = 5
      WHERE    Pnumber=10;
```

Additional Features of SQL

- Techniques for specifying complex retrieval queries
- Writing programs in various programming languages that include SQL statements
- Set of commands for specifying physical database design parameters, file structures for relations, and access paths
- Transaction control commands

Additional Features of SQL (cont'd.)

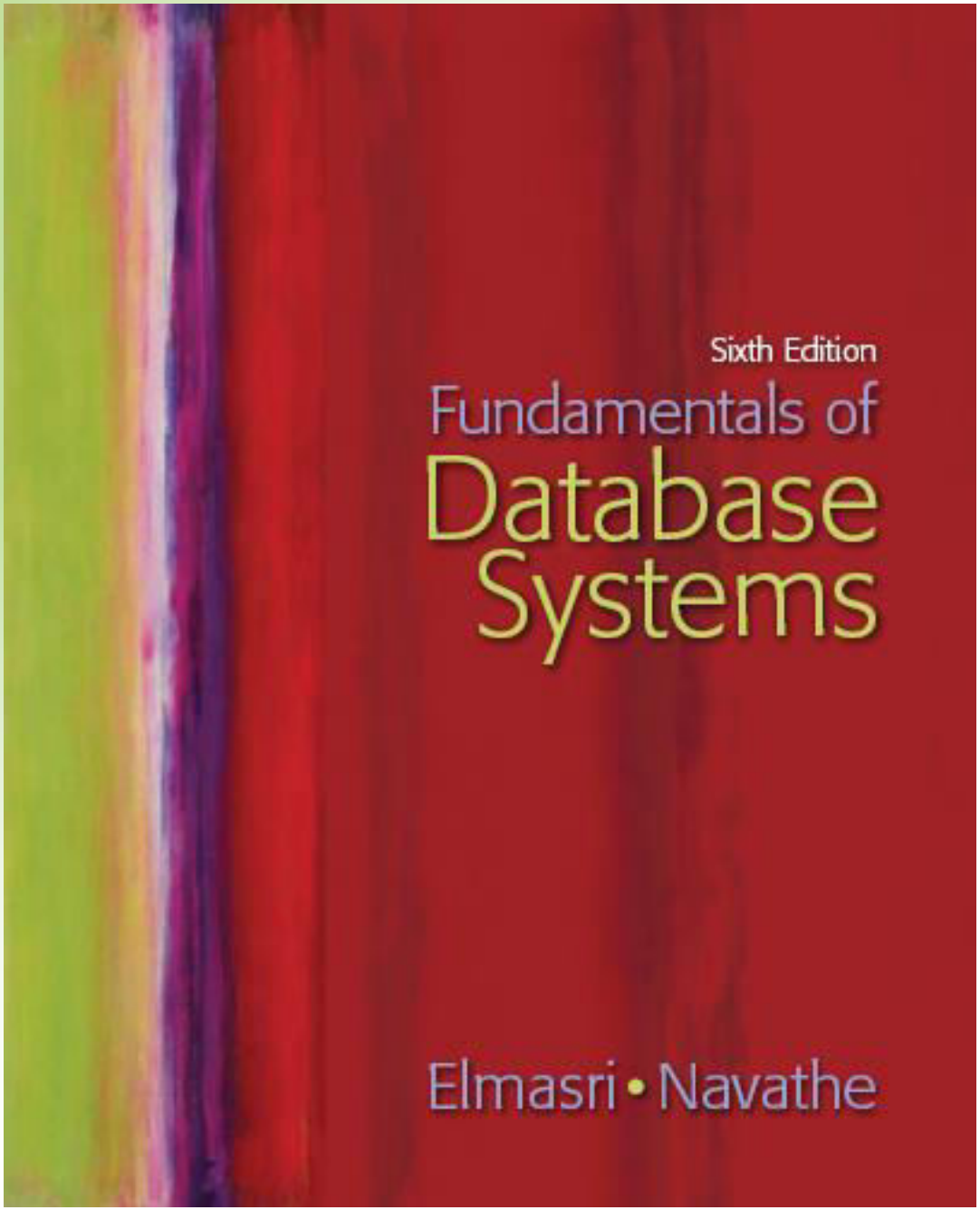
- Specifying the granting and revoking of privileges to users
- Constructs for creating triggers
- Enhanced relational systems known as object-relational
- New technologies such as XML and OLAP

Summary

- SQL
  - Comprehensive language
  - Data definition, queries, updates, constraint specification, and view definition
- Covered in Chapter 4:
  - Data definition commands for creating tables
  - Commands for constraint specification
  - Simple retrieval queries
  - Database update commands

Chapter 5

More SQL: Complex Queries, Triggers, Views, and Schema Modification



Addison-Wesley
is an imprint of
PEARSON

Chapter 5 Outline

- More Complex SQL Retrieval Queries
- Specifying Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, outer joins, aggregate functions, and grouping

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS` or `IS NOT NULL`

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within WHERE clause of another query
  - **Outer query**
- Comparison operator `IN`
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries (cont'd.)

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

Nested Queries (cont'd.)

- Use other comparison operators to compare a single value *v*
  - = ANY (or = SOME) operator
    - Returns TRUE if the value *v* is equal to some value in the set *V* and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL  ( SELECT   Salary
                         FROM     EMPLOYEE
                         WHERE    Dno=5 );
```

Nested Queries (cont'd.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT   E.Fname, E.Lname
      FROM     EMPLOYEE AS E
      WHERE    E.Ssn IN ( SELECT   Essn
                          FROM     DEPENDENT AS D
                          WHERE    E.Fname=D.Dependent_name
                          AND E.Sex=D.Sex );
```

Correlated Nested Queries

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

The EXISTS and UNIQUE Functions in SQL

- `EXISTS` function
  - Check whether the result of a correlated nested query is empty or not
- `EXISTS` and `NOT EXISTS`
  - Typically used in conjunction with a correlated nested query
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause
- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query

```
Q8A:    SELECT    E.Lname AS Employee_name, S.Lname AS Supervisor_name
        FROM      EMPLOYEE AS E, EMPLOYEE AS S
        WHERE     E.Super_ssn=S.Ssn;
```

Joined Tables in SQL and Outer Joins

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table

```
Q1A:  SELECT  Fname, Lname, Address
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```

Joined Tables in SQL and Outer Joins (cont'd.)

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S

Joined Tables in SQL and Outer Joins (cont'd.)

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
  - Every tuple in left table must appear in result
  - If no matching tuple
    - Padded with NULL values for attributes of right table

Joined Tables in SQL and Outer Joins (cont'd.)

- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for the attributes of left table
- FULL OUTER JOIN
- Can nest join specifications

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- **Grouping**
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- Functions can be used in the `SELECT` clause or in a `HAVING` clause

Aggregate Functions in SQL (cont'd.)

- NULL values discarded when aggregate functions are applied to a particular column

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE   Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT  COUNT (*)
      FROM    EMPLOYEE;

Q22:  SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND DNAME='Research';
```

Grouping: The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

- **`HAVING`** clause
  - Provides a condition on the summary information

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:  SELECT   Dnumber, COUNT (*)
      FROM     DEPARTMENT, EMPLOYEE
      WHERE    Dnumber=Dno AND Salary>40000 AND
               ( SELECT       Dno
                 FROM         EMPLOYEE
                 GROUP BY Dno
                 HAVING       COUNT (*) > 5)
```

Discussion and Summary of SQL Queries

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

Specifying Constraints as Assertions and Actions as Triggers

- **`CREATE ASSERTION`**
  - Specify additional types of constraints outside scope of built-in relational model constraints
- **`CREATE TRIGGER`**
  - Specify automatic actions that database system will perform when certain events and conditions occur

Specifying General Constraints as Assertions in SQL

- CREATE ASSERTION
  - Specify a query that selects any tuples that violate the desired condition
  - Use only in cases where it is not possible to use CHECK on attributes and domains

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS  ( SELECT     *
                      FROM       EMPLOYEE E, EMPLOYEE M,
                                 DEPARTMENT D
                      WHERE      E.Salary>M.Salary
                                 AND E.Dno=D.Dnumber
                                 AND D.Mgr_ssn=M.Ssn ) );
```

Introduction to Triggers in SQL

- `CREATE TRIGGER` statement
  - Used to monitor the database
- Typical trigger has three components:
  - **Event(s)**
  - **Condition**
  - **Action**

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table

Specification of Views in SQL

- **`CREATE VIEW`** command
  - Give table name, list of attribute names, and a query to specify the contents of the view

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
          FROM      EMPLOYEE, PROJECT, WORKS_ON
          WHERE     Ssn=Essn AND Pno=Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
          FROM      DEPARTMENT, EMPLOYEE
          WHERE     Dnumber=Dno
          GROUP BY  Dname;
```

Specification of Views in SQL (cont'd.)

- Specify SQL queries on a view
- View always up-to-date
  - Responsibility of the DBMS and not the user
- **`DROP VIEW`** command
  - Dispose of a view

View Implementation, View Update, and Inline Views

- Complex problem of efficiently implementing a view for querying
- **Query modification** approach
  - Modify view query into a query on underlying base tables
  - Disadvantage: inefficient for views defined via complex queries that are time-consuming to execute

View Implementation

- **View materialization approach**
  - Physically create a temporary view table when the view is first queried
  - Keep that table on the assumption that other queries on the view will follow
  - Requires efficient strategy for automatically updating the view table when the base tables are updated

View Implementation (cont'd.)

- **Incremental update strategies**
  - DBMS determines what new tuples must be inserted, deleted, or modified in a materialized view table

View Update and Inline Views

- Update on a view defined on a single table without any aggregate functions
  - Can be mapped to an update on underlying base table
- View involving joins
  - Often not possible for DBMS to determine which of the updates is intended

View Update and Inline Views (cont'd.)

- Clause **`WITH CHECK OPTION`**
  - Must be added at the end of the view definition if a view is to be updated
- **In-line view**
  - Defined in the `FROM` clause of an SQL query

Schema Change Statements in SQL

- **Schema evolution commands**
  - Can be done while the database is operational
  - Does not require recompilation of the database schema

The DROP Command

- `DROP` command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP SCHEMA COMPANY CASCADE;`

The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- Example:
  - `ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);`
- To drop a column
  - Choose either `CASCADE` or `RESTRICT`

The ALTER Command (cont'd.)

- Change constraints specified on a table
  - Add or drop a named constraint

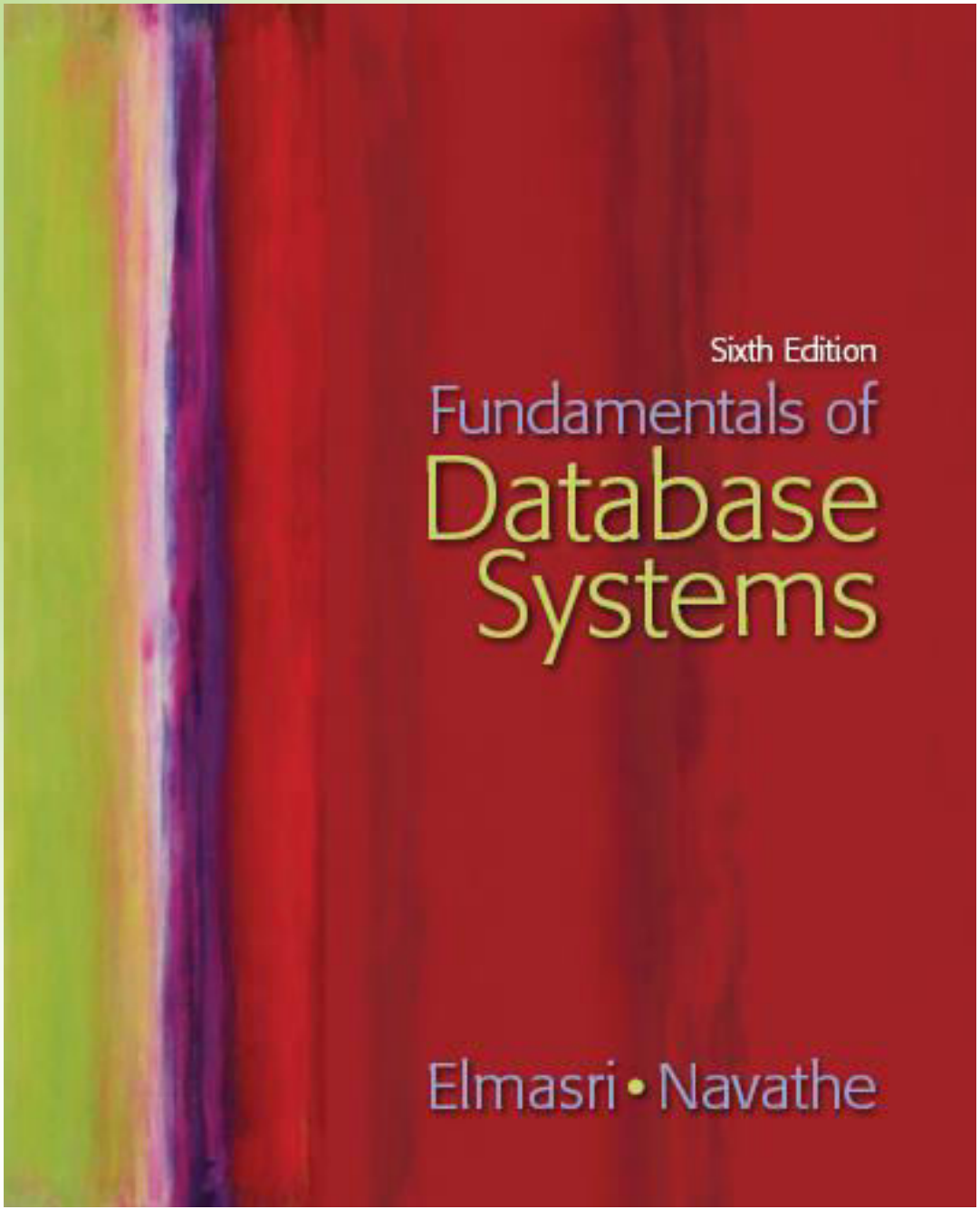
```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Summary

- Complex SQL:
  - Nested queries, joined tables, outer joins, aggregate functions, grouping
- `CREATE ASSERTION` and `CREATE TRIGGER`
- Views
  - Virtual or derived tables

Chapter 7

Data Modeling Using the Entity-Relationship (ER) Model



Addison-Wesley
is an imprint of
PEARSON

Chapter 7 Outline

- Using High-Level Conceptual Data Models for Database Design
- A Sample Database Application
- Entity Types, Entity Sets, Attributes, and Keys
- Relationship Types, Relationship Sets, Roles, and Structural Constraints
- Weak Entity Types

Chapter 7 Outline (cont'd.)

- Refining the ER Design for the COMPANY Database
- ER Diagrams, Naming Conventions, and Design Issues
- Example of Other Notation: UML Class Diagrams
- Relationship Types of Degree Higher than Two

Data Modeling Using the Entity-Relationship (ER) Model

- **Entity-Relationship (ER) model**
  - Popular high-level conceptual data model
- **ER diagrams**
  - Diagrammatic notation associated with the ER model
- **Unified Modeling Language (UML)**

Using High-Level Conceptual Data Models for Database Design

- **Requirements collection and analysis**
  - Database designers interview prospective database users to understand and document data requirements
  - Result: **data requirements**
  - **Functional requirements** of the application

Using High-Level Conceptual Data Models (cont'd.)

- **Conceptual schema**
  - Conceptual design
  - Description of data requirements
  - Includes detailed descriptions of the entity types, relationships, and constraints
  - Transformed from high-level data model into implementation data model

Using High-Level Conceptual Data Models (cont'd.)

- **Logical design** or **data model mapping**
  - Result is a database schema in implementation data model of DBMS
- **Physical design phase**
  - Internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files specified

A Sample Database Application

- COMPANY
  - Employees, departments, and projects
  - Company is organized into departments
  - Department controls a number of projects
  - Employee: store each employee's name, Social Security number, address, salary, sex (gender), and birth date
  - Keep track of the dependents of each employee

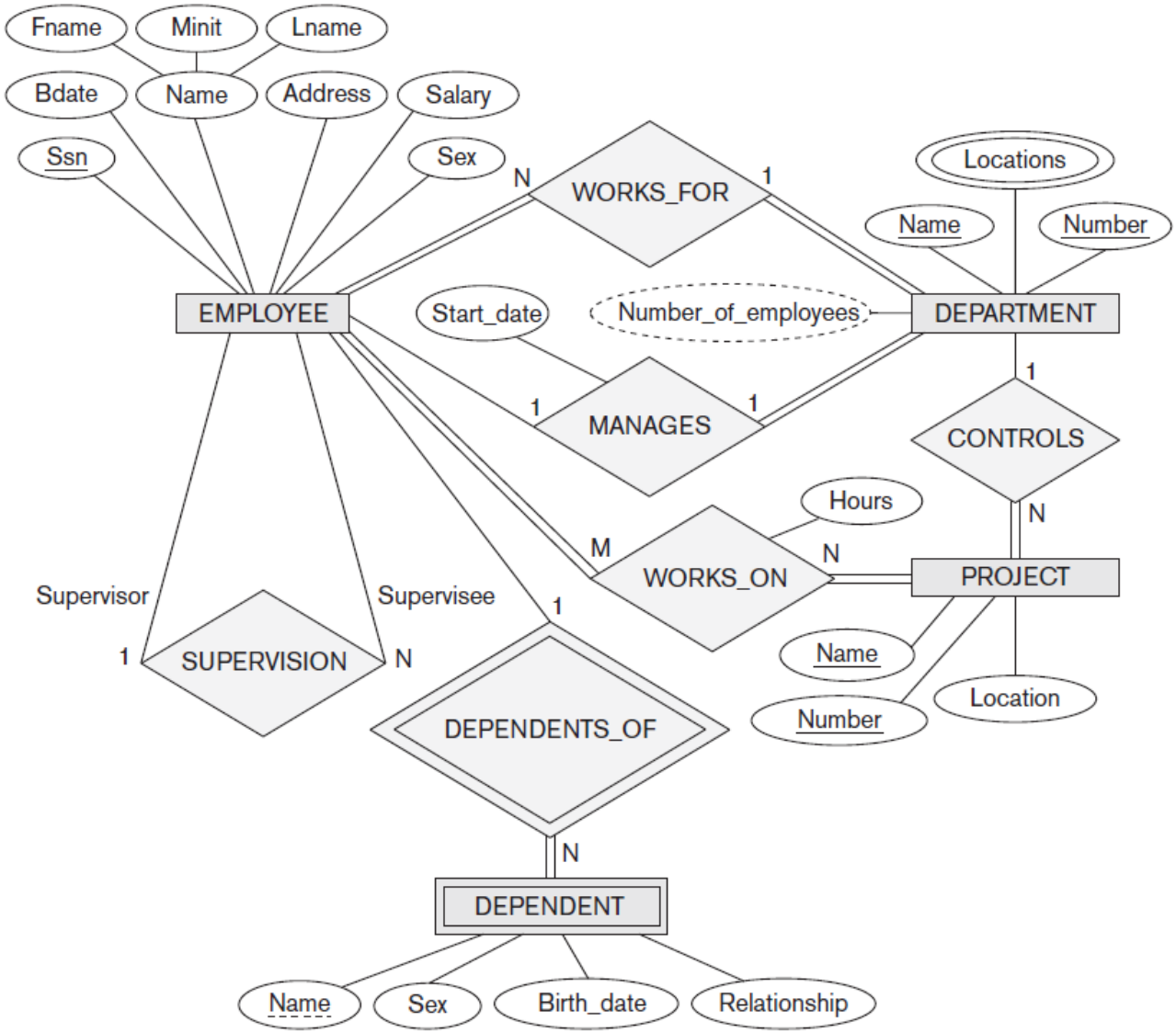


Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

Entity Types, Entity Sets, Attributes, and Keys

- ER model describes data as:
  - Entities
  - Relationships
  - Attributes

Entities and Attributes

- **Entity**
  - Thing in real world with independent existence
- **Attributes**
  - Particular properties that describe entity
  - Types of attributes:
    - *Composite* versus *simple* (atomic) *attributes*
    - **Single-valued** versus **multivalued** attributes
    - **Stored** versus **derived** attributes
    - **NULL** values
    - **Complex** attributes

Entities and Attributes (cont'd.)

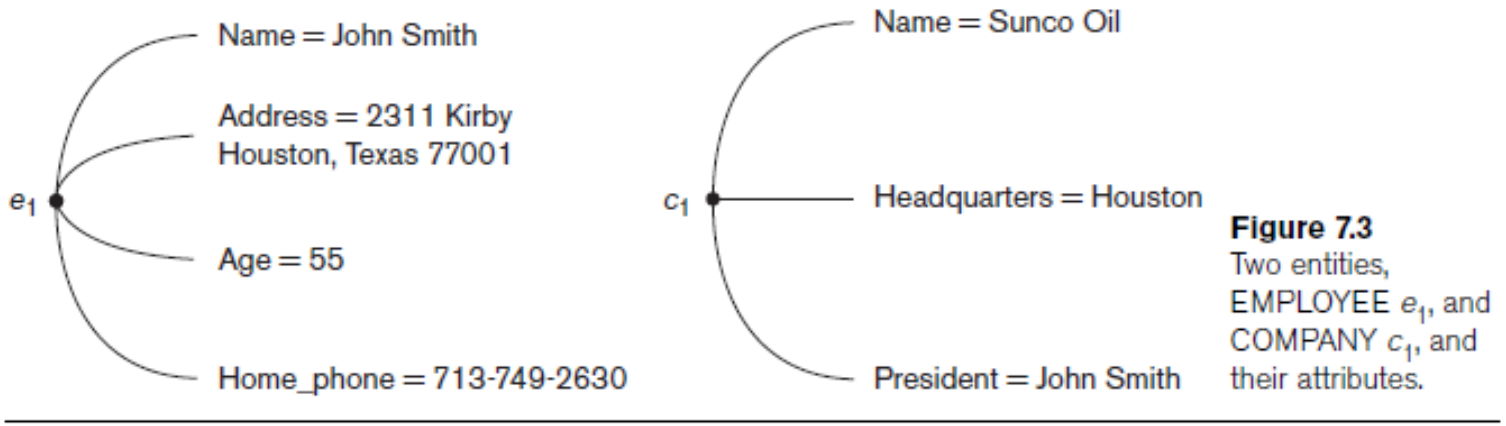


Figure 7.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

Entity Types, Entity Sets, Keys, and Value Sets

- **Entity type**
  - Collection (or set) of entities that have the same attributes

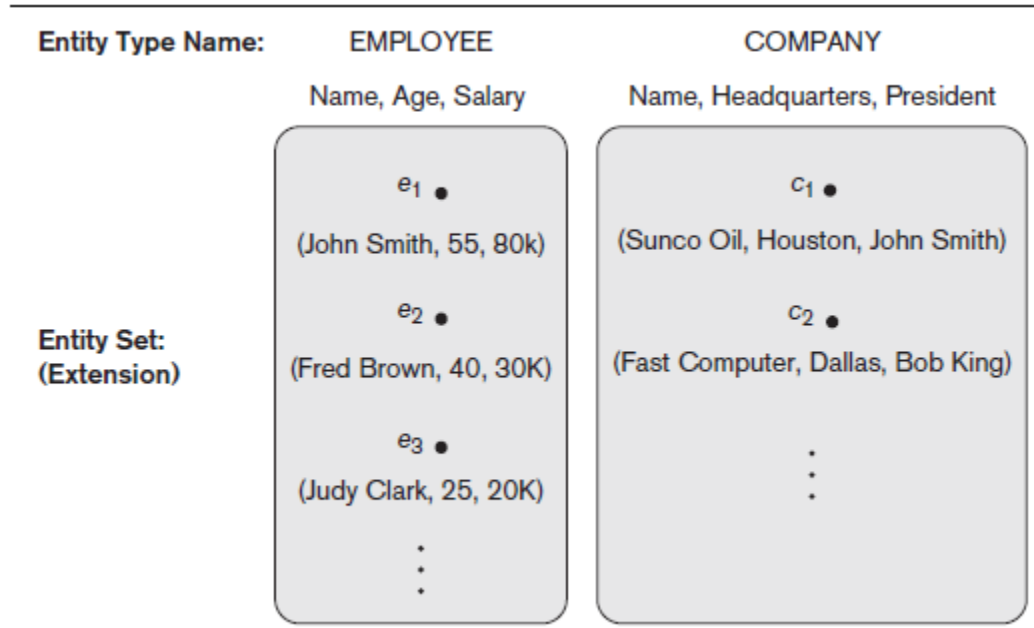


Figure 7.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

Entity Types, Entity Sets, Keys, and Value Sets (cont'd.)

- **Key** or **uniqueness constraint**
  - Attributes whose values are distinct for each individual entity in entity set
  - **Key attribute**
    - Uniqueness property must hold for every entity set of the entity type
- **Value sets** (or **domain of values**)
  - Specifies set of values that may be assigned to that attribute for each individual entity

Initial Conceptual Design of the COMPANY Database

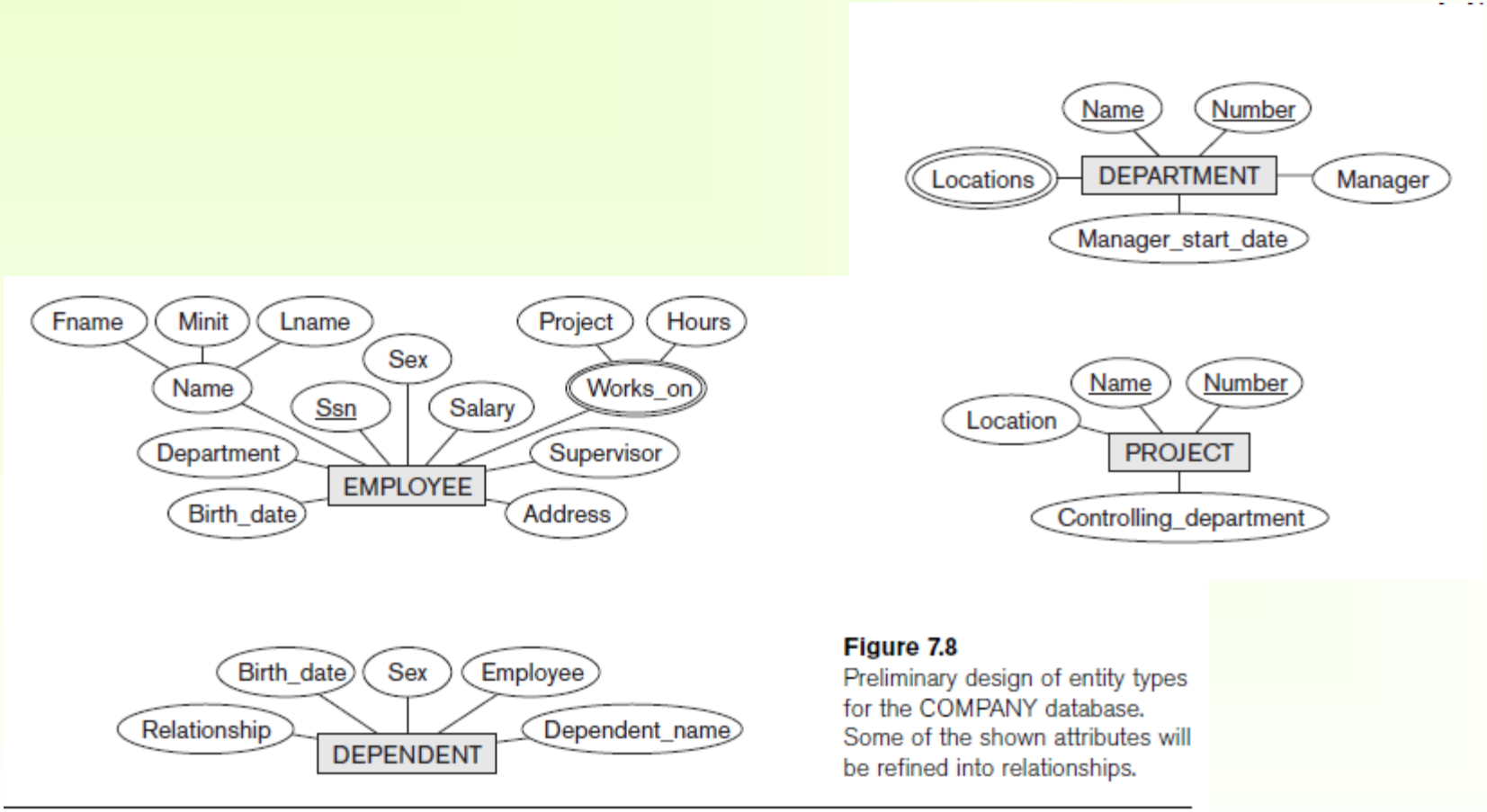


Figure 7.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship Types, Relationship Sets, Roles, and Structural Constraints

- **Relationship**
  - When an attribute of one entity type refers to another entity type
  - Represent references as relationships not attributes

Relationship Types, Sets, and Instances

- **Relationship type** R among n entity types $E_1, E_2, ..., E_n$
  - Defines a set of associations among entities from these entity types
- **Relationship instances** r_i
  - Each r_i associates n individual entities ($e_1, e_2, ..., e_n$)
  - Each entity e_j in r_i is a member of entity set E_j

Relationship Degree

- **Degree** of a relationship type
  - Number of participating entity types
  - **Binary**, **ternary**
- Relationships as attributes
  - Think of a binary relationship type in terms of attributes

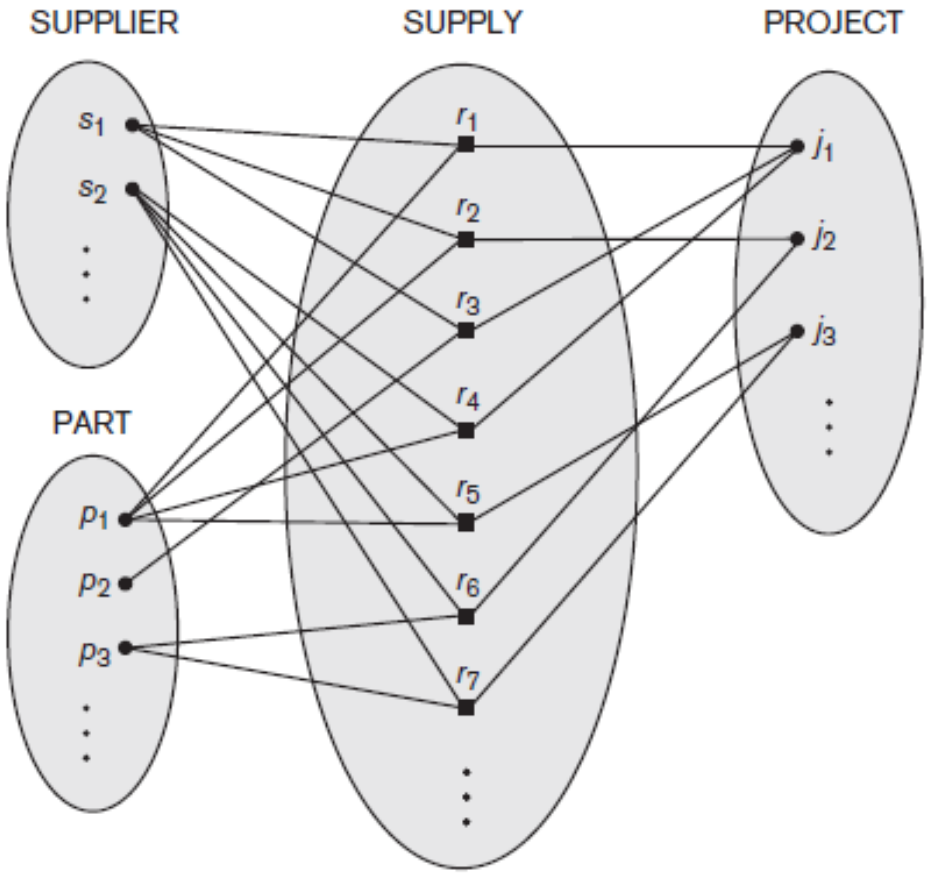


Figure 7.10
Some relationship instances in the SUPPLY ternary relationship set.

Role Names and Recursive Relationships

- **Role names** and recursive relationships
  - Role name signifies role that a participating entity plays in each relationship instance
- **Recursive** relationships
  - Same entity type participates more than once in a relationship type in different roles
  - Must specify role name

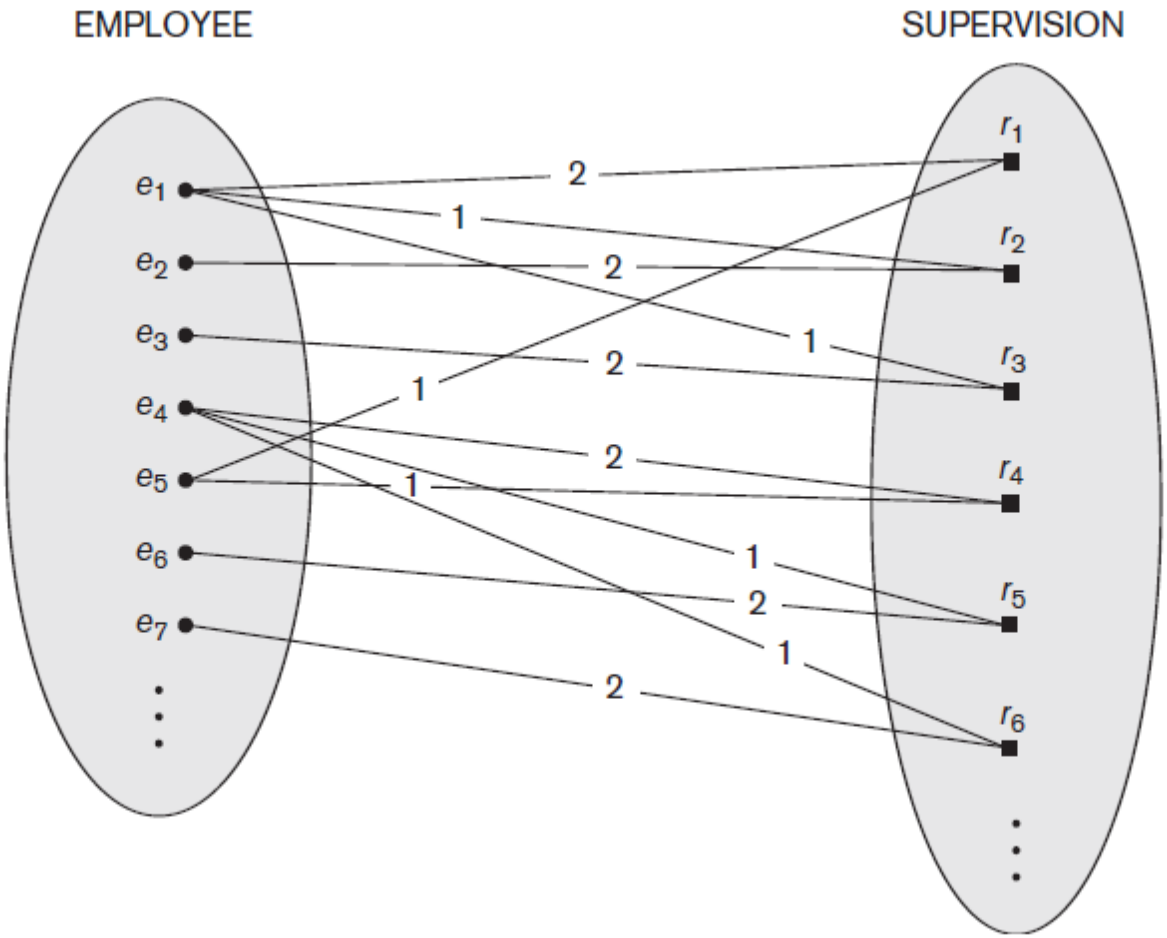


Figure 7.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Constraints on Binary Relationship Types

- **Cardinality ratio** for a binary relationship
  - Specifies maximum number of relationship instances that entity can participate in
- **Participation constraint**
  - Specifies whether existence of entity depends on its being related to another entity
  - Types: **total** and **partial**

Attributes of Relationship Types

- Attributes of 1:1 or 1:N relationship types can be migrated to one entity type
- For a 1:N relationship type
  - Relationship attribute can be migrated only to entity type on N-side of relationship
- For M:N relationship types
  - Some attributes may be determined by combination of participating entities
  - Must be specified as relationship attributes

Weak Entity Types

- Do not have key attributes of their own
  - Identified by being related to specific entities from another entity type
- **Identifying relationship**
  - Relates a weak entity type to its owner
- Always has a total participation constraint

Refining the ER Design for the COMPANY Database

- Change attributes that represent relationships into relationship types
- Determine cardinality ratio and participation constraint of each relationship type

ER Diagrams, Naming Conventions, and Design Issues

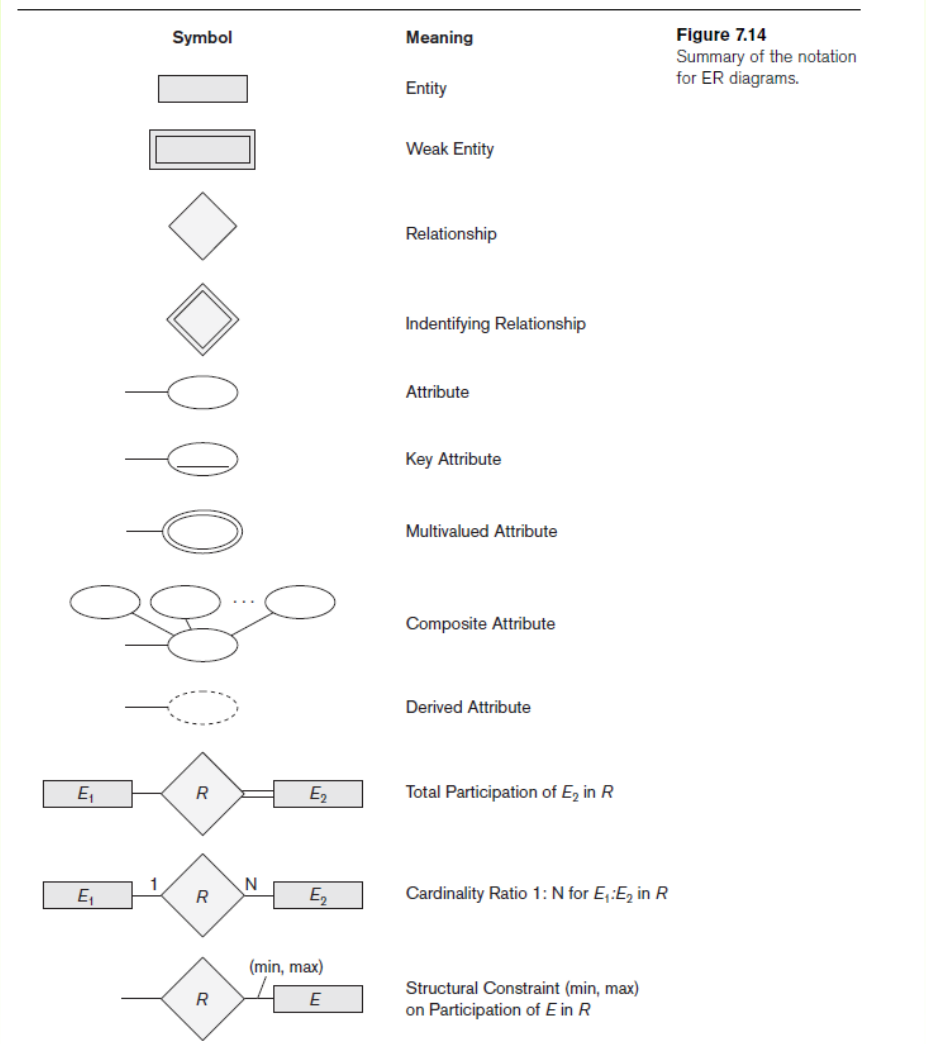


Figure 7.14
Summary of the notation for ER diagrams.

Proper Naming of Schema Constructs

- Choose names that convey meanings attached to different constructs in schema
- Nouns give rise to entity type names
- Verbs indicate names of relationship types
- Choose binary relationship names to make ER diagram readable from left to right and from top to bottom

Design Choices for ER Conceptual Design

- Model concept first as an attribute
  - Refined into a relationship if attribute is a reference to another entity type
- Attribute that exists in several entity types may be elevated to an independent entity type
  - Can also be applied in the inverse

Alternative Notations for ER Diagrams

- Specify structural constraints on relationships
  - Replaces cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints
  - Associate a pair of integer numbers (min, max) with each participation of an entity type *E* in a relationship type *R*, where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$

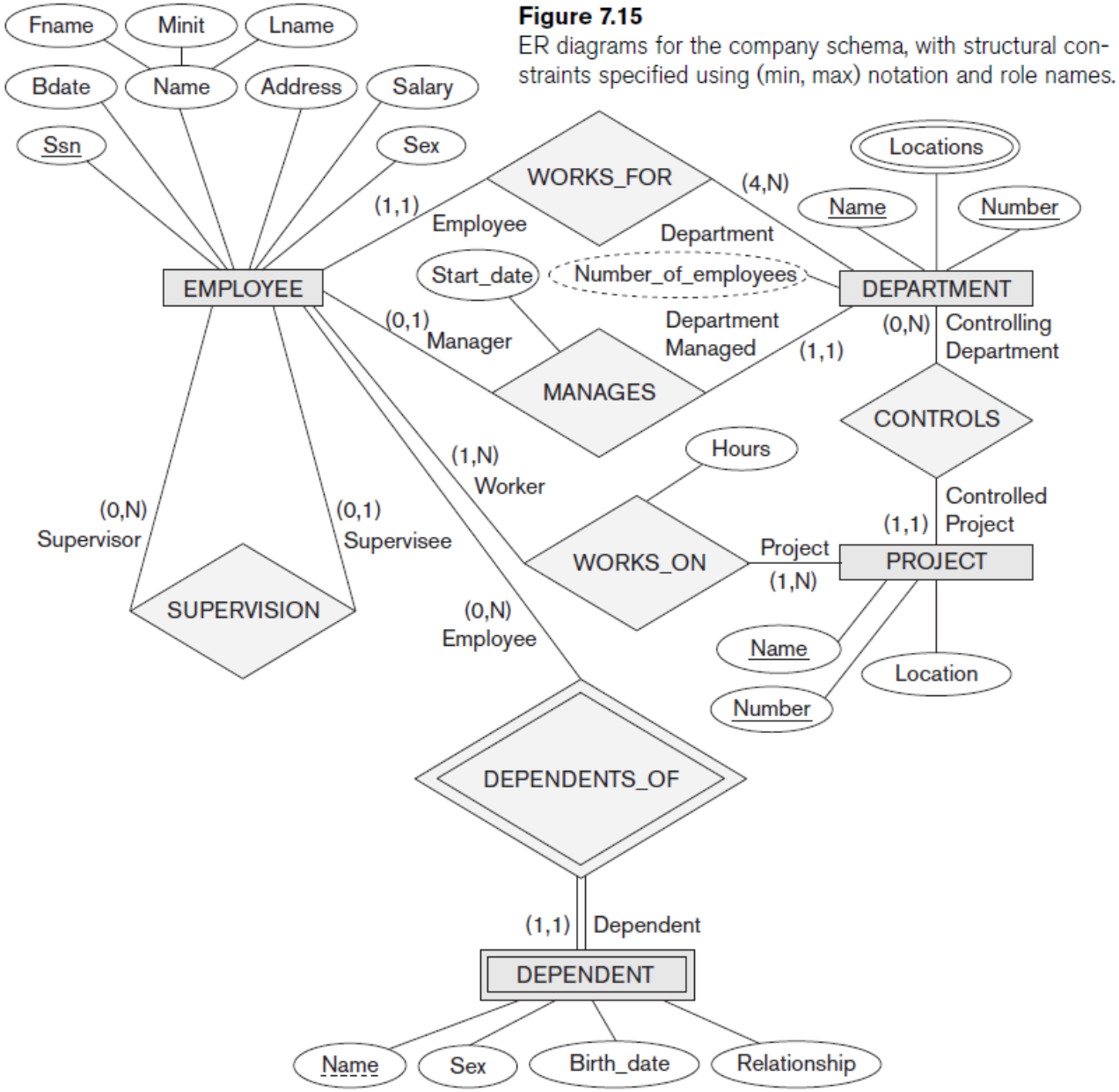
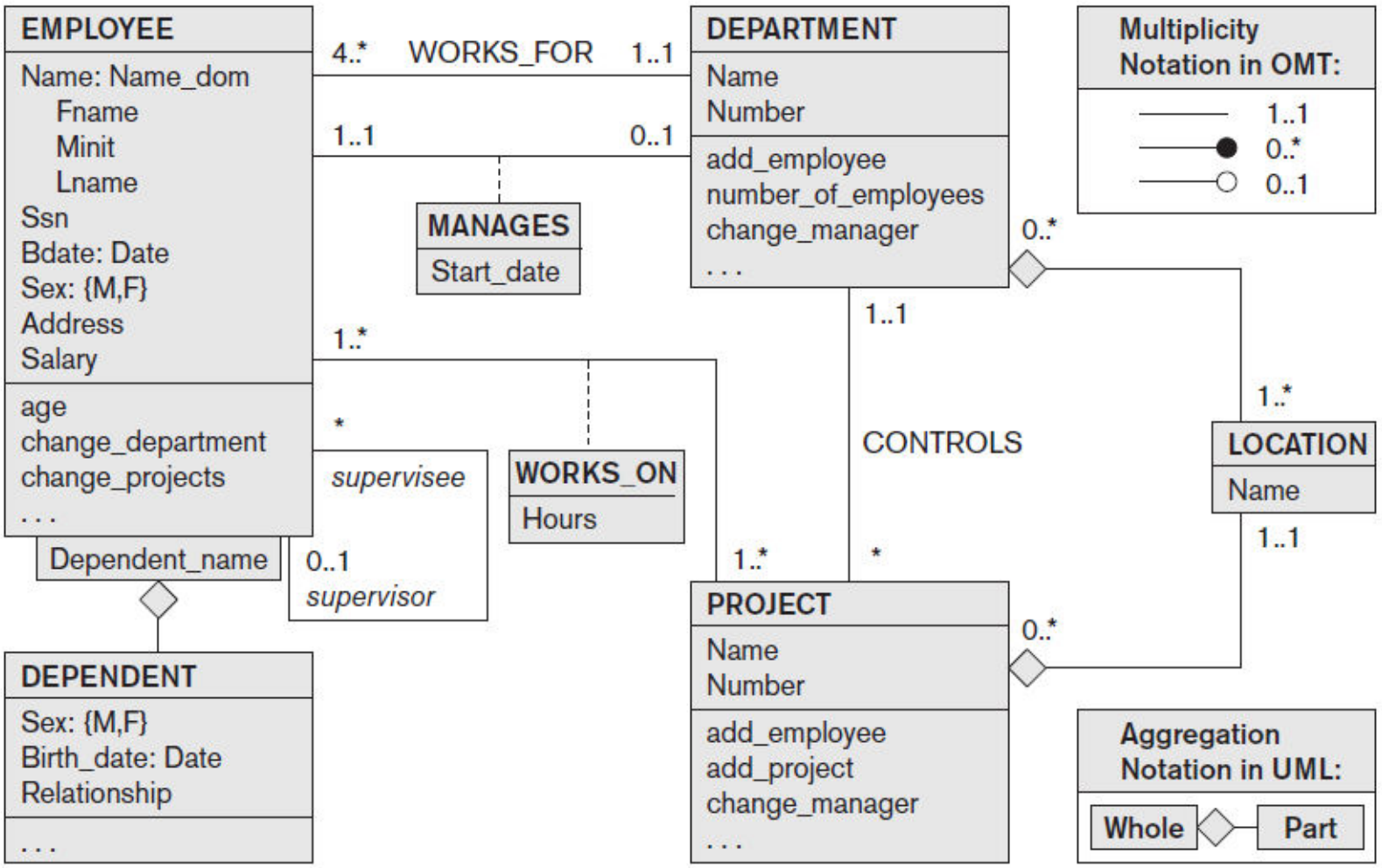


Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Example of Other Notation: UML Class Diagrams

- UML methodology
  - Used extensively in software design
  - Many types of diagrams for various software design purposes
- UML class diagrams
  - Entity in ER corresponds to an object in UML

Figure 7.16
The COMPANY conceptual schema in UML class diagram notation.



Example of Other Notation: UML Class Diagrams (cont'd.)

- **Class** includes three sections:
  - Top section gives the class name
  - Middle section includes the attributes;
  - Last section includes operations that can be applied to individual objects

Example of Other Notation: UML Class Diagrams (cont'd.)

- **Associations**: relationship types
- **Relationship instances**: links
- Binary association
  - Represented as a line connecting participating classes
  - May optionally have a name
- Link attribute
  - Placed in a box connected to the association's line by a dashed line

Example of Other Notation: UML Class Diagrams (cont'd.)

- **Multiplicities**: min..max, asterisk (*) indicates no maximum limit on participation
- Types of relationships: **association** and **aggregation**
- Distinguish between **unidirectional** and **bidirectional** associations
- Model weak entities using **qualified association**

Relationship Types of Degree Higher than Two

- **Degree** of a relationship type
  - Number of participating entity types
- *Binary*
  - Relationship type of degree two
- *Ternary*
  - Relationship type of degree three

Choosing between Binary and Ternary (or Higher-Degree) Relationships

- Some database design tools permit only binary relationships
  - Ternary relationship must be represented as a weak entity type
  - No partial key and three identifying relationships
- Represent ternary relationship as a regular entity type
  - By introducing an artificial or surrogate key

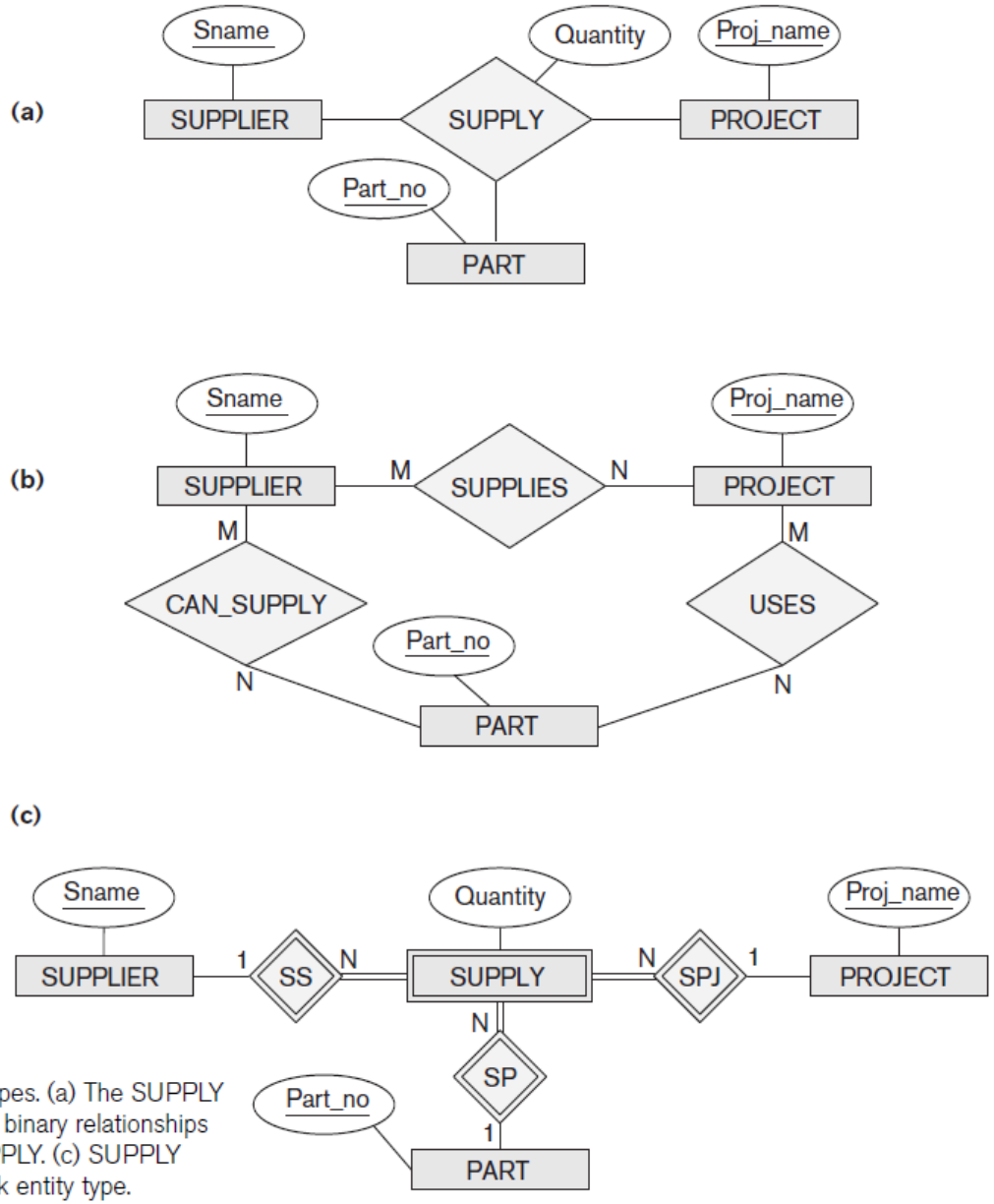


Figure 7.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Constraints on Ternary (or Higher-Degree) Relationships

- Notations for specifying structural constraints on *n*-ary relationships
  - Should both be used if it is important to fully specify structural constraints

Summary

- Basic ER model concepts of entities and their attributes
  - Different types of attributes
  - Structural constraints on relationships
- ER diagrams represent E-R schemas
- UML class diagrams relate to ER modeling concepts