

# **Chapter 1**

## **Databases and Database Users**

Sixth Edition  
**Fundamentals of  
Database  
Systems**

**Elmasri • Navathe**

**Addison-Wesley**  
is an imprint of

**PEARSON**

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

# 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
- **Mineworld 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

# Typical DBMS Functionality

- *Define* a particular database in terms of its data types, structures, and constraints
- *Construct* or Load the initial database contents on a secondary storage medium
- *Manipulating* the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing the database through Web applications
- *Processing* and *Sharing* by a set of concurrent users and application programs – yet, keeping all data valid and consistent

# Typical DBMS Functionality

- Other features:
  - Protection or Security measures to prevent unauthorized access
  - “Active” processing to take internal actions on data
  - Presentation and Visualization of data
  - Maintaining the database and associated programs over the lifetime of the database application
    - Called database, software, and system maintenance

# 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.

# 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

**Figure 1.2**  
A database that stores student and course information.

**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

## 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.

# Main Characteristics of the Database Approach

- **Self-describing nature of a database system:**
  - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
  - The description is called **meta-data**.
  - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data:**
  - Called **program-data independence**.
  - Allows changing data structures and storage organization without having to change the DBMS access programs.

# Example of a simplified database catalog

## 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.

# Main Characteristics of the Database Approach (continued)

- **Data Abstraction:**
  - A **data model** is used to hide storage details and present the users with a conceptual view of the database.
  - Programs refer to the data model constructs rather than data storage details
- **Support of multiple views of the data:**
  - Each user may see a different view of the database, which describes **only** the data of interest to that user.

# Main Characteristics of the Database Approach (continued)

- **Sharing of data and multi-user transaction processing:**
  - Allowing a set of **concurrent users** to retrieve from and to update the database.
  - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
  - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database
  - **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

# Database Users

- Users may be divided into
  - Those who actually use and control the database content, and those who design, develop and maintain database applications (called “Actors on the Scene”), and
  - Those who design and develop the DBMS software and related tools, and the computer systems operators (called “Workers Behind the Scene”).

# Database Users

- Actors on the scene

- **Database administrators:**

- Responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software and hardware resources, controlling its use and monitoring efficiency of operations.

- **Database Designers:**

- Responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.

# Categories of End-users

- Actors on the scene (continued)
  - **End-users:** They use the data for queries, reports and some of them update the database content. End-users can be categorized into:
    - **Casual:** access database occasionally when needed
    - **Naïve** or Parametric: they make up a large section of the end-user population.
      - They use previously well-defined functions in the form of “canned transactions” against the database.
      - Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

# Categories of End-users (continued)

- **Sophisticated:**

- These include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities.
- Many use tools in the form of software packages that work closely with the stored database.

- **Stand-alone:**

- Mostly maintain personal databases using ready-to-use packaged applications.
- An example is a tax program user that creates its own internal database.
- Another example is a user that maintains an address book

# 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 Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
  - Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing Storage Structures (e.g. indexes) for efficient Query Processing

# Advantages of Using the Database Approach (continued)

- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing inferences and actions from the stored data using deductive and active rules

# Additional Implications of Using the Database Approach (continued)

- Flexibility to change data structures:
  - Database structure may evolve as new requirements are defined.
- Availability of current information:
  - Extremely important for on-line transaction systems such as airline, hotel, car reservations.
- Economies of scale:
  - Wasteful overlap of resources and personnel can be avoided by consolidating data and applications across departments.

# Historical Development of Database Technology

- Early Database Applications:
  - The Hierarchical and Network Models were introduced in mid 1960s and dominated during the seventies.
  - A bulk of the worldwide database processing still occurs using these models, particularly, the hierarchical model.
- Relational Model based Systems:
  - Relational model was originally introduced in 1970, was heavily researched and experimented within IBM Research and several universities.
  - Relational DBMS Products emerged in the early 1980s.

# Historical Development of Database Technology (continued)

- Object-oriented and emerging applications:
  - Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.
    - Their use has not taken off much.
  - Many relational DBMSs have incorporated object database concepts, leading to a new category called *object-relational* DBMSs (ORDBMSs)
  - *Extended relational* systems add further capabilities (e.g. for multimedia data, XML, and other data types)

# Historical Development of Database Technology (continued)

- Data on the Web and E-commerce Applications:
  - Web contains data in HTML (Hypertext markup language) with links among pages.
  - This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language). (see Ch. 27).
  - Script programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database (see Ch. 26).
    - Also allow database updates through Web pages

# Extending Database Capabilities

- New functionality is being added to DBMSs in the following areas:
  - Scientific Applications
  - XML (eXtensible Markup Language)
  - Image Storage and Management
  - Audio and Video Data Management
  - Data Warehousing and Data Mining
  - Spatial Data Management
  - Time Series and Historical Data Management
- The above gives rise to *new research and development* in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.

# When not to use a DBMS

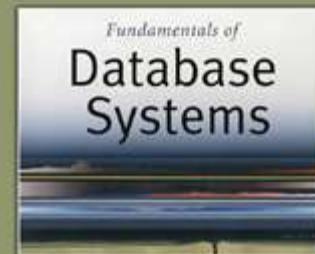
- Main inhibitors (costs) of using a DBMS:
  - High initial investment and possible need for additional hardware.
  - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- When a DBMS may be unnecessary:
  - If the database and applications are simple, well defined, and not expected to change.
  - If there are stringent real-time requirements that may not be met because of DBMS overhead.
  - If access to data by multiple users is not required.

# When not to use a DBMS

- When no DBMS may suffice:
  - If the database system is not able to handle the complexity of data because of modeling limitations
  - If the database users need special operations not supported by the DBMS.

# Chapter 2

## Database System Concepts and Architecture



Elmasri / Navathe

# Outline

- Data Models and Their Categories
- History of Data Models
- Schemas, Instances, and States
- Three-Schema Architecture
- Data Independence
- DBMS Languages and Interfaces
- Database System Utilities and Tools
- Centralized and Client-Server Architectures
- Classification of DBMSs

# Data Models

- **Data Model:**
  - A set of concepts to describe the ***structure*** of a database, the ***operations*** for manipulating these structures, and certain ***constraints*** that the database should obey.
- **Data Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include ***elements*** (and their ***data types***) as well as groups of elements (e.g. ***entity, record, table***), and ***relationships*** among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times

# Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many users perceive data.
    - (Also called *entity-based* or *object-based* data models.)
- **Physical (low-level, internal) data models:**
  - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Implementation (representational) data models:**
  - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

# Schemas versus Instances

- Database Schema:
  - The ***description*** of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- Schema Diagram:
  - An ***illustrative*** display of (most aspects of) a database schema.
- Schema Construct:
  - A ***component*** of the schema or an object within the schema, e.g., STUDENT, COURSE.

# Schemas versus Instances

## ■ Database State:

- The actual data stored in a database at a ***particular moment in time***. This includes the collection of all the data in the database.
- Also called database instance (or occurrence or snapshot).
  - The term *instance* is also applied to individual database components, e.g. *record instance*, *table instance*, *entity instance*

# Database Schema vs. Database State

- Database State:
  - Refers to the ***content*** of a database at a moment in time.
- Initial Database State:
  - Refers to the database state when it is initially loaded into the system.
- Valid State:
  - A state that satisfies the structure and constraints of the database.

# Database Schema vs. Database State (continued)

- Distinction
  - The ***database schema*** changes very infrequently.
  - The ***database state*** changes every time the database is updated.
- **Schema** is also called **intension**.
- **State** is also called **extension**.

# Example of a Database Schema

## 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
----------------	--------------------	-------

**Figure 2.1**

Schema diagram for the database in Figure 1.2.

# Example of a database state

**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	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	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.

# Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program-data independence.**
  - Support of **multiple views** of the data.
- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization

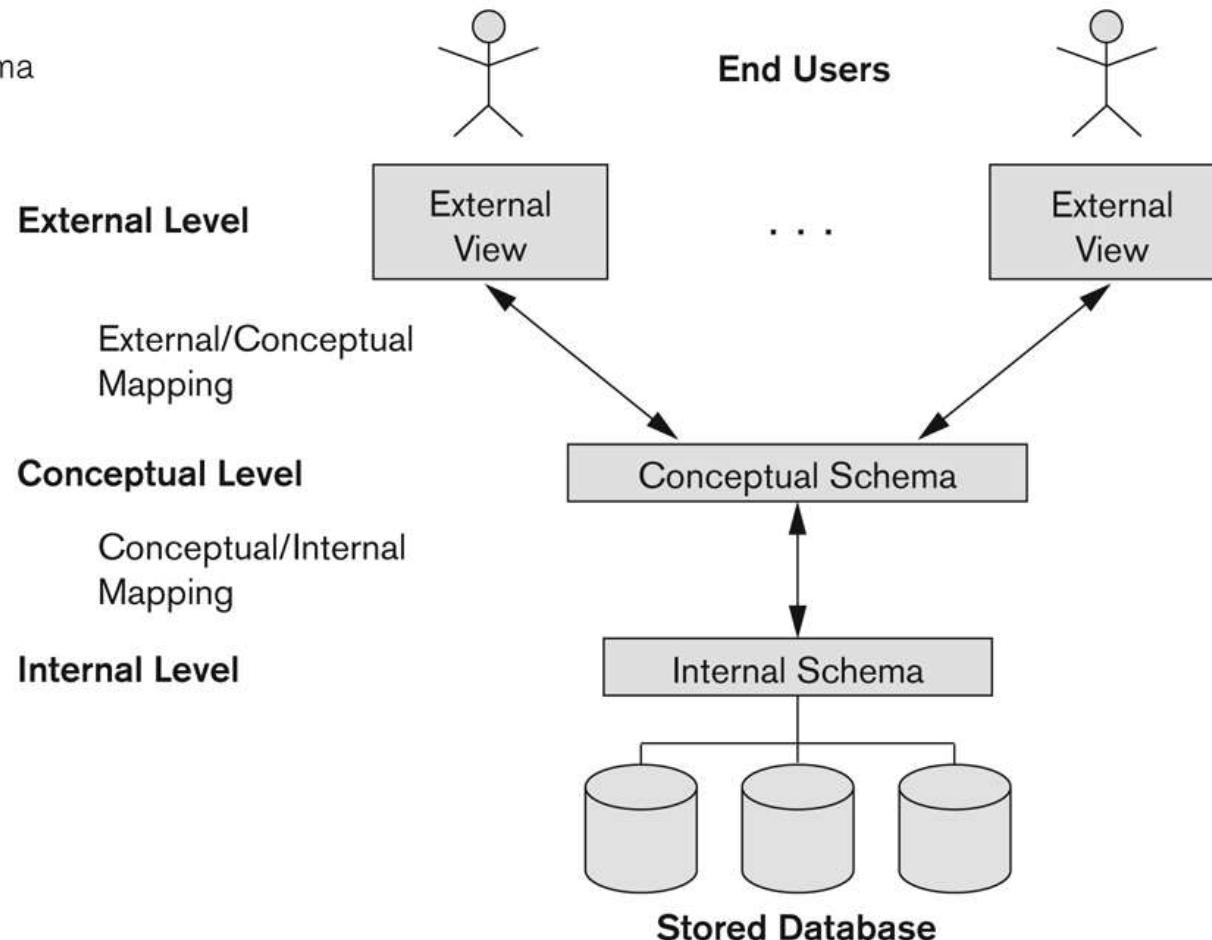
# Three-Schema Architecture

- Defines DBMS schemas at *three* levels:
  - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g indexes).
    - Typically uses a **physical** data model.
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
    - Uses a **conceptual** or an **implementation** data model.
  - **External schemas** at the external level to describe the various user views.
    - Usually uses the same data model as the conceptual schema.

# The three-schema architecture

**Figure 2.2**

The three-schema architecture.



# Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

# Data Independence

- **Logical Data Independence:**
  - The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.
- **Physical Data Independence:**
  - The capacity to change the internal schema without having to change the conceptual schema.
  - For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance

# Data Independence (continued)

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.
- The higher-level schemas themselves are **unchanged**.
  - Hence, the application programs need not be changed since they refer to the external schemas.

# DBMS Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language

# DBMS Languages

## ■ Data Definition Language (DDL):

- Used by the DBA and database designers to specify the conceptual schema of a database.
- In many DBMSs, the DDL is also used to define internal and external schemas (views).
- In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.
  - SDL is typically realized via DBMS commands provided to the DBA and database designers

# DBMS Languages

- **Data Manipulation Language (DML):**
  - Used to specify database retrievals and updates
  - DML commands (data sublanguage) can be *embedded* in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
    - A library of functions can also be provided to access the DBMS from a programming language
  - Alternatively, stand-alone DML commands can be applied directly (called a *query language*).

# Types of DML

- **High Level or Non-procedural Language:**
  - For example, the SQL relational language
  - Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
  - Also called **declarative** languages.
- **Low Level or Procedural Language:**
  - Retrieve data one record-at-a-time;
  - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

# DBMS Interfaces

- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- GUI-based
  - (Point and Click, Drag and Drop, etc.)
- Natural language: requests in written English
- Combinations of the above:
  - For example, both menus and forms used extensively in Web database interfaces

# Other DBMS Interfaces

- Speech as Input and Output
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
  - Creating user accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access paths

# Database System Utilities

- To perform certain functions such as:
  - Loading data stored in files into a database.  
Includes data conversion tools.
  - Backing up the database periodically on tape.
  - Reorganizing database file structures.
  - Report generation utilities.
  - Performance monitoring utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.

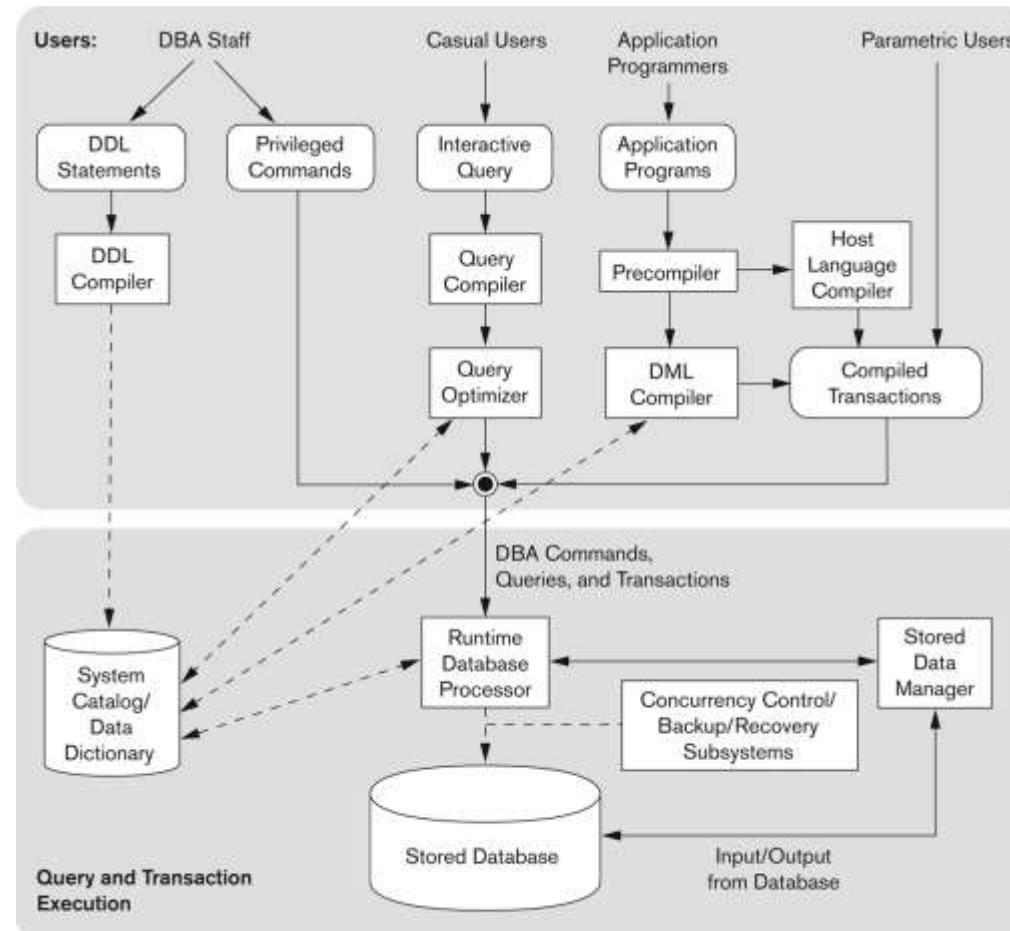
# Other Tools

- Data dictionary / repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - **Active data dictionary** is accessed by DBMS software and users/DBA.
  - **Passive data dictionary** is accessed by users/DBA only.

# Other Tools

- Application Development Environments and CASE (computer-aided software engineering) tools:
- Examples:
  - PowerBuilder (Sybase)
  - JBuilder (Borland)
  - JDeveloper 10G (Oracle)

# Typical DBMS Component Modules



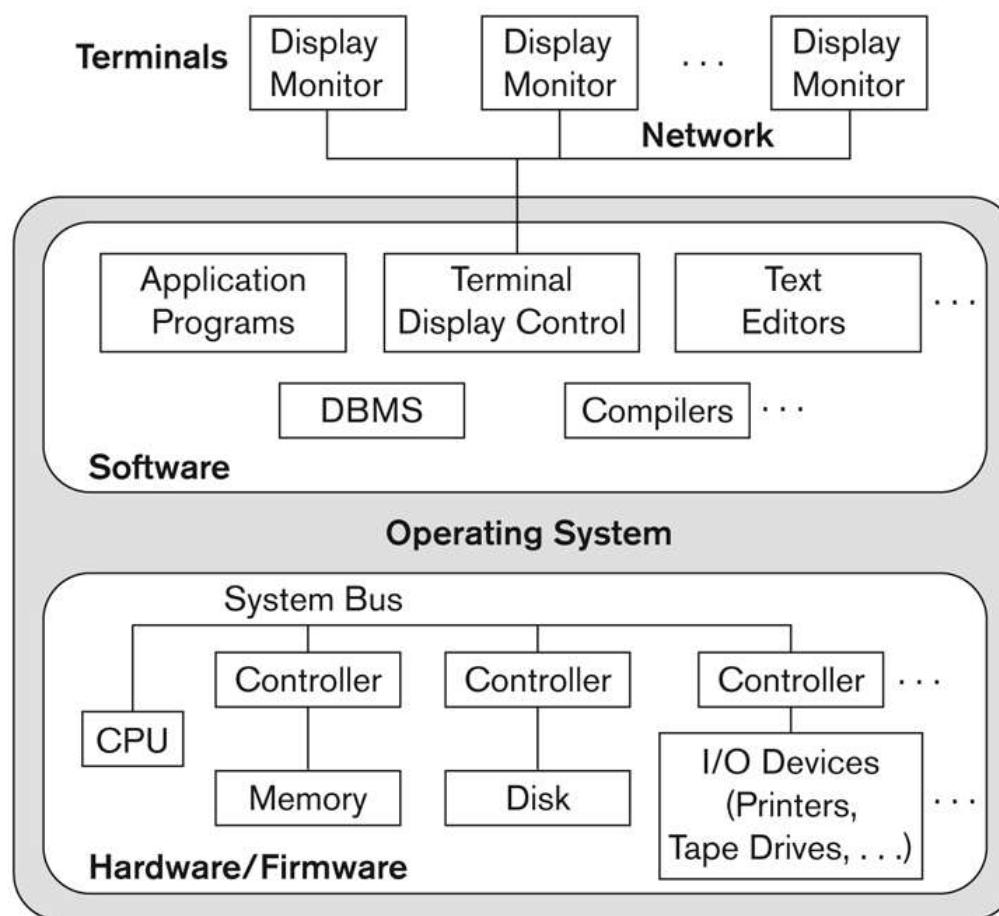
**Figure 2.3**  
Component modules of a DBMS and their interactions.

# Centralized and Client-Server DBMS Architectures

## ■ Centralized DBMS:

- Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
- User can still connect through a remote terminal – however, all processing is done at centralized site.

# A Physical Centralized Architecture



**Figure 2.4**

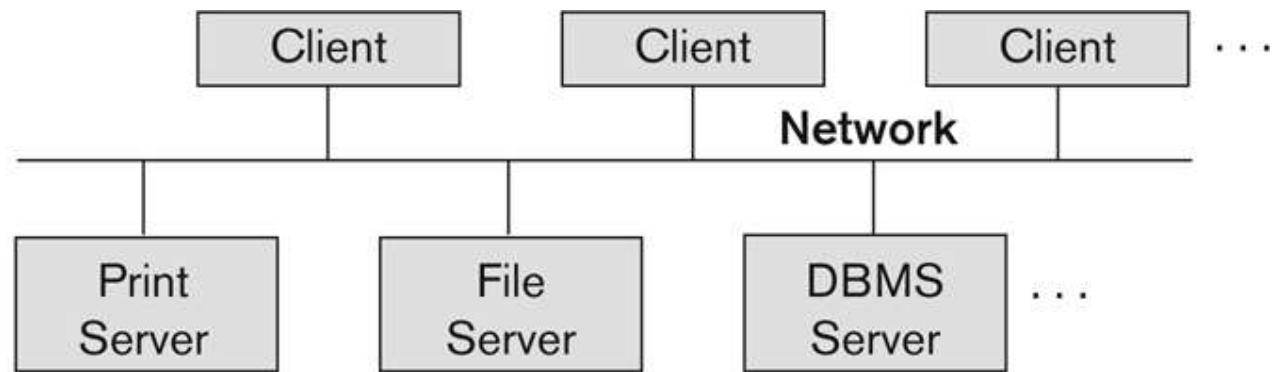
A physical centralized architecture.

# Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions
  - Print server
  - File server
  - DBMS server
  - Web server
  - Email server
- Clients can access the specialized servers as needed

# Logical two-tier client server architecture

**Figure 2.5**  
Logical two-tier  
client/server  
architecture.



# Clients

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
  - (LAN: local area network, wireless network, etc.)

# DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC
- See Chapter 9

# Two Tier Client-Server Architecture

- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

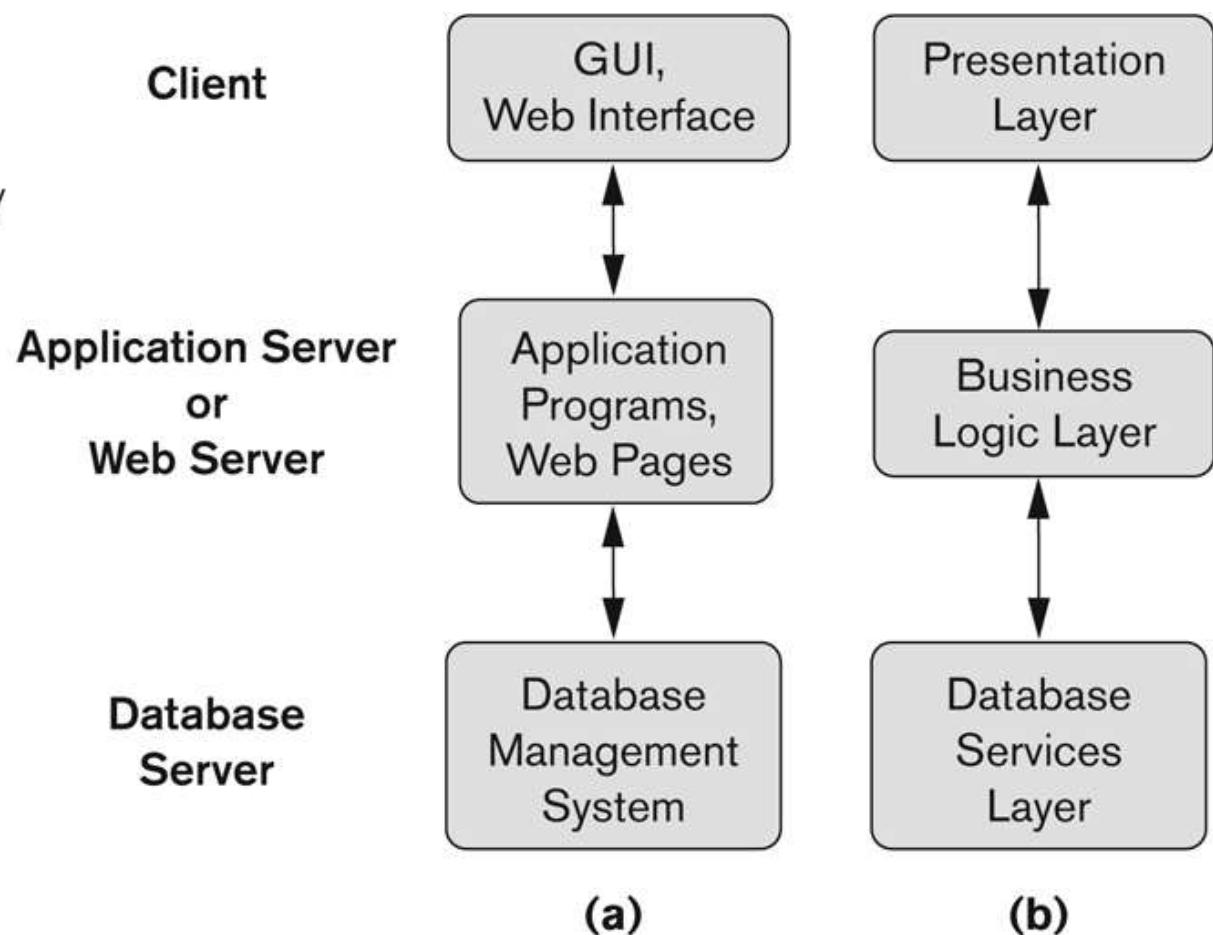
# Three Tier Client-Server Architecture

- Common for Web applications
- Intermediate Layer called Application Server or Web Server:
  - Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
  - Acts like a conduit for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
  - Database server only accessible via middle tier
  - Clients cannot directly access database server

# Three-tier client-server architecture

**Figure 2.7**

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



**CS 380**

# **Introduction to Database Systems**

**(Chapter 3: Data Modeling Using the Entity-Relationship Model)**

# Outline

- Main Phases of Database Design
- Example (Company Example)
- Entities and Attributes
- Key Attribute(s)
- Value Sets (Domain) of Attributes
- Relationships
- Constraints on Relationship Types (Structural Constraints)
- Attributes for Relationship Types
- Weak Entity Type
- Summary of ER-Diagram Notation
- Alternative Notations for ER Diagrams
- Reading: Chapter 3

# Main Phases of Database Design

- **1<sup>st</sup> Step : Requirements collection and analysis:**
  - Designers interview users to understand and document their **data requirements** as a set of detailed user requirements
  - In parallel, it is useful to specify the known **functional requirements** of the application, which consists of user defined operations ( updates and retrievals) that will be applied to the database.

# Main Phases of Database Design

- **2<sup>nd</sup> Step : Creating a conceptual schema for the database using high-level conceptual data model:**
  - This step is called conceptual database design.
  - Conceptual schema is a concise high level description of user data requirements including description of entity types, relationships and constraints.
  - ER model is used.
  - Does not include implementation details.
  - Users and developers can easily understand and discuss
  - Used to ensure all user requirements are met.
  - Enables designers to concentrate on specifying the properties of the data, without being concerned about storage details.

# Main Phases of Database Design

- **3<sup>rd</sup> Step: Specifying the high-level user operations:**
  - During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user operations identified during functional analysis.
  - Serves to confirm that the conceptual schema can meet all the identified functional requirement.

# Main Phases of Database Design

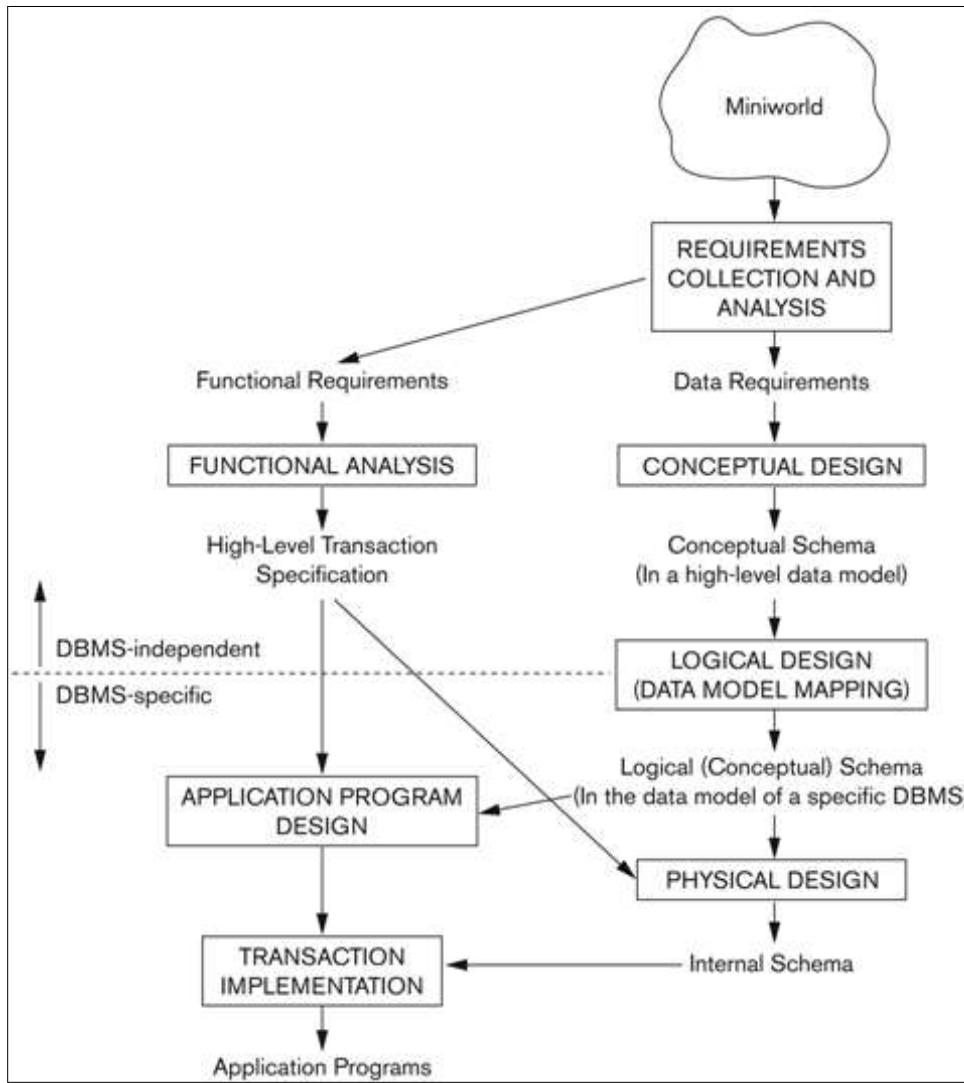
- 4<sup>th</sup> Step : **Actual implementation of the database using DBMS:**
  - This step is called **logical design** or data model mapping and its result is a database schema in the implementation data model of the DBMS. In other words, conceptual schema is transformed from the high level data model (ER model) into implementation data model (relational model).
- 5<sup>th</sup> Step : **Schema Refinement:**
  - This step analyses the collection of relational database schema to identify potential problems and to refine it. We use the theory normalizing relations i.e. restructuring them to ensure some desirable properties.

# Main Phases of Database Design

- **6<sup>th</sup> Step: Physical database design phase:**
  - The internal storage structures, indexes, access paths and file organizations for the database files are specified.
- **7<sup>th</sup> step: Application and Security Design:**

In parallel with these activities, application programs are designed and implemented. At the same time ensure security measures on the access of database.

# Main Phases of Database Design



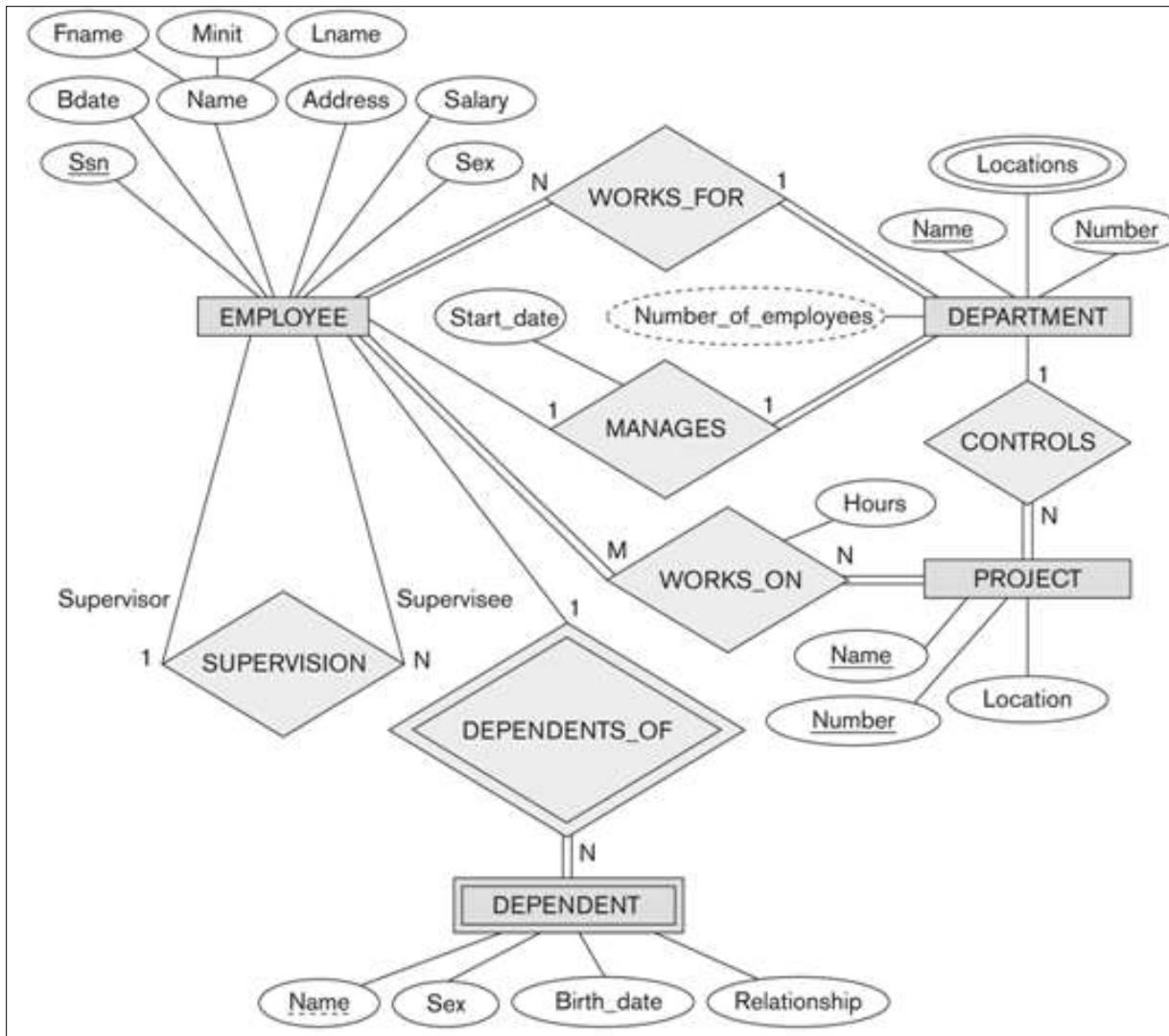
# Example (Company)

- The COMPANY database keeps track of a company's employees, departments and projects.
- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

# Example (Company)

- We store each employee's name, social security number, address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

# Example (Company Example)- ER Diagram



## Entities and Attributes

The ER Model describes data as entities, relationships and attributes.

- **Entity:**

- The basic object that the ER model represents is an entity, which is a “thing” in the real world with an independent existence.
- Could be an object with a physical existence (person, car) or conceptual existence (company, course).
- Each entity has attributes.

PERSON

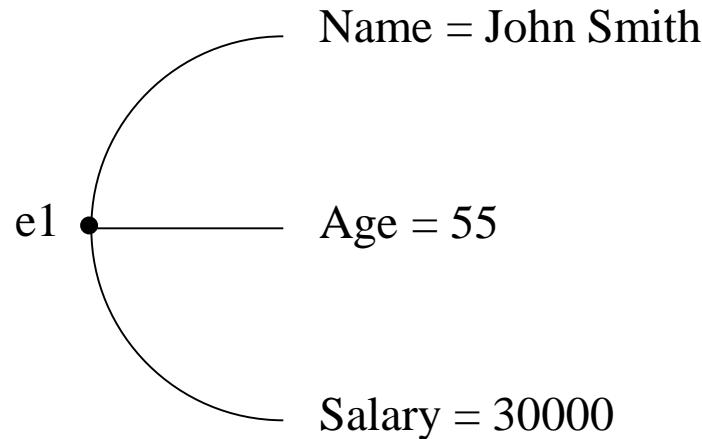
- **Attribute:**

- Property describing an entity.
- E.g. Employee name, age, salary,...etc.

SALARY

# Entities with values of their attributes

- An entity will have a value for each of its attributes; e.g. A specific employee entity may have Name='John Smith', ...etc.

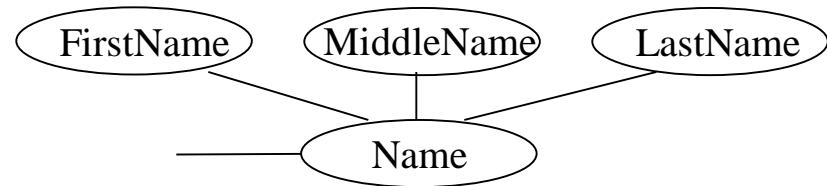


The attribute values that describe each entity become a part of the data stored in the database.

# Types of Attributes - Simple Versus Composite

- **Simple:**

- Each entity has a single atomic value for the attribute.
- e.g. Salary.



- **Composite:**

- Composite attribute is composed of several components, each of which represent more basic attributes with independent meaning.
- E.g. Name (FirstName, MiddleName, LastName).
- Composition may form a hierarchy where some components are themselves composite.

The value of a composite attribute is the concatenation of the values of its constituent simple attributes.

# Types of Attributes - Single-Valued Versus Multivalued

- **Single-valued:**

- One single value for a particular entity.
- E.g. Salary.



- **Multi-valued:**

- A set of values for the same entity.
- E.g. colours attribute for a car - {Color} or
- College degrees attribute for a person - {college degree}.
- A multi-valued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity e.g. the colors attribute of a car may have between 1 and 3 values.



# Types of Attributes - Stored Versus Derived

- **Stored:**

- E.g. Birth\_date.



- **Derived:**

- When two or more attributes are related, the value of one attribute can be obtained from other.
- E.g. The age can be determined from the current date and the value of the birth date of a person, so age is a derived attribute and birth\_date is a stored attribute.
- Some attribute values can be derived from related entities; e.g. an attribute Number\_of\_employees of a DEPARTMENT entity can be derived by counting the number of employees related to that department.



# Types of Attributes

- **NULL values:**
  - **Unknown value:** a particular person has a date of birth but it is unknown, so it is represented by NULL in the database.
  - **Unavailable value:** a person has a home phone but does not want it to be listed, so it is represented as NULL in the database.
  - **Not applicable attribute:** an attribute CollegeDegree would be NULL for a person who has no college degrees.
- **Complex Attributes:**
  - Composite and multivalued attributes can be nested in an arbitrary way.
  - E.g. { AddressPhone{ (Phone(AreaCode, PhoneNumber), { Address(StreetAddress(Number, Street, ApartmentNumber), City, State, Zip) } ) }

# Entity Types, Entity Sets

- A database usually contains groups of entities that are similar.  
E.g. A company storing similar information for each employee.
- The employee entities share the same attributes but each entity has its own value(s) for each attribute.
- An entity type defines a collection (or set) of entities with the same attributes.

# Entity Types, Entity Sets

## ENTITY TYPE

Name:

## EMPLOYEE

Name, Age, Salary

## ENTITY SET: (EXTENSION)

e1•

(John Smith, 55, 30000)

e2•

(Fred Brown, 40, 25000)

e3•

(Judy Clark, 25, 20000)

:

:

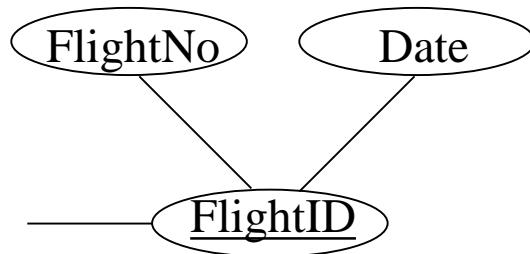
# Key Attribute(s)

- An important constraint on the entity of an entity type is the key or uniqueness constraint on attributes.
- An entity type usually has an attribute whose values are distinct for each individual entity in the collection, called key attribute.
- The value of the key attribute is used to identify each entity uniquely.



# Key Attribute(s)

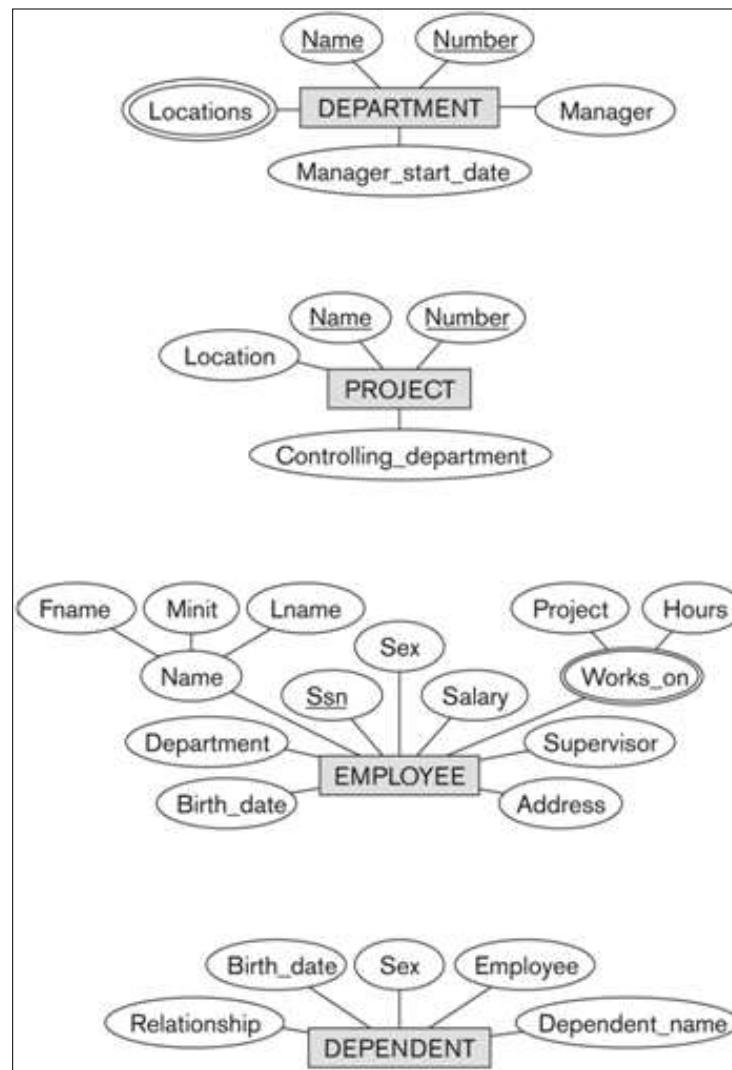
- Sometimes, several attributes together form a key meaning that the combination of the attribute values must be distinct for each entity. In this case it is called a composite key.



# Value Sets (Domain) of attributes

- The domain is the set of permitted values for each attribute.
- Each simple attribute of an entity type is associated with a value set (or domain of values) which specifies the set of values that may be assigned to that attribute for each entity.

# Initial Conceptual Design of the Company Example



# Relationships

- Whenever an attribute of one entity refers to another entity type, some relationship exists.
- E.g. The attribute Manager of Department refers to an employee who manages the department.
- In ER Model these references should not be represented as attributes but as relationships.

# Relationship Types, Sets, and Instances

- A relationship relates two or more distinct entities with a specific meaning.  
e.g. EMPLOYEE John Smith works on the ProductX PROJECT.
- Relationships of the same type are grouped or typed into a relationship type.  
E.g. The WORKS-ON relationship type in which EMPLOYEES and PROJECTs participate.



# Relationship Types, Sets, and Instances

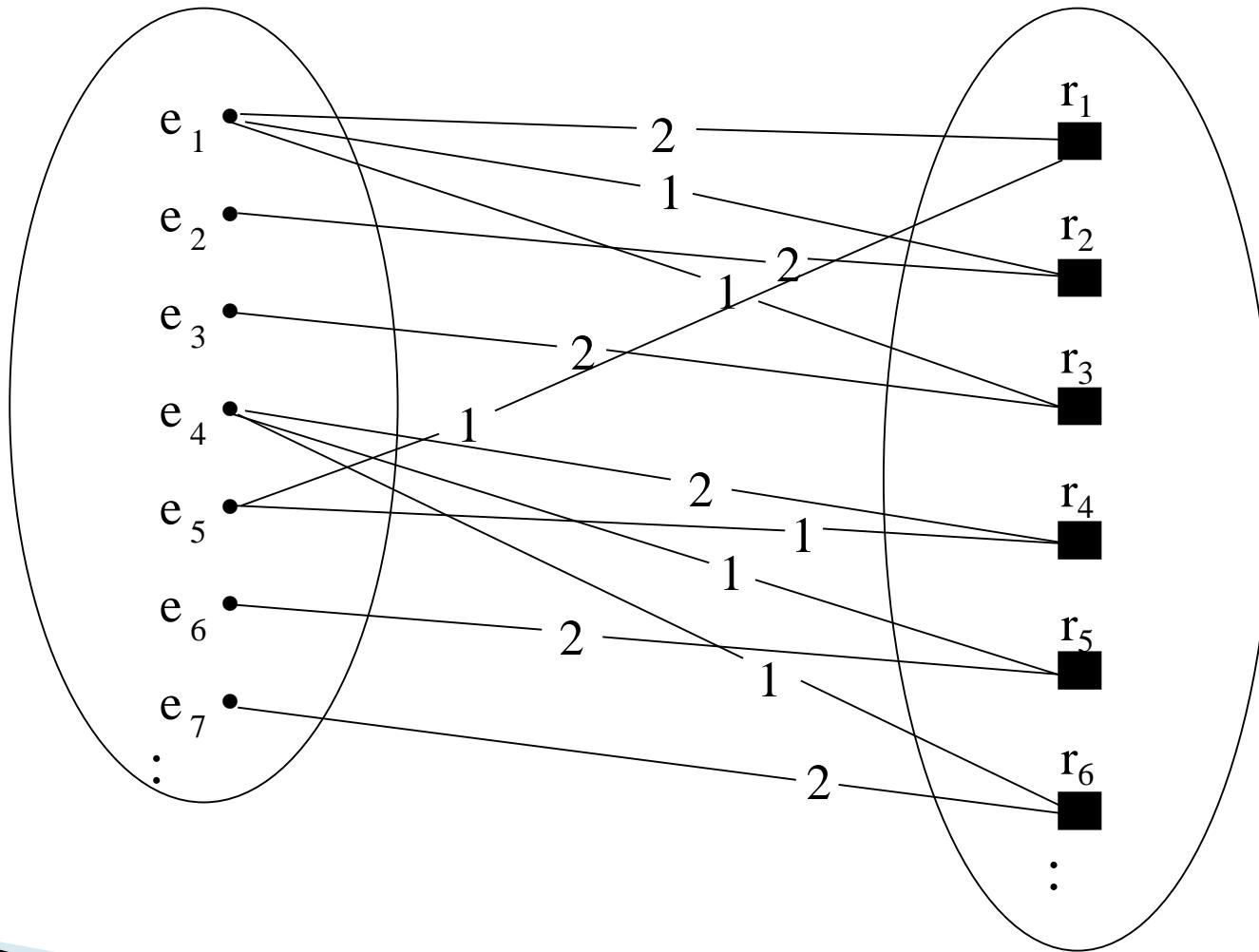
- A relationship R among n entity types E1, E2,...En defines a set of associations (or a relationship set) among entities from these types.
- Mathematically, the relationship set R is a set of relationship instances  $r_i$ , where each  $r_i$  associates n individual entities ( $e_1, e_2, \dots, e_n$ ).

# Relationship Degree

- Degree of a relationship is the number of entity types that participate in it:
  - Unary (or recursive) relationship.
  - Binary Relationship.
  - Ternary relationship.

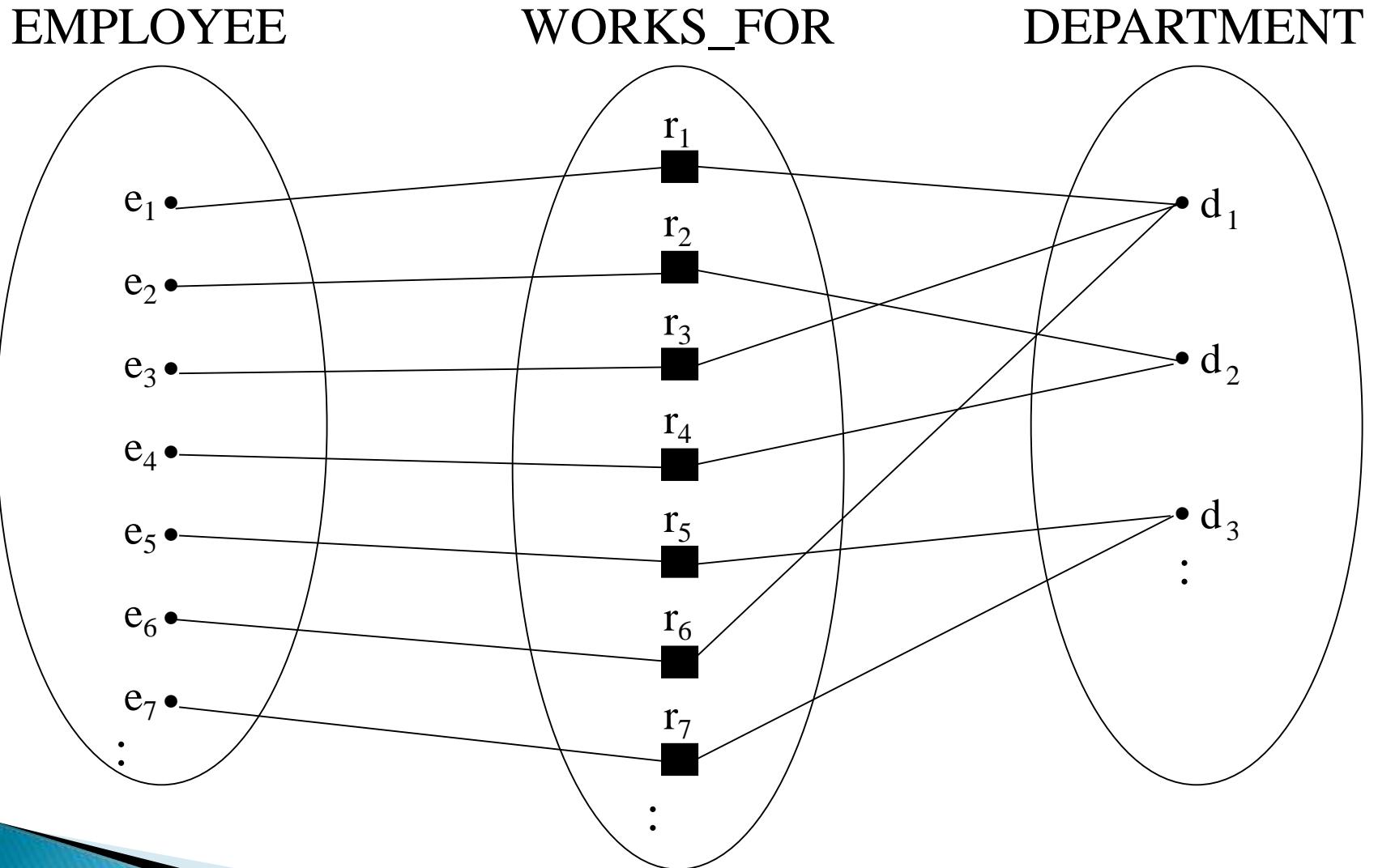
# Relationship Degree - Unary relationship

EMPLOYEE

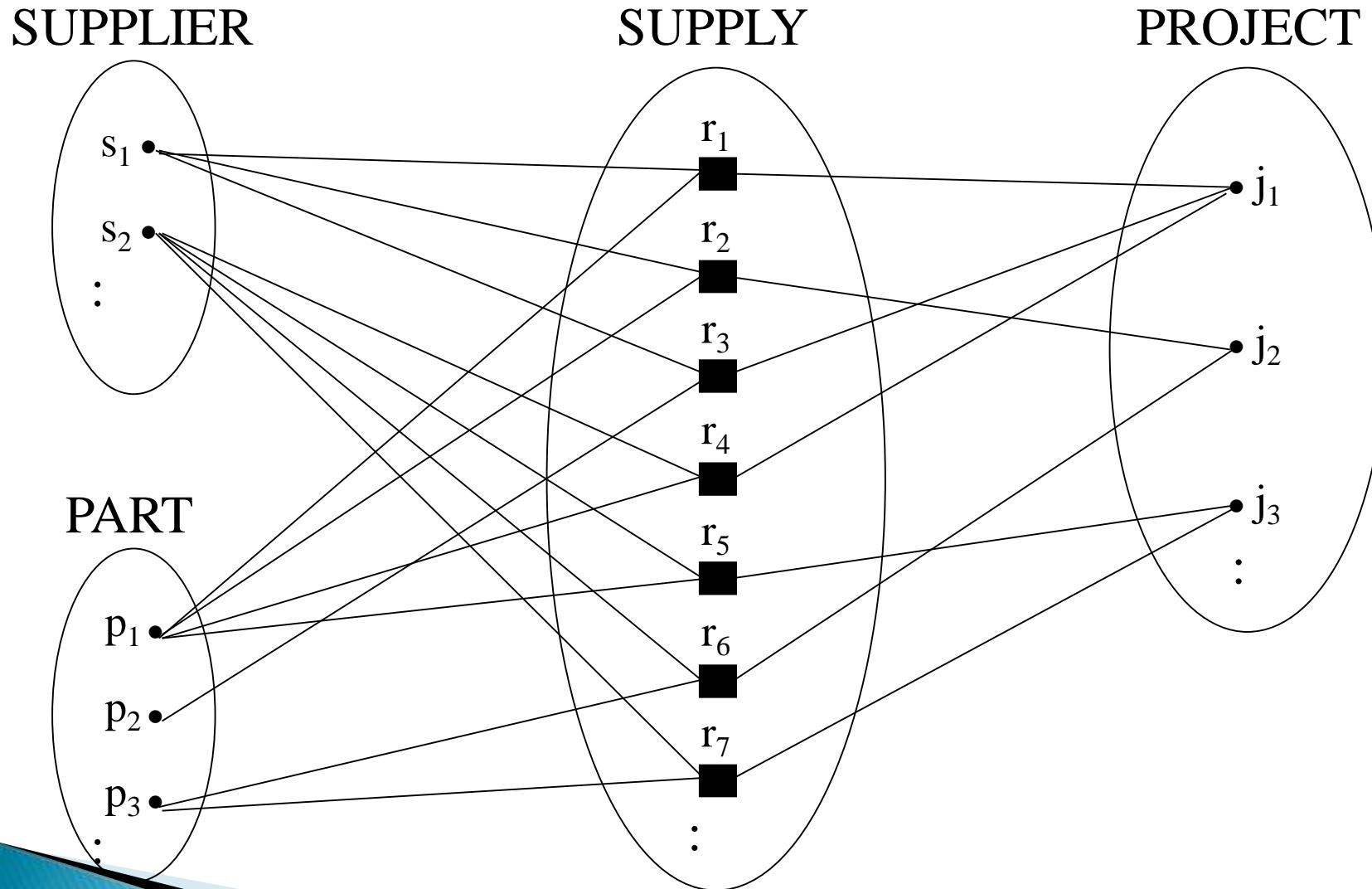


SUPERVISION

# Relationship Degree - Binary relationship



# Relationship Degree - Ternary relationship



# Role Names and Recursive Relationships

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- E.g. In the WORKS-FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.
- Role names are not necessary in relationships where all entities are distinct since each entity name can be used as the role name.
- However, in some cases the role names are essential for distinguishing the meaning of participation.
- Such relationship types are called unary relationships (recursive relationships).

## Constraints on Relationship Types (Structural Constraints)

- There are two types of relationship constraints:
  - Cardinality ratio.
  - Participation.

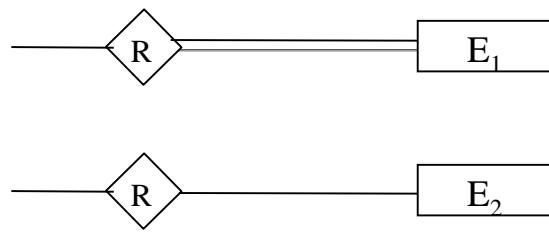
# Cardinality Ratio for Binary Relationships

- The cardinality ratio specifies the **maximum** number of relationship instances that an entity can participate in.
- E.g. In the WORKS-FOR binary relationship type, EMPLOYEE:DEPARTMENT is of cardinality N:1 meaning that an employee can be related to only one department but each department can be related to many employees.
- The possible cardinality ratios for binary relationships are:  
1:1, 1:N, N:1, M:N.



# Participation for Binary Relationships

- The constraints specifies
  - Whether the existence of an entity depends ~~on its~~ being related to another entity via the relationship type.
  - the **minimum** number of relationship instances that each entity can participate in, and is called **minimum cardinality constraint**
- 
- There are two types of participation constraints:
  - Total.
  - Partial.



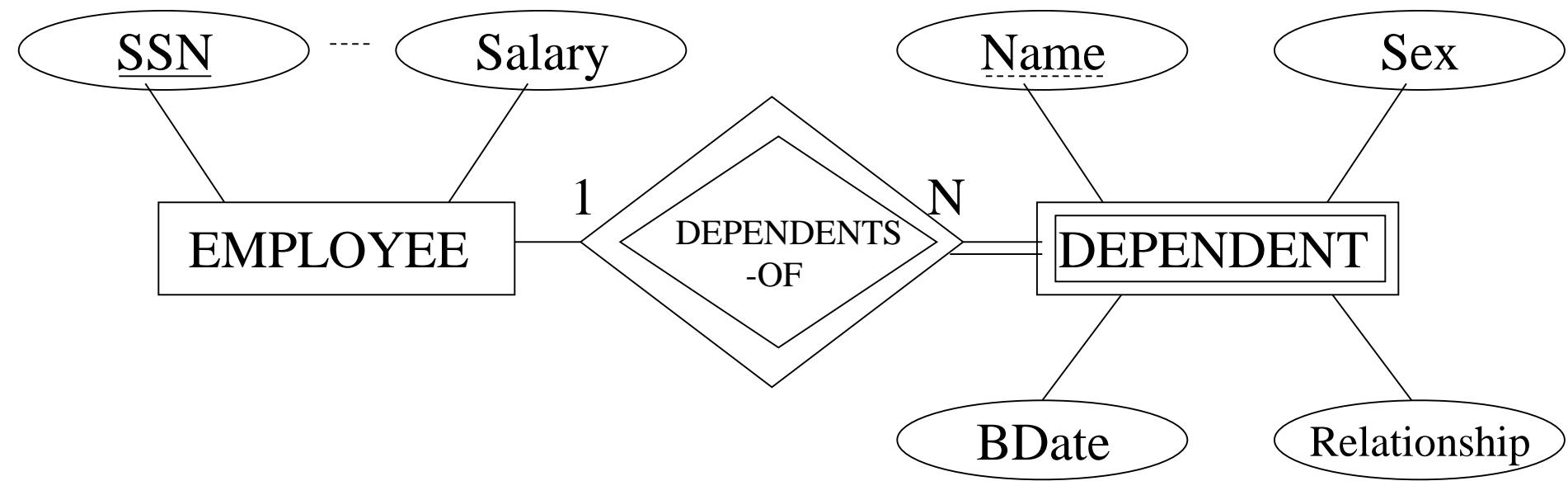
# Attributes of Relationship Types

- Relationship types can also have attributes.
- E.g. To record the number of hours per week that an employee works on a particular project. The attribute Hours can be included to the WORKS-ON relationship type.
- Attributes of 1:1 relationship types can be migrated to one of the participating entity types.
- Attributes of 1:N or N:1 relationship types can be migrated only to the entity type N-side of the relationship.
- Attributes of M:N relationship types must be specified as relationship attributes.

# Weak Entity Type

- A **strong entity** type is an entity that have a key attribute.  
(I.e. It exists independently of other entity types)
- A **weak entity** type is an entity that does not have a key attribute.  
(I.e. Its existence depend on some other entity type)
- The entity type which the weak entity type depends on is called the identifying entity type (owner).
- Weak entities are identified by the combination of:
  - A **partial key (discriminator)** of the weak entity type.
  - The **particular entity** they are related to in the identifying entity type.

# Weak Entity Type



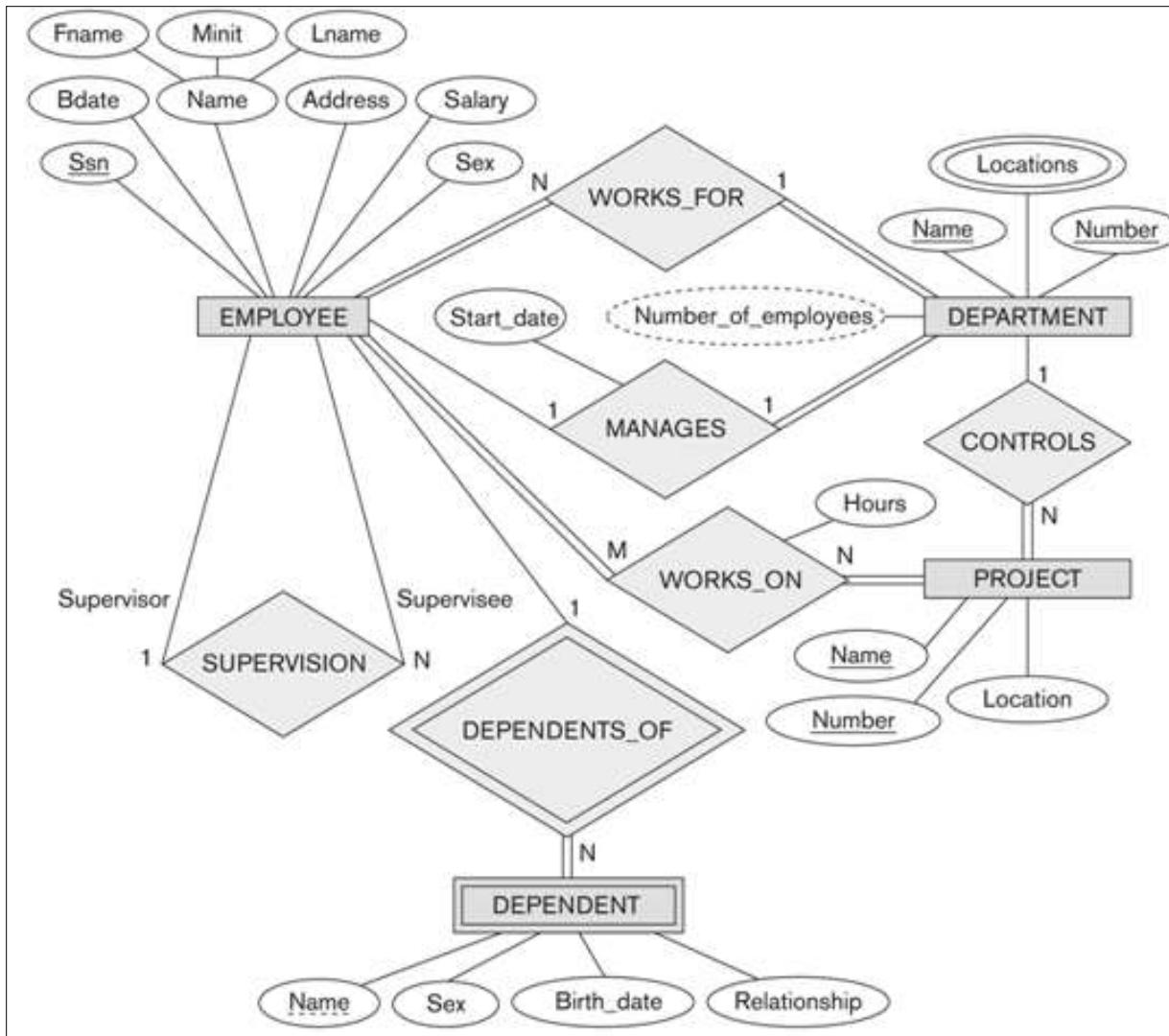
# ER Diagram Conventions

- One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema.
- Nouns for entity type and attribute names and verbs for relationship type names.
- Choosing binary relationship names to make the ER diagram readable from left to right and from top to bottom.
- Entity type and relationship type names in uppercase letters, attribute names capitalized, and roll names in lowercase.

# Summary of Notation For ER-Diagrams

<u>Symbol</u>	<u>Meaning</u>
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E <sub>2</sub> IN R
	CARDINALITY RATIO 1:N FOR E <sub>1</sub> :E <sub>2</sub> IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

# Example (Company Example)



# Alternative (min, max) notation for ER Diagrams

- Specified on each participation of an entity type E in a relationship type R.
- Specifies that each entity e in E participates in at least min and at most max relationship instances in R.
- Default (no constraint):  $\text{min} = 0, \text{max} = n$ .
- Must have  $0 \leq \text{min} \leq \text{max}$  and  $\text{max} \geq 1$ .
- Derived from the knowledge of mini-world constraints.

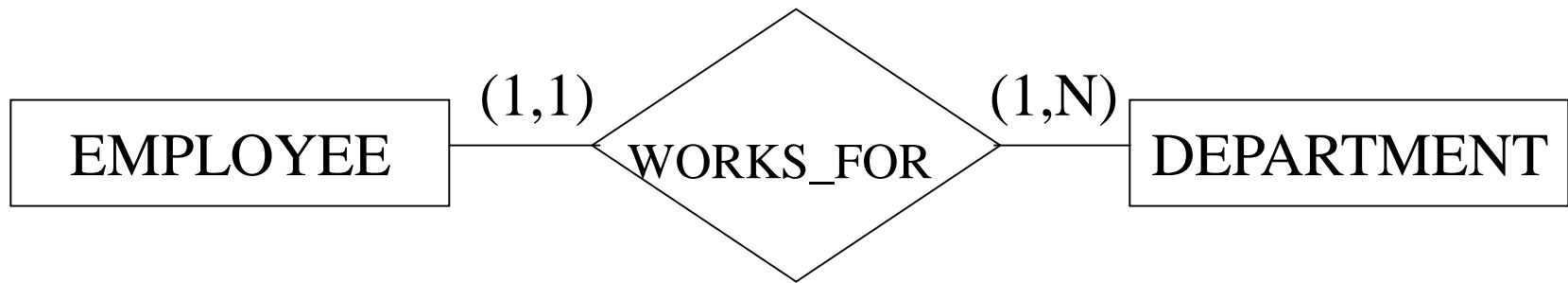
# Alternative (min, max) notation for ER Diagrams

- A department has exactly one manager and an employee can manage at most one department:
  - Specify (0,1) for participation of EMPLOYEE in MANAGES.
  - Specify (1,1) for participation of DEPARTMENT in MANAGES.

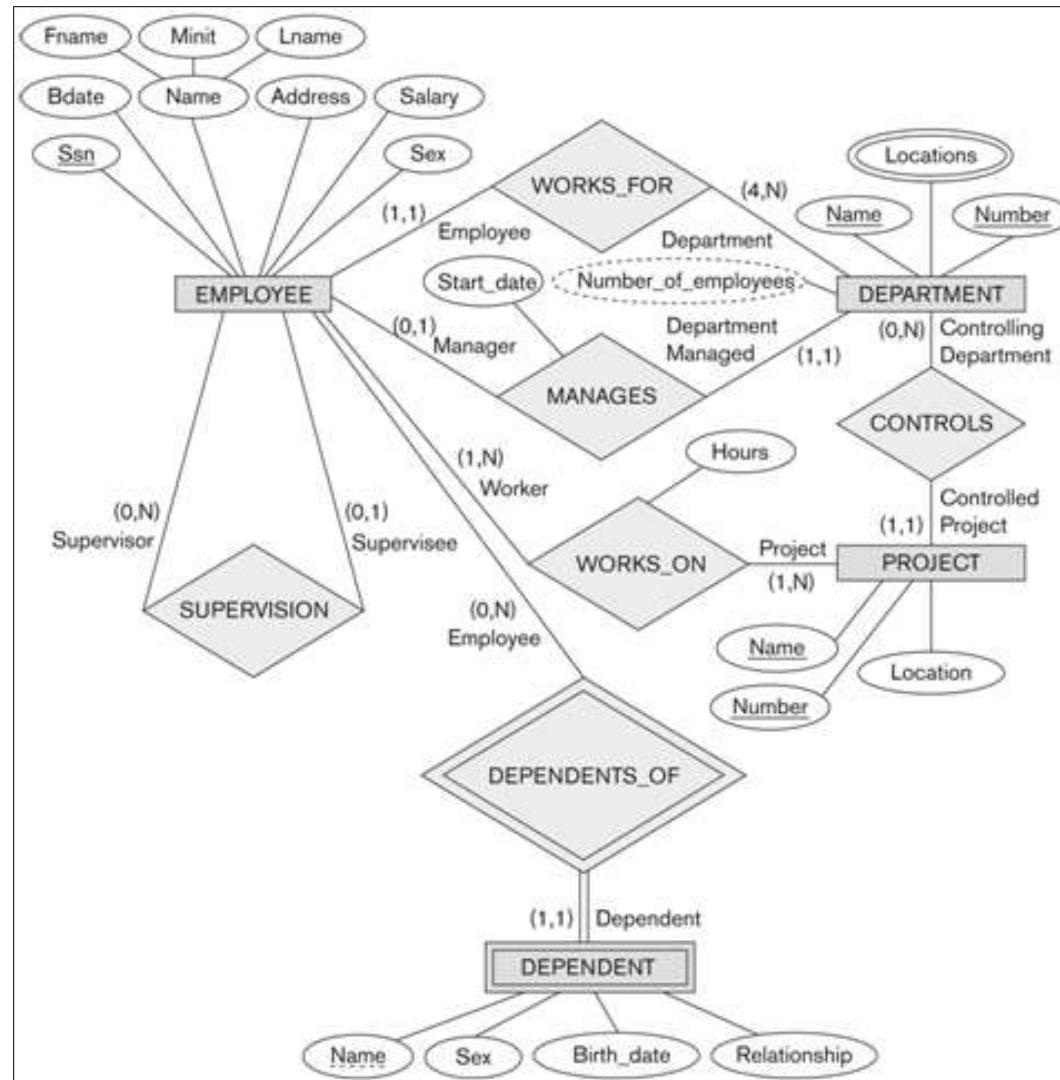


# Alternative (min, max) notation for ER Diagrams

- An employee can work for exactly one department but a department can have any number of employees:
  - Specify (1,1) for participation of EMPLOYEE in WORK\_FOR.
  - Specify (1,N) for participation of DEPARTMENT in WORKS\_FOR.



# Alternative (min, max) notation for ER Diagrams



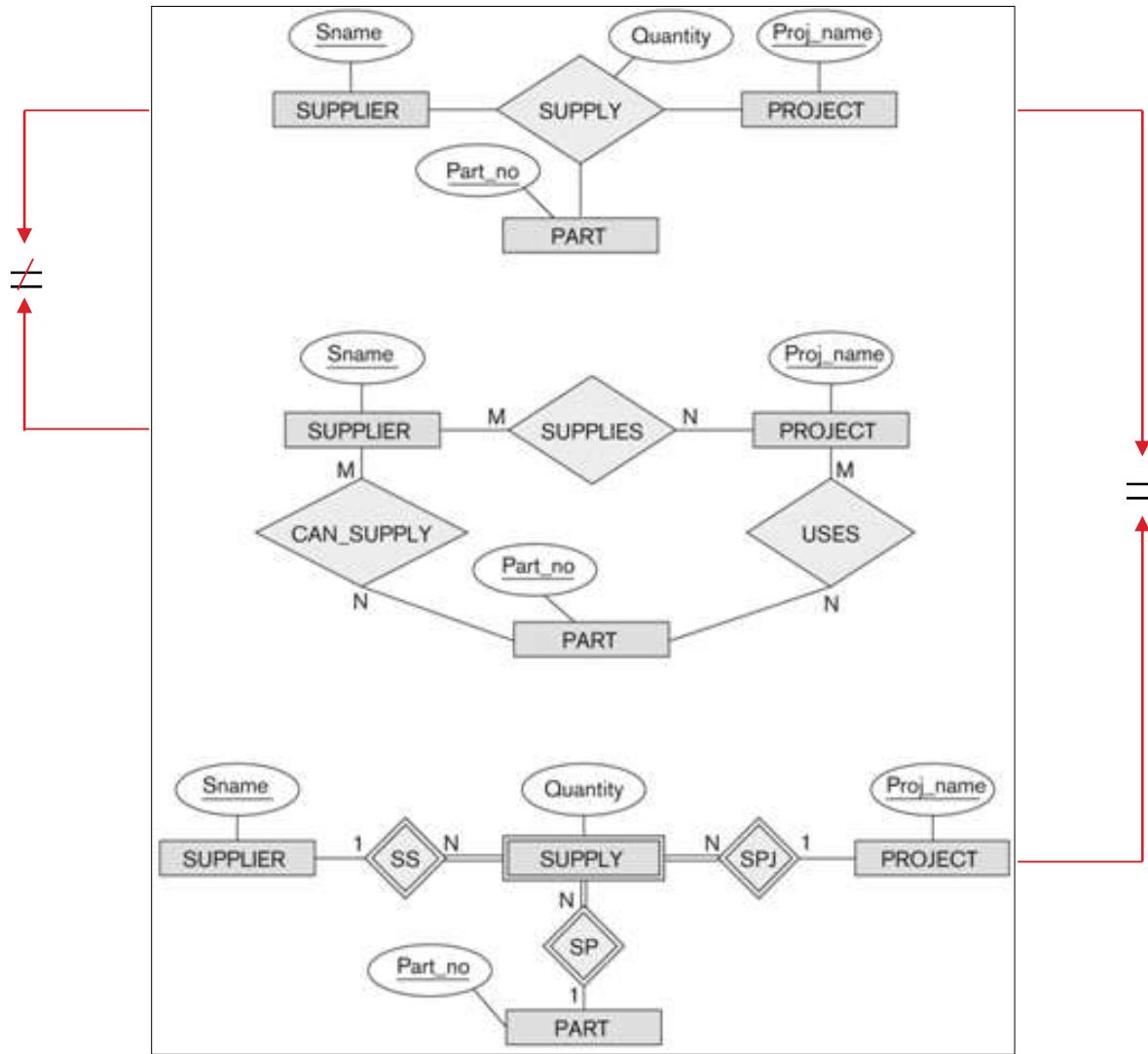
# Alternative Notations for ER Diagrams

- ER diagrams is one popular example for displaying database schemas.
- Many other notations exist in the literature and in various database design and modeling tools.
- Appendix A illustrates some of the alternative notations that have been used.
- UML class diagrams is representative of another way of displaying ER concepts that is used in several commercial design tools.

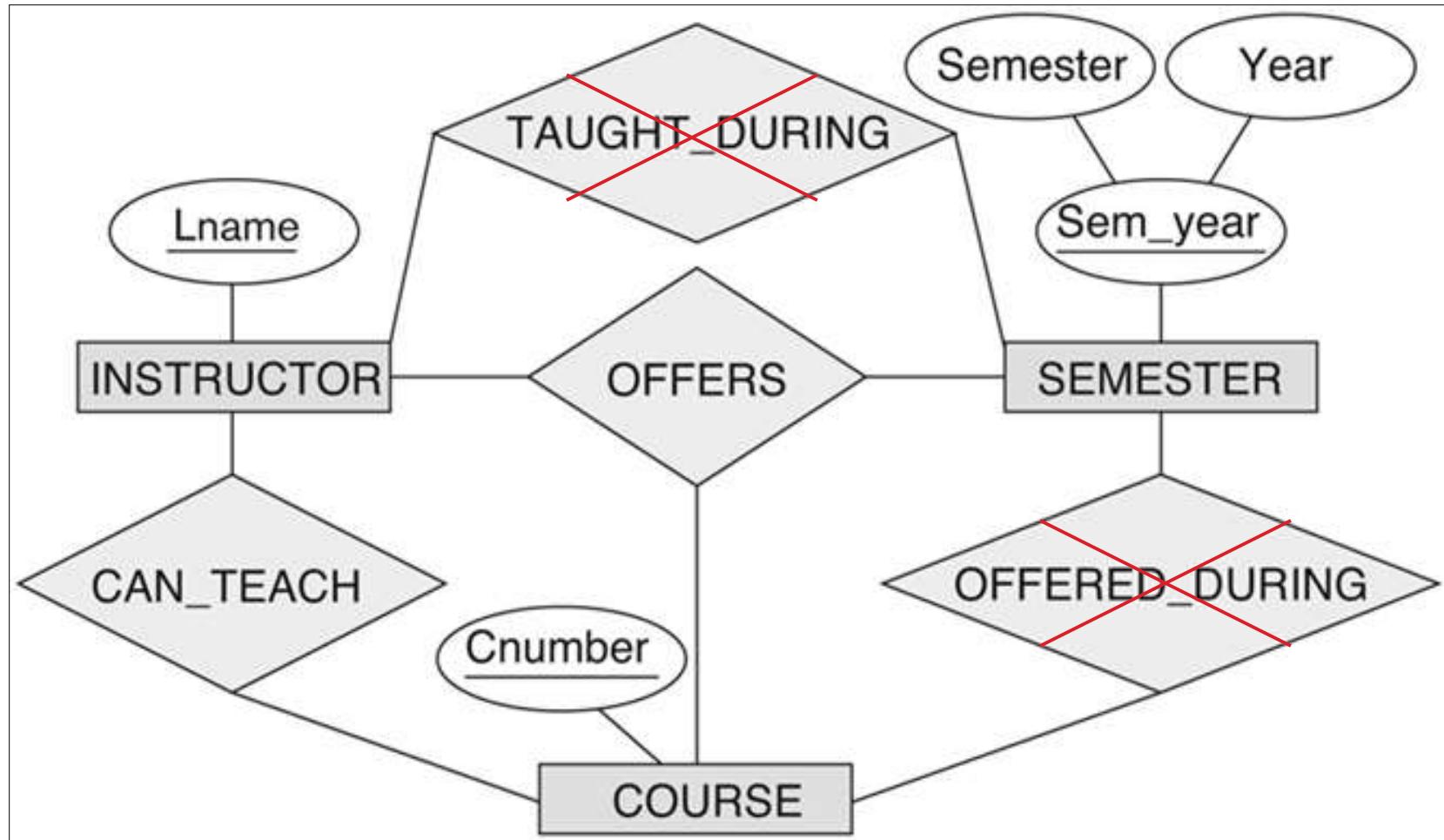
# Relationships of Higher Degree

- A relationship with degree  $n$  ( $n > 2$ ) is called  $n$ -ary.
- In general, an  $n$ -ary relationship is not equivalent to  $n$  binary relationships.
- Constraints are harder to specify for higher-degree relationships ( $n > 2$ ) than for binary relationships.

# Ternary Relationships



# Ternary VS. Binary Relationships



# Displaying Constraints on Higher-Degree Relationships

- The (min, max) constraints can be displayed on the edges. However, they do not fully describe the constraints.
- Displaying a 1, M, or N indicates additional constraints:
  - An M or N indicates no constraint.
  - A 1 indicates that an entity can participate in at most one relationship instance that has a particular combination of the other participating entities.
- In general, both (min, max) and 1, M, or N are needed to describe fully the constraints.

# **CS 380**

# **Introduction to Database Systems**

**(Chapter 4: The Relational Data Model and Relational Database Constraints)**

# Outline

- Introduction
- Informal Definition
- Formal Definition
- Attributes, Domains, & Relations
- Characteristics of Relations
- Relational Model Constraints
- Relational Databases and Relational Database Schemas
- Update Operations & Dealing With Violations

# Introduction

- The relational model was first introduced by Ted Codd of IBM research in 1970.
- The relational model uses the concept of a mathematical relation.
- A mathematical relation concept is based on the ideas of sets.
- It has been the dominant technology since the 1980s.

# Informal Definition

- Each relation resembles a table of values.
- A relation may be thought of as a set of rows or columns.
- Each row represents a collection of related data values.
- Each row represents a fact that corresponds to a real-world entity or relationship.
- Each row has a value of an item(s) that uniquely identifies the row in the table.
- Each column is called by its name or column header or attribute name.
- All values in a column are of the same data type.

# Formal Definition

- The table is called a **relation**.
- A row is called a **tuple**.
- A column header is called an **attribute**.
- A **domain** is the set of possible values for an attribute.

Relation                          Attribute

Department	Dnumber	Dname	Location
	1	Accounting	New York
	2	Research	Dallas
	3	Sales	Chicago
	4	Operations	Boston

Tuple →

Domain for location:  
New York, Dallas,  
Chicago, Boston,  
Manchester, London,  
etc

# Attributes, Domain, & Relations

- No composite or multivalued attributes are permitted.  
This property is known as the First Normal Form.
- The domain of values for an attribute contains only atomic values.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- Each domain is given a name, datatype, & format.  
E.g. Name = SALARY, datatype = integer, format = 5 digits.

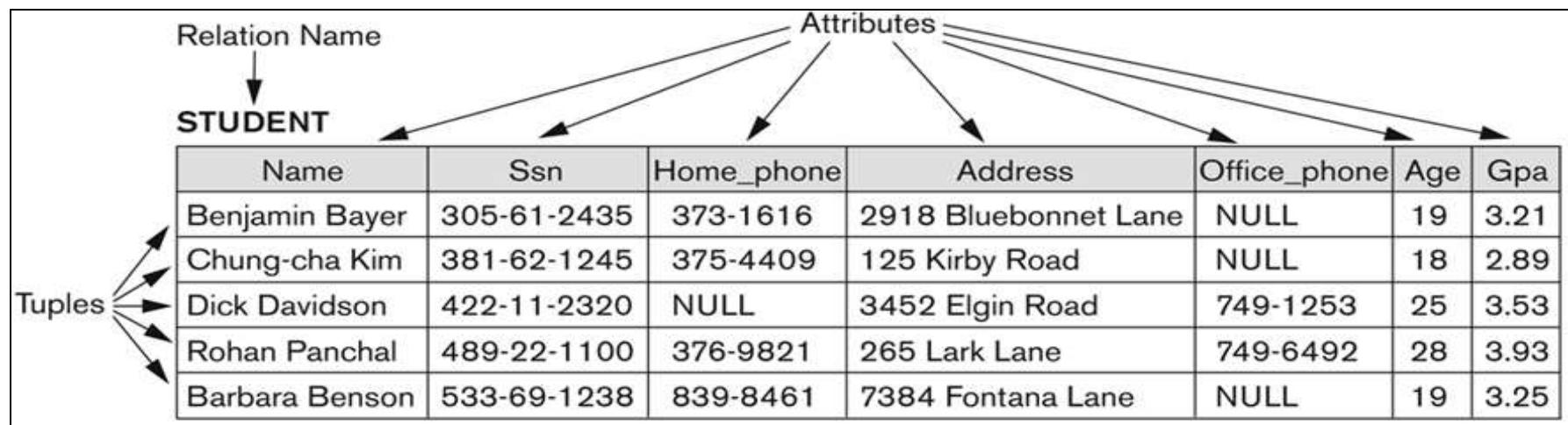
# Attributes, Domains, & Relations

- The degree of a relation is the number of attributes n of its relation schema.

Degree = 4

Department	Dnumber	Dname	Location	Phone
	1	Accounting	New York	749-1111
	2	Research	Dallas	null
	3	Sales	Chicago	null
	4	Operations	Boston	null

# Attributes, Domains, & Relations



# Characteristics of Relations

- **Ordering of tuples in a relation:** the tuples are not considered to be ordered, even though they appear to be in the tabular form.
- **Ordering of values within a tuple:** the attributes in R and the values in t are considered to be ordered. (although, a more general definition of relation does not require this ordering).
- **Values and nulls in the tuples:** all values are considered atomic. A special null value is used to represent values that are unknown.

# Characteristics of Relations

## Ordering of tuples

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

**Figure 5.2**

The relation STUDENT from Figure 5.1 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	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

# Characteristics of Relations

## Ordering of values

The following two tuples are equal:

$$t = \langle \{Name, Dick\ Davidson\}, \{Ssn, 422-11-2320\}, \{Home\_phone, Null\}, \\ \{Address, 3452 Eign Road\}, \{Office\_Phone, 749-1253\}, \{Age, 25\}, \\ \{Gpa, 3.53\} \rangle$$
$$t = \langle \{Address, 3452 Eign Road\}, \{Name, Dick\ Davidson\}, \{Ssn, 422-11-2320\}, \{Age, 25\}, \\ \{Office\_Phone, 749-1253\}, \{Gpa, 3.53\}, \{Home\_phone, Null\} \rangle$$

# Characteristics of Relations

## Values and nulls in the tuples

- Each value in a tuple is considered to be atomic (i.e. it is not divisible into components). Hence the composite or multi-valued attributes are not allowed. This model is called Flat Relational Model. This assumption is called the first Normal Form.
  - Multi-valued attributes are represented by separate relations
  - Composite attributes are represented only by their simple component attributes in the basic relational model.
- Null values are used to represent values of attributes that are unknown, unavailable, or not applicable to a tuple. A special value NULL is used for these cases.

# Relational Model Constraints

- **Inherent model-based constraints:** constraints that are inherent in the data model.
- **Schema-based constraints:** constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL.
- **Application-based constraints:** constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs.

# Inherent Model-based (implicit) Constraints

- Ordering of tuples in a relation is not important.
- Ordering of values within a tuple is important.
- A relation cannot have duplicate tuples.
- Interpretation (Meaning) of a Relation. Every relation represents facts about entities or relationships.

# Schema-Based (Explicit) Constraints

- Include:
  - Domain constraints.
  - Key constraints.
  - Constraints on nulls.
  - Entity integrity constraints.
  - Referential integrity constraints.

# Domain Constraints

- It specifies that within each tuple, the value of each attribute must be an atomic value from the domain.
- The data types associated with domains: integers, real numbers, characters, booleans, strings, ...etc.

# Key Constraints

- **Super key:** a set of attributes SK of R such that no two tuples in any valid relation instance will have the same value for SK.
- **Key:** a “minimal” superkey; that is, a superkey from which we can not remove any attribute(s) and still have the uniqueness constraint hold.
- E.g. CAR (State, Reg#, SerialNo, Make, Model, Year)
  - Has two keys:
    - Key 1 = {State, Reg#}.
    - Key 2 = {SerialNo}.
  - {SerialNo, Make} is a superkey but not a key.
- If a relation has several **candidate keys**, one is chosen arbitrary to be the **primary key**.

# Key Constraints

- It specifies that no two tuples can have the same value for their primary keys.

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 5.4**

The CAR relation, with two candidate keys: License\_number and Engine\_serial\_number.

# Constraints on Null values

- Specifies whether null values are or are not permitted. Ex: if every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then the Name of STUDENT is constrained to be NOT NULL.

# Entity Integrity Constraints

- The primary key attributes PK of each relation schema R in S cannot have null values in any tuple. This is because primary key values are used to identify the individual tuples.
  - $t[PK] \neq \text{null}$  for any tuple t in  $r(R)$
  - If PK has several attributes, null is not allowed in any of these attributes
- Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

# Referential Integrity Constraints

- A constraint involving two relations  
The previous constraints involve a single relation.
- Used to specify a relationship among tuples in two relations:  
The referencing relation and the referenced relation.

# Referential Integrity Constraints

- It is used to maintain the consistency among tuples in the two relations (the referencing relation and the referenced relation).
- Tuples in the referencing relation have attributes FK (**foreign key**) that reference the primary key attribute PK of the referenced relation.
- The value in the foreign key of the referencing relation can be either:
  - A value of an existing primary key value of the corresponding primary key in the referenced relation. Or,
  - A null.

# Referential Integrity Constraints (Example)

CUSTOMER

<u>CustomerID</u>	CustomerName
-------------------	--------------

ORDER

<u>OrderID</u>	OrderDate	CustomerID
----------------	-----------	------------

Foreign Key

# Relational Databases and Relational Database Schemas

- A relational Database schema  $S$  is a set of relation schema  $S = \{ R_1, R_2, \dots, R_m \}$  and a set of integrity Constraints IC.
- A relational Database State DB of  $S$  is a set of relation states  $DB = \{ r_1, r_2, \dots, r_m \}$  such that each  $r_i$  is a state of  $R_i$  and such that the  $r_i$  relation states satisfy the integrity constraints specified in IC.
- A database state that does not obey all the integrity constraints is called an **invalid state**, and a state that satisfies all the constraints in IC is called a **valid state**.

# Relational Databases and Relational Database Schemas

- Attributes that represent the same real world concept may or may not have identical names in different relation. ( Ex: Ssn and Super-ssn, Dnumber and Dnum )
- **Attributes that represent different concepts may have the same name in different relations.**

# COMPANY Database Schema

## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	Dlocation
----------------	-----------

## PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

## DEPENDENT

<u>Essn</u>	Dependent_name	Sex	Bdate	Relationship
-------------	----------------	-----	-------	--------------

**Figure 5.5**

Schema diagram for the COMPANY relational database schema.

# One Possible Database State for COMPANY

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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	MGRSSN	MGRSTARTDATE	DEPT_LOCATIONS	DNUMBER	DLOCATION
	Research	5	333445555	1988-05-22		1	Houston
	Administration	4	987654321	1995-01-01		4	Stafford
	Headquarters	1	888665555	1981-06-19		5	Bellaire
						5	Sugarland
						5	Houston

WORKS_ON	ESSN	PNO	HOURS	PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	123456789	1	32.5		ProductX	1	Bellaire	5
	123456789	2	7.5		ProductY	2	Sugarland	5
	666884444	3	40.0		ProductZ	3	Houston	5
	453453453	1	20.0		Computerization	10	Stafford	4
	453453453	2	20.0		Reorganization	20	Houston	1
	333445555	2	10.0		Newbenefits	30	Stafford	4
	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					

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

# Referential Integrity Constraint for COMPANY

**Figure 5.7**

Referential integrity constraints displayed on the COMPANY relational database schema.

## EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	-----	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
-------	---------	---------	----------------

## DEPT\_LOCATIONS

Dnumber	Dlocation
---------	-----------

## PROJECT

Pname	Pnumber	Plocation	Dnum
-------	---------	-----------	------

## WORKS\_ON

Essn	Pno	Hours
------	-----	-------

## DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
------	----------------	-----	-------	--------------

# Update Operations & Dealing With Violations

- The operations of the relational model can be categorized into retrievals and updates.
- Update or modification operations:
  - Insert.
  - Delete.
  - Update.

# The Insert Operation

- Used to provide a list of attribute values for a new tuple.
- Can violate the:
  - Domain constraints: if a value given doesn't appear in the domain.
  - Key constraints: if a key value already exist in the relation.
  - Entity integrity constraints: if the primary key of the new tuple is null
  - Referential integrity constraints: if a foreign key value refers to a non existent tuple.
- When violation occurs the default option is to reject the insertion.  
(or correct the reason for rejection)

# The Insert Operation

DEPARTMENT

DeptNO	DName	Location
1	Accounting	New York
2	Research	Dallas
3	Sales	Chicago
4	Operations	Boston

- INSERT (1.5, Marketing, Los Angeles) into DEPARTMENT X
- INSERT (1, Marketing, Los Angeles) into DEPARTMENT X
- INSERT (null, Marketing, Los Angeles) into DEPARTMENT X

# The Delete Operation

- Can violate only the referential integrity constraint if the tuple being deleted is referenced by the foreign key(s) from other relation.
- To specify deletion, a condition on the attributes of the relation selects the tuple(s) to be deleted.
- Three options are available if deletion causes a violation:
  - To reject the deletion.
  - To attempt to cascade the deletion by detecting tuples that reference the tuple that is being deleted.
  - To modify the referencing attribute values causing violation by either setting it to null or changing it to reference another valid tuple.

# The Delete Operation

DEPARTMENT

DeptNO	DNname
1	Accounting
2	Research

EMPLOYEE

EmplNO	...	DeptNO
111	...	1
222	...	2

Deleting this tuple → violation of referential integrity

# The Update Operation

- Used to change the values of an attribute(s) in a tuple of R.
- It is necessary to specify a condition on the attribute(s) of the relation to select the tuple(s) to be modified.
- Updating an attribute that is neither a primary key nor a foreign key causes no problems. DBMS needs only to check the datatype and domain.
- All issues of insert & delete apply if updating a primary key or a foreign key.

# The Update Operation

Any of the other constraints may also be violated, depending on the attribute being updated:

- Updating the primary key (PK):
  - Similar to a DELETE followed by an INSERT
  - Need to specify similar options to DELETE
- Updating a foreign key (FK):
  - May violate referential integrity

# In-Class Exercise

(Taken from Exercise 5.16)

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK\_ADOPTION(Course#, Quarter, Book\_ISBN)

TEXT(Book\_ISBN, Book\_Title, Publisher, Author)

**Specify the foreign keys for this schema, then draw a relational schema diagram.**

Answer:

The schema of this question has the following four foreign keys:

1. the attribute SSN of relation ENROLL that references relation STUDENT,
2. the attribute Course# in relation ENROLL that references relation COURSE,
3. the attribute Course# in relation BOOK\_ADOPTION that references relation COURSE, and
4. the attribute Book\_ISBN of relation BOOK\_ADOPTION that references relation TEXT.

# **CS 380**

# **Introduction to Database Systems**

**(Chapter 5: Relational Database Design by ER- and EER-to-Relational Mapping)**



# Outline

- ER-to-Relational Mapping Algorithm:
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relationship Types
  - Step 4: Mapping of Binary 1:N Relationship Types
  - Step 5: Mapping of Binary M:N Relationship Types
  - Step 6: Mapping of Mutivalued Attributes
  - Step 7: Mapping of N-ary Relationship Types
- Mapping EER Model Constructs to Relations:
  - Step 8: Options for Mapping Specialization or Generalization
  - Step 9: Mapping of Union Types (Categories)

# **Relational Database Design by ER- and EER-to-Relational Mapping**

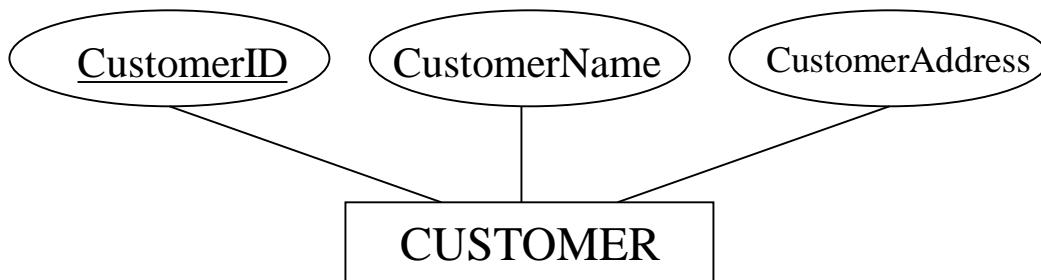
- This corresponds to the logical database design (data model mapping) step.

# Step 1: Mapping of Regular Entity Types

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key for R.
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

# Step 1: Mapping of Regular Entity Types

- CUSTOMER entity type with simple attributes:



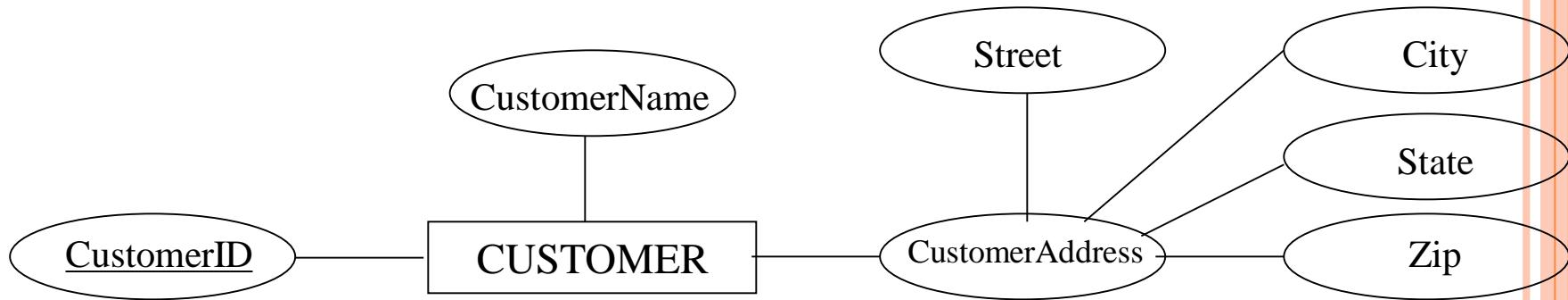
- CUSTOMER relation:

CUSTOMER

<u>CustomerID</u>	CustomerName	CustomerAddress
-------------------	--------------	-----------------

# Step 1: Mapping of Regular Entity Types

- CUSTOMER entity type with a composite attribute:



- CUSTOMER relation with address detail:

CUSTOMER

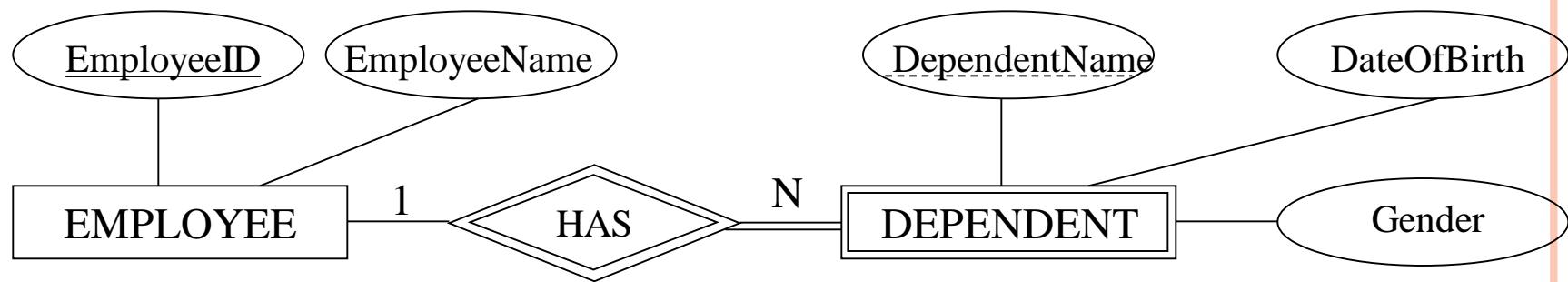
<u>CustomerID</u>	CustomerName	Street	City	State	Zip
-------------------	--------------	--------	------	-------	-----

## Step 2: Mapping of Weak Entity Types

- For each weak entity type  $W$  in the ER schema with owner entity type  $E$ , create a relation  $R$  and include all simple attributes of  $W$  as attributes of  $R$ .
- Include as Foreign key attributes of  $R$  the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The primary key of  $R$  is composed of:
  - Partial key of the weak entity type  $W$ .
  - Primary key(s) of the owner(s).

## Step 2: Mapping Weak Entities to Relations

- Weak entity DEPENDENT:



- Relations:

### EMPLOYEE

<u>EmployeeID</u>	EmployeeName
-------------------	--------------

### DEPENDENT

<u>DependentName</u>	<u>EmployeeID</u>	DateOfBirth	Gender
----------------------	-------------------	-------------	--------

Composite primary key

Foreign key

## Step 3: Mapping OF Binary 1:1 Relationship Types

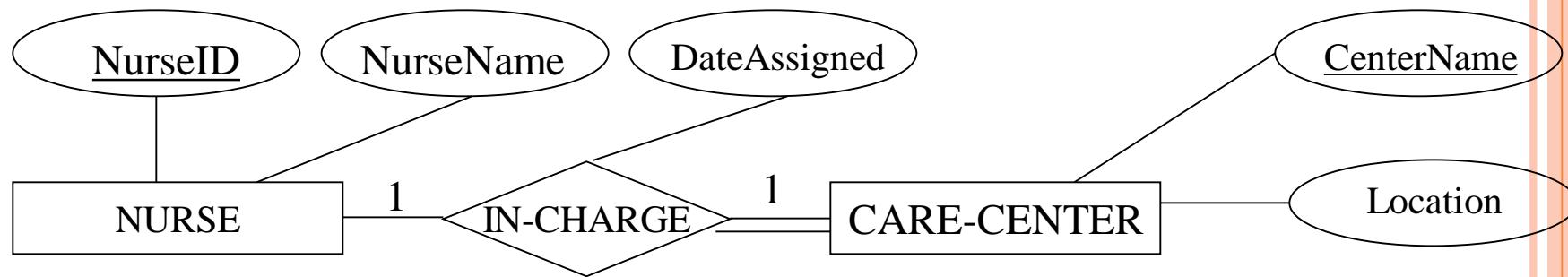
- For each binary 1:1 relationship type R in the ER schema, identify relations S and T that correspond to the entity types participating in R.

# Step 3: Mapping OF Binary 1:1 Relationship Types

- There are three possible approaches:
  - **Foreign key approach:**
    - Choose one of the relations - S, say - and include as a foreign key in S the primary key of T.
    - It is better to choose an entity type with total participation in R in the role of S.
    - Include all the simple attributes of the 1:1 relationship type R as attributes of S.
  - **Merged relation option:**
    - Merging the two entity types and the relationship into a single relation. (Used when both participations are total)
  - **Cross-reference or relationship relation option:**
    - Setting up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

# Step 3: Mapping OF Binary 1:1 Relationship Types

- one-to-one relationship:



- Mapping the relationship (**foreign key approach**):

NURSE

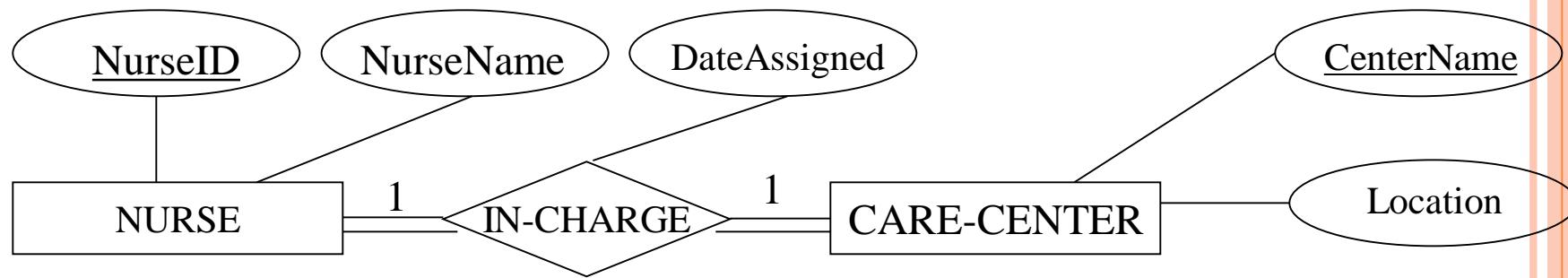
<u>NurseID</u>	NurseName
----------------	-----------

CARE-CENTER

<u>CenterName</u>	Location	NurseInCharge	DateAssigned
-------------------	----------	---------------	--------------

# Step 3: Mapping OF Binary 1:1 Relationship Types

- one-to-one relationship:



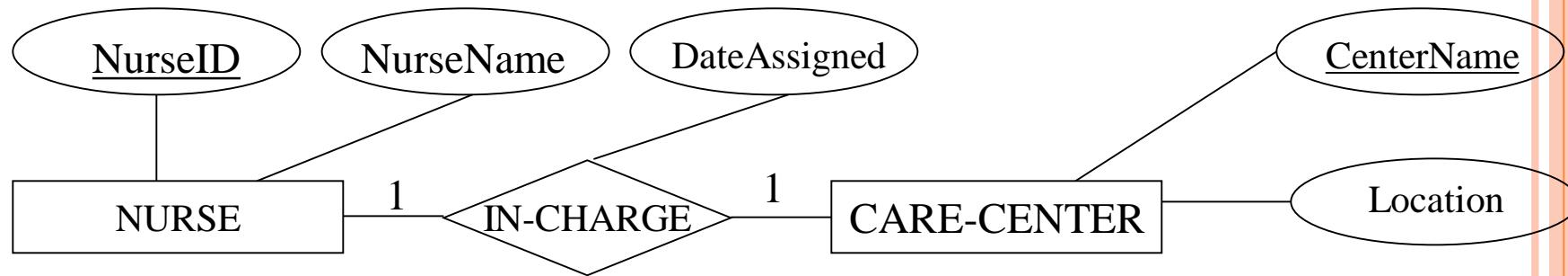
- Mapping the relationship (**Merged relation option**):

**NURSE**

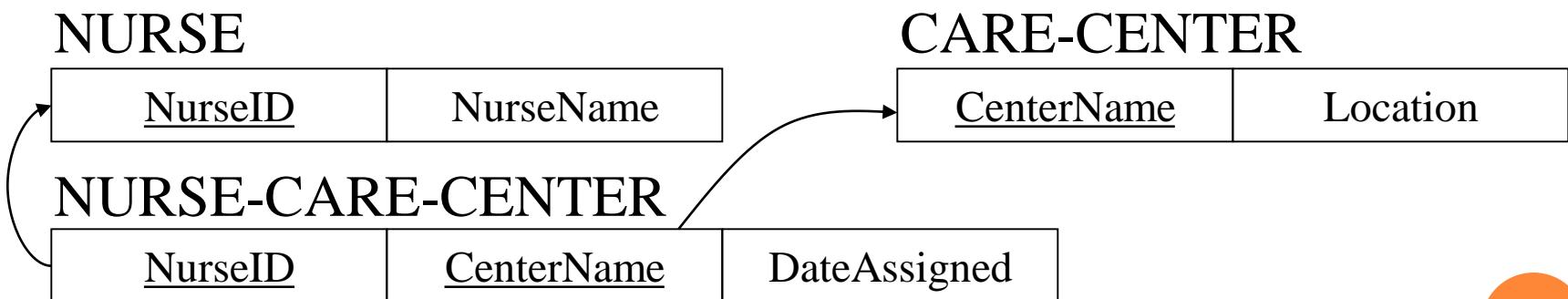
<u>NurseID</u>	NurseName	CenterName	Location	DateAssigned
----------------	-----------	------------	----------	--------------

# Step 3: Mapping OF Binary 1:1 Relationship Types

- one-to-one relationship:



- Mapping the relationship (**Cross-reference option**):



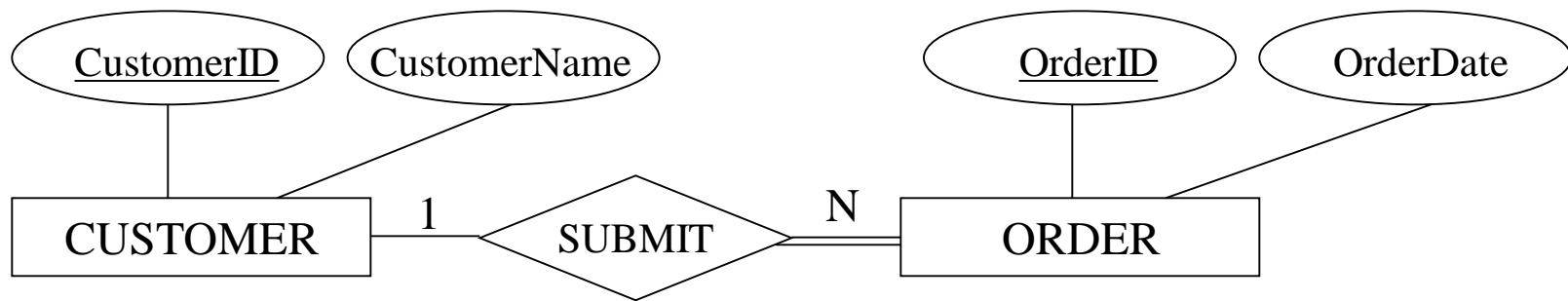
## Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type.
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
- Include any simple attribute of the 1:N relationship type as attributes of S.



# Step 4: Mapping of Binary 1:N Relationship Types

- One-to-many relationship:



- Mapping the relationship (**foreign key approach**):

CUSTOMER

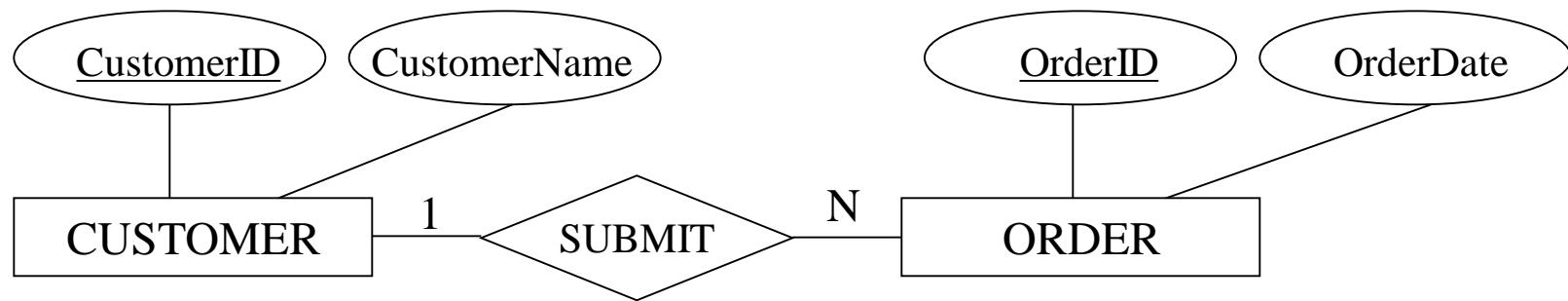
<u>CustomerID</u>	CustomerName
-------------------	--------------

ORDER

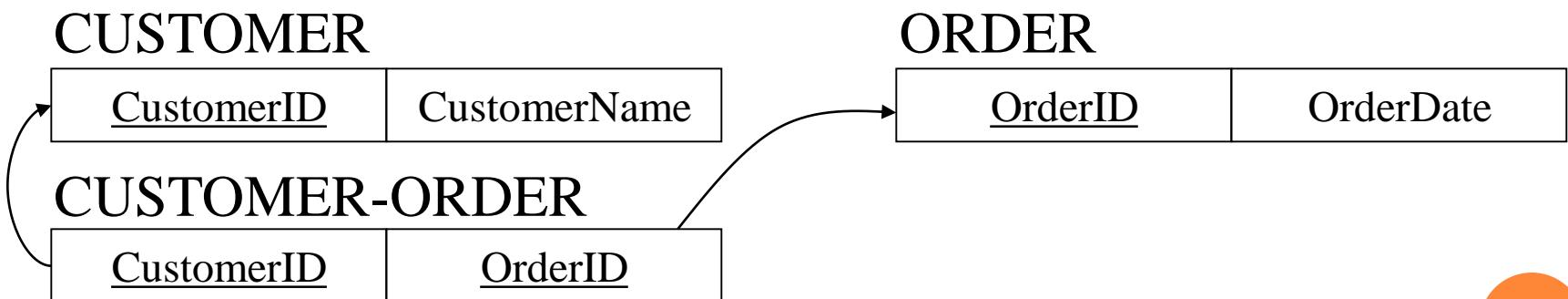
<u>OrderID</u>	OrderDate	CustomerID
----------------	-----------	------------

# Step 4: Mapping of Binary 1:N Relationship Types

- One-to-many relationship:



- Mapping the relationship (**Cross-reference option**):



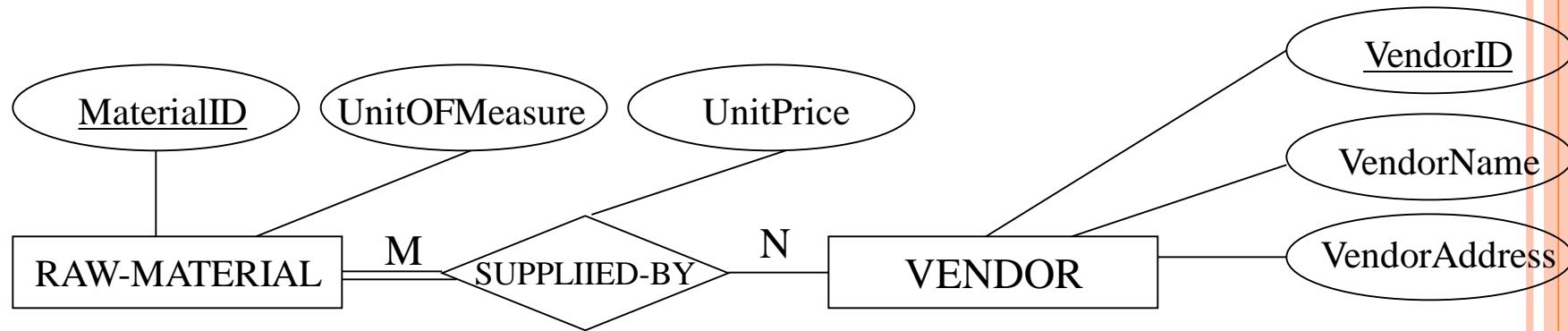
## Step 5: Mapping of Binary M:N Relationship Types

- For each binary M:N relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
- Include any simple attributes of the M:N relationship type as attributes of S.

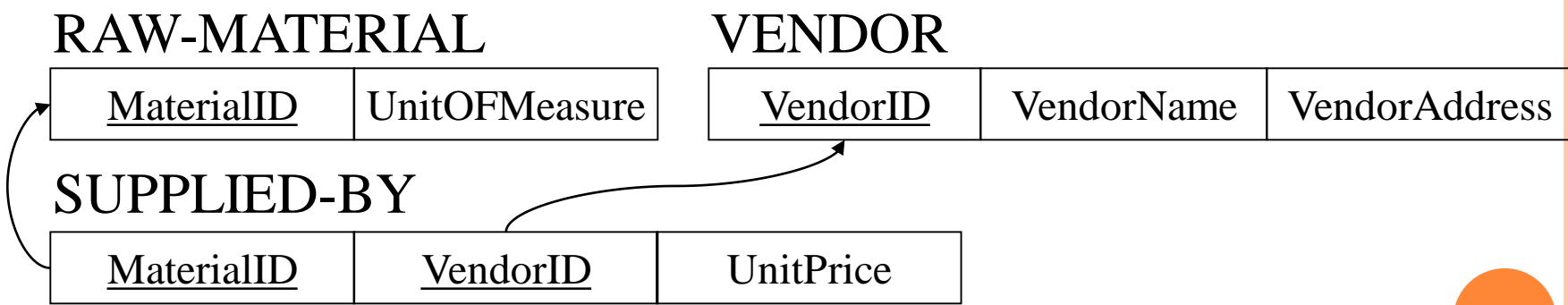


# Step 5: Mapping of Binary M:N Relationship Types

- Many-to-many relationship:



- Mapping the relationship:

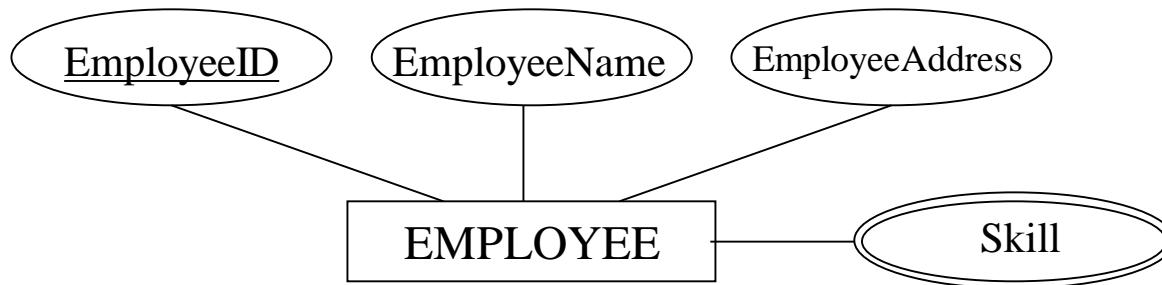


## Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K - as a foreign key in R - of the relation that represents the entity type or relationship type that has A as an attribute.
- The primary key of R is the combination of A and K.
- If the multivalued attribute is composite, include its simple components.

# Step 6: Mapping of Multivalued Attributes

- EMPLOYEE entity type with a multivalued attribute:



- Multivalued attribute becomes a separate relation with a foreign key:

EMPLOYEE		
<u>EmployeeID</u>	EmployeeName	EmployeeAddress
EMPLOYEE_SKILL		
<u>EmployeeID</u>		<u>Skill</u>

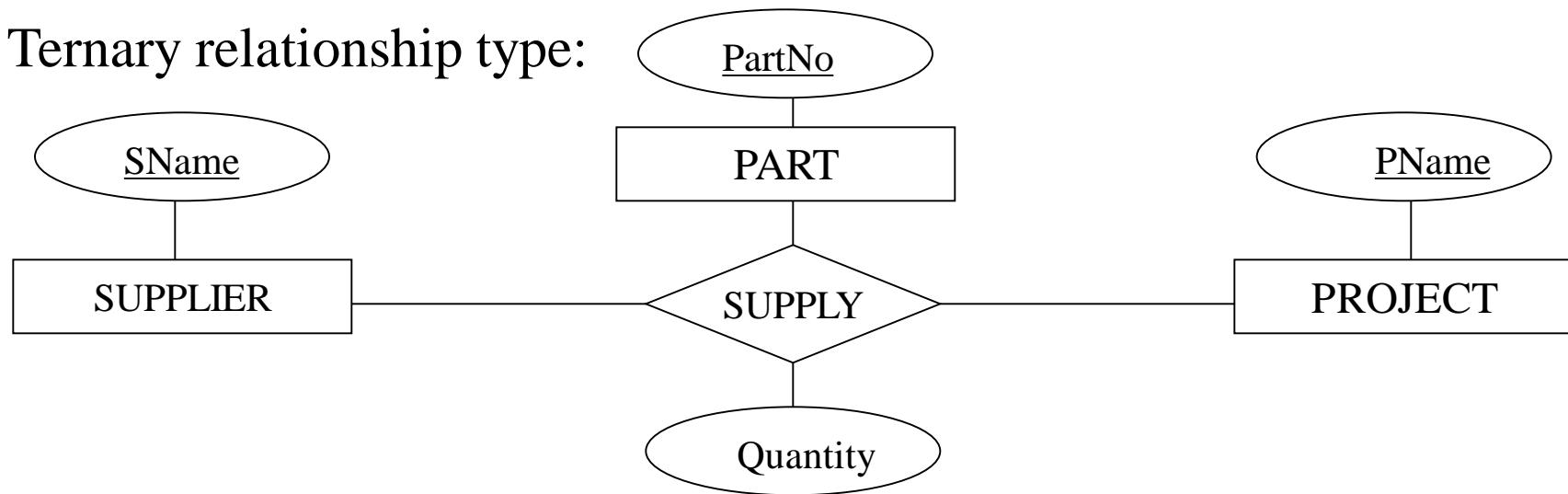
## Step 7: Mapping of N-ary Relationship Types

- For each n-ary relationship type R, where  $n > 2$ , create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Include any simple attributes of the n-ary relationship type as attributes of S.

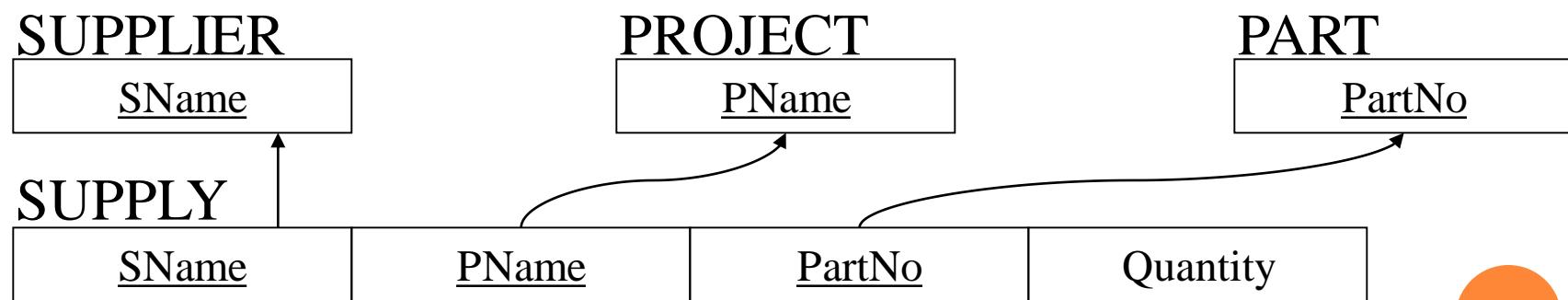


# Step 7: Mapping of N-ary Relationship Types

- Ternary relationship type:



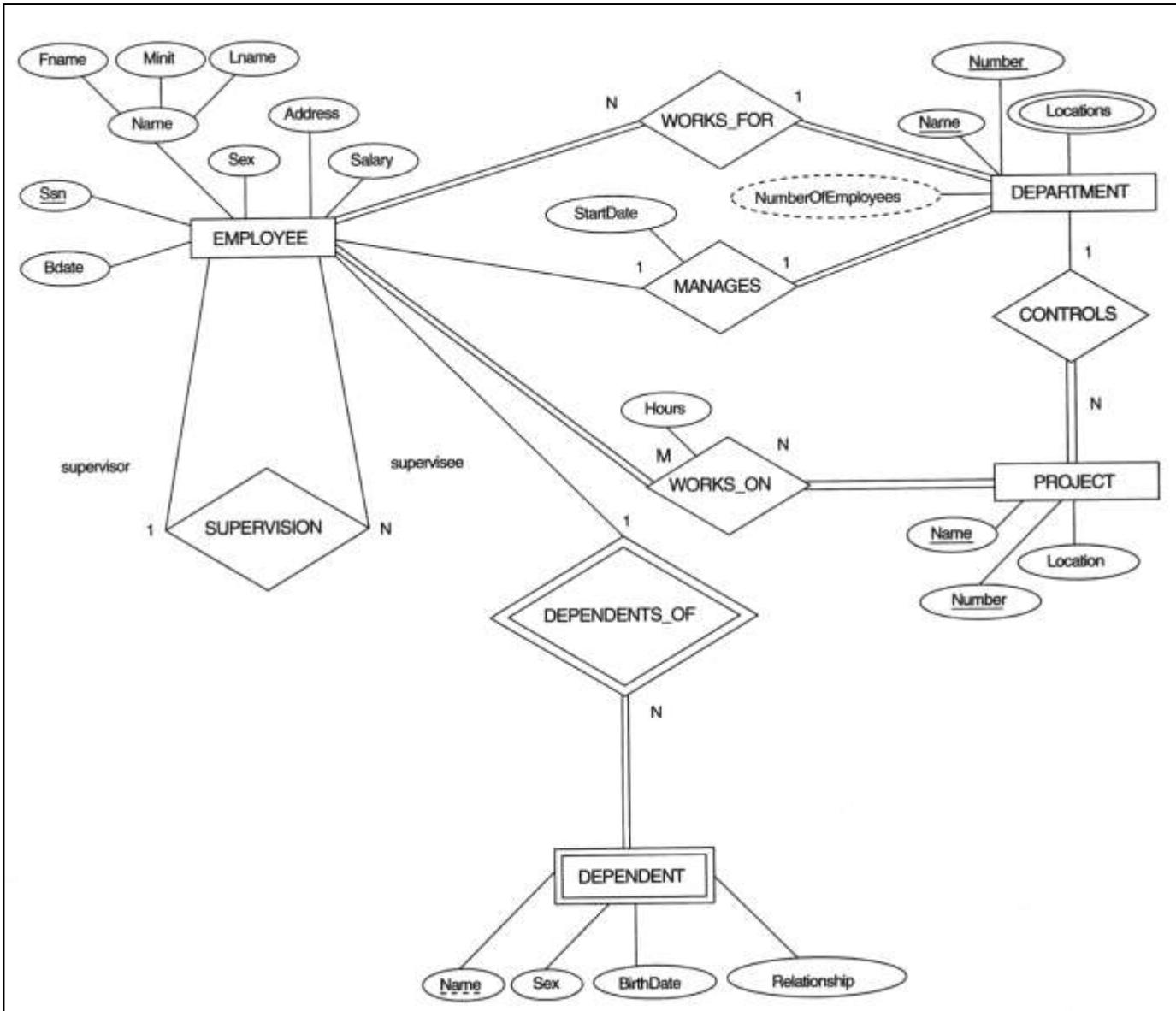
- Mapping the n-ary relationship type **SUPPLY**:



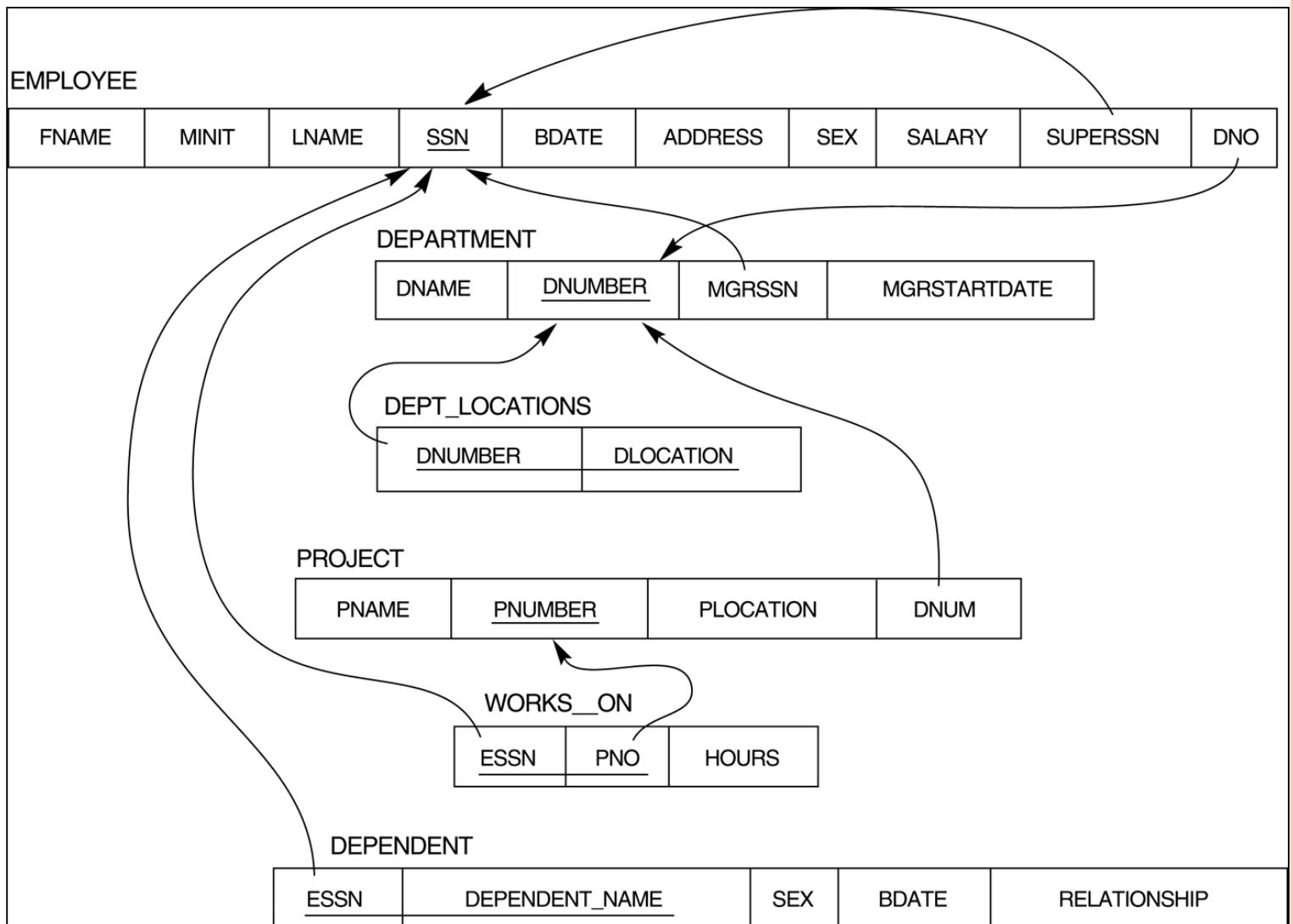
# Correspondence Between ER and Relational Models

ER Model	Relational Model
Entity type	“Entity” relation
1:1 or 1:N relationship type	Foreign key (or “relationship” relation)
M:N relationship type	“Relationship” relation and two foreign keys
n-ary relationship type	“Relationship” relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Mutivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

# Example



# Example



# Step 8: Options for Mapping Specialization or Generalization

## Option 8A: Multiple Relations - Superclass and Subclasses

This option works for any specialization.

### SECRETARY

<u>SSN</u>	TypingSpeed
------------	-------------

### TECHNICIAN

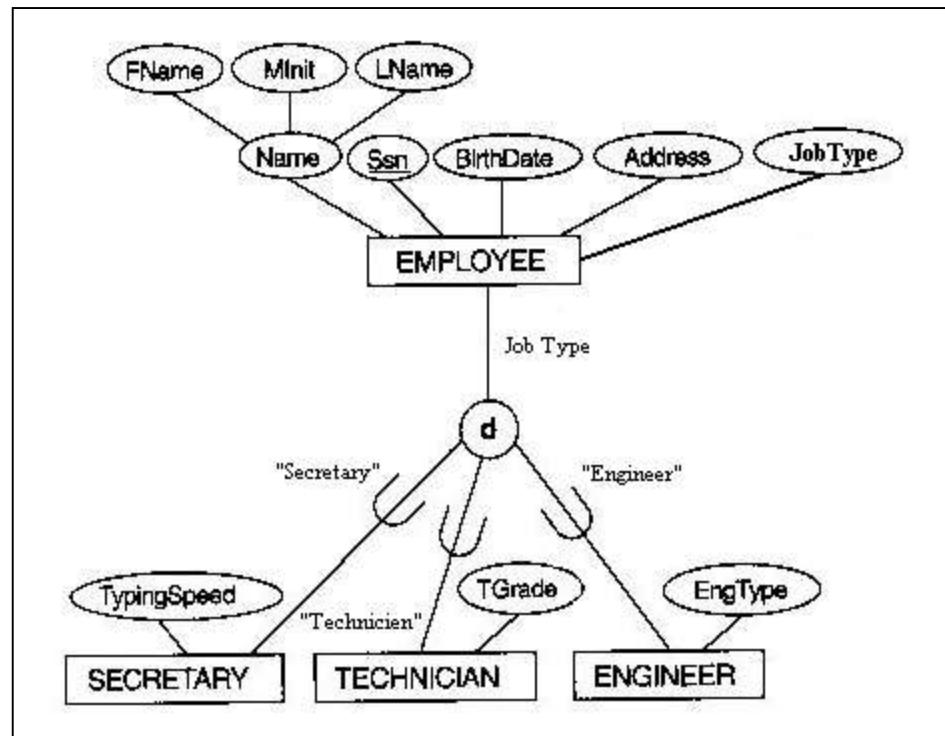
<u>SSN</u>	TGrade
------------	--------

### ENGINEER

<u>SSN</u>	EngType
------------	---------

### EMPLOYEE

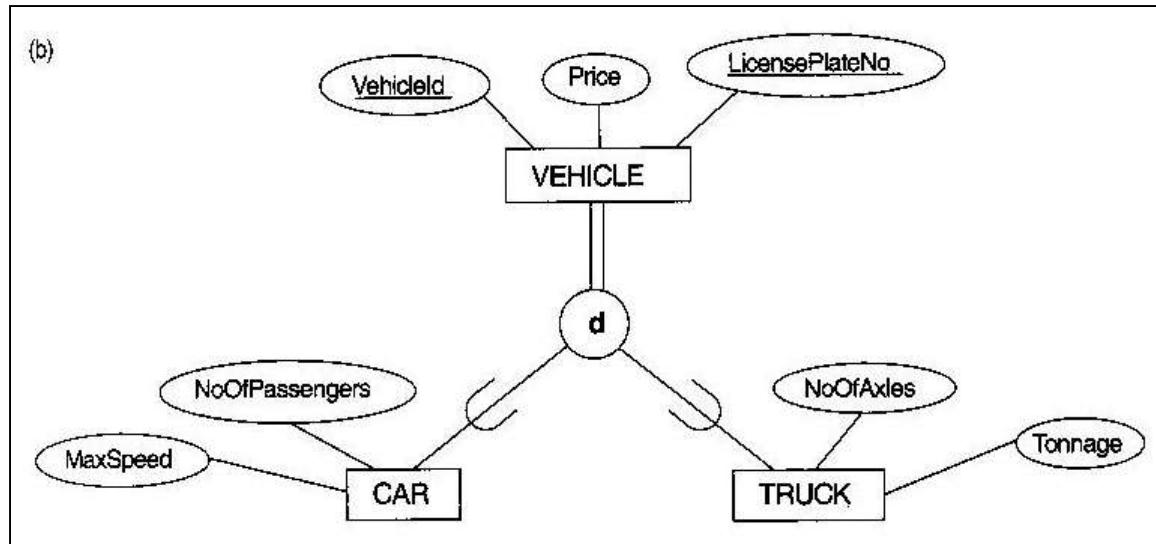
<u>SSN</u>	FName	Minit	LName	BirthDate	Address	JobType
------------	-------	-------	-------	-----------	---------	---------



# Step 8: Options for Mapping Specialization or Generalization

## Option 8B: Multiple Relations - Subclass Relations Only

This option only works for a specialization whose subclasses are total.



### CAR

<u>VehicleID</u>	LicensePlateNo	Price	MaxSpeed	NoOfPassengers
------------------	----------------	-------	----------	----------------

### TRUCK

<u>VehicleID</u>	LicensePlateNo	Price	NoOfAxies	Tonnage
------------------	----------------	-------	-----------	---------

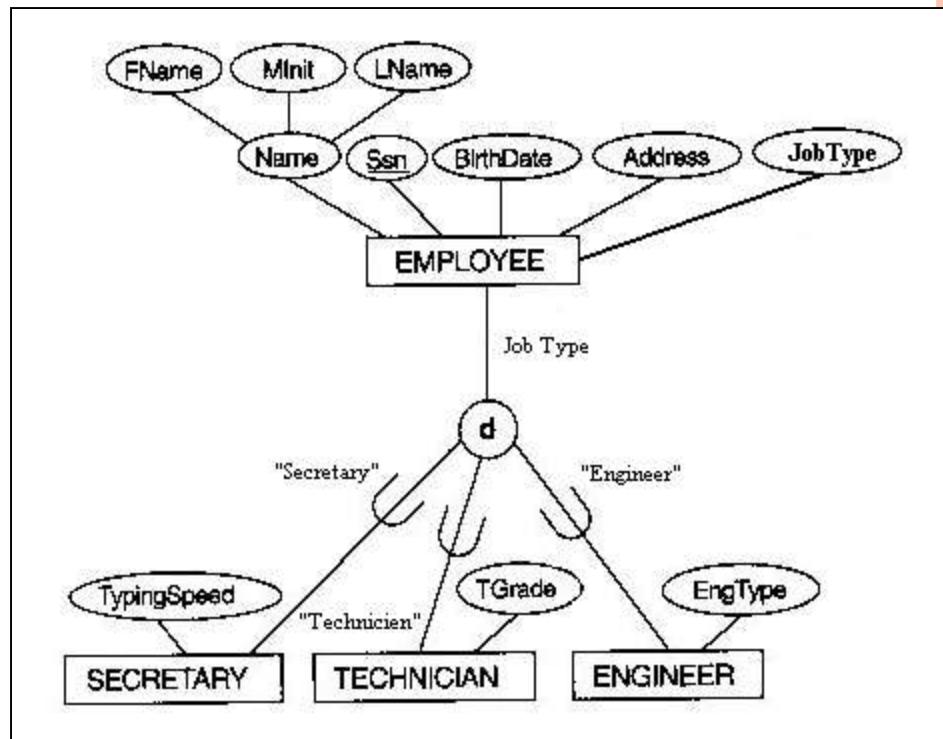


# Step 8: Options for Mapping Specialization or Generalization

## Option 8C: Single Relation With One Type Attribute

This option:

- Works only for a specialization whose subclasses are disjoint.
- Has the potential for generating many null values if many specific attributes exist in the subclasses.



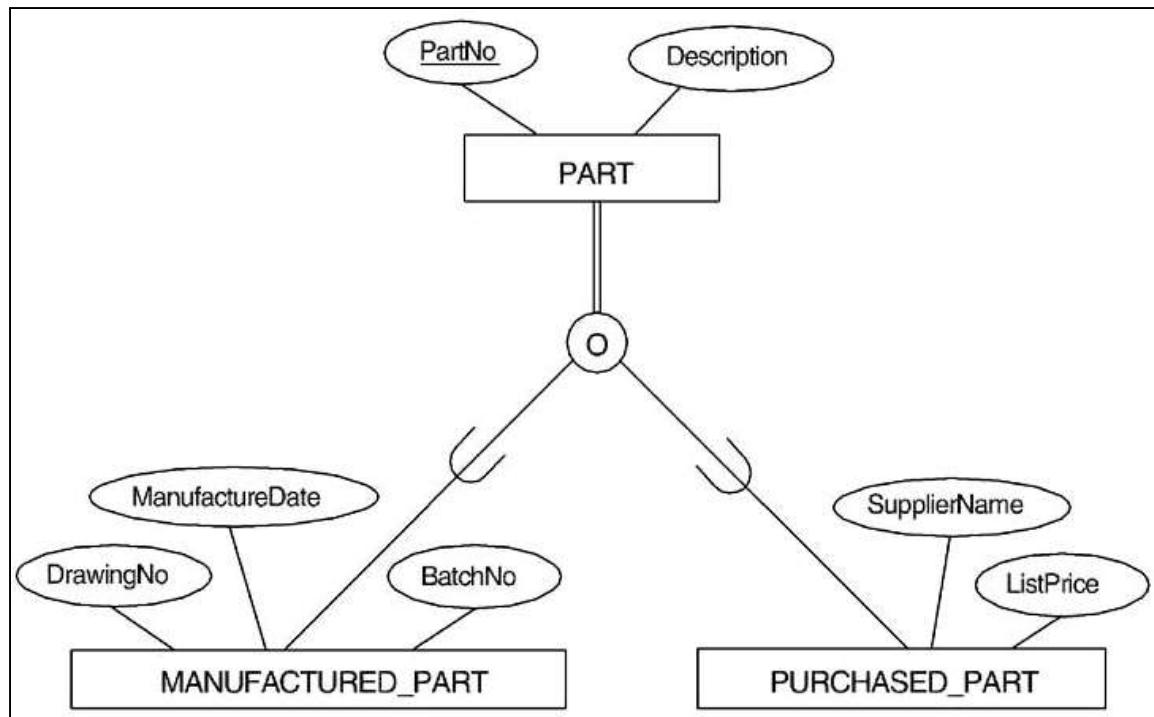
## EMPLOYEE

SSN	FName	Minit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	EngType
-----	-------	-------	-------	-----------	---------	---------	-------------	--------	---------

## Step 8: Options for Mapping Specialization or Generalization

### Option 8D: Single Relation With Multiple Type Attributes

This option works for a specialization whose subclasses are overlapping  
(It also works for a disjoint specialization).



### PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice
--------	-------------	-------	-----------	-----------------	---------	-------	--------------	-----------

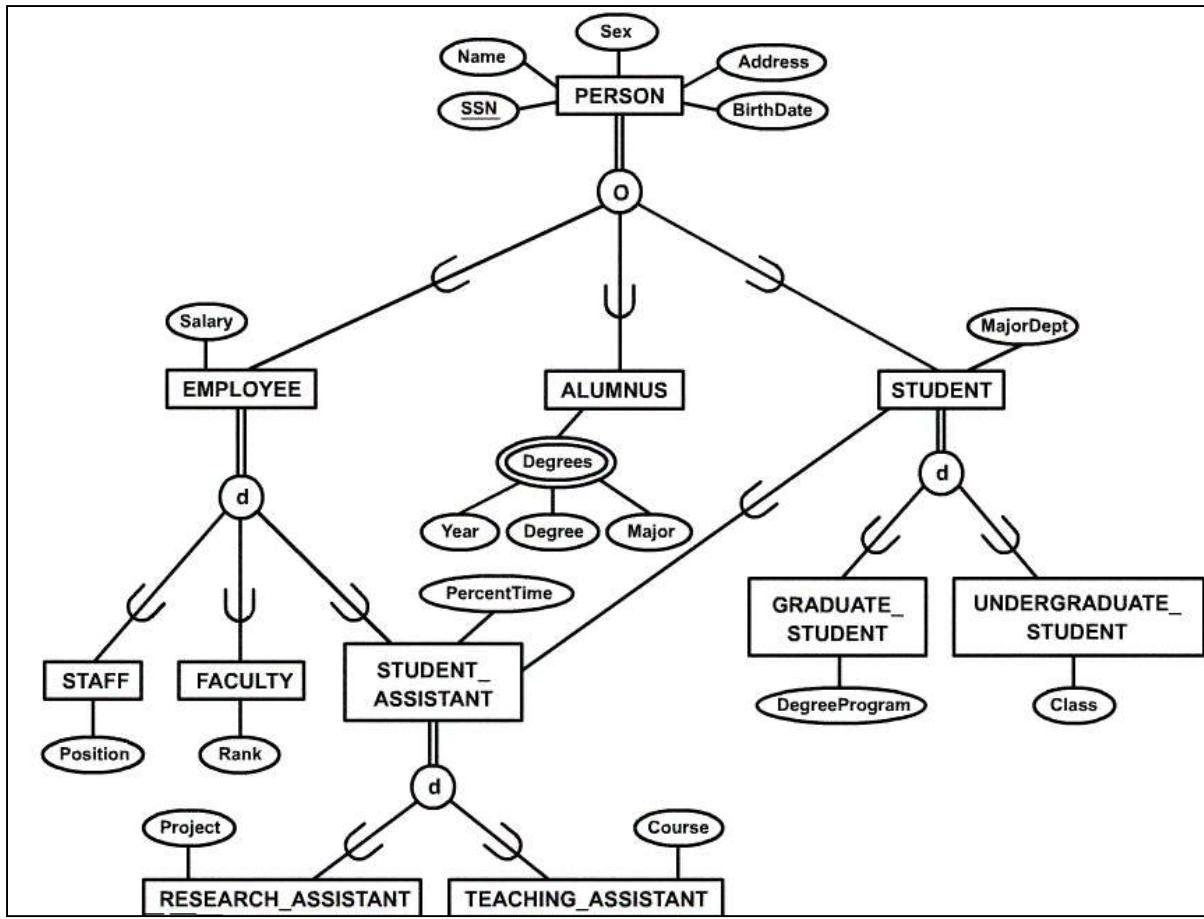
## Step 8: Options for Mapping Specialization or Generalization

### Mapping of Shared Subclasses (Multiple Inheritance)

- Shared subclasses must have the same key attribute; otherwise the shared subclass would be modeled as a category.
- Any of the options discussed in step 8 can be applied to a shared subclass.

# Step 8: Options for Mapping Specialization or Generalization

## Mapping of Shared Subclasses (Multiple Inheritance)



## Step 8: Options for Mapping Specialization or Generalization

### Mapping of Shared Subclasses (Multiple Inheritance)

#### PERSON

<u>SSN</u>	Name	BirthDate	Sex	Address
------------	------	-----------	-----	---------

#### EMPLOYEE

<u>SSN</u>	Salary	EmployeeType	Position	Rank	PercentTime	RAFlag	TAFlag	Project	Course
------------	--------	--------------	----------	------	-------------	--------	--------	---------	--------

#### STUDENT

<u>SSN</u>	MajorDept	GradFlag	UndergradFlag	DegreeProgram	Class	StudAssistFlag
------------	-----------	----------	---------------	---------------	-------	----------------

#### ALUMNUS

<u>SSN</u>
------------

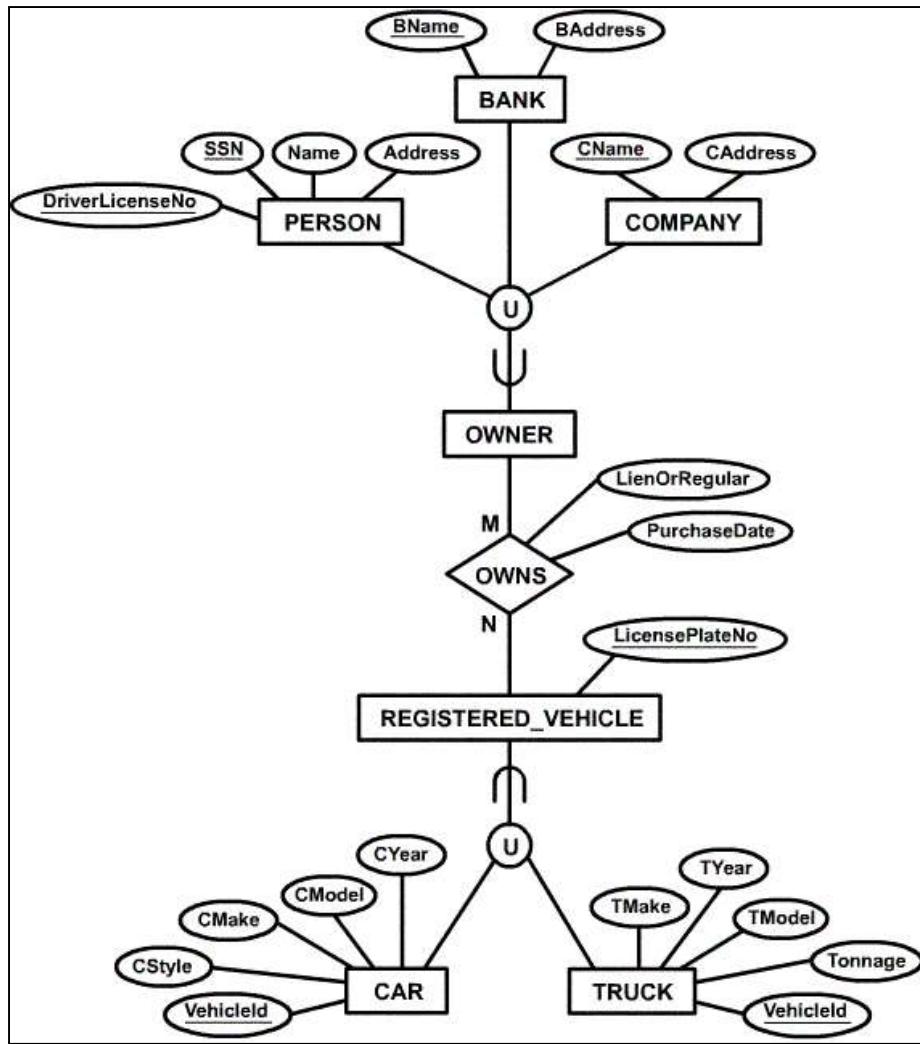
#### ALUMNUS-DEGREE

<u>SSN</u>	<u>Year</u>	<u>Degree</u>	<u>Major</u>
------------	-------------	---------------	--------------

## Step 9: Mapping of Union Types (Categories)

- For mapping a category whose defining subclasses have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.

# Step 9: Mapping of Union Types (Categories)



# Step 9: Mapping of Union Types (Categories)

## PERSON

<u>SSN</u>	DriverLicenseNo	Name	Address	OwnerID
------------	-----------------	------	---------	---------

## BANK

<u>BName</u>	BAddress	OwnerID
--------------	----------	---------

## COMPANY

<u>CName</u>	CAddress	OwnerID
--------------	----------	---------

## OWNER

<u>OwnerID</u>
----------------

## REGISTERED-VEHICLE

<u>VehicleID</u>	LicensePlateNumber
------------------	--------------------

## CAR

<u>VehicleID</u>	CStyle	CMake	CModel	CYear
------------------	--------	-------	--------	-------

## TRUCK

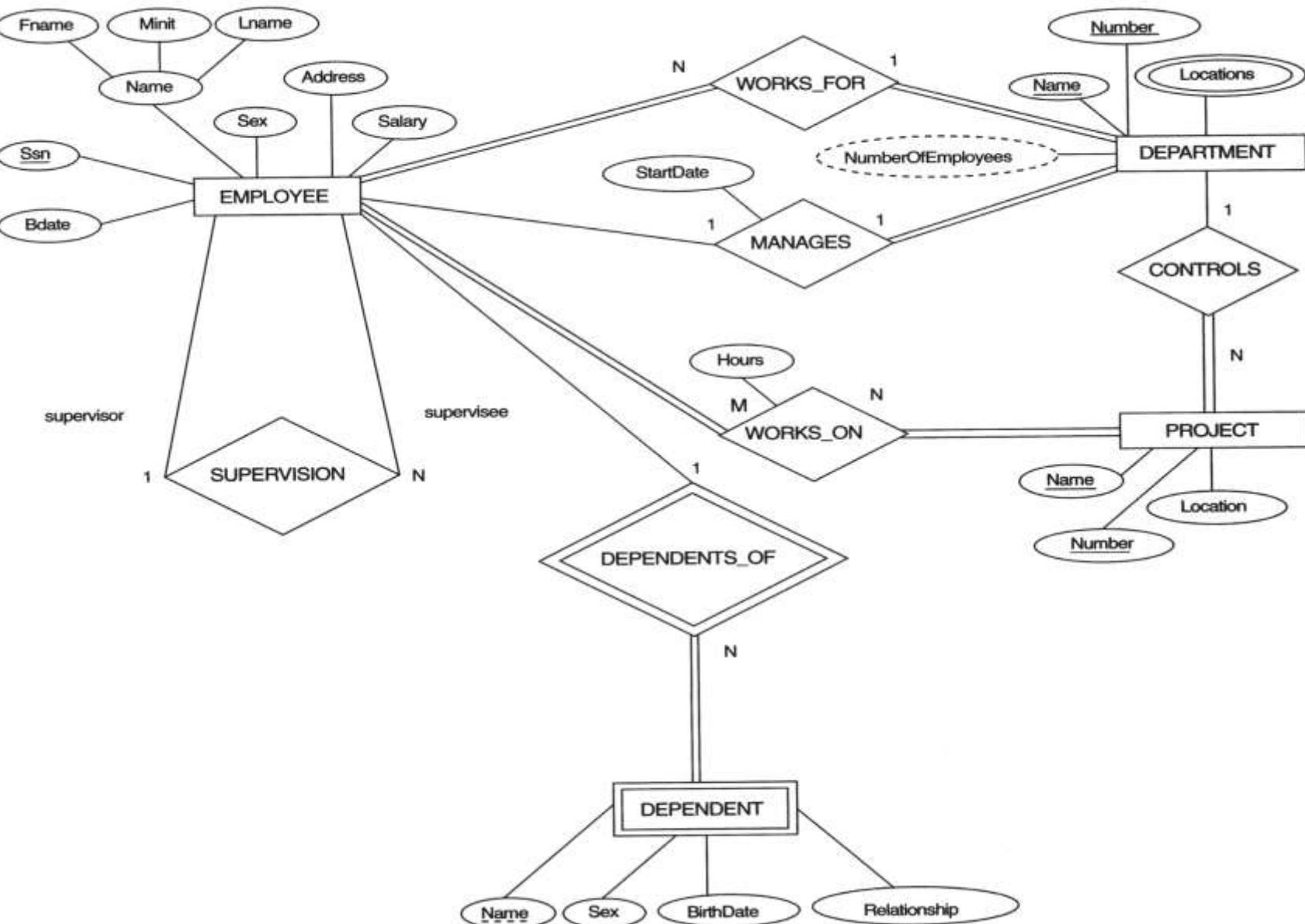
<u>VehicleID</u>	TStyle	TMake	Tonnage	TYear
------------------	--------	-------	---------	-------

## OWNS

<u>OwnerID</u>	<u>VehicleID</u>	PurchaseDate	LienOrRegular
----------------	------------------	--------------	---------------

## Hand Outs

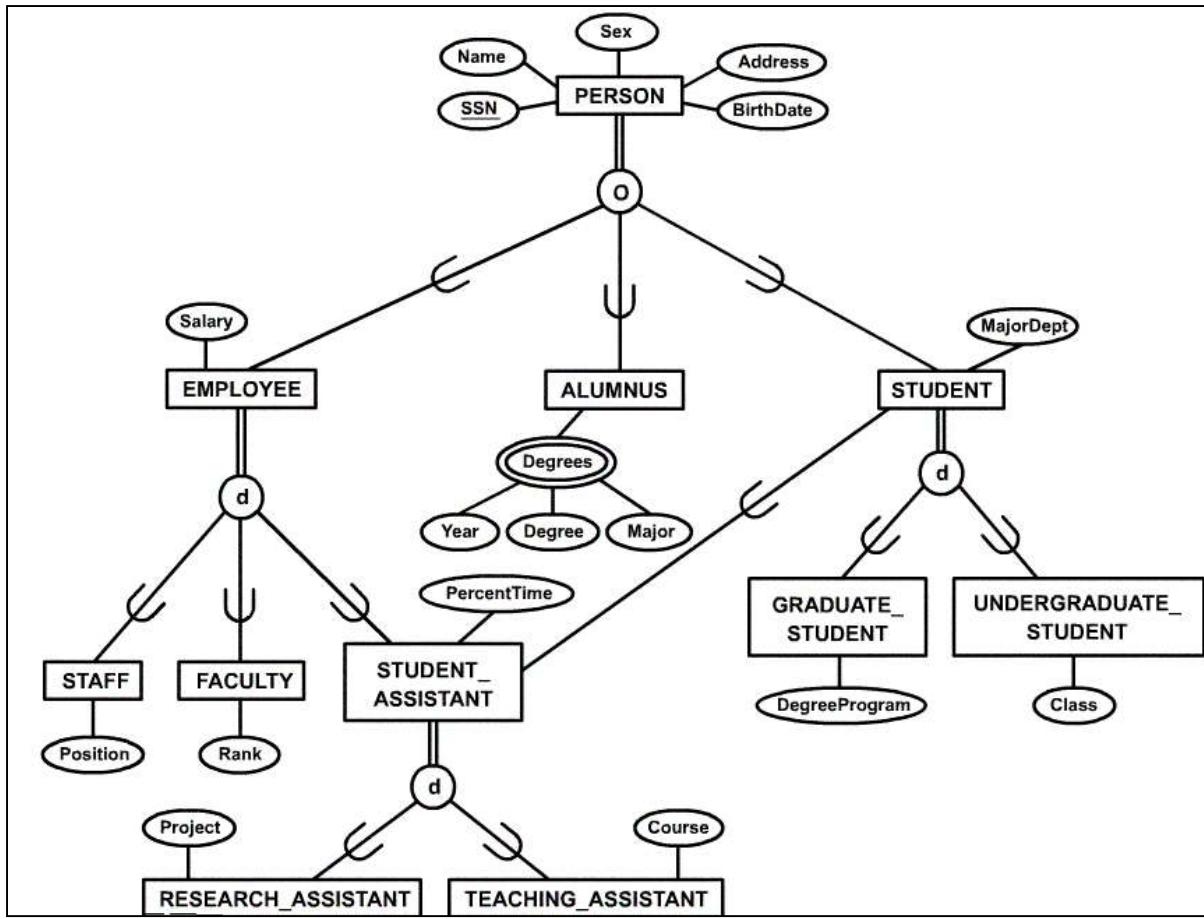




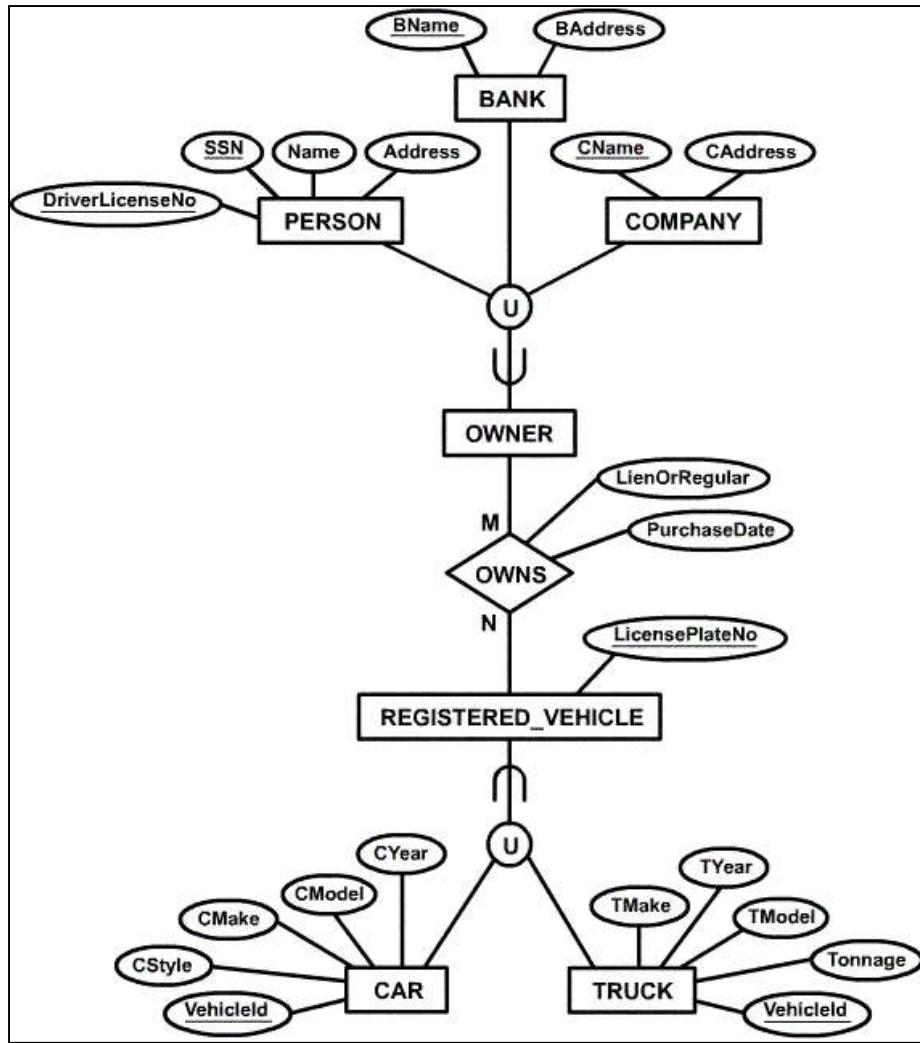


# Step 8: Options for Mapping Specialization or Generalization

## Mapping of Shared Subclasses (Multiple Inheritance)



# Step 9: Mapping of Union Types (Categories)



**CS 380**  
**Introduction to Database Systems**

**Chapter 6: The Relational Algebra and Relational Calculus**

# Outline

- Introduction
- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations from Set Theory
  - Binary Relational Operations
  - Additional relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus
  - Introduction to Tuple Relational Calculus
  - Introduction to Domain Relational Calculus

# Introduction

- A data model must include a set of operations to manipulate relations and produce new relations as answers to queries.
- Formal languages for the relational model:
  - Relational algebra: specifies a sequence of operations to specify a query.
  - Relational calculus: specifies the result of a query.  
(without specifying how to produce the query result)
    - Tuple calculus.
    - Domain Calculus.

# Example

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	Locatoin
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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

# Relational Algebra

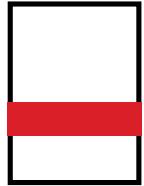
- The basic set of operations for the relational model is known as the **relational algebra**.
- These operations enable a user to specify basic retrieval requests.
- The result of a retrieval is a new relation, which may have been formed from one or more relations.
- A sequence of relational algebra operations forms a **relational algebra expression**.

# Relational Algebra

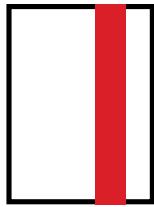
- Two basic sets of operations:
  - Relational operators (specific for relational databases):
    - Select.
    - Project.
    - Join.
    - Division.
  - Set Theoretic Operators:
    - Union.
    - Intersection.
    - Minus.
    - Cartesian product.

# Basic Relational Algebra Operations

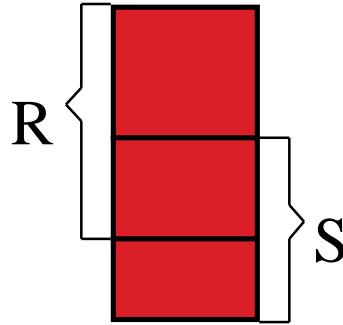
Select



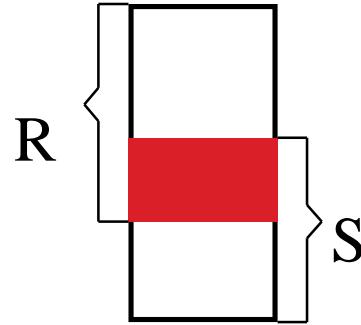
Project



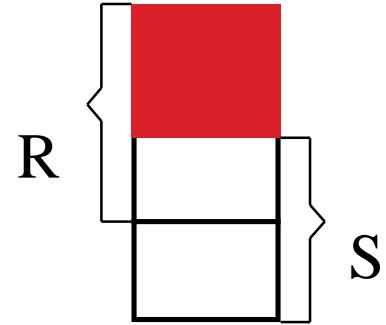
Union



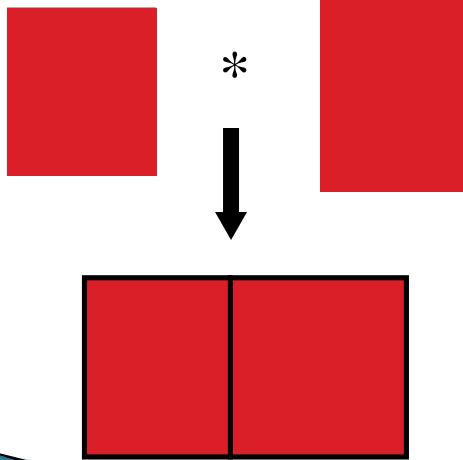
Intersection



Difference



Cartesian Product



# Unary Relational Operations - SELECT

- Selects a subset of the tuples from a relation that satisfy a selection condition.

- Syntax:

$$\sigma_{<\text{selection condition}>} (<\text{relation name}>)$$

- Examples:

- Select the EMPLOYEE tuples whose department is 4.

$$\sigma_{Dno=4} (\text{EMPLOYEE})$$

- Select the tuples for all employees who either work in department 4 and make over \$25,000 per year, or work in department 5 and make over \$30,000.

$$\sigma_{(Dno=4 \text{ AND } \text{Salary}>25000) \text{ OR } (Dno=5 \text{ AND } \text{Salary}>30000)} (\text{EMPLOYEE})$$

# Unary Relational Operations - SELECT

$\sigma_{(Dno=4 \text{ AND } \text{Salary}>25000) \text{ OR } (Dno=5 \text{ AND } \text{Salary}>30000)}(\text{EMPLOYEE})$

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

# Unary Relational Operations - SELECT

- The SELECT operation  $\sigma_{<\text{selection condition}>} (R)$  produces a relation S that has the same schema as R

- The select operation is commutative:

$$\sigma_{<\text{condition1}>} (\sigma_{<\text{condition2}>} (R)) = \sigma_{<\text{condition2}>} (\sigma_{<\text{condition1}>} (R))$$

- A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions:

$$\begin{aligned} & \sigma_{<\text{condition1}>} (\sigma_{<\text{condition2}>} (\sigma_{<\text{condition3}>} (R))) \\ &= \sigma_{<\text{condition1}> \text{AND} <\text{condition2}> \text{ AND} <\text{condition3}>} (R) \end{aligned}$$

# Unary Relational Operations - PROJECT

- Selects certain columns from the table and discards the other columns.

- Syntax:

$$\pi_{<\text{attribute list}>} (<\text{relation name}>)$$

- Example: list each employee's first and last name and salary.

$$\pi_{\text{Lname, Fname, Salary}} (\text{EMPLOYEE})$$

- The project operation removes any duplicate tuples, so the result of the project operation is a set of tuples and hence a valid relation.

# Unary Relational Operations - PROJECT

$\pi$  Lname, Fname, Salary (EMPLOYEE)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

# Sequence of Operations & RENAME Operation

- We can:
  - Nest the relational algebra operations as a single expression, or
  - Apply one operation at a time and create intermediate result relations, and give it a name.
- Example: retrieve the first name, last name, and salary of all employees who work in department number 5.

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

or

$$\text{TEMP} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$
$$R \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{TEMP})$$

- The rename operator is  $\rho$ .

# Sequence of Operations & RENAME Operation

$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

$\text{TEMP} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$

$\text{R}(\text{First\_name}, \text{Last\_name}, \text{Salary}) \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{TEMP})$

TEMP									
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
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

R		
First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

# Relational Algebra Operations From Set Theory

## UNION, INTERSECTION, & MINUS

- Operands need to be union compatible for the result to be a valid relation.
- In practice, it is rare that two relations are union compatible (occurs most often in derived relations).

# Relational Algebra Operations From Set Theory - UNION

- The result of this operation, denoted by  $R \cup S$ , is a relation that includes all tuples that are either in  $R$  or in  $S$  or in both  $R$  and  $S$ .
- Duplicate tuples are eliminated.

# Relational Algebra Operations From Set Theory - UNION

- Example: retrieve the SSN of all employees who either work in department 5 or directly supervise an employee who works in department 5.

$$\text{DEP5_EMPS} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$
$$\text{RESULT1} \leftarrow \pi_{\text{Ssn}}(\text{DEP5_EMPS})$$
$$\text{RESULT2}(\text{Ssn}) \leftarrow \pi_{\text{Super\_ssn}}(\text{DEP5_EMPS})$$
$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

RESULT1	RESULT2	RESULT
Ssn	Ssn	Ssn
123456789		123456789
333445555	333445555	333445555
666884444	888665555	666884444
453453453		453453453
		888665555

# Relational Algebra Operations From Set Theory

## - INTERSECTION

- The result of this operation, denoted by  $R \cap S$ , is a relation that includes all tuples that are in both  $R$  and  $S$ .

## Relational Algebra Operations From Set Theory - MINUS

- The result of this operation, denoted by  $R - S$ , is a relation that includes all tuples that are in  $R$  but not in  $S$ .

# The Set Operations

(a) STUDENT		INSTRUCTOR	
Fn	Ln	Fname	Lname
Susan	Yao	John	Smith
Ramesh	Shah	Ricardo	Browne
Johnny	Kohler	Susan	Yao
Barbara	Jones	Francis	Johnson
Amy	Ford	Ramesh	Shah
Jimmy	Wang		
Ernest	Gilbert		

(b)			
Fn	Ln		
Susan	Yao		
Ramesh	Shah		
Johnny	Kohler		
Barbara	Jones		
Amy	Ford		
Jimmy	Wang		
Ernest	Gilbert		
John	Smith		
Ricardo	Browne		
Francis	Johnson		

(c)		(d)		(e)	
Fn	Ln	Fn	Ln	Fname	Lname
Susan	Yao	Johnny	Kohler	John	Smith
Ramesh	Shah	Barbara	Jones	Ricardo	Browne
		Amy	Ford	Francis	Johnson
		Jimmy	Wang		
		Ernest	Gilbert		

- b. STUDENT  $\cup$  INSTRUCTOR
- c. STUDENT  $\cap$  INSTRUCTOR
- d. STUDENT - INSTRUCTOR
- e. INSTRUCTOR - STUDENT

# Relational Algebra Operations From Set Theory

## CARTESIAN PRODUCT

- This operation is used to combine tuples from two relations in a combinational fashion.
- The result denoted by  $R_1 \times R_2$  is a relation that includes all the possible combinations of tuples from  $R_1$  and  $R_2$ .
- It is not a very useful operation by itself but it is used in conjunction with other operations.

# Relational Algebra Operations From Set Theory

## CARTESIAN PRODUCT

- Example: retrieve a list of names of each female employee's dependents.

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{Sex}='F'}(\text{EMPLOYEE})$$
$$\text{EMP NAMES} \leftarrow \pi_{\text{Fname}, \text{Lname}, \text{Ssn}}(\text{FEMALE\_EMPS})$$
$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMP NAMES} \times \text{DEPENDENT}$$
$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn}=\text{Essn}}(\text{EMP\_DEPENDENTS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname}, \text{Lname}, \text{Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$$

# Example

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	Locatoin
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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

# Relational Algebra Operations From Set Theory

## CARTESIAN PRODUCT

FEMALE_EMPS	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES	FNAME	LNAME	SSN
	Alicia	Zelaya	999887777
	Jennifer	Wallace	987654321
	Joyce	English	453453453

EMP_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	***
	Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	***
	Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	***
	Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	***
	Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	***
	Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	***
	Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	***
	Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	***
	Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	***
	Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	***
	Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	***
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	***
	Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	***
	Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	***
	Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	***
	Joyce	English	453453453	333445555	Alice	F	1986-04-05	***
	Joyce	English	453453453	333445555	Theodore	M	1983-10-25	***
	Joyce	English	453453453	333445555	Joy	F	1958-05-03	***
	Joyce	English	453453453	987654321	Abner	M	1942-02-28	***
	Joyce	English	453453453	123456789	Michael	M	1988-01-04	***
	Joyce	English	453453453	123456789	Alice	F	1988-12-30	***
	Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	***

ACTUAL_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	***
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	***

RESULT	FNAME	LNAME	DEPENDENT_NAME
	Jennifer	Wallace	Abner

# Completeness of Relational Algebra

- SELECT, PROJECT, UNION, MINUS, and CARTESIAN PRODUCT are the basic operators of the relational algebra.
- Additional operators are defined as combination of two or more of the basic operations.
- Example:
  - JOIN = CARTESIAN PRODUCT + SELECT.
  - DIVISION = PROJECT + CARTESIAN PRODUCT + MINUS.

# Binary Relational Operations - JOIN

- The JOIN operation is used to combine related tuples from two relations into single tuples.
- Syntax:  $R \bowtie_{\text{join condition}} S$  (does not require union compatibility of R and S).
- Example: retrieve the name of the manager of each department.  
 $\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE}$   
 $\text{RESULT} \leftarrow \pi_{\text{Dname}, \text{Lname}, \text{Fname}} (\text{DEPT\_MGR})$

DEPT_MGR								
Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

# Binary Relational Operations - EQUIJOIN

- Joins conditions with equality comparisons only.
- In the result of an EQUIJOIN, one or more pairs of attributes always have identical values in every tuple.

# Binary Relational Operations - NATURAL JOIN

- Because one of each pair of attributes with identical values is superfluous, a new operation called NATUARAL JOIN was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
- The standard definition requires that the two join attributes, or each pair of corresponding join attributes, have the same name in both relations.
- Syntax:  $R * S$

# Binary Relational Operations - NATURAL JOIN

- To apply a NATURAL JOIN on the DNUMBER attribute of DEPARTMENT and DEPT\_LOCATIONS, it is sufficient to write:  
$$\text{DEPT\_LOCS} \leftarrow \text{DEPARTMENT} * \text{DEPT\_LOCATIONS}$$

DEPT_LOCS				
Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

# Binary Relational Operations - NATURAL JOIN

- To apply a NATURAL JOIN on the department number attribute of DEPARTMENT and PROJECT:

$$\text{DEPT} \leftarrow \rho_{(\text{Dname}, \text{Dnum}, \text{Mgr\_ssn}, \text{Mgr\_start\_date})} (\text{DEPARTMENT})$$
$$\text{PROJ\_DEPT} \leftarrow \text{PROJECT} * \text{DEPT}$$

PROJ_DEPT						
Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

# Binary Relational Operations - DIVISION

- The division operation is applied to two relations  $R(Z) \div S(X)$ , where  $X$  is a subset from  $Z$ .
- Example: retrieve the names of employees who work on all the projects that ‘John Smith’ works on.

$$SMITH \leftarrow \sigma_{Fname='John' \text{ AND } Lname='Smith'}(EMPLOYEE)$$
$$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$$
$$SSN\_PNOS \leftarrow \pi_{Essn, Pno}(WORKS\_ON)$$
$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$
$$RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMPLOYEE)$$

# Binary Relational Operations - DIVISION

SSN_PNOS		SMITH_PNOS		SSNS	
Essn	Pno	Pno		Ssn	
123456789	1	1		123456789	
123456789	2	2		453453453	
666884444	3				
453453453	1				
453453453	2				
333445555	2				
333445555	3				
333445555	10				
333445555	20				
999887777	30				
999887777	10				
987987987	10				
987987987	30				
987654321	30				
987654321	20				
888665555	20				

# Additional Relational Operations

## Aggregate Functions and Grouping

- The first type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collection of values from the database.
- Common functions applied to collections of numeric values include:
  - SUM.
  - AVERAGE.
  - MAXIMUM.
  - MINIMUM.
- The COUNT function is used for counting tuples or values.

# Additional Relational Operations

## Aggregate Functions and Grouping

- Another common type of request involves grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.
- Syntax:  $\underset{\text{grouping attributes}}{\mathfrak{T}} \underset{\text{function list}}{\mathcal{J}} (R)$
- Example: COUNT<sub>Ssn</sub>, AVERAGE<sub>Salary</sub> (EMPLOYEE)

Count_ssn	Average_salary
8	35125

# Additional Relational Operations

## Aggregate Functions and Grouping

- Example:  $\text{Dno} \not\exists \text{ COUNT}_{\text{Ssn}}, \text{AVERAGE}_{\text{Salary}} (\text{EMPLOYEE})$

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

- Example:  $\rho_{(\text{Dno}, \text{No\_of\_employees}, \text{Average\_sal})} (\text{Dno} \not\exists \text{COUNT}_{\text{Ssn}}, \text{AVERAGE}_{\text{Salary}} (\text{EMPLOYEE}))$

R		
Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

# Additional Relational Operations

## OUTER JOIN Operations

- In NATURAL JOIN, the following tuples are eliminated from the join result:
  - Tuples without a matching (or related) tuple.
  - Tuples with null in the join attributes.
- A set of operations, called OUTER JOINS, can be used when we want to keep all the tuples:
  - in R, or
  - in S, or
  - in both relationsin the result of the join.

# Additional Relational Operations

## OUTER JOIN Operations

- The left outer join operation ( $R \bowtie S$ ) keeps every tuple in R, if no matching tuple is found in S, then the attributes of S in the join result are filled with null values.
- The right outer join operation ( $R \bowtie^r S$ ) keeps every tuple in S, if no matching tuple is found in R, then the attributes of R in the join result are filled with null values.
- The full outer join operation ( $R \bowtie^f S$ ) keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

# **Additional Relational Operations**

## **OUTER UNION Operation**

- The outer union operation was developed to take the union of tuples from two relations if the relations are not union compatible.

# Examples of Queries in Relational Algebra

- Query 1: Retrieve the name and address of all employees who work for the ‘Research’ department.

$$R\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$
$$R\_EMPS \leftarrow (R\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Lname, Fname, Address}(R\_EMPS)$$

# Examples of Queries in Relational Algebra

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

$$S\_PROJS \leftarrow \sigma_{Plocation='Stafford'}(PROJECT)$$
$$C\_DEPT \leftarrow (S\_PROJS \bowtie_{Dnum=Dnumber} DEPARTMENT)$$
$$P\_DEPT\_MGR \leftarrow (C\_DEPT \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Pnumber, Dnum, Lname, Address, Bdate}(P\_DEPT\_MGR)$$

# Examples of Queries in Relational Algebra

- Query 3: Find the names of employees who work on all projects controlled by the department number 5.

```
D_5_PROJS(Pno) ← πPnumber(σDnum=5(PROJECT))
EMP_PROJ(Ssn, Pno) ← πEssn, Pno(WORKS_ON)
RESULT_EMP_SSNS ← EMP_PROJ ÷ D_5_PROJS
RESULT ← πLname, Fname(RESULT_EMP_SSNS * EMPLOYEE)
```

# Examples of Queries in Relational Algebra

- Query 4: Make a list of 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.

$$\text{SMITHS}(\text{Essn}) \leftarrow \pi_{\text{Ssn}} (\sigma_{\text{Lname}=\text{'Smith'}} (\text{EMPLOYEE}))$$
$$\text{SMITH\_W\_PROJ} \leftarrow \pi_{\text{Pno}} (\text{WORKS\_ON} * \text{SMITHS})$$
$$\text{MGRS} \leftarrow \pi_{\text{Lname, Dnumber}} (\text{EMPLOYEE} \bowtie_{\text{Ssn}=\text{Mgr\_ssn}} \text{DEPARTMENT})$$
$$\text{SMITH\_M\_DEPTS}(\text{Dnum}) \leftarrow \pi_{\text{Dnumber}} (\sigma_{\text{Lname}=\text{'Smith'}} (\text{MGRS}))$$
$$\text{SMITH\_M\_PROJS}(\text{Pno}) \leftarrow \pi_{\text{Pnumber}} (\text{SMITH\_M\_DEPTS} * \text{PROJECT})$$
$$\text{RESULT} \leftarrow (\text{SMITH\_W\_PROJS} \cup \text{SMITH\_M\_PROJS})$$

# Examples of Queries in Relational Algebra

- Query 5: list the names of all employees with two or more dependents.

$$\begin{aligned} T1(Ssn, No\_of\_deps) &\leftarrow \text{Essn } \delta \text{ COUNT}_{\text{Dependent\_name}}(\text{DEPENDENT}) \\ T2 &\leftarrow \sigma_{No\_of\_deps > 1}(T1) \\ \text{RESULT} &\leftarrow \pi_{Lname, Fname}(T2 * \text{EMPLOYEE}) \end{aligned}$$

# Examples of Queries in Relational Algebra

- Query 6: Retrieve the names of employees who have no dependents.

$\text{ALL\_EMPS} \leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$

$\text{EMPS\_WITH\_DEPS}(\text{Ssn}) \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$

$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMP} - \text{EMPS\_WITH\_DEPS})$

$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$

# Examples of Queries in Relational Algebra

- Query 7: list the names of managers who have at least one dependent.

$$\text{MGRS}(\text{Ssn}) \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$$
$$\text{EMPS\_WITH\_DEPS}(\text{Ssn}) \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$$
$$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$$

# The Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range:
  - Over rows of stored database relations (in **tuple calculus**), or
  - Over columns of the stored relations (in **domain calculus**).
- In a calculus expression, there is no order of operations to specify how to retrieve the query result.
- A calculus expression specifies only what information the result should contain.

# Introduction to Tuple Relational Calculus

- Is based on specifying a number of tuple variables, each tuple variable usually ranges over a particular database relation.
- A simple tuple relational calculus query is of the form:  
 $\{t \mid \text{COND}(t)\}$
- Example: find all employees whose salary is above \$50,000.  
 $\{t \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{Salary} > 50000\}$
- Example: find the first and last names of all employees whose salary is above \$50,000.  
 $\{t.\text{Lname}, t.\text{Fname} \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{Salary} > 50000\}$

# The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas:
  - Existential quantifier ( $\exists$ ).
  - Universal quantifier ( $\forall$ ).
- Informally, a tuple variable  $t$  is bound if it is quantified, meaning that it appears in an  $(\exists t)$  or  $(\forall t)$  clause; otherwise, it is free.

# Example Queries Using Existential Quantifier

- Query 1: retrieve the name and address of all employees who work for the ‘Research’ department.

Q1: {t.Lname, t.Fname, t.Address | EMPLOYEE(t)

AND ( $\exists d$ ) (DEPARTMENT(d) AND d.Dname=‘Research’  
AND d.Dnumber=t.Dno)}

- Query 2: for every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, birth date, and address.

Q2: {p.Pnumber, p.Dnum, m.Lname, m.Bdate, m.Address|  
PROJECT(p) AND EMPLOYEE(m) AND  
p.Plocation=‘Stafford’ AND (( $\exists d$ )(DEPARTMENT(d) AND  
p.Dnum=d.Dnumber AND d.Mgr\_ssn=m.Ssn))}

# Introduction to Domain Relational Calculus

- Example: Retrieve the name and address of all employees who work for the ‘Research’ department.

$$\{qsv \mid (\exists z) (\exists l) (\exists m) (\text{EMPLOYEE}(qrstuvwxyz) \text{ AND } \text{DEPARTMENT}(lmno) \text{ AND } l = \text{'Research'} \text{ AND } m = z)\}$$

- Example: for every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, birth date, and address.

$$\{iksv \mid (\exists j) (\exists m) (\exists n) (\exists t) (\text{PROJECT}(hijk) \text{ AND } \text{EMPLOYEE}(qrstuvwxyz) \text{ AND } \text{DEPARTMENT}(lmno) \text{ AND } k = m \text{ AND } n = t \text{ AND } j = \text{'Stafford'})\}$$

# **CS 380**

# **Introduction to Database Systems**

**(Chapter 7: SQL-99: Schema Definition, Constraints, Queries, and Views)**

# Outline

- Introduction
- Data Definition Language
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- Data Manipulation Language
  - SELECT Statement
  - INSERT Statement
  - DELETE Statement
  - UPDATE Statement
- Assertions
- Views

# Introduction

- SQL: Structured Query Language.
- Standard for commercial relational DBMSs, a joint effort by ANSI & ISO.
- Benefits of a standardized relational language:
  - Reduced training costs.
  - Productivity.
  - Application portability.
  - Application longevity.
  - Reduced dependence on a single vendor.
  - Cross-system communication.

# “Declarative” Versus “Procedural”

- Most DBMSs provide the user with a high-level declarative language interface.
- The user has just to specify “what” results he/she would like, leaving the “how” to the DBMS.

# Schema and Catalog Concepts in SQL

- The concept of an SQL Schema was incorporated to group together tables & other constructs that belong to the same database application.
- Schema: the structure that contains descriptions of objects created by the user (tables, constraints, views, domains, character sets, triggers, rules).
- Catalog: a set of schemas that constitute the description of a database.

# Types of SQL Statements

- Data Definition Language (DDL):
  - Commands that define a database.
  - E.g. CREATE, ALTER, DROP, ...etc.
- Data manipulation language (DML):
  - Commands that maintain and query a database.
  - E.g. SELECT, INSERT, UPDATE, DELETE.
- Data Control Language (DCL):
  - Commands that control a database, including administering privileges and committing data.
  - E.g. CONNECT, GRANT, REVOKE, ...etc.

# CREATE Statements

- Major CREATE statements:
  - CREATE SCHEMA:
    - Defines a portion of the database owned by a particular user.
    - CREATE SCHEMA *COMPANY* AUTHORIZATION *JSMITH*;
  - CREATE TABLE:
    - Defines a table and its columns.
  - CREATE VIEW:
    - Defines a logical table from one or more views.
- Other CREATE statements: DOMAIN, CHARACTER SET, COLLATION, TRANSLATION, ASSERTION.

# CREATE TABLE Construct

- CREATE TABLE [*schema.*] *name*  
(*column-1* *TYPE* [DEFAULT *value*] [*constraints*],  
*column-2* *TYPE* [DEFAULT *value*] [*constraints*],  
*column-n* *TYPE* [DEFAULT *value*] [*constraints*],  
[*table-constraints*]);
- Where:
  - [*schema.*]: schema name followed by a dot.
  - *name*: table name.
  - *column-1* to *column-n*: column names.
  - *TYPE*: data type.
  - [DEFAULT *value*]: optional default value.
  - [*constraints*]: optional constraints, can be specified at column level or at table level.

# Some SQL Data Types (From Oracle)

- **CHAR(*n*)**: a fixed-length string *n* characters long (default 1, max 255 character).
- **VARCHAR(*n*)**: a variable string up to *n* character long (max 2000).
- **INT**: an integer number.
- **DECIMAL(*n,d*)**: a number *n* digits long including *d* decimal places.
- **DATE**: a valid date YYYY-MM-DD.

# CREATE DOMAIN Construct

- A domain can be declared, and the domain name is used with the attribute specification.
- This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability.
- Examples:
  - CREATE DOMAIN SSN\_TYPE AS CHAR(9);
  - CREATE DOMAIN D\_NUM AS INT CHECK (D\_NUM>0 AND D\_NUM<21);

# Constraints

- There are essentially five types of constraints:
  - NOT NULL constraints.
  - UNIQUE value constraints.
  - PRIMARY KEY constraints.
  - FOREIGN KEY constraints.
  - CHECK constraints.
- The syntax of these constraints varies according to whether they are specified at column level or at table level.
- Any constraints which apply to multiple columns must be defined as table level constraints.

# Constraints

- The syntax for each of the column level constraints is:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - REFERENCES *table* (*key*)
  - CHECK (*condition*)

# Constraints

- The syntax for each of the table level constraints is:
  - UNIQUE (*column-list*)
  - [CONSTRAINT *name*] PRIMARY KEY (*column-list*)
  - [CONSTRAINT *name*] FOREIGN KEY (*column-list*)  
  REFERENCES *table* (*column-list*)
  - CHECK (*condition*)
- An additional ON DELETE clause may be applied to a foreign key constraint:
  - ON DELETE CASCADE causes all referencing child records to be deleted whenever a parent record is deleted.
  - ON DELETE SET NULL causes all referencing child records to be set to NULL whenever a parent record is deleted.

# Steps in Table Creation

- Identify data types for attributes.
- Determine default values.
- Identify columns that can and cannot be null.
- Identify columns that must be unique (candidate keys).
- Identify primary key - foreign key mates.
- Identify constraints on columns (domain specifications).
- Create the table and associated indexes. To improve access speed

# CREATE TABLE Example

- 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	CHECK (LOWER(SEX) IN ('m', 'f')),
Salary	DECIMAL(10,2),	
Super_ssn	CHAR(9),	
Dno	INT	NOT NULL,

CONSTRAINT EMPPK PRIMARY KEY (Ssn),  
CONSTRAINT EMPSUPERFK FOREIGN KEY (Super\_ssn) REFERENCES  
EMPLOYEE (Ssn) ON DELETE SET NULL,  
CONSTRAINT EMPDEPTFK FOREIGN KEY (Dno) REFERENCES  
DEPARTMENT (Dnumber) ON DELETE SET NULL);

# **ALTER TABLE Construct**

- Tables can be changed in a number of ways using the SQL statement ALTER TABLE.
- Possible actions include:
  - Adding or dropping a column.
  - Adding or dropping a constraint.
  - Changing a column definition.

# ALTER TABLE Construct

- To add a column to the EMPLOYEE table:

```
ALTER TABLE EMPLOYEE ADD  
Job VARCHAR(12);
```

- To add a constraint to the DEPARTMENT table:

```
ALTER TABLE DEPARTMENT ADD CONSTRAINT  
DEPTMGRFK FOREIGN KEY (Mgr_ssn)  
REFERENCES EMPLOYEE (Ssn) ON DELETE SET NULL;
```

# **ALTER TABLE Construct**

- To drop the attribute Address from the EMPLOYEE table:

```
ALTER TABLE EMPLOYEE DROP COLUMN  
Address CASCADE CONSTRAINTS;
```

- To drop a constraint named EMPSUPERFK from EMPLOYEE table:

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

# ALTER TABLE Construct

- To increase the width of Lname column:

```
ALTER TABLE EMPLOYEE MODIFY  
Lname VARCHAR2(20);
```

- To add a NOT NULL constraint to the Address column:

```
ALTER TABLE EMPLOYEE MODIFY  
Address NOT NULL;
```

- To drop an existing default clause:

```
ALTER TABLE DEPARTMENT ALTER  
Mgr_ssn DROP DEFAULT;
```

- To define a new default clause:

```
ALTER TABLE DEPARTMENT ALTER  
Mgr_ssn SET DEFAULT '333445555';
```

# DROP SCHEMA Construct

- The DROP SCHEMA command can be used to drop a schema.

- Syntax:

DROP SCHEMA *name* [CASCADE or RESTRICT]

- Where:

- *name*: is the name of the schema.
- CASCADE: removes the schema and all its tables, domains, and other elements.
- RESTRICT: removes the schema only if it has no elements.

# DROP TABLE Construct

- The DROP TABLE command deletes all information about the dropped relation from the database.

- Syntax:

DROP TABLE *name* [CASCADE CONSTRAINTS]

- Where:

- *name*: is the name of the table to be dropped.
- CASCADE CONSTRAINTS: drops all referential integrity constraints which refer to keys in the dropped table.

# Basic Queries in SQL

- SQL has one basic statement for retrieving data from a database, the SELECT statement.
- This is not the same as the SELECT operation of the relational algebra.

# The SQL SELECT Statement

- It consists of the following basic clauses:

Mandatory

<b>SELECT</b>	specifies the columns which are to be retrieved
<b>FROM</b>	specifies the tables from which data is to be retrieved
<b>WHERE</b>	defines any selection criteria required
<b>GROUP BY</b>	specifies how output is to be grouped & summarized
<b>HAVING</b>	determines which group summaries to include in the output
<b>ORDER BY</b>	defines how the output will be stored

# The SELECT-FROM-WHERE Structure

- The basic form of the SELECT statement is:

```
SELECT      [DISTINCT]  <attribute-list>
FROM        <table-list>
WHERE       <condition>
```

- Where:
  - *item-list*: a comma-separated list of items to be retrieved.  
(database attributes, functions, fixed text, ...etc.)
  - *table-list*: a comma-separated list of tables from which the attributes are to be retrieved.
  - *condition*: a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.
  - [DISTINCT]: ensures only unique tuples will be returned.
- The result of an SQL query is a relation (can be a multiset)

# The SELECT-FROM-WHERE Structure

- Query: retrieve the birthdate and address of the employee(s) whose name is ‘John B. Smith’.

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

- Is it similar to a tuple relational expression? Any differences?

SELECT Statement	Relational Algebra
SELECT clause	project operation
FROM clause	cartesian product operation
WHERE clause	select condition

# The SELECT-FROM-WHERE Structure

- Query: retrieve the name and address of all employees who work for the ‘Research’ department.

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

- Query: for every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birthdate.

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

# Ambiguous Attribute Names, Aliasing, and Tuple Variables

- In SQL if the same attribute name is used in more than one relation, the attribute names must be qualified.
- Assume that DNO attribute of the EMPLOYEE relation was called DNUMBER.
- Query: retrieve the name and address of all employees who work for the ‘Research’ department.

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

# Ambiguous Attribute Names, Aliasing, and Tuple Variables

- Ambiguity also arise in the case of queries that refer to the same relation twice, as in the following example.
- Query: for each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT      E.Fname, E.Lname, S.Fname, S.Lname  
FROM        EMPLOYEE E, EMPLOYEE S  
WHERE       E.Super_ssn=S.Ssn;
```

# Unspecified WHERE Clause

- Query: select all EMPLOYEE SSNs.

```
SELECT      Ssn  
FROM       EMPLOYEE;
```

- Query: select all combinations of EMPLOYEE SSN and DEPARTMENT DNAME.

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

# The Use of Asterisk

- Query: retrieve all the attribute values of any EMPLOYEE who work in DEPARTMENT 5.

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

- Query: retrieve all the attributes of an EMPLOYEE and all the attributes of the DEPARTMENT in which he or she works for every employee of the ‘Research’ department.

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

# Tables as Sets in SQL

- Query: retrieve the salary of every employee.

```
SELECT      Salary  
FROM        EMPLOYEE;
```

- Query: retrieve all distinct salary values.

```
SELECT      DISTINCT    Salary  
FROM        EMPLOYEE;          ] A set
```

# The UNION, INTERSECT, & MINUS SET OPERATORS

- The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result.
- Query: 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.

```
SELECT      DISTINCT      Pnumber      ←  
FROM        PROJECT, DEPARTMENT, EMPLOYEE  
WHERE       Dnum=Dnumber AND Mgr_ssn=Ssn AND  
           Lname='Smith'
```

Union  
compatible

```
UNION  
SELECT      DISTINCT      Pnumber      ←  
FROM        PROJECT, WORKS_ON, EMPLOYEE  
WHERE       Pnumber=Pno AND Essn=Ssn AND Lname='Smith';
```

# Conditional Operators

- Main conditional operators include: `=, <>, >, <, >=, <=`.
- Other conditional operators:
  - **IN**: tests if an item value appears in a specified list of values.
  - **BETWEEN**: tests if an item value lies (inclusively) between two specified values.
  - **IS NULL**: tests if an item has a Null value. Null is neither zero (in a numeric item), nor spaces (in a character item).
  - **LIKE**: tests if an item value matches a string containing wildcard characters. The wildcard characters are:
    - **%**: meaning zero or more occurrences of any character.
    - **\_**: meaning a single occurrence of any character.
- The keyword **NOT** can be used in conjunction with all the above operators to revise the test.

# Conditional Operators

- Query: retrieve the names of all employees whose salary is not (20000, 30000, or 40000).

```
SELECT      Fname, Lname  
FROM        EMPLOYEE  
WHERE       Salary NOT IN (20000, 30000, 40000);
```

- Query: retrieve all employees in department 5 whose salary is between \$30,000 and \$40,000.

```
SELECT      *  
FROM        EMPLOYEE  
WHERE       (Salary BETWEEN 30000 AND 40000) AND  
           Dno = 5;
```

# Conditional Operators

- Query: retrieve the names of all employees who do not have supervisors.

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

- Query: retrieve all employees whose address is in Houston, Texas.

```
SELECT      Fname, Lname  
FROM        EMPLOYEE  
WHERE       Address LIKE '%Houston, TX%';
```

# Arithmetic Operations

- The standard arithmetic operations (+, -, \*, /) can be applied to numeric values or attributes with numeric domains.
- Query: show the resulting salaries if every employee working on the ‘ProductX’ project is given 10 percent raise.

```
SELECT      Fname, Lname, 1.1*Salary INCREASED_SAL
FROM        EMPLOYEE, WORKS_ON, PROJECT
WHERE       Ssn=Essn AND Pno=Pnumber AND
           Pname='ProductX';
```

# Ordering of Query Results

- SQL allows the user to order the tuples in the result of a query by the values of one or more attributes, using the ORDER BY clause.
- Query: retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

```
SELECT      Dname, Lname, Fname, Pname  
FROM        DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT  
WHERE       Dnumber=Dno AND Ssn=Essn AND  
           Pno=Pnumber  
ORDER BY    Dname, Lname, Fname;
```

- The default order is in ascending order.
- **ORDER BY** Dname **DESC**, Lname **ASC**, Fname **ASC**

# Nested Queries

- Some queries require that existing values in the database be fetched and then used in a comparison condition.
- Such queries can be conveniently formulated by using nested queries.
- Nested queries: complete select-from-where blocks usually within the WHERE clause of another query. That other query is called the outer query.

# Nested Queries

- Query: 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.

```
SELECT      DISTINCT      Pnumber
FROM        PROJECT
WHERE       Pnumber IN (SELECT      Pnumber
                        FROM        PROJECT, EMPLOYEE,
                        DEPARTMENT
                        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

- SQL allows the use of tuples of values in comparisons by placing them within parentheses.
- Example:

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

# Nested Queries

- In addition to the IN operator, a number of other comparison operators can be used to compare a single value (attribute) to a set or multiset.
- The keywords ANY or ALL can be combined with the following operators:  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $\neq$ . ( $=$  ANY is equivalent to IN)
- Query: list the names of employees whose salary is greater than the salary of all the employees in department 5.

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

# The EXISTS Function in SQL

- This function checks whether the result of a correlated nested query is empty (contains no tuples) or not.
- Query: retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
SELECT      E.Fname, E.Lname
FROM        EMPLOYEE E
WHERE       EXISTS (SELECT      *
                    FROM        DEPENDENT
                    WHERE       E.Ssn=Essn AND
                                E.Sex=Sex AND
                                E.Fname=Dependent_name);
```

# The EXISTS Function in SQL

- Query: retrieve the names of employees who have no dependents.

```
SELECT      Fname, Lname
FROM        EMPLOYEE
WHERE       NOT EXISTS (SELECT      *
                        FROM        DEPENDENT
                        WHERE       Ssn=Essn);
```

# Aggregate Function in SQL

- A number of built-in functions exist: COUNT, SUM, MAX, MIN, and AVG.
- These functions can be used in the SELECT clause or in the HAVING clause.
- The functions SUM, MAX, MIN, and AVG are applied to a set or multiset of numeric values.
- The functions MAX and MIN can also be used with attributes that have nonnumeric domains if the domain values have a total ordering among one another.
- Null values are discarded when aggregate functions are applied to a particular attribute.

# Aggregate Function in SQL

- Query: find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
SELECT      SUM (Salary), MAX (Salary), MIN (Salary),
            AVG (Salary)
FROM        EMPLOYEE;
```

- Query: 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.

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

# Aggregate Function in SQL

- Query: retrieve the total number of employees in the company.

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

- Query: retrieve the total number of employees in the ‘Research’ department.

```
SELECT      COUNT(*)  
FROM        EMPLOYEE, DEPARTMENT  
WHERE       Dno=Dnumber AND Dname='Research';
```

# Aggregate Function in SQL

- Query: Count the number of distinct salary values in the database.

```
SELECT      COUNT(DISTINCT Salary)  
FROM        EMPLOYEE;
```

- Query: retrieve the names of all employees who have two or more dependents.

```
SELECT      Lname, Fname  
FROM        EMPLOYEE  
WHERE       (SELECT      COUNT(*)  
              FROM        DEPENDENT  
              WHERE       Ssn=Essn) >= 2;
```

# The GROUP BY Clause

- This clause enables aggregates to be accumulated and grouped on output.
- Query: for each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT      DNO, COUNT(*), AVG(Salary)  
FROM        EMPLOYEE  
GROUP BY Dno;
```

- If Nulls exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.

# The HAVING Clause

- This clause specifies which groups are to be selected for output.
- Groups are only output if they match the condition(s) specified by the HAVING clause.
- Query: for each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT      Pnumber, Pname, COUNT(*)  
FROM        PROJECT, WORKS_ON  
WHERE       Pnumber=Pno  
GROUP BY    Pnumber, Pname  
HAVING      COUNT(*) >2;
```

# **INSERT, DELETE, and UPDATE Statements**

- In SQL, three commands can be used to modify the database:
  - INSERT.
  - DELETE.
  - UPDATE.

# The INSERT Statement

- Using the SQL INSERT statement, rows may be inserted into tables in two ways:
  - As lists of values, one row at a time.
  - From another table using a query.

# INSERT Using a Value List

- Syntax:

```
INSERT INTO table [(attribute-list)]  
VALUES  
(value-list)
```

- Where:

- *table*: the name of the table to update.
- *attribute-list*: a list of attributes in the specified table.
- *value-list*: the list of values to be inserted for the row.

# INSERT Using a Value List

- Example:

```
INSERT INTO PROJECT  
VALUES ('ProductZ', 1, NULL, 2)
```

- Example:

```
INSERT INTO PROJECT (PNAME, PNUMBER, DNUM)  
VALUES ('ProductZ', 1, 2)
```

- Example:

```
INSERT INTO PROJECT (PNAME, PNUMBER, DNUM)  
VALUES ('&PN', &P_NO, &D_NO)
```



Each time it is run, the user will be prompted for a new set of values

# INSERT Using a Query

- Syntax:

INSERT INTO *table* [(*attribute-list*)]

SELECT statement

- There are no limitation concerning the SELECT statement.

# INSERT Using a Query

- Example:

```
CREATE TABLE DEPTS_INFO  
(DEPT_NAME      VARCHAR(15),  
 NO_OF_EMPS     INTEGER,  
 TOTAL_SAL      INTEGER);
```

```
INSERT      INTO DEPTS_INFO  
SELECT      DNAME, COUNT(*), SUM (Salary)  
FROM        DEPARTMENT, EMPLOYEE  
WHERE       Dnumber=Dno  
GROUP BY   Dname;
```

# The DELETE Statement

- Syntax:

```
DELETE FROM table
[WHERE conditions]
```

- Example:

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

- Example:

```
DELETE FROM EMPLOYEE
WHERE Dno IN (SELECT Dnumber
               FROM DEPARTMENT
               WHERE Dname='Research');
```

- Example:

```
DELETE FROM EMPLOYEE;
```

# The UPDATE Statement

- The UPDATE statement can be used:
  - With individual values.
  - In conjunction with a SELECT statement.

# UPDATE Using Fixed Values

- Syntax:

```
UPDATE    table
SET        attribute-1 = value-1,
          attribute-2 = value-2,
          :
          attribute-n = value-n
```

[WHERE *conditions*]

- Example:

```
UPDATE    PROJECT
SET        Plocation = 'Bellaire'
WHERE      Pnumber=10;
```

- Example:

```
UPDATE    EMPLOYEE
SET        Salary = Salary * 1.25;
```

# UPDATE Using a Query

- Syntax:

UPDATE      *table*  
SET            (*attribute-list*)  
=                 
(SELECT statement)  
[WHERE *conditions*]

- Example:

Update      EMPLOYEE  
SET            (Salary)  
=                 
( SELECT      Salary  
        FROM      EMPLOYEE  
        WHERE     Ssn='987654321')  
WHERE        Fname = 'John' AND Lname='Smith';

# Concept of a View (Virtual Table) in SQL

- A view is a customized representation of one or more underlying tables or other views (or a mixture of both).
- A view does not necessarily exist in physical form.
- Allows full query operations.
- Allows limited update operations (since the table may not physically be stored).
- Convenience for expressing certain operations.

# Concept of a View (Virtual Table) in SQL

- Advantages of views:
  - Simplify query commands.
  - Provide data security.
  - Enhance programming productivity.

# Specification of Views in SQL

- Example:

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

- Example:

```
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

- Query: retrieve the last name and first name of all employees who work on ‘ProjectX’.

```
SELECT      Fname, Lname  
FROM        WORKS_ON1  
WHERE       Pname='ProjectX';
```

- A view is supposed to be always up to date.
- When no longer needed, a view can be dropped:  
`DROP VIEW name`

# Effective View Implementation

- Two main approaches have been suggested:
  - Query modification: involves modifying the view query into a query on the underlying base tables. I.e, the previous query would be automatically modified to:

```
SELECT      Fname, Lname
FROM        EMPLOYEE, PROJECT, WORKS_ON
WHERE      Ssn=Essn AND Pno=Pnumber AND
          Pname='ProjectX';
```
  - View Materialization: involves physically creating a temporary view table (should be kept up to date).

# View Update

- A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint that do not have default values specified.
- Views defined on multiple tables using joins are generally not updatable.
- Views defined using grouping and aggregate functions are not updatable.

# View Update

- In SQL, the clause WITH CHECK OPTION must be added at the end of the new definition if the view is to be updated.

**King Saud University**  
**College of Computer & Information Sciences**  
**Computer Science Department**



**CS 380**  
**Introduction to Database Systems**

**Functional Dependencies and Normalization for Relational Databases**

# Outline

- Introduction
- Informal Design Guidelines For Relation Schemas
- Functional Dependencies
- Inference Rules for Functional Dependencies
- Normalization of Relations
- Steps in Data Normalization
  - First Normal Form
  - Second Normal Form
  - Third Normal Form
- Boyce-Codd Normal Form (BCNF)
- Advantages of Normalization
- Disadvantages of Normalization
- Conclusion

# Introduction

- Relational database design: is the grouping of attributes to form “good” relation schemas.
- There are two levels of relation schemas:
  - The logical “user view” level.
  - The storage “base relation” level
- Design is concerned mainly with base relations.
- *What are the criteria for “good” base relations?*

# **Informal Design Guidelines For Relation Schemas**

## **1. Semantics of the Relation Attributes**

- Whenever attributes are grouped to form a relation schema, it is assumed that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them.
- In general the easier it is to explain the semantics of the relation, the better the relation schema design will be.

Guideline 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

# **Informal Design Guidelines For Relation Schemas**

## **2. Redundant Information in Tuples and Update Anomalies**

- One goal of schema design is to minimize the storage space used by the base relations.
- Grouping attributes into relation schemas has a significant effect on storage space.
- Mixing attributes of multiple entities may cause problems  
→ Information is stored redundantly wasting storage.

# Informal Design Guidelines For Relation Schemas

## 2. Redundant Information in Tuples and Update Anomalies

EMPLOYEE					DEPARTMENT		
ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DNUMBER	DMGRSSN
Smith,John B.	123456789	1965-01-09	731 Fondren,Houston,TX	5	Research	5	333445555
Wong,Franklin T.	333445555	1955-12-08	638 Voss,Houston,TX	5	Administration	4	987654321
Zelaya,Alicia J.	999887777	1968-07-19	3321 Castle, Spring,TX	4	Headquarters	1	888665555
Wallace,Jennifer S.	987654321	1941-06-20	291 Berry,Bellaire,TX	4			
Narayan,Remesh K.	666884444	1962-09-15	975 Fire Oak,Humble,TX	5			
English, Joyce A.	453453453	1972-07-31	5631 Rice,Houston,TX	5			
Jabbar,Ahmad V.	987987987	1969-03-29	980 Dallas,Houston,TX	4			
Borg,James E.	888665555	1937-11-10	450 Stone,Houston,TX	1			

EMP_DEPT						
ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
Smith,John B.	123456789	1965-01-09	731 Fondren,Houston,TX	5	Research	333445555
Wong,Franklin T.	333445555	1955-12-08	638 Voss,Houston,TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring,TX	4	Administration	987654321
Wallace,Jennifer S.	987654321	1941-06-20	291 Berry,Bellaire,TX	4	Administration	987654321
Narayan,Ramesh K.	666884444	1962-09-15	975 FireOak,Humble,TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice,Houston,TX	5	Research	333445555
Jabbar,Ahmad V.	987987987	1969-03-29	980 Dallas,Houston,TX	4	Administration	987654321
Borg,James E.	888665555	1937-11-10	450 Stone,Houston,TX	1	Headquarters	888665555

redundancy

# **Informal Design Guidelines For Relation Schemas**

## **2. Redundant Information in Tuples and Update Anomalies**

- It is important to distinguish between redundancy and duplicated data:
  - Duplicated data exists when an attribute has two or more identical values in a table.
  - Redundancy exists if data can be deleted without any information being lost.
- Redundancy may be viewed as unnecessary duplication.

# **Informal Design Guidelines For Relation Schemas**

## **2. Redundant Information in Tuples and Update Anomalies**

- Another serious problem is the problem of update anomalies.
- Update anomalies:
  - Insertion anomalies.
  - Deletion anomalies.
  - Modification anomalies.

# Informal Design Guidelines For Relation Schemas

## 2. Redundant Information in Tuples and Update Anomalies

EMP\_PROJ

SSN	PNumber	Hours	EName	PName	PLocation

- **Insertion Anomalies:**

- Occurs when it is impossible to store a fact until another fact is known.
- Example:
  - Cannot insert a project unless an employee is assigned to.
  - Cannot insert an employee unless he/she is assigned to a project.

# Informal Design Guidelines For Relation Schemas

## 2. Redundant Information in Tuples and Update Anomalies

EMP\_PROJ

SSN	PNumber	Hours	EName	PName	PLocation
-----	---------	-------	-------	-------	-----------

- **Delete anomalies:**
  - Occurs when the deletion of a fact causes other facts to be deleted.
  - Example:
    - When a project is deleted, it will result in deleting all the employees who work on that project.
    - If an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

# Informal Design Guidelines For Relation Schemas

## 2. Redundant Information in Tuples and Update Anomalies

EMP\_PROJ

SSN	PNumber	Hours	EName	PName	PLocation
-----	---------	-------	-------	-------	-----------

- **Modification Anomalies:**

- Occurs when a change in a fact causes multiple modifications to be necessary.
- Example: changing the name of project number P1 (for example) may cause this update to be made for all employees working on that project.

# **Informal Design Guidelines For Relation Schemas**

## **2. Redundant Information in Tuples and Update Anomalies**

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

# Informal Design Guidelines For Relation Schemas

## 3. Null Values in Tuples

- In some schema designs many attributes may be grouped together into a “flat” relation.
- If many of the attributes do not apply to all tuples in the relation, many null values will appear in those tuples.

Guideline 3: As far as possible, avoid placing attributes in a base relation whose values may frequently be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

# Informal Design Guidelines For Relation Schemas

## 4. Generation of Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations.

EMP_PROJ1				
SSN	PNUMBER	HOURS	PNAME	PLOCATION
123456789	1	32.5	Product X	Bellaire
123456789	2	7.5	Product Y	Sugarland
666884444	3	40.0	Product Z	Houston
453453453	1	20.0	Product X	Bellaire
453453453	2	20.0	Product Y	Sugarland
333445555	2	10.0	Product Y	Sugarland
333445555	3	10.0	Product Z	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston

EMP_LOCS	
ENAME	PLOCATION
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford

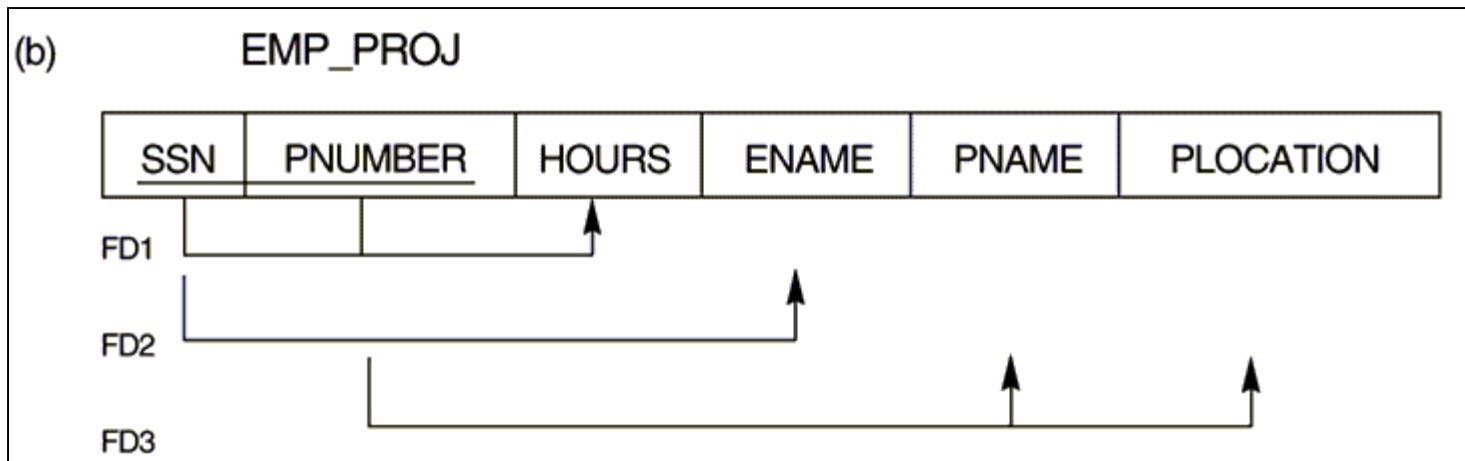
# Functional Dependencies

- Functional dependencies (FDs) are used to specify formal measures of the “goodness” of relational designs.
- FDs and keys are used to define normal forms for relations.
- FDs are constraints that are derived from the meaning and interrelationships of the data attributes.
- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

# Functional Dependencies

- $X \rightarrow Y$  holds if whenever two tuples have the same value for  $X$ , they must have the same value for  $Y$
- $X \rightarrow Y$  in  $R$  specifies a constraint on all relation instances  $r(R)$ .

# Functional Dependencies



- $\{SSN, PNUMBER\} \rightarrow HOURS$
- $SSN \rightarrow ENAME$
- $PNUMBER \rightarrow \{PNAME, PLOCATION\}$

# Functional Dependencies

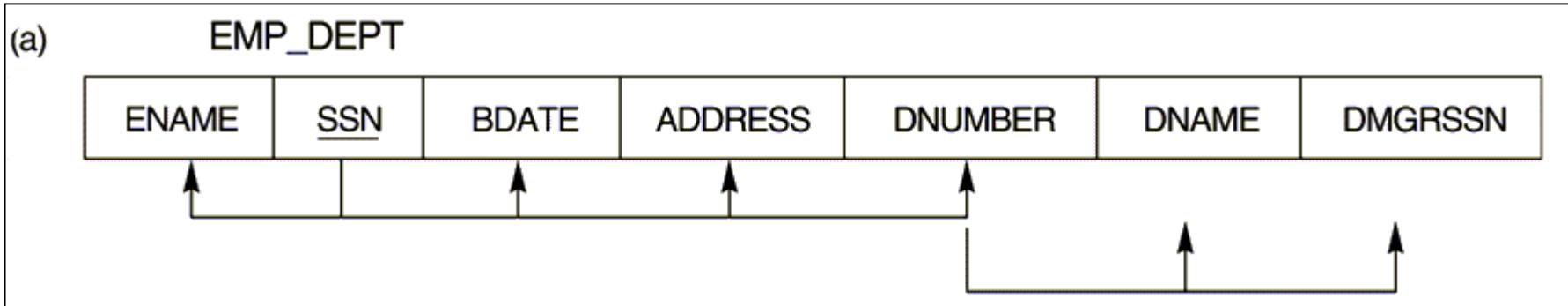
TEACH		
TEACHER	COURSE	TEXT
Smith	Data Structures	Bartram
Smith	Data Management	Al-Nour
Hall	Compilers	Hoffman
Brown	Data Structures	Augenthaler

- $\text{TEXT} \rightarrow \text{COURSE}$  ✓
- $\text{TEACHER} \rightarrow \text{COURSE}$  ✗

# Inference Rules for Functional Dependencies

- Given a set of FDs  $F$ , we can infer additional FDs that hold whenever the FDs in  $F$  hold using the following rules:
  - IR1 (reflexive rule): If  $X \supseteq Y$ , then  $X \rightarrow Y$ .
  - IR2 (augmentation rule):  $\{X \rightarrow Y\}$  then  $XZ \rightarrow YZ$ .
  - IR3 (transitive rule):  $\{X \rightarrow Y, Y \rightarrow Z\}$  then  $X \rightarrow Z$ .
  - IR4 (decomposition, or projective, rule):  $\{X \rightarrow YZ\}$  then  $X \rightarrow Y$ .
  - IR5 (union, or additive, rule):  $\{X \rightarrow Y, X \rightarrow Z\}$  then  $X \rightarrow YZ$ .
  - IR6 (pseudotransitive rule):  $\{X \rightarrow Y, WY \rightarrow Z\}$  then  $WX \rightarrow Z$ .
  - Form a sound and complete set of inference rules.
- The set of all dependencies that include  $F$  as well as all dependencies that can be inferred from  $F$  is called the closure of  $F$ ; denoted by  $F^+$ .

# Inference Rules for Functional Dependencies



- $\text{SSN} \rightarrow \{\text{ENAME}, \text{BDATE}, \text{ADDRESS}, \text{DNUMBER}\}$
- $\text{DNUMBER} \rightarrow \{\text{DNAME}, \text{DMGRSSN}\}$
- Some additional functional dependencies that we can infer are:
- $\text{SSN} \rightarrow \{\text{DNAME}, \text{DMGRSSN}\}$
- $\text{DNUMBER} \rightarrow \text{DNAME}$

# Normalization of Relations

- **Normalization** is the process of decomposing relations with anomalies to produce smaller, well structured relations.
- Normalization can be accomplished and understood in stages, each of which corresponds to a normal form.
- **Normal form** is a state of a relation that results from applying simple rules regarding functional dependencies (or relationships between attributes) to that relation.

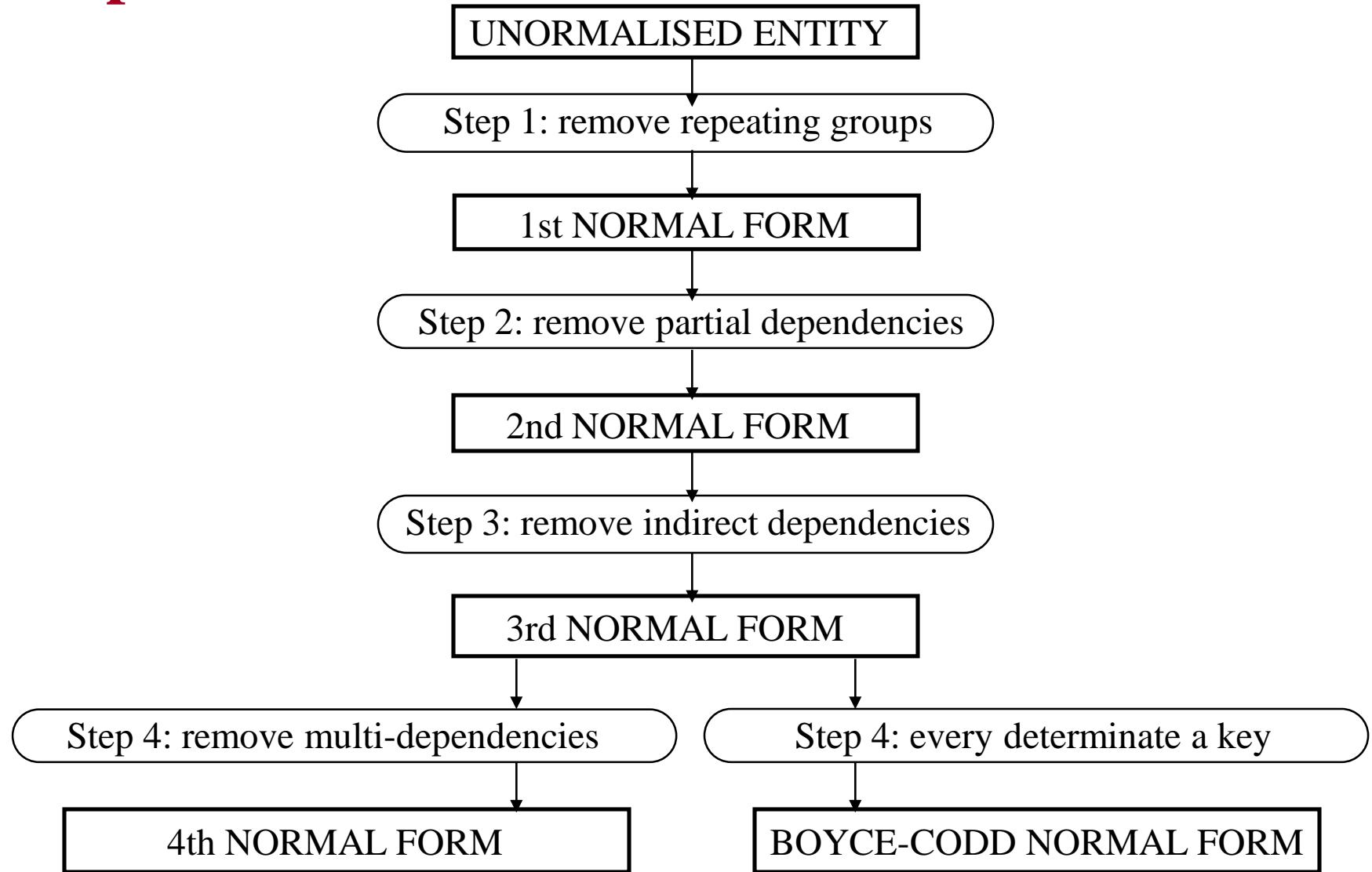
# Normalization of Relations

- Normal forms:
  - First Normal Form (1NF).
  - Second Normal Form (2NF).
  - Third Normal Form (3NF).
  - Boyce-Codd Normal Form (BCNF). — A stronger definition of 3NF
  - Fourth Normal Form (4NF).
  - Fifth Normal Form (5NF).
- Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or 4NF.
- The database designers need not normalize to the highest possible normal form.

# Normalization of Relations

- Normal forms, when considered in isolation from other factors, do not guarantee a good database design.
- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should process. These include two properties:
  - The **lossless join or nonadditive join property**, which guarantees that the spurious tuple generation problem does not occur. — Critical
  - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition. — Desirable

# Steps in Data Normalization



# Steps in Data Normalization

## 1. First Normal Form

- 1NF is now considered to be part of the formal definition of a relation in the basic (flat) relational model.
- It was defined to disallow multivalued attributes, composite attributes, and their combinations. (I.e. The only attribute values permitted by 1NF are single atomic values).

# Steps in Data Normalization

## 1. First Normal Form

(a)

DEPARTMENT

DNAME	<u>DNUMBER</u>	DMGRSSN	DLOCATIONS



(b)

DEPARTMENT

DNAME	<u>DNUMBER</u>	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

# Steps in Data Normalization

## 1. First Normal Form

- There are three main techniques to achieve first normal form for such a relation:
  - Remove the attribute DLOCATIONS that violates 1NF and place it in a separate relation DEPT\_LOCATIONS along with the primary key DNUMBER of DEPARTMENT.
  - Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. — Redundancy
  - If a maximum number of values is known for the attribute (e.g. 3) replace the DLOCATIONS attribute by three atomic attributes: DLOCATION1, DLOCATION2, DLOCATION3. — Null values

# Steps in Data Normalization

## 1. First Normal Form

(a) <b>DEPARTMENT</b>			
Dname	Dnumber	Dmgr_ssn	Dlocations
(b) <b>DEPARTMENT</b>			
Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}
(c) <b>DEPARTMENT</b>			
Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

# Steps in Data Normalization

## 1. First Normal Form

(a) **EMP\_PROJ**

SSN	ENAME	PROJS	
		PNUMBER	HOURS

(b) **EMP\_PROJ**

SSN	ENAME	PNUMBER	HOURS
123456789	Smith,John B.	1	32.5
		2	7.5
666884444	Narayan,Ramesh K.	3	40.0
453453453	English,Joyce A.	1	20.0
		2	20.0
333445555	Wong,Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya,Alicia J.	30	30.0
		10	10.0
987987987	Jabbar,Ahmad V.	10	35.0
		30	5.0
987654321	Wallace,Jennifer S.	30	20.0
		20	15.0
888665555	Borg,James E.	20	null

(c) **EMP\_PROJ1**

SSN	ENAME
<b>EMP_PROJ2</b>	
SSN	PNUMBER

# Steps in Data Normalization

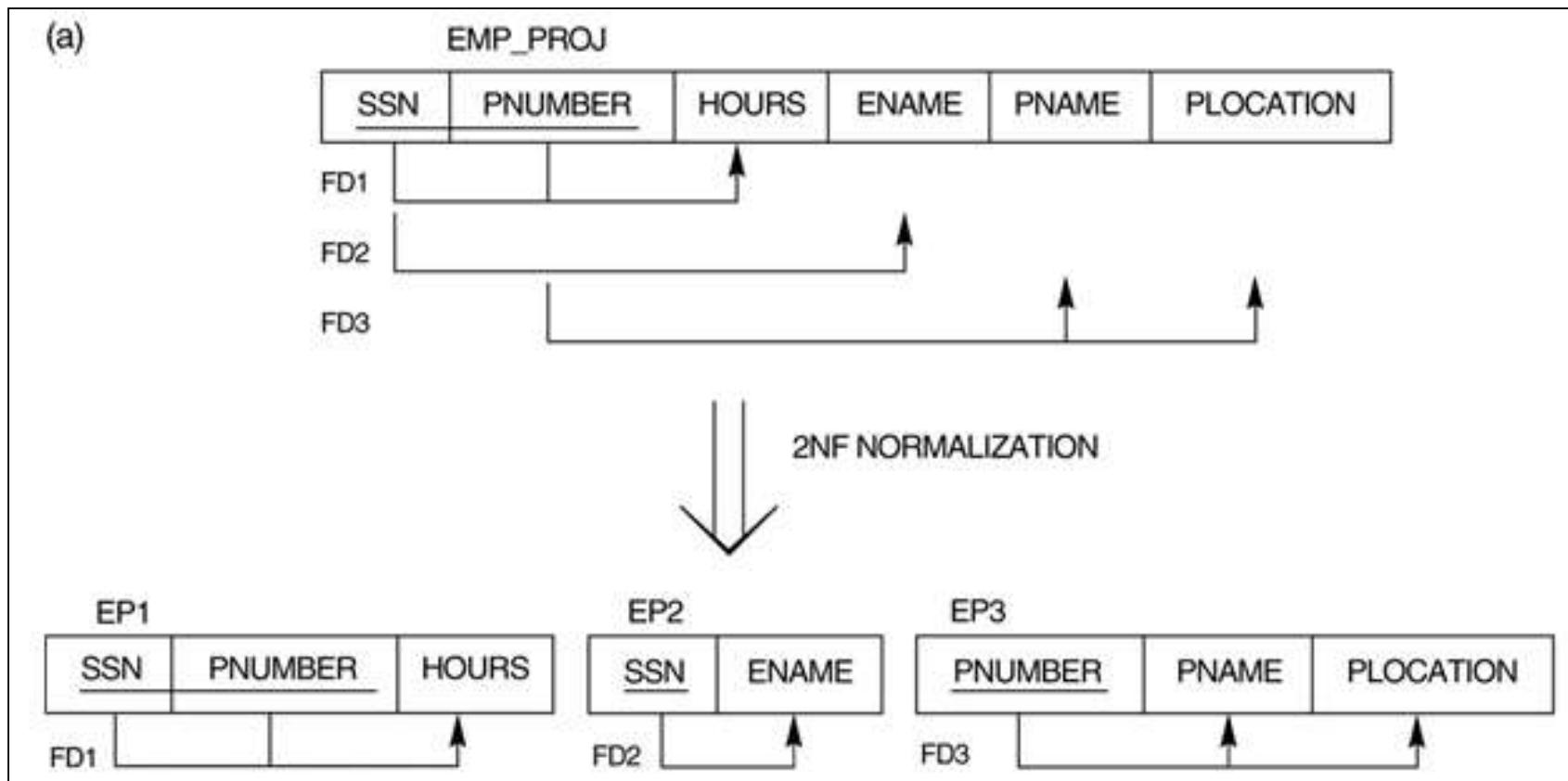
## 2. Second Normal Form

Not a member of  
any candidate key

- A relation is in 2NF if it is in 1NF and every nonprime attribute is fully functionally dependent on the primary key.
- I.e. remove any attributes which are dependent on part of the compound key.
- These attributes are put into a separate table along with that part of the compound key.

# Steps in Data Normalization

## 2. Second Normal Form



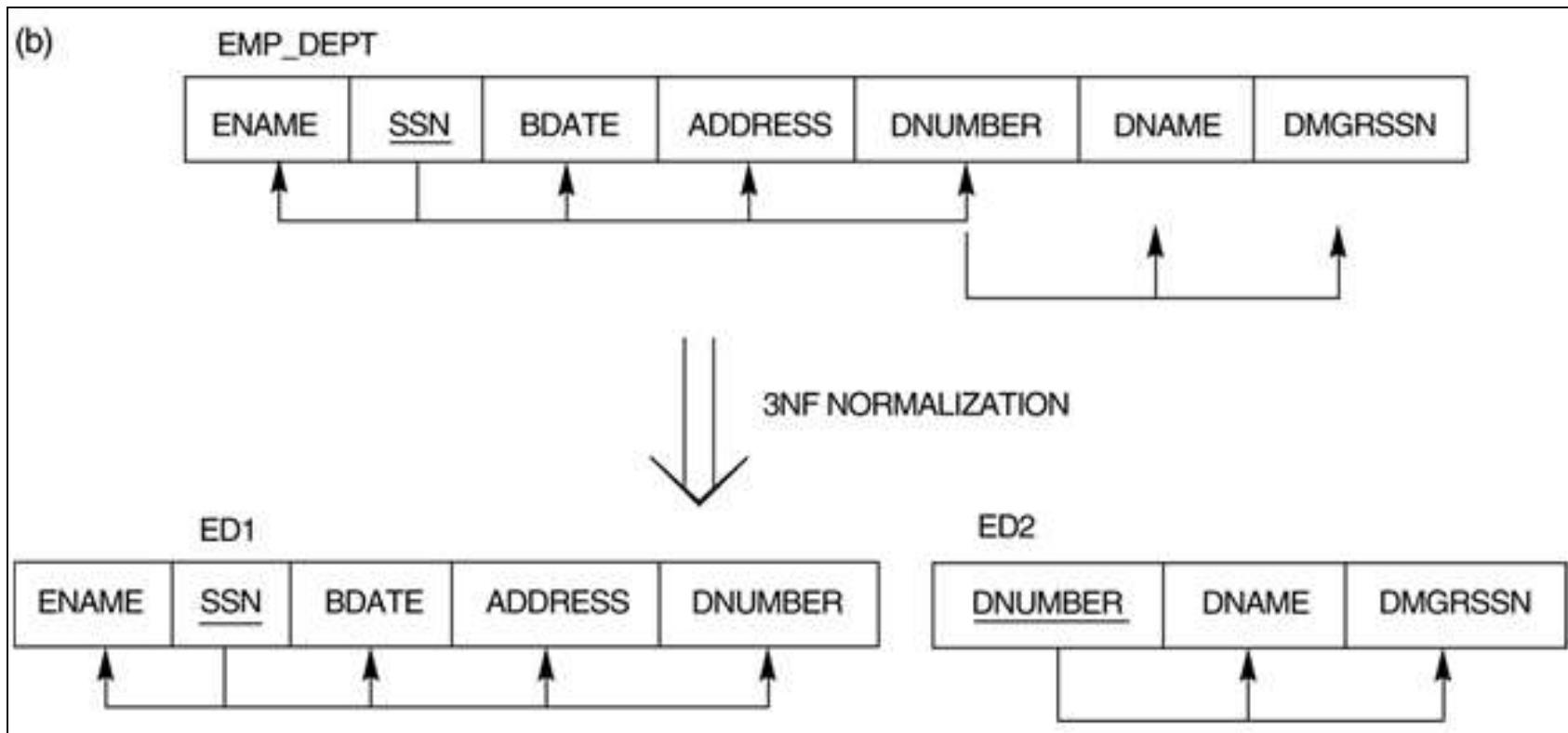
# Steps in Data Normalization

## 3. Third Normal Form

- A relation is in 3NF if it is in 2NF and no nonprime attribute A in R is transitively dependent on the primary key.
- I.e. Separate attributes which are dependent on another attribute other than the primary key within the table.

# Steps in Data Normalization

## 3. Third Normal Form



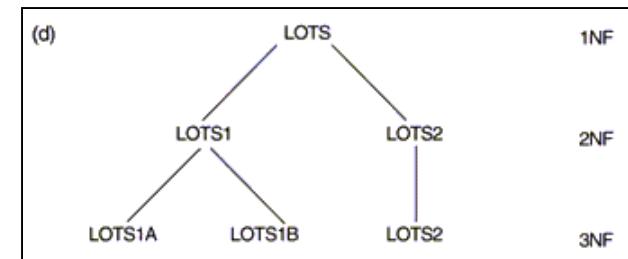
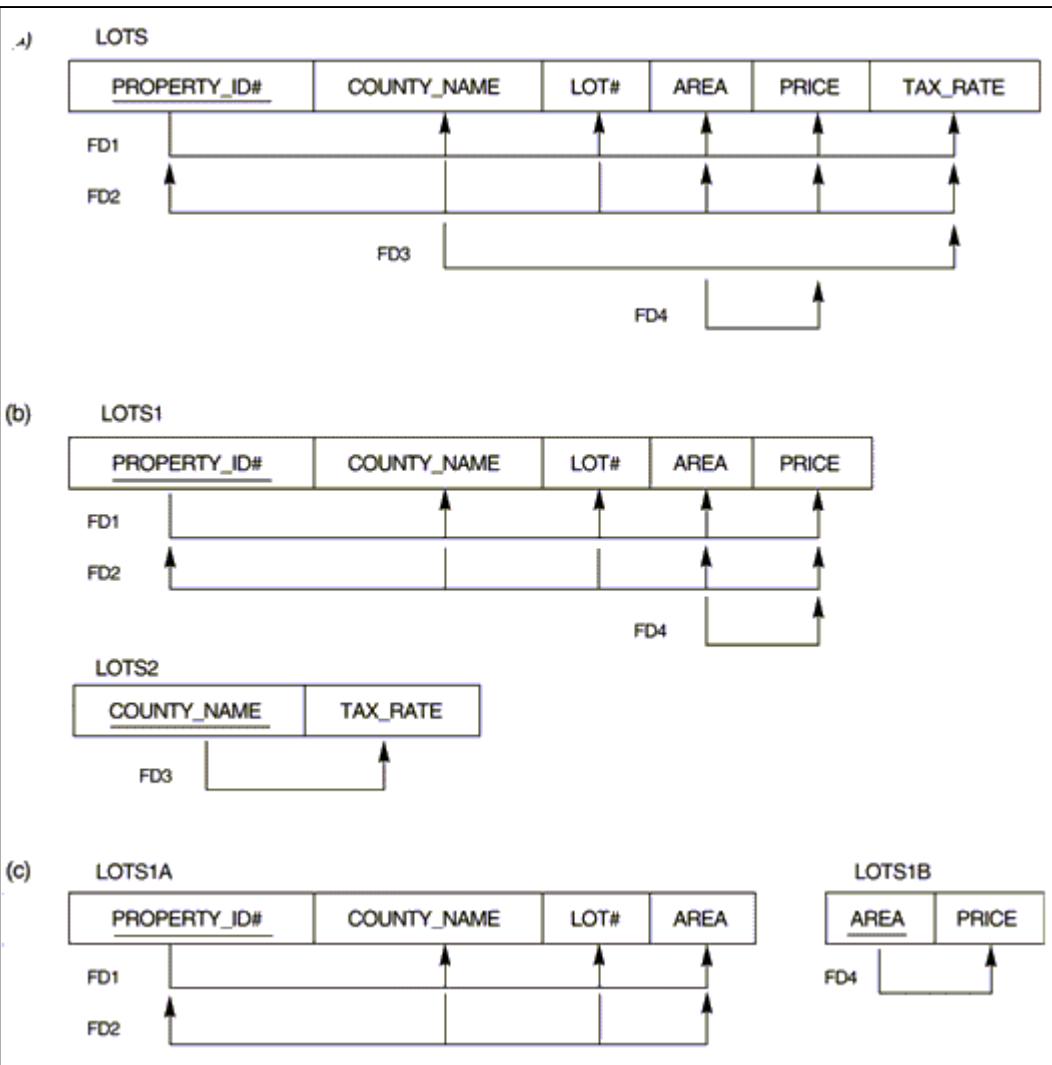
# Normal Forms Defined Informally

- 1<sup>st</sup> normal form → All attributes depend on the key.
- 2<sup>nd</sup> normal form → All attributes depend on the whole key.
- 3<sup>rd</sup> normal form → All attributes depend on nothing but the key.

## General Definitions of Second and Third Normal Forms

- The previous definitions consider the primary key only.
- The following more general definitions take into account relations with multiple candidate keys.
- A relation is in 2NF if it is in 1NF and every nonprime attribute is fully functionally dependent on every key.
- A relation is in 3NF if it is in 2NF and if whenever a FD  $X \rightarrow A$  holds in R, then either:
  - X is a superkey of R, or
  - A is a prime attribute of R.

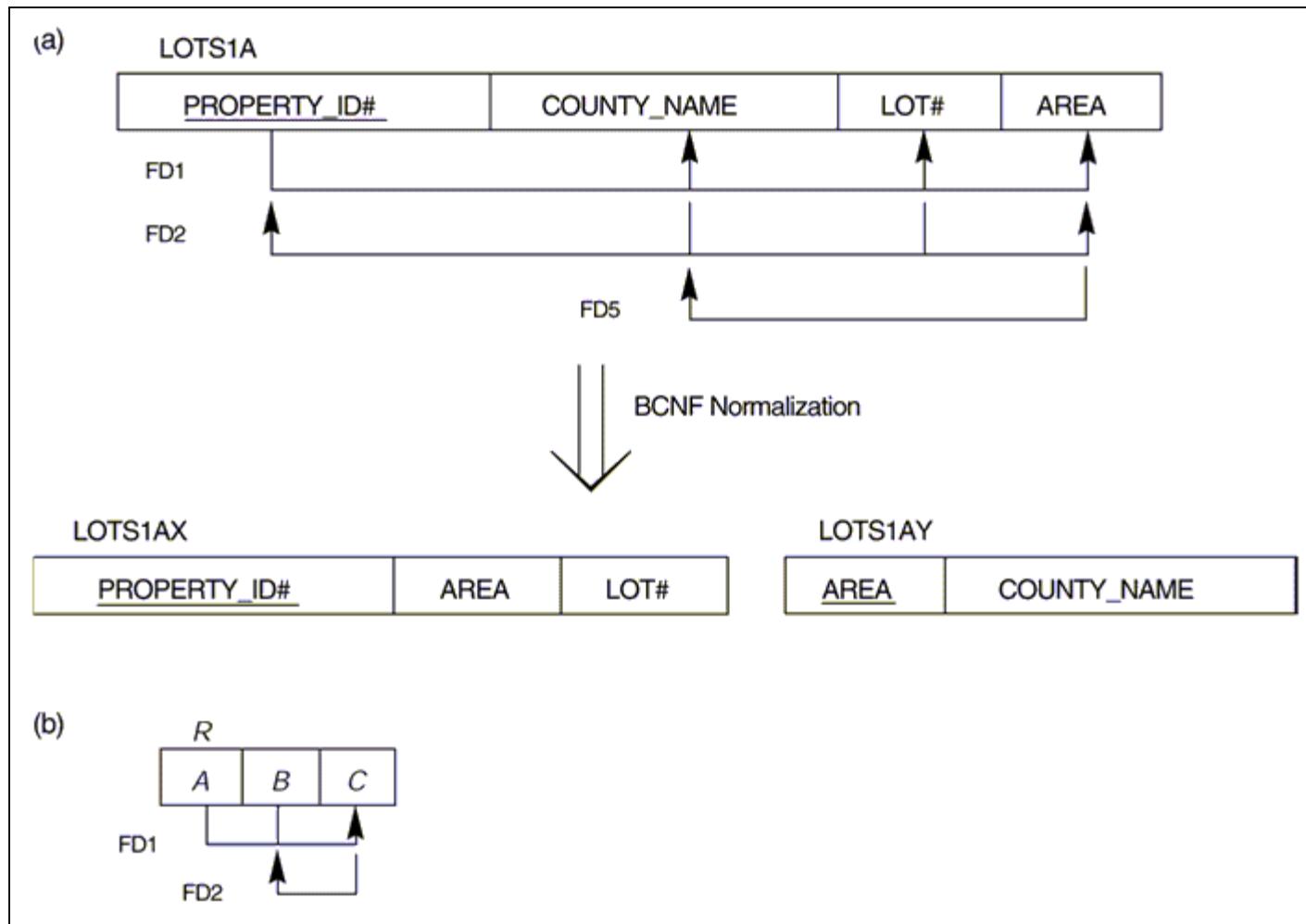
# General Definitions of Second and Third Normal Forms



# Boyce-Codd Normal Form (BCNF)

- BCNF was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
- A relation schema R is in BCNF if whenever a FD  $X \rightarrow A$  holds in R, then X is a superkey of R.
- I.e. A relation is in BCNF if every determinant is a key.
- Thus:
  - Every relation in BCNF is also in 3NF.
  - A relation in 3NF is not necessarily in BCNF.
- The goal is to have each relation in BCNF (or 3NF).

# Boyce-Codd Normal Form (BCNF) - Example 1



# Boyce-Codd Normal Form (BCNF) - Example 2

- DIRECTORY (EmployeeNo, EmployeeName, DepartmentName, RoomNo, TelNo)
- Where:
  - No employee works for more than one department.
  - Many employees may occupy one room.
  - Employee numbers are unique.
  - No room is shared by between departments.
- Two FDs exist in the relation DIRECTORY:
  - $\text{EmployeeNo} \rightarrow \{\text{EmployeeName}, \text{DepartmentName}, \text{RoomNo}, \text{TelNo}\}$
  - $\text{RoomNo} \rightarrow \text{DepartmentName}$

## Boyce-Codd Normal Form (BCNF) - Example 2

- All attributes are dependent on EmployeeNo (the primary key).
- RoomNo is also a determinant, but not a candidate key.
- This violates the definition of BCNF and therefore DIRECTOTY must be decomposed into two relations:
  - EMP (EmployeeNo, EmployeeName, RoomNo, TelNo).
  - ALLOC (RoomNo, DepartmentName).

# Boyce-Codd Normal Form (BCNF) - Example 3

- TEACH(Student, Course, Instructor).
- Two FDs exist in the relation TEACH:
  - $\{ \text{Student}, \text{Course} \} \rightarrow \text{Instructor}$
  - $\text{Instructor} \rightarrow \text{Course}$
- $\{ \text{Student}, \text{Course} \}$  is a candidate key for this relation.
- This relation is in 3NF but not in BCNF.

# Boyce-Codd Normal Form (BCNF) - Example 3

- Three possible decomposition for relation TEACH:
  - {Student, instructor} and {Student, Course}
  - {Course, Instructor} and {Course, Student}
  - {Instructor, Course} and {instructor, Student}
- All three decompositions will lose FD1.
- Only the 3rd decomposition will not generate spurious tuples after join.

# Advantages of Normalization

- Greater overall database organization will be gained.
- The amount of unnecessary redundant data is reduced.
- Data integrity is easily maintained within the database.
- The database & application design processes are much more flexible.
- Security is easier to manage.

# Disadvantages of Normalization

- Produces lots of tables with a relatively small number of columns.
- Probably requires joins in order to put the information back together in the way it needs to be used - effectively reversing the normalization.
- Impacts computer performance (CPU, I/O, memory).

# Conclusion

- Data normalization is a bottom-up technique that ensures the basic properties of the relational model:
  - No duplicate tuples.
  - No nested relations.
- A more appropriate approach is to complement conceptual modeling with data normalization.