

DBMS: ERD, Normalization, QueryOpt

Dr. Debasish Jana

***BE(CS)(JU), MMATH(CS)(UW, Canada), MBA(Fin)(IGNOU), PhD(CS, JU),
FIE(I), FIETE, SMIEEE, SMACM
debasishj871 at gmail dot com***

2023

Example COMPANY Database

- We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:
 - The company is organized into DEPARTMENTS. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
 - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

Example COMPANY Database (Continued)

- The database will store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
 - Each employee *works for* one department but may *work on* several projects.
 - The DB will keep track of the number of hours per week that an employee currently works on each project.
 - It is required to keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of DEPENDENTS.
 - For each dependent, the DB keeps a record of name, sex, birthdate, and relationship to the employee.

ER Model Concepts

- Entities and Attributes

- Entity is a basic concept for the ER model. Entities are specific things or objects in the mini-world that are represented in the database.
 - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
- Attributes are properties used to describe an entity.
 - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate
- A specific entity will have a value for each of its attributes.
 - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55'
- Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, date, enumerated type, ...

Types of Attributes (1)









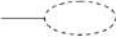


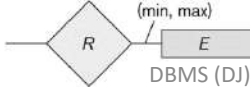
- Simple
 - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- Composite
 - The attribute may be composed of several components. For example:
 - Address(Apt#, House#, Street, City, State, ZipCode, Country), or
 - Name(FirstName, MiddleName, LastName).
 - Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
 - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.
 - Denoted as {Color} or {PreviousDegrees}.

Types of Attributes (2)

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.
 - For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}
 - Multiple PreviousDegrees values can exist
 - Each has four subcomponent attributes:
 - College, Year, Degree, Field

NOTATION for ER diagrams

Figure 3.14
Summary of the
notation for ER
diagrams.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R DBMS (DJ)

Entity Type CAR with two keys and a corresponding Entity Set

(a)

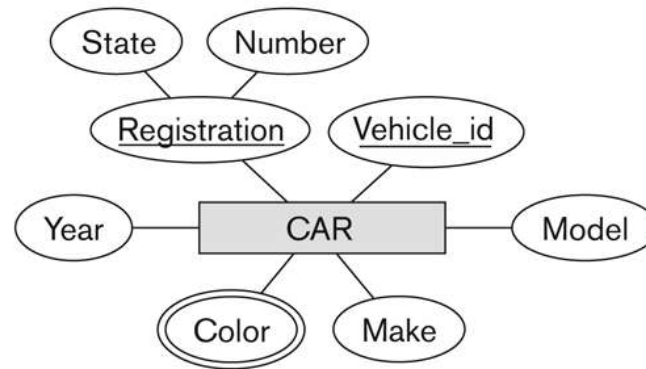


Figure 3.7

The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR₁
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR₂
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR₃
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Initial Design of Entity Types:

EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

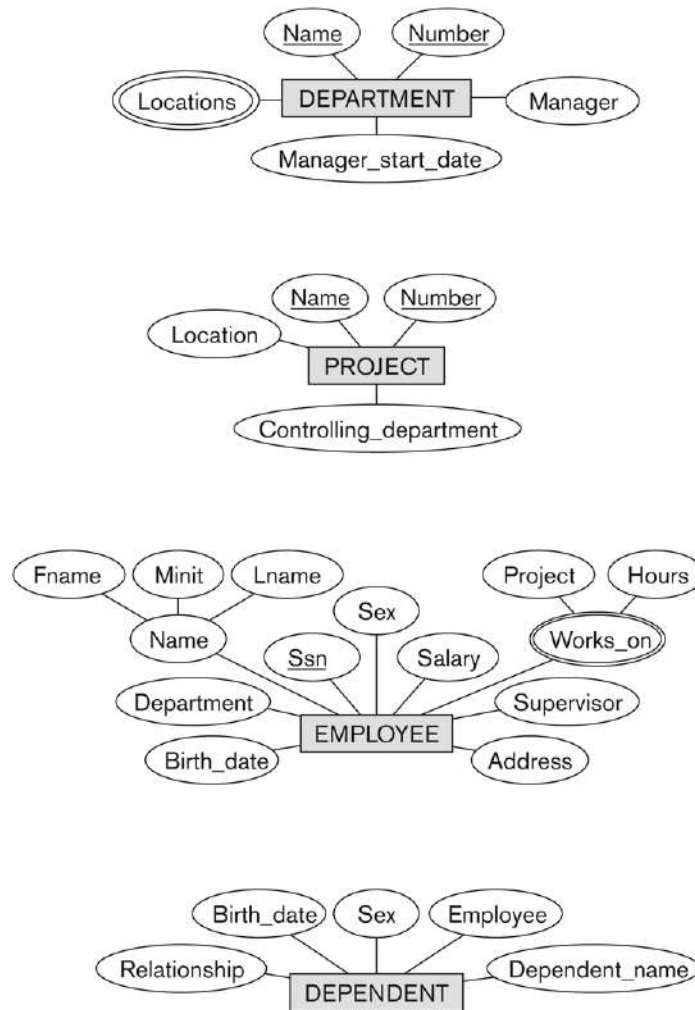


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Refining the initial design by introducing **relationships**

- The initial design is typically not complete
- Some aspects in the requirements will be represented as **relationships**
- ER model has three main concepts:
 - Entities (and their entity types and entity sets)
 - Attributes (simple, composite, multivalued)
 - Relationships (and their relationship types and relationship sets)
- We introduce relationship concepts next

Relationships and Relationship Types (1)

- A **relationship** relates two or more distinct entities with a specific meaning.
 - For example, EMPLOYEE John Smith *works on* the ProductX PROJECT, or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
 - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
 - Both MANAGES and WORKS_ON are *binary* relationships.

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

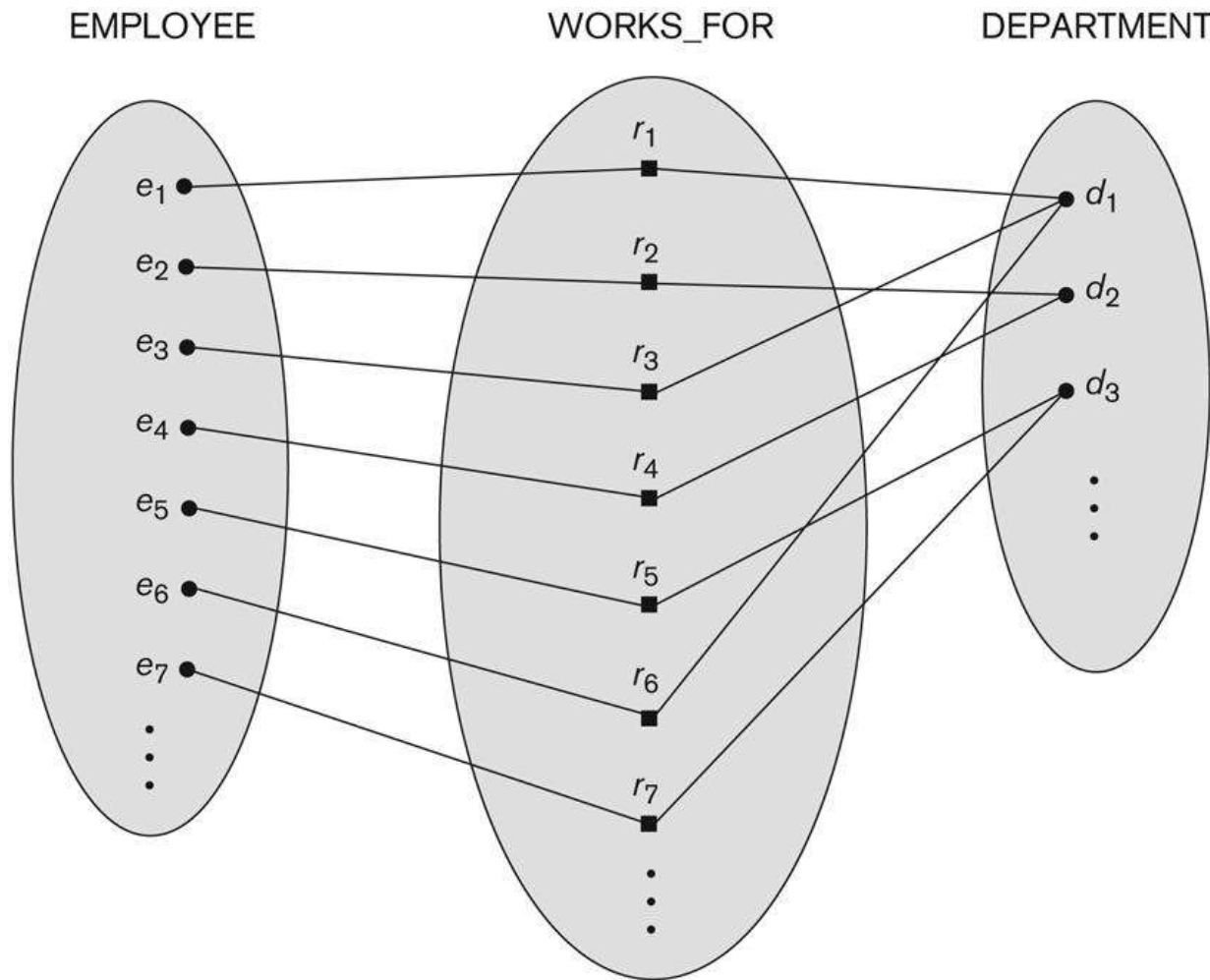


Figure 3.9

Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

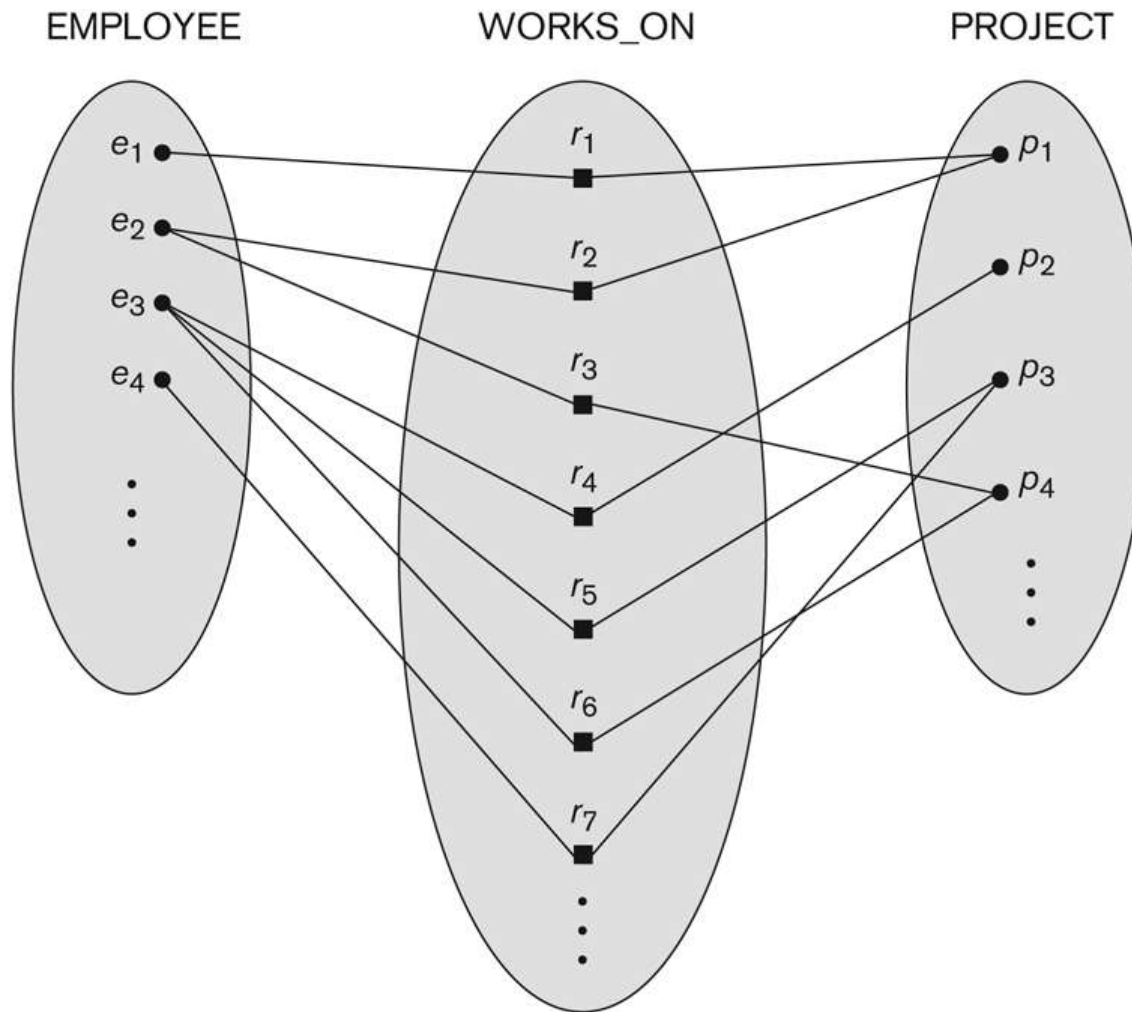


Figure 3.13
An M:N relationship,
WORKS_ON.

Relationship type vs. relationship set (1)

- Relationship Type:
 - Is the schema description of a relationship
 - Identifies the relationship name and the participating entity types
 - Also identifies certain relationship constraints
- Relationship Set:
 - The current set of relationship instances represented in the database
 - The current *state* of a relationship type

Relationship type vs. relationship set (2)

- Previous figures displayed the relationship sets
- Each instance in the set relates individual participating entities – one from each participating entity type
- In ER diagrams, we represent the *relationship type* as follows:
 - Diamond-shaped box is used to display a relationship type
 - Connected to the participating entity types via straight lines
 - Note that the relationship type is not shown with an arrow. The name should be typically be readable from left to right and top to bottom.

Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are *binary* relationships(degree 2)
- Listed below with their participating entity types:
 - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
 - MANAGES (also between EMPLOYEE, DEPARTMENT)
 - CONTROLS (between DEPARTMENT, PROJECT)
 - WORKS_ON (between EMPLOYEE, PROJECT)
 - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
 - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER DIAGRAM – Relationship Types are:

WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF

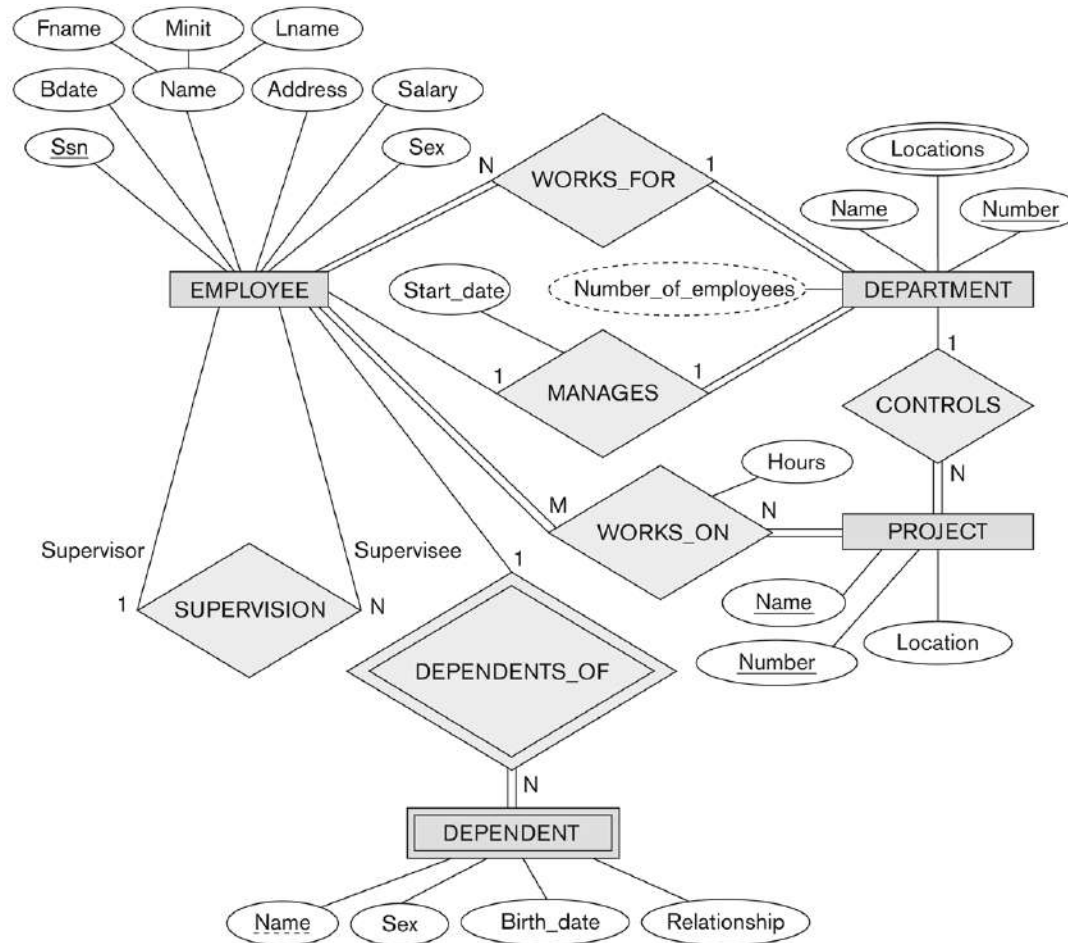


Figure 3.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
 - Manager of DEPARTMENT -> MANAGES
 - Works_on of EMPLOYEE -> WORKS_ON
 - Department of EMPLOYEE -> WORKS_FOR
 - etc
- In general, more than one relationship type can exist between the same participating entity types
 - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
 - Different meanings and different relationship instances.

Constraints on Relationships

- Constraints on Relationship Types
 - (Also known as ratio constraints)
 - Cardinality Ratio (specifies *maximum* participation)
 - One-to-one (1:1)
 - One-to-many (1:N) or Many-to-one (N:1)
 - Many-to-many (M:N)
 - Existence Dependency Constraint (specifies *minimum* participation) (also called participation constraint)
 - zero (optional participation, not existence-dependent)
 - one or more (mandatory participation, existence-dependent)

Many-to-one (N:1) Relationship

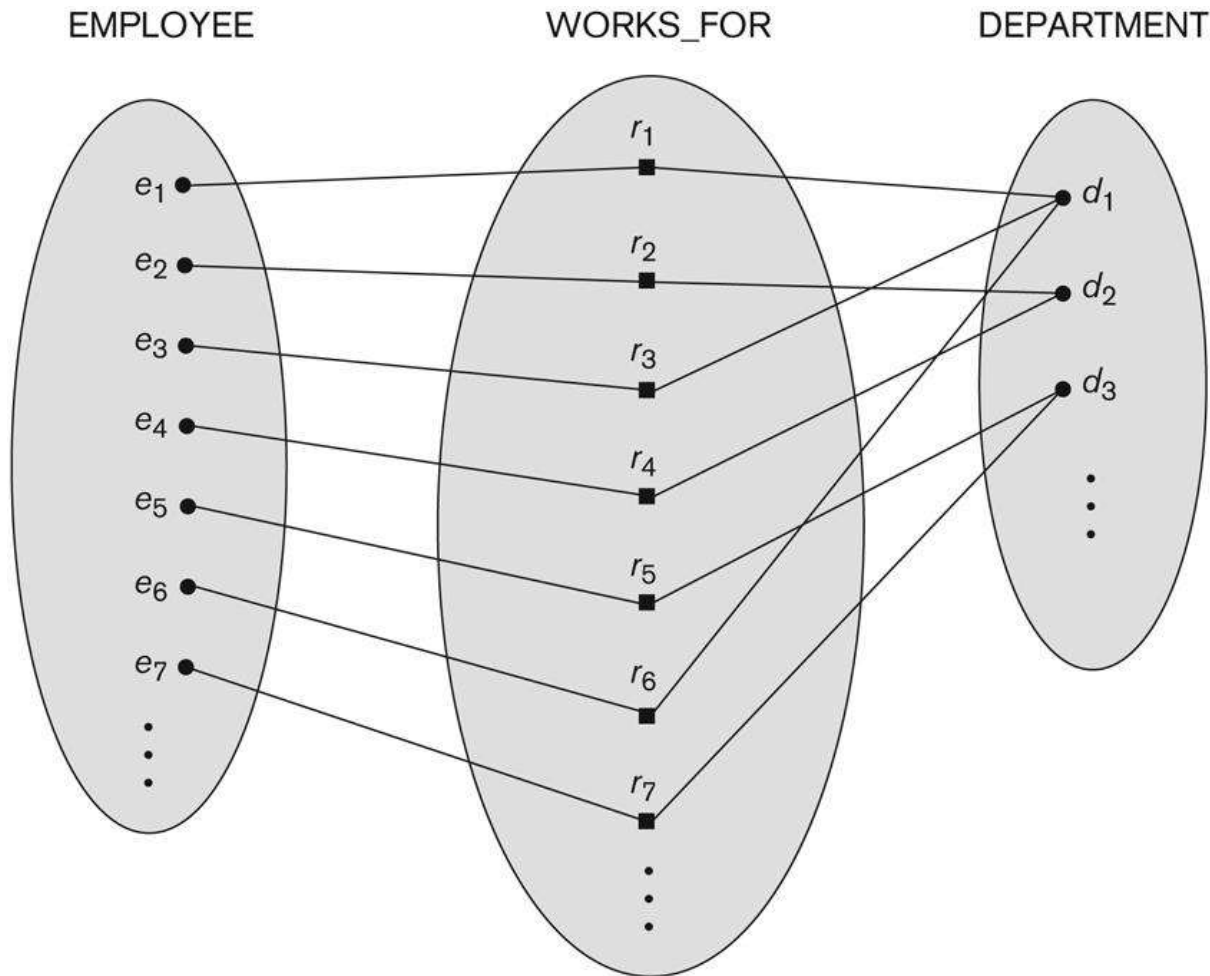


Figure 3.9

Some instances in the **WORKS_FOR** relationship set, which represents a relationship type **WORKS_FOR** between **EMPLOYEE** and **DEPARTMENT**.

Many-to-many (M:N) Relationship

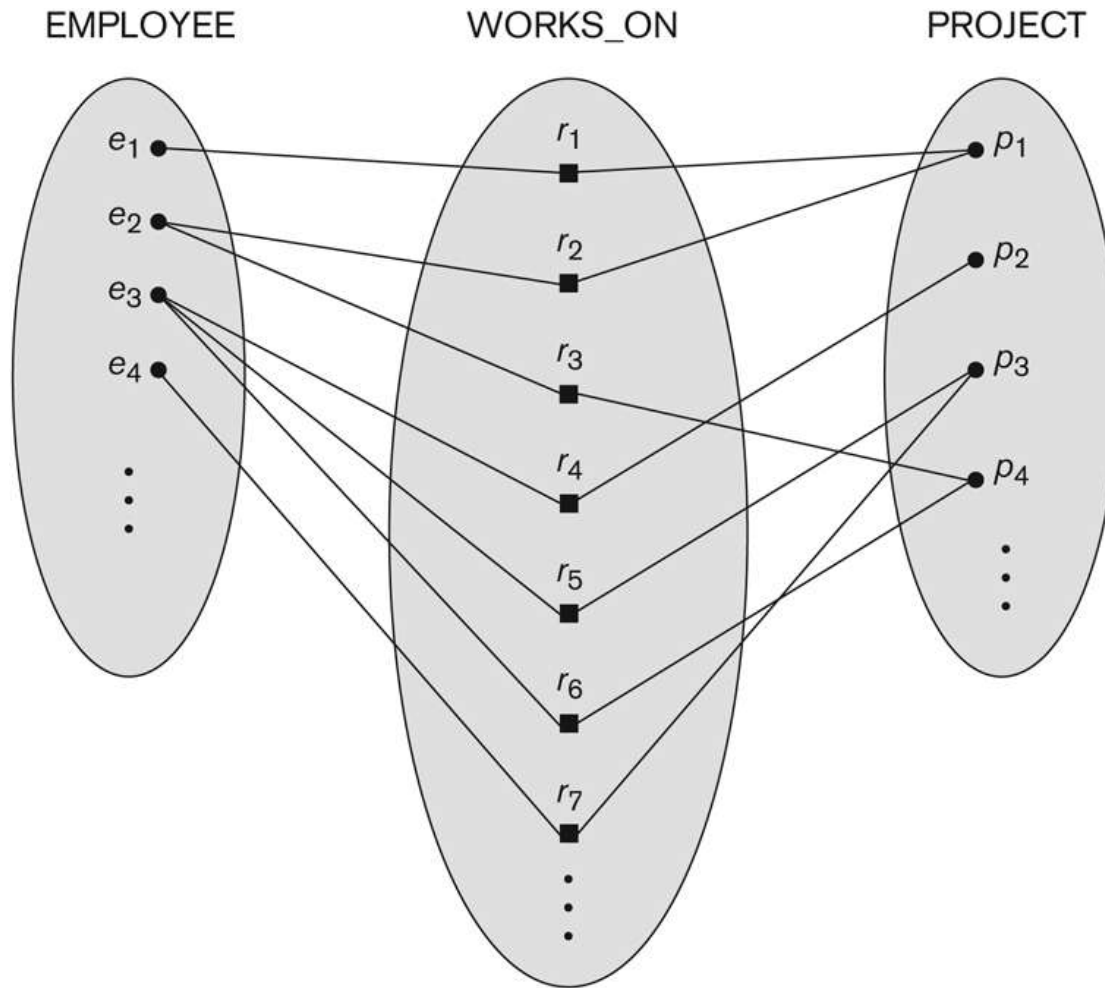


Figure 3.13
An M:N relationship,
WORKS_ON.

Recursive Relationship Type

- A relationship type between the same participating entity type in **distinct roles**
- Also called a **self-referencing** relationship type.
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
 - supervisor (or boss) role
 - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
 - One employee in *supervisor* role
 - One employee in *supervisee* role

Displaying a recursive relationship

- In a recursive relationship type.
 - Both participations are same entity type in different roles.
 - For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

A Recursive Relationship Supervision`

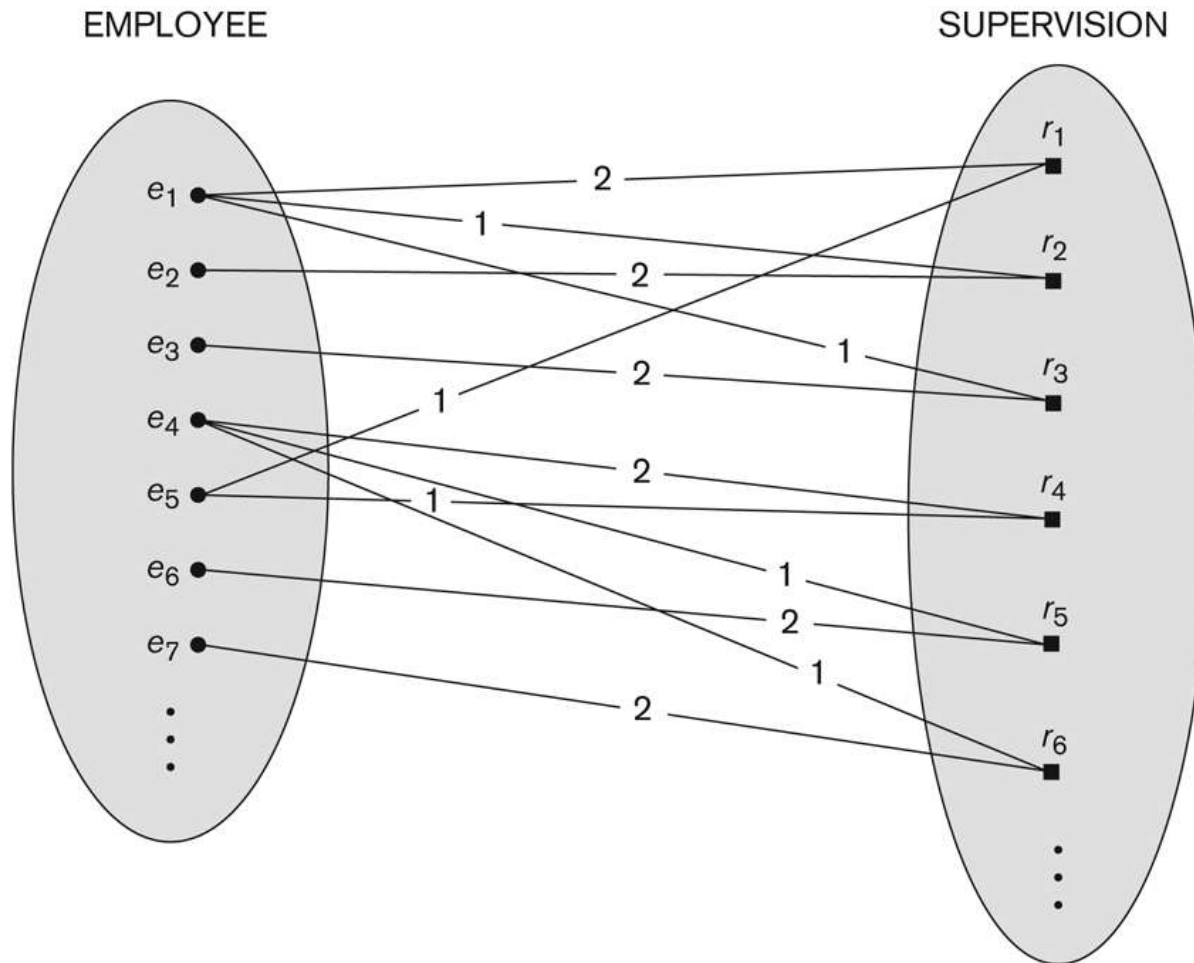


Figure 3.11
A recursive relationship **SUPERVISION** between **EMPLOYEE** in the *supervisor* role (1) and **EMPLOYEE** in the *subordinate* role (2).

Recursive Relationship Type is: SUPERVISION (participation role names are shown)

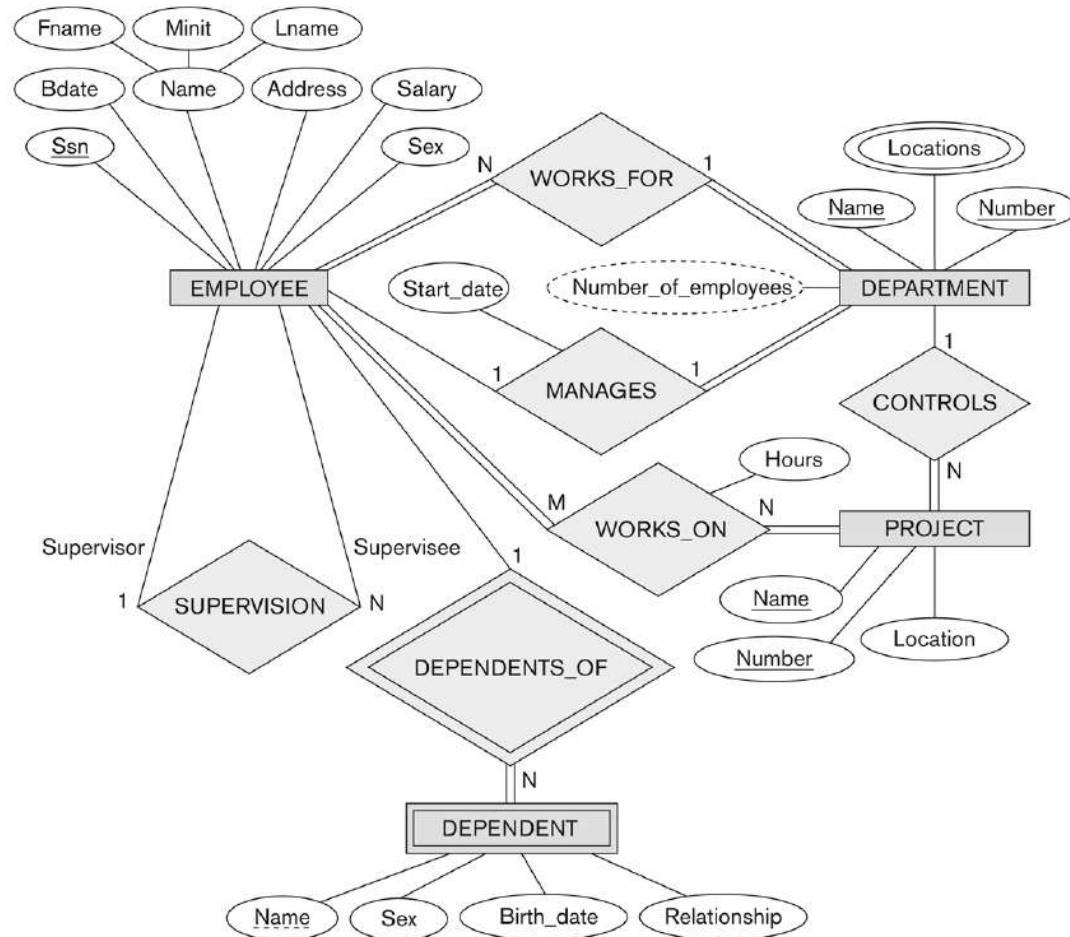


Figure 3.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Weak Entity Types

- An entity that does not have a key attribute and that is identification-dependent on another entity type.
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
 - A partial key of the weak entity type
 - The particular entity they are related to in the identifying relationship type
- **Example:**
 - A DEPENDENT entity is identified by the dependent's first name, *and* the specific EMPLOYEE with whom the dependent is related
 - Name of DEPENDENT is the *partial key*
 - DEPENDENT is a *weak entity type*
 - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Attributes of Relationship types

- A relationship type can have attributes:
 - For example, HoursPerWeek of WORKS_ON
 - Its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.
 - A value of HoursPerWeek depends on a particular (employee, project) combination
- Most relationship attributes are used with M:N relationships
 - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship

Example Attribute of a Relationship Type: Hours of WORKS_ON

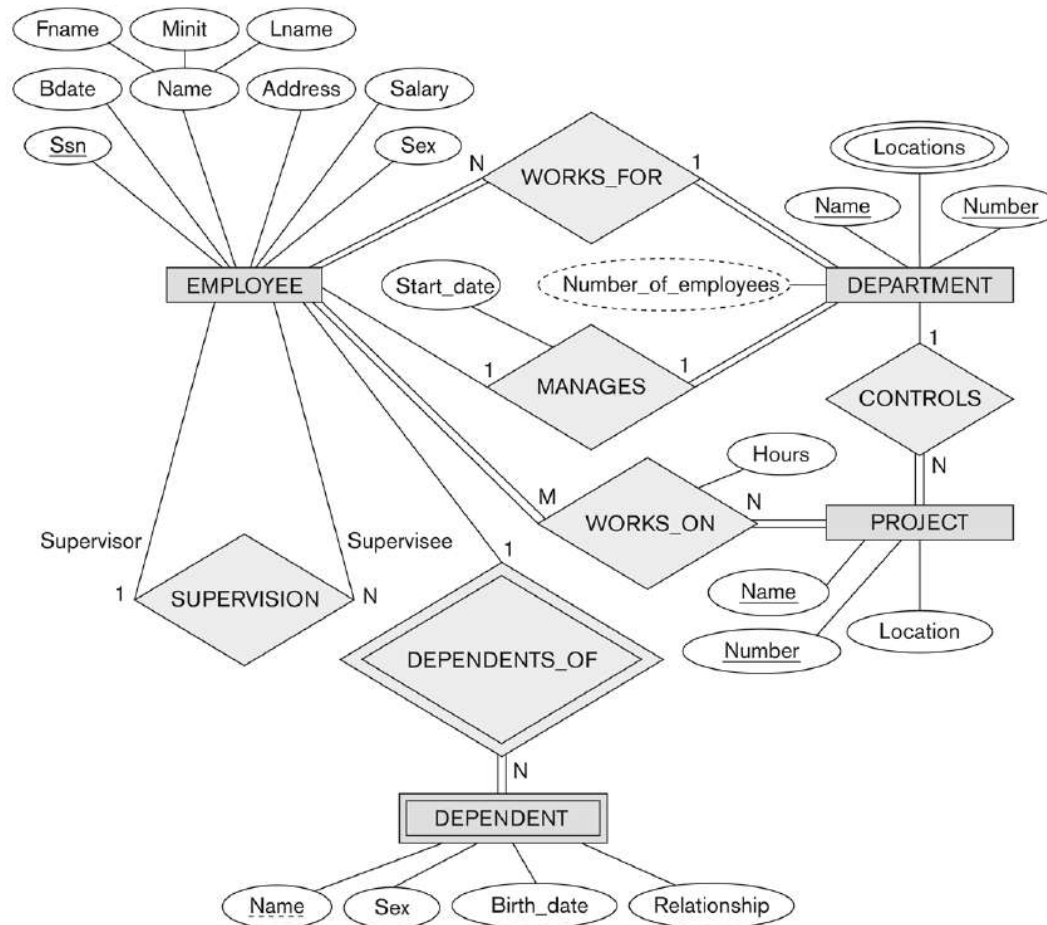


Figure 3.2

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

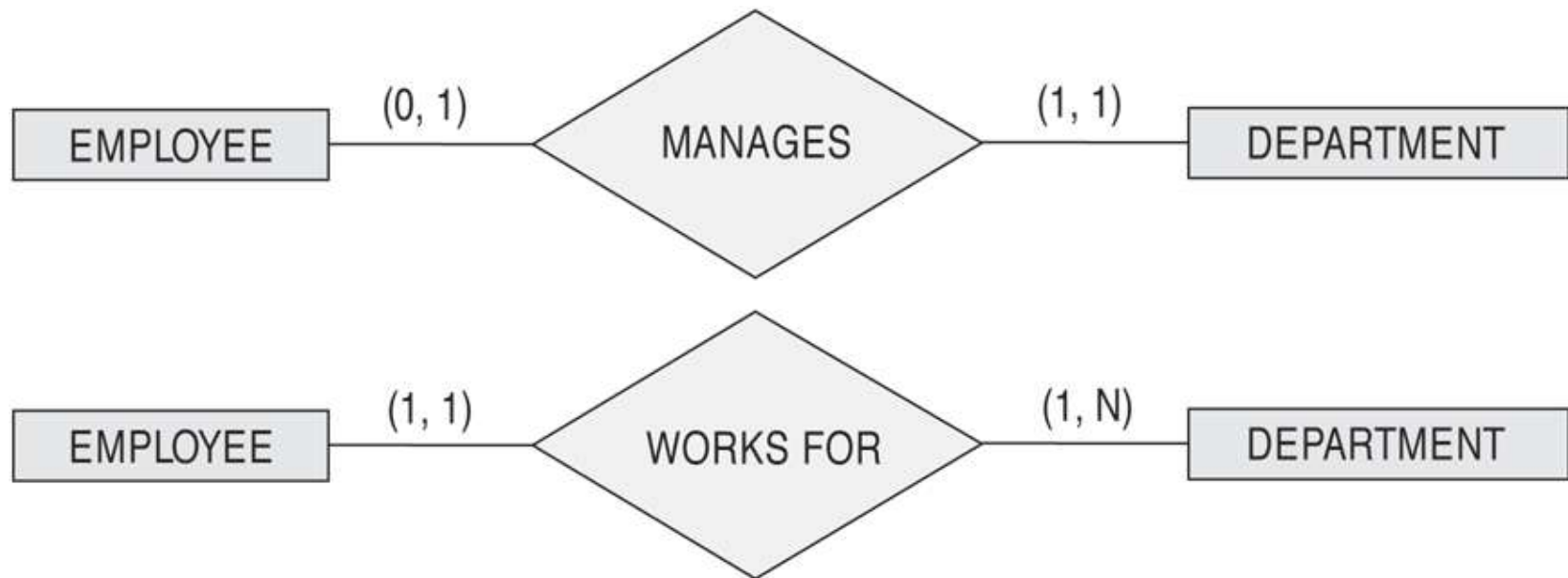
Notation for Constraints on Relationships

- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
 - Shown by placing appropriate numbers on the relationship edges.
- Participation constraint (on each participating entity type): total (called existence dependency) or partial.
 - Total shown by double line, partial by single line.
- NOTE: These are easy to specify for Binary Relationship Types.

Alternative (min, max) notation for relationship structural constraints:

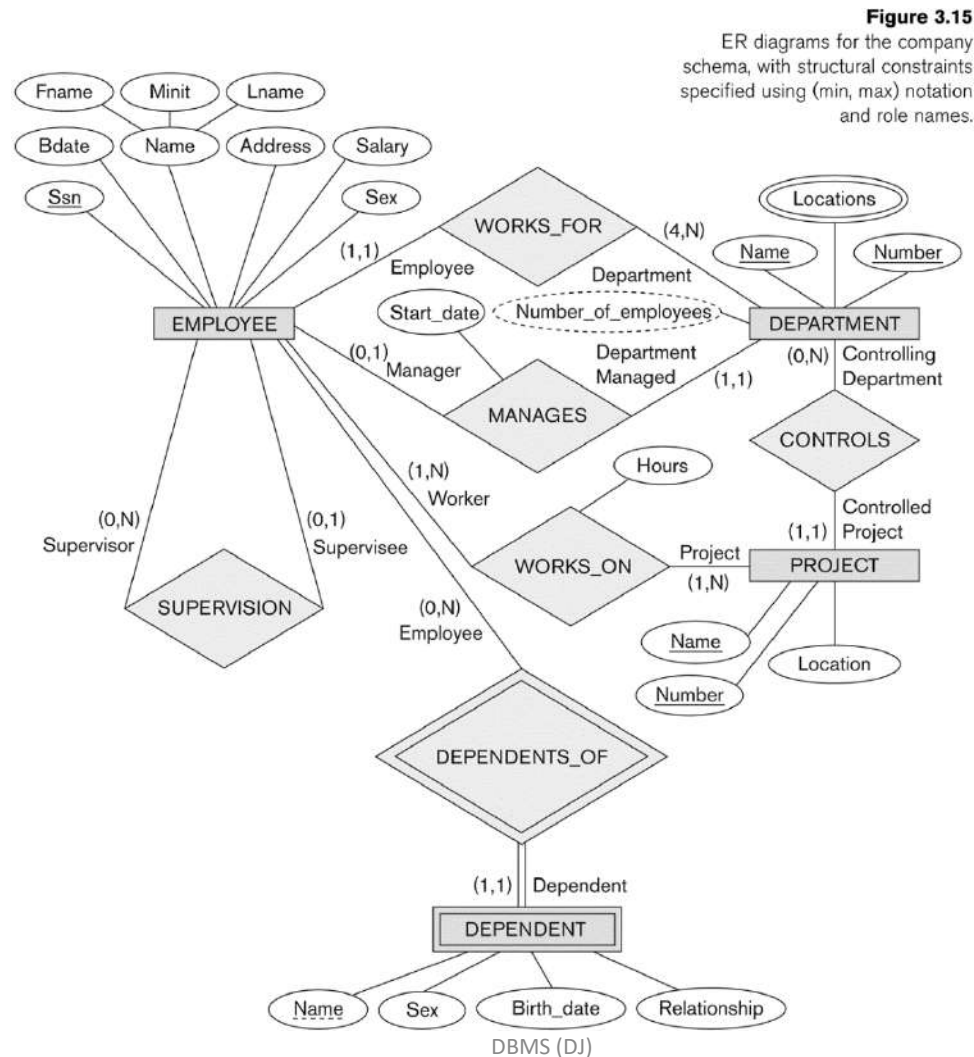
- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have $\text{min} \leq \text{max}$, $\text{min}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

The (min,max) notation for relationship constraints



Read the min,max numbers next to the entity type and looking **away from** the entity type

COMPANY ER Schema Diagram using (min, max) notation









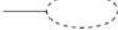
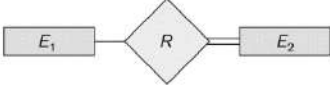

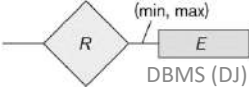


Alternative diagrammatic notation

- 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

Summary of notation for ER diagrams

Figure 3.14
Summary of the
notation for ER
diagrams.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R DBMS (DJ)

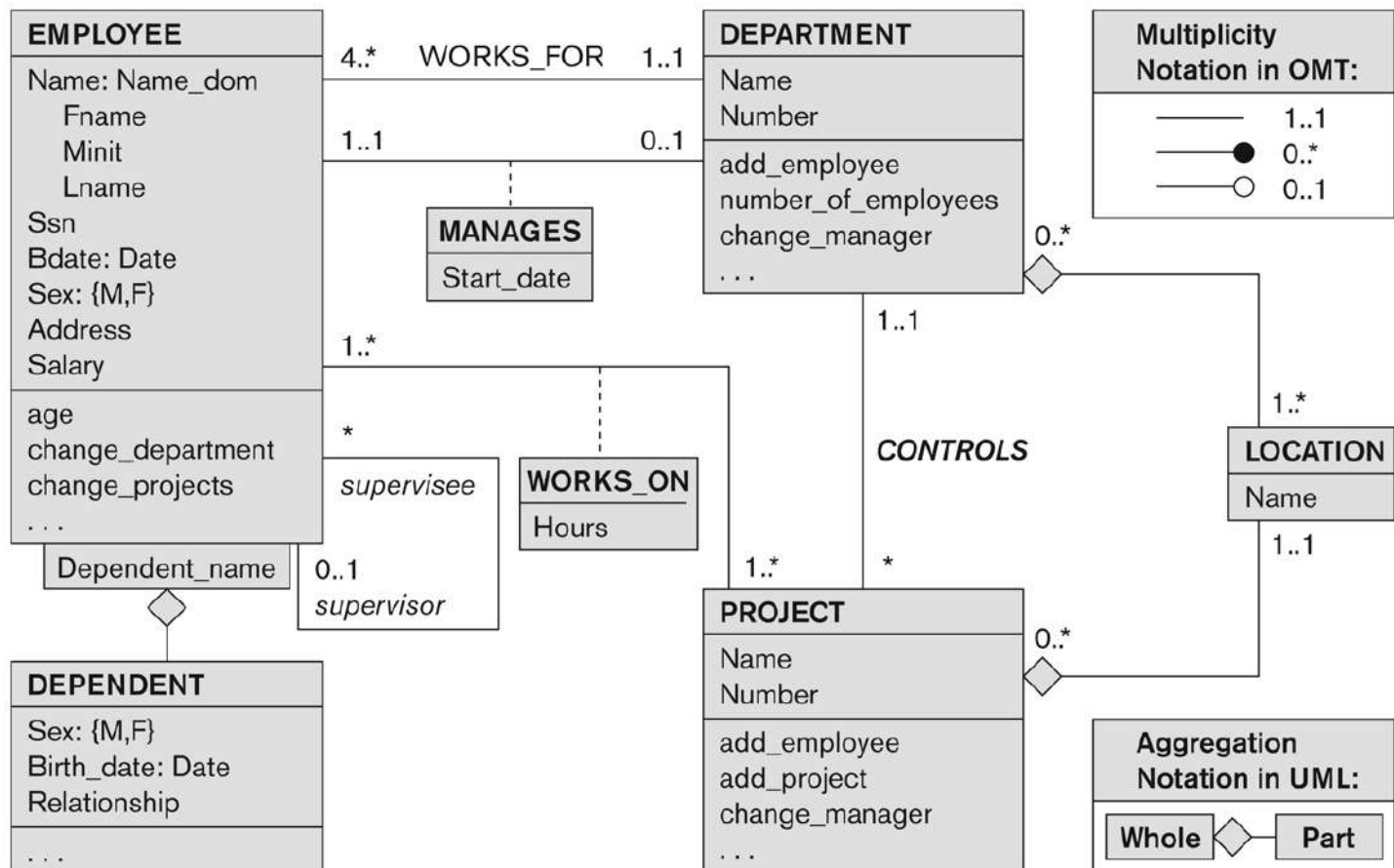
UML class diagrams

- Represent classes (similar to entity types) as large rounded boxes with three sections:
 - Top section includes entity type (class) name
 - Second section includes attributes
 - Third section includes class operations (operations are not in basic ER model)
- Relationships (called associations) represented as lines connecting the classes
 - Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design

UML class diagram for COMPANY database schema

Figure 3.16

The COMPANY conceptual schema in UML class diagram notation.



Other alternative diagrammatic notations

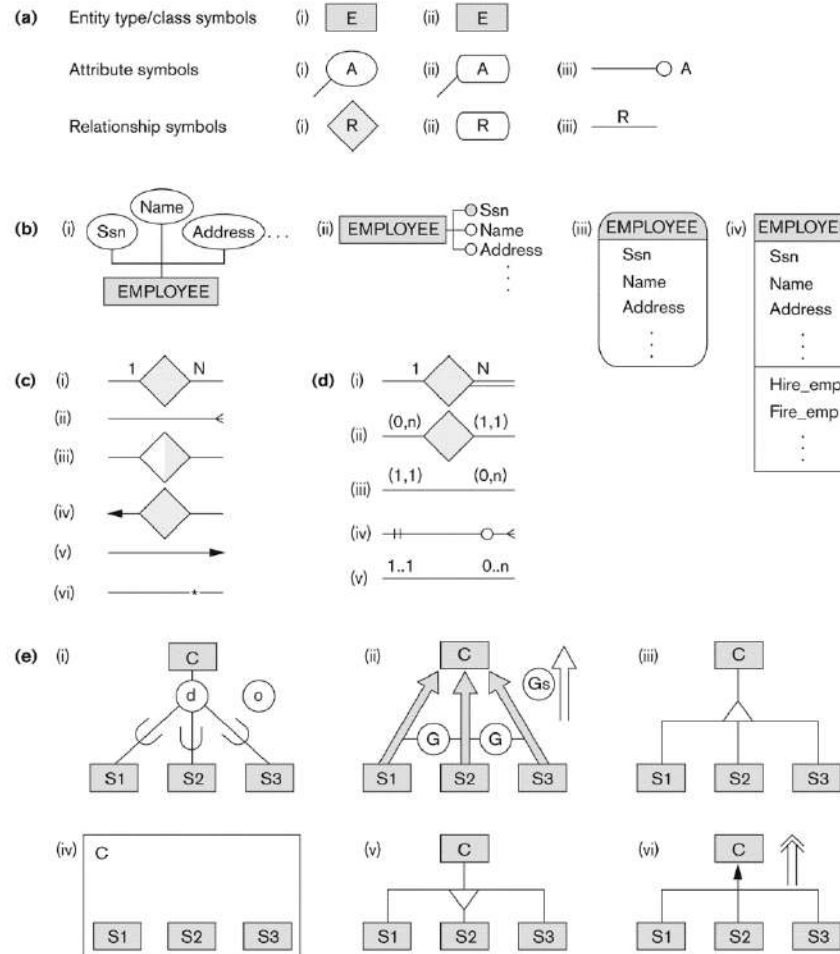


Figure A.1

Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

DBMS (DJ)

Relationships of Higher Degree

- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are 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

Discussion of n-ary relationships ($n > 2$)

- In general, 3 binary relationships can represent different information than a single ternary relationship (see Figure 3.17a and b on next slide)
- If needed, the binary and n-ary relationships can all be included in the schema design (see Figure 3.17a and b, where all relationships convey different meanings)
- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types) (see Figure 3.17c)

Example of a ternary relationship

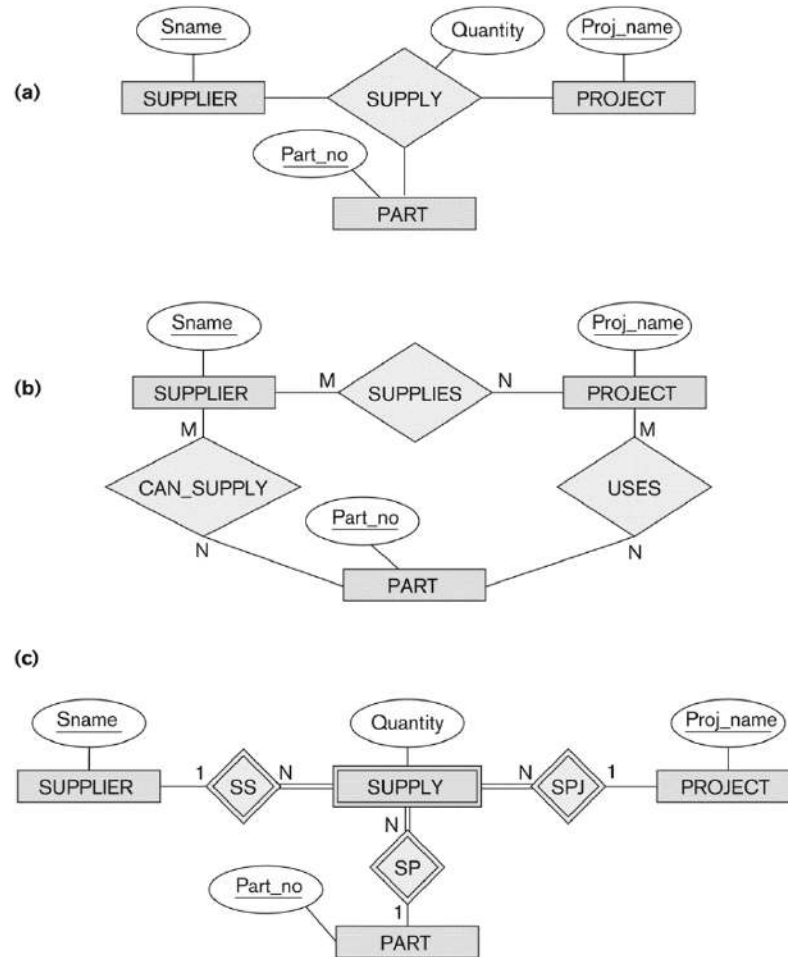


Figure 3.17

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

DBMS (DJ)

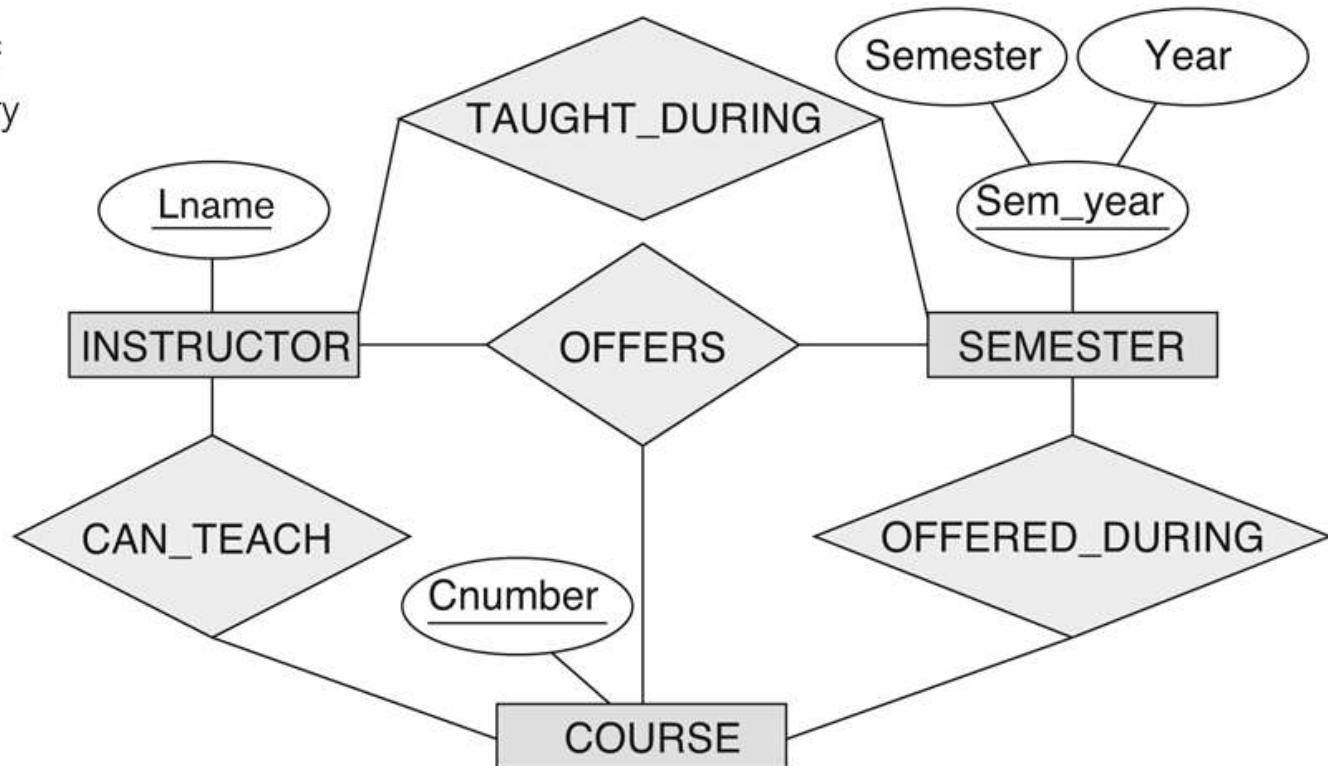
Discussion of n-ary relationships ($n > 2$)

- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant
- For example, the TAUGHT_DURING binary relationship in Figure 3.18 (see next slide) can be derived from the ternary relationship OFFERS (based on the meaning of the relationships)

Another example of a ternary relationship

Figure 3.18

Another example of ternary versus binary relationship types.



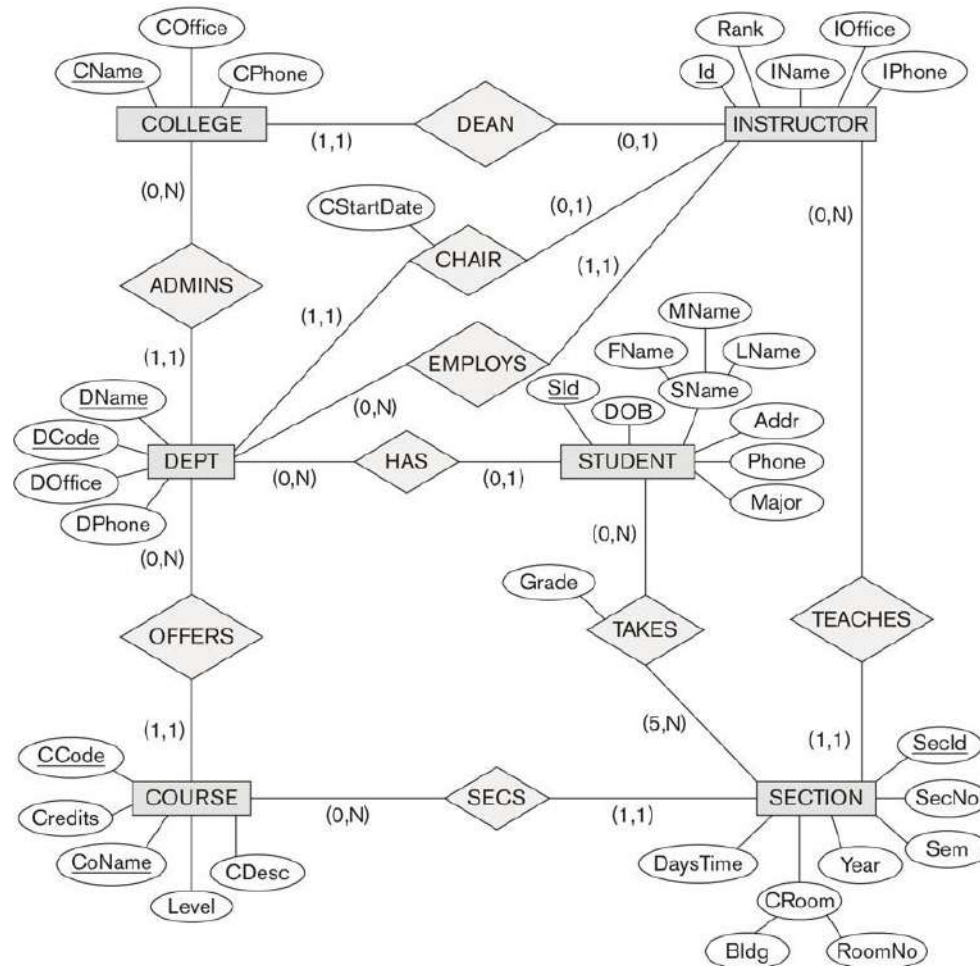
Displaying constraints on higher-degree relationships

- 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
- Overall, the constraint specification is difficult and possibly ambiguous when we consider relationships of a degree higher than two.

Another Example: A UNIVERSITY Database

- To keep track of the enrollments in classes and student grades, another database is to be designed.
- It keeps track of the COLLEGES, DEPARTMENTS within each college, the COURSEs offered by departments, and SECTIONs of courses, INSTRUCTORs who teach the sections etc.
- These entity types and the relationships among these entity types are shown on the next slide in Figure 3.20.

UNIVERSITY database conceptual schema



Data Modeling Tools (Additional Material)

- A number of popular tools that cover conceptual modeling and mapping into relational schema design.
 - Examples: ERWin, S- Designer (Enterprise Application Suite), ER- Studio, etc.
- POSITIVES:
 - Serves as documentation of application requirements, easy user interface - mostly graphics editor support
- NEGATIVES:
 - Most tools lack a proper distinct notation for relationships with relationship attributes
 - Mostly represent a relational design in a diagrammatic form rather than a conceptual ER-based design

Some of the Automated Database Design Tools (Note: Not all may be on the market now)

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration, space and security management
Oracle	Developer 2000/Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum (Computer Associates)	Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational (IBM)	Rational Rose	UML Modeling & application generation in C++/JAVA
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance
Sybase	Enterprise Application Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design/reengineering Visual Basic/C++

Extended Entity-Relationship (EER) Model (in the next chapter)

- The entity relationship model in its original form did not support the specialization and generalization abstractions
- Next chapter illustrates how the ER model can be extended with
 - Type-subtype and set-subset relationships
 - Specialization/Generalization Hierarchies
 - Notation to display them in EER diagrams

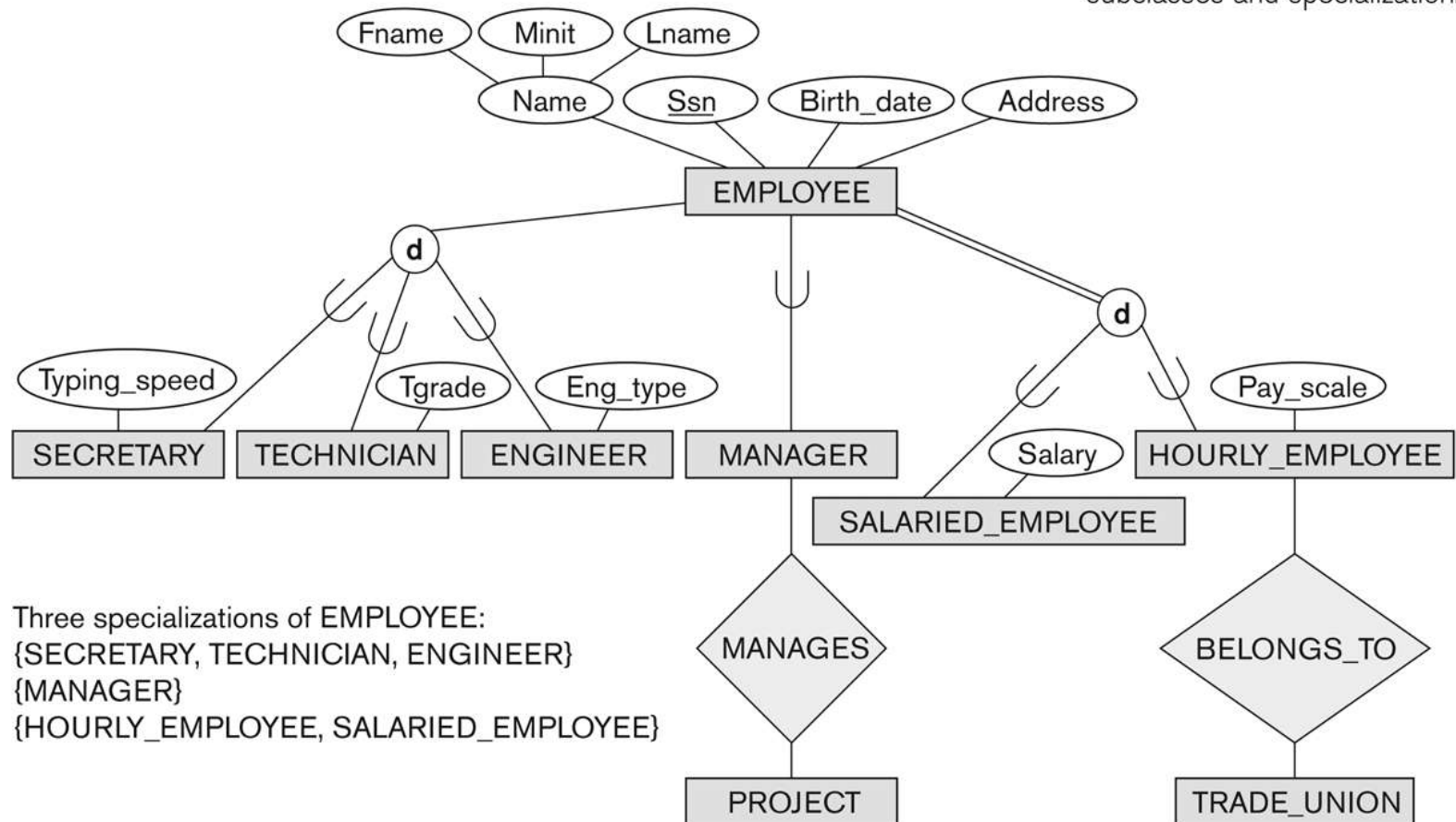
Subclasses and Superclasses (1)

- An entity type may have additional meaningful subgroupings of its entities
 - Example: EMPLOYEE may be further grouped into:
 - SECRETARY, ENGINEER, TECHNICIAN, ...
 - Based on the EMPLOYEE's Job
 - MANAGER
 - EMPLOYEES who are managers (the role they play)
 - SALARIED_EMPLOYEE, HOURLY_EMPLOYEE
 - Based on the EMPLOYEE's method of pay
- EER diagrams extend ER diagrams to represent these additional subgroupings, called *subclasses* or *subtypes*

Subclasses and Superclasses

Figure 4.1

EER diagram notation to represent subclasses and specialization.



Subclasses and Superclasses (2)

- Each of these subgroupings is a subset of EMPLOYEE entities
- Each is called a subclass of EMPLOYEE
- EMPLOYEE is the superclass for each of these subclasses
- These are called superclass/subclass relationships:
 - EMPLOYEE/SECRETARY
 - EMPLOYEE/TECHNICIAN
 - EMPLOYEE/MANAGER
 - ...

Subclasses and Superclasses (3)

- These are also called IS-A relationships
 - SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE,
- Note: An entity that is member of a subclass represents the same real-world entity as some member of the superclass:
 - The subclass member is the same entity in a *distinct specific role*
 - An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
 - A member of the superclass can be optionally included as a member of any number of its subclasses

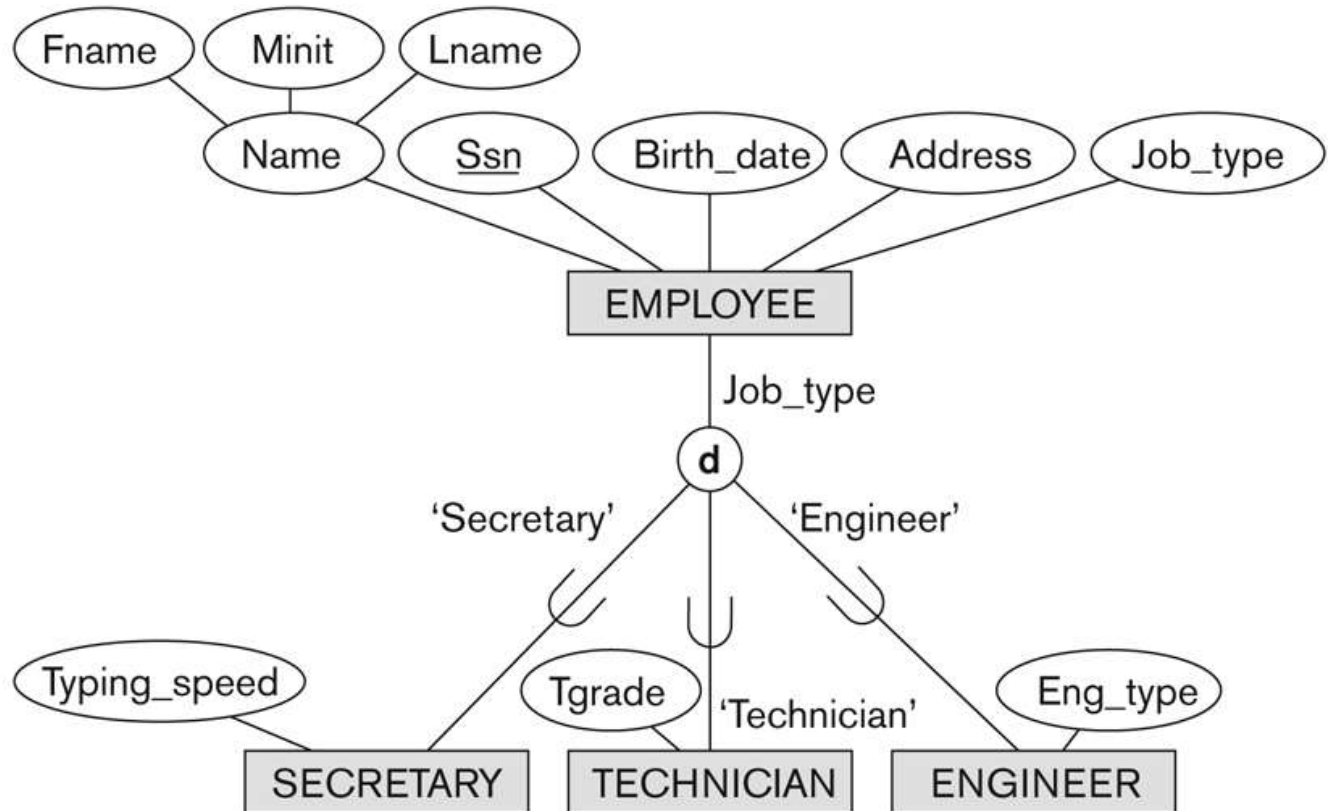
Subclasses and Superclasses (4)

- Examples:
 - A salaried employee who is also an engineer belongs to the two subclasses:
 - ENGINEER, and
 - SALARIED_EMPLOYEE
 - A salaried employee who is also an engineering manager belongs to the three subclasses:
 - MANAGER,
 - ENGINEER, and
 - SALARIED_EMPLOYEE
- It is not necessary that every entity in a superclass be a member of some subclass

Representing Specialization in EER Diagrams

Figure 4.4

EER diagram notation for an attribute-defined specialization on Job_type.



Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass *inherits*
 - All attributes of the entity as a member of the superclass
 - All relationships of the entity as a member of the superclass
- Example:
 - In the previous slide, SECRETARY (as well as TECHNICIAN and ENGINEER) inherit the attributes Name, SSN, ..., from EMPLOYEE
 - Every SECRETARY entity will have values for the inherited attributes

Specialization (1)

- Specialization is the process of defining a set of subclasses of a superclass
- The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass
 - Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type*.
 - Example: MANAGER *is a specialization of EMPLOYEE based on the role the employee plays*
 - May have several specializations of the same superclass

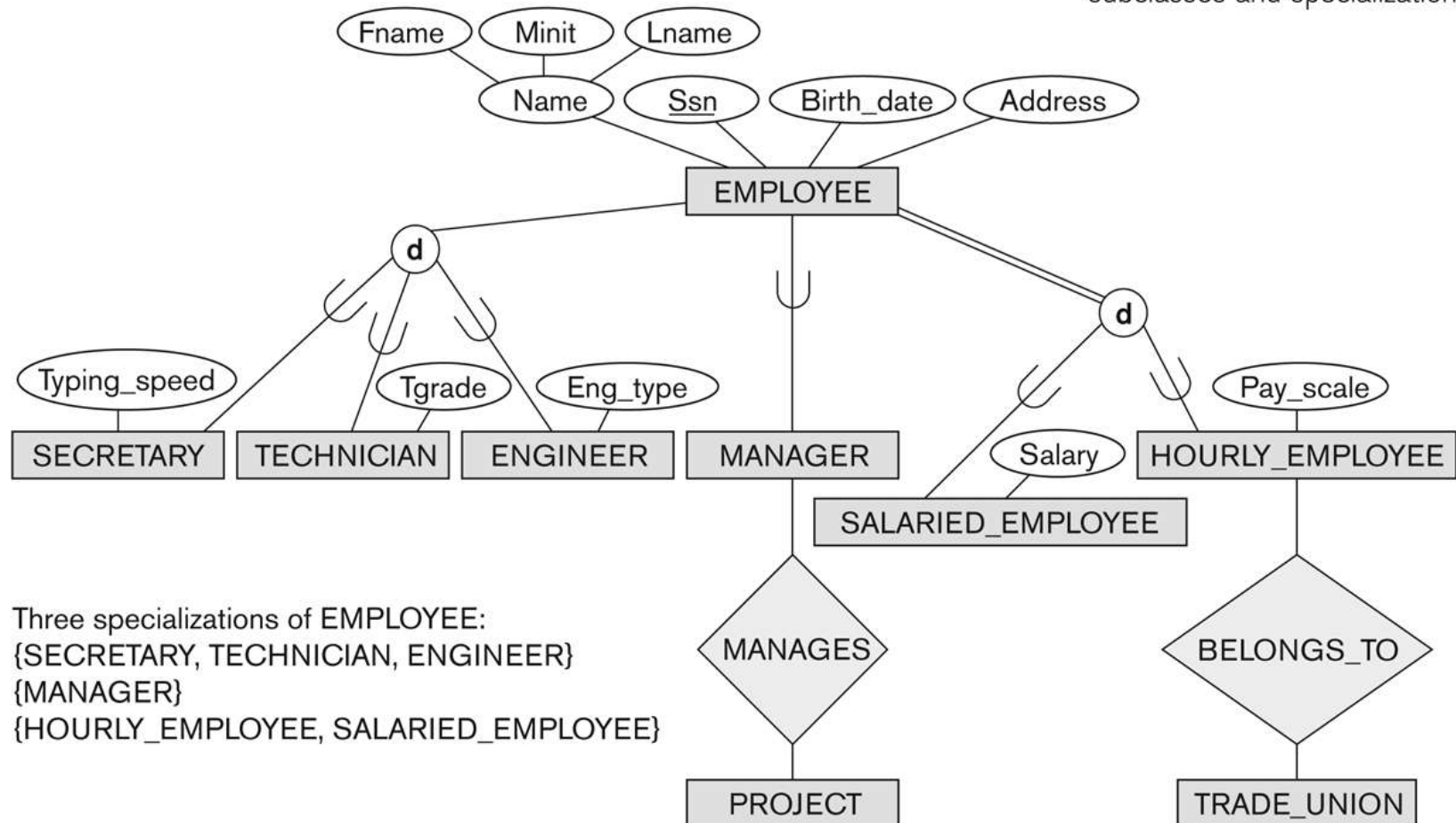
Specialization (2)

- Example: Another specialization of EMPLOYEE based on *method of pay* is {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}.
 - Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams
 - Attributes of a subclass are called *specific* or *local* attributes.
 - For example, the attribute TypingSpeed of SECRETARY
 - The subclass can also participate in specific relationship types.
 - For example, a relationship BELONGS_TO of HOURLY_EMPLOYEE

Specialization (3)

Figure 4.1

EER diagram notation to represent subclasses and specialization.



Generalization

- Generalization is the reverse of the specialization process
- Several classes with common features are generalized into a superclass;
 - original classes become its subclasses
- Example: CAR, TRUCK generalized into VEHICLE;
 - both CAR, TRUCK become subclasses of the superclass VEHICLE.
 - We can view {CAR, TRUCK} as a specialization of VEHICLE
 - Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK

Generalization (2)

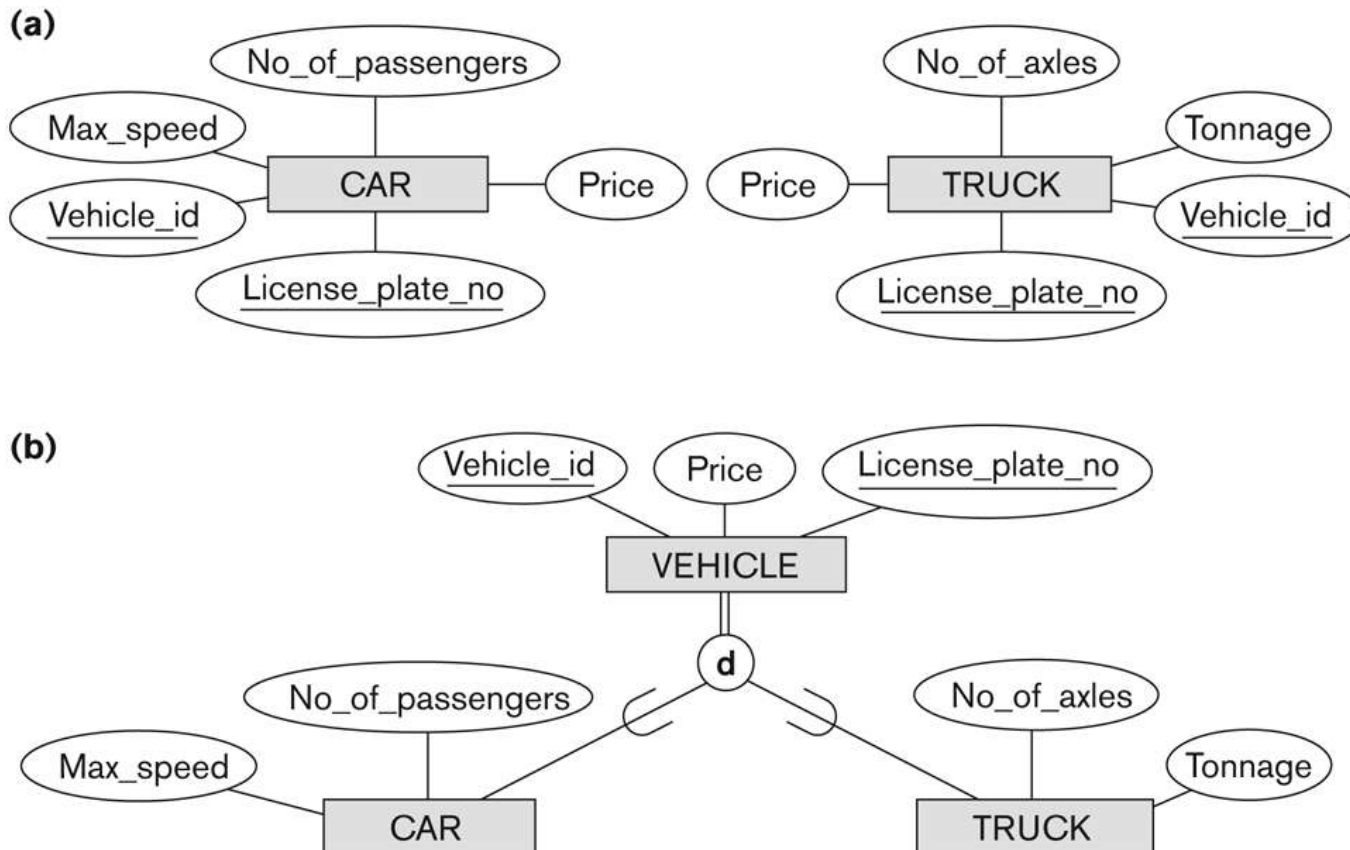


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Generalization and Specialization (1)

- Diagrammatic notations are sometimes used to distinguish between generalization and specialization
 - Arrow pointing to the generalized superclass represents a generalization
 - Arrows pointing to the specialized subclasses represent a specialization
 - We *do not use* this notation because it is often subjective as to which process is more appropriate for a particular situation
 - We advocate not drawing any arrows

Generalization and Specialization (2)

- Data Modeling with Specialization and Generalization
 - A superclass or subclass represents a collection (or set or grouping) of entities
 - It also represents a particular *type of entity*
 - Shown in rectangles in EER diagrams (as are entity types)
 - We can call all entity types (and their corresponding collections) **classes**, whether they are entity types, superclasses, or subclasses

Types of Specialization

- Predicate-defined (or condition-defined) : based on some predicate. E.g., based on value of an attribute, say, Job-type, or Age.
- Attribute-defined: shows the name of the attribute next to the line drawn from the superclass toward the subclasses (see Fig. 4.1)
- User-defined: membership is defined by the user on an entity by entity basis

Constraints on Specialization and Generalization (1)

- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses
 - Condition is a constraint that determines subclass members
 - Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its superclass

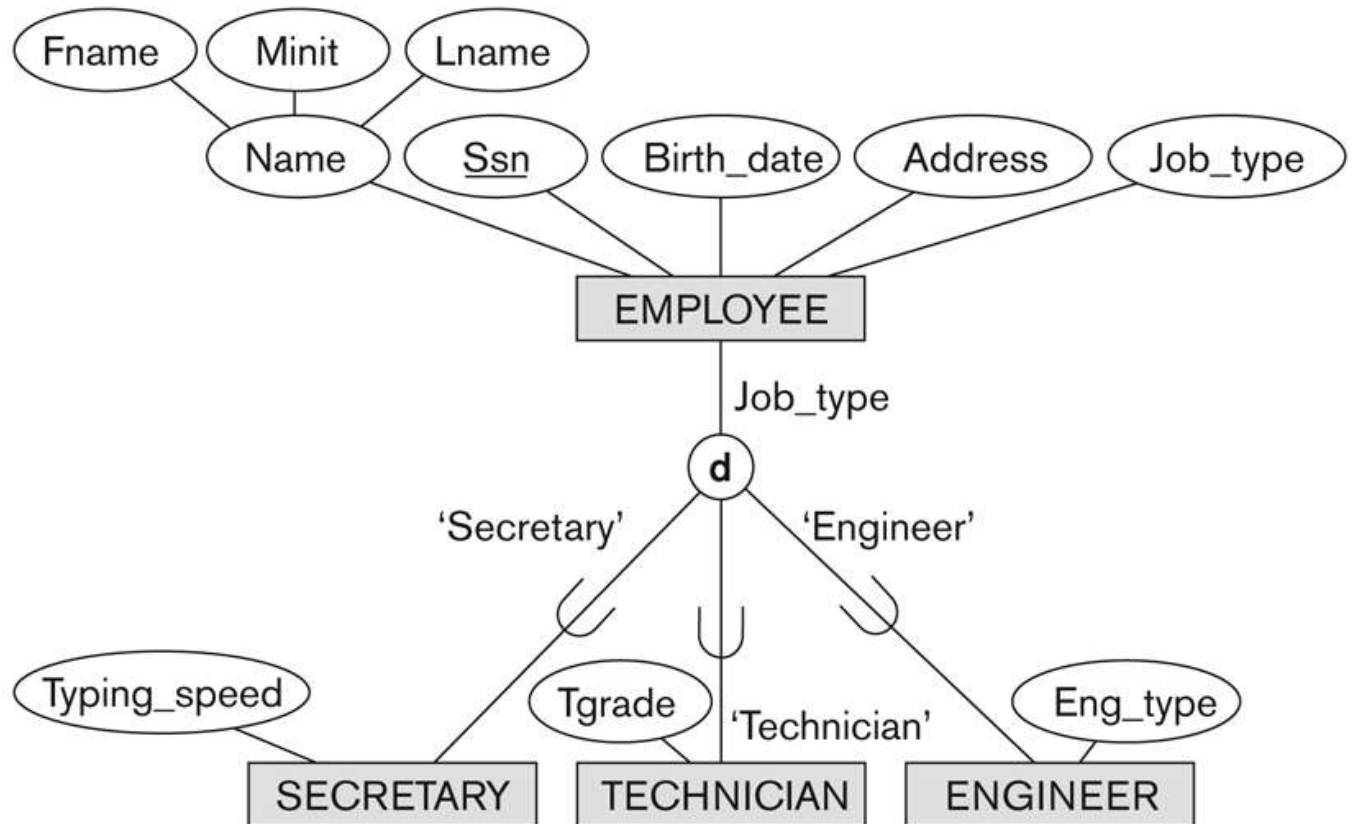
Constraints on Specialization and Generalization (2)

- If all subclasses in a specialization have membership condition on same attribute of the superclass, specialization is called an attribute-defined specialization
 - Attribute is called the defining attribute of the specialization
 - Example: JobType is the defining attribute of the specialization {SECRETARY, TECHNICIAN, ENGINEER} of EMPLOYEE
- If no condition determines membership, the subclass is called user-defined
 - Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
 - Membership in the subclass is specified individually for each entity in the superclass by the user

Displaying an attribute-defined specialization in EER diagrams

Figure 4.4

EER diagram notation for an attribute-defined specialization on Job_type.



Constraints on Specialization and Generalization (3)

- Two basic constraints can apply to a specialization/generalization:
 - Disjointness Constraint:
 - Completeness Constraint:

Constraints on Specialization and Generalization (4)

- Disjointness Constraint:
 - Specifies that the subclasses of the specialization must be *disjoint*:
 - an entity can be a member of at most one of the subclasses of the specialization
 - Specified by **d** in EER diagram
 - If not disjoint, specialization is *overlapping*:
 - that is the same entity may be a member of more than one subclass of the specialization
 - Specified by **o** in EER diagram

Constraints on Specialization and Generalization (5)

- Completeness (Exhaustiveness) Constraint:
 - *Total* specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
 - Shown in EER diagrams by a **double line**
 - *Partial* allows an entity not to belong to any of the subclasses
 - Shown in EER diagrams by a single line

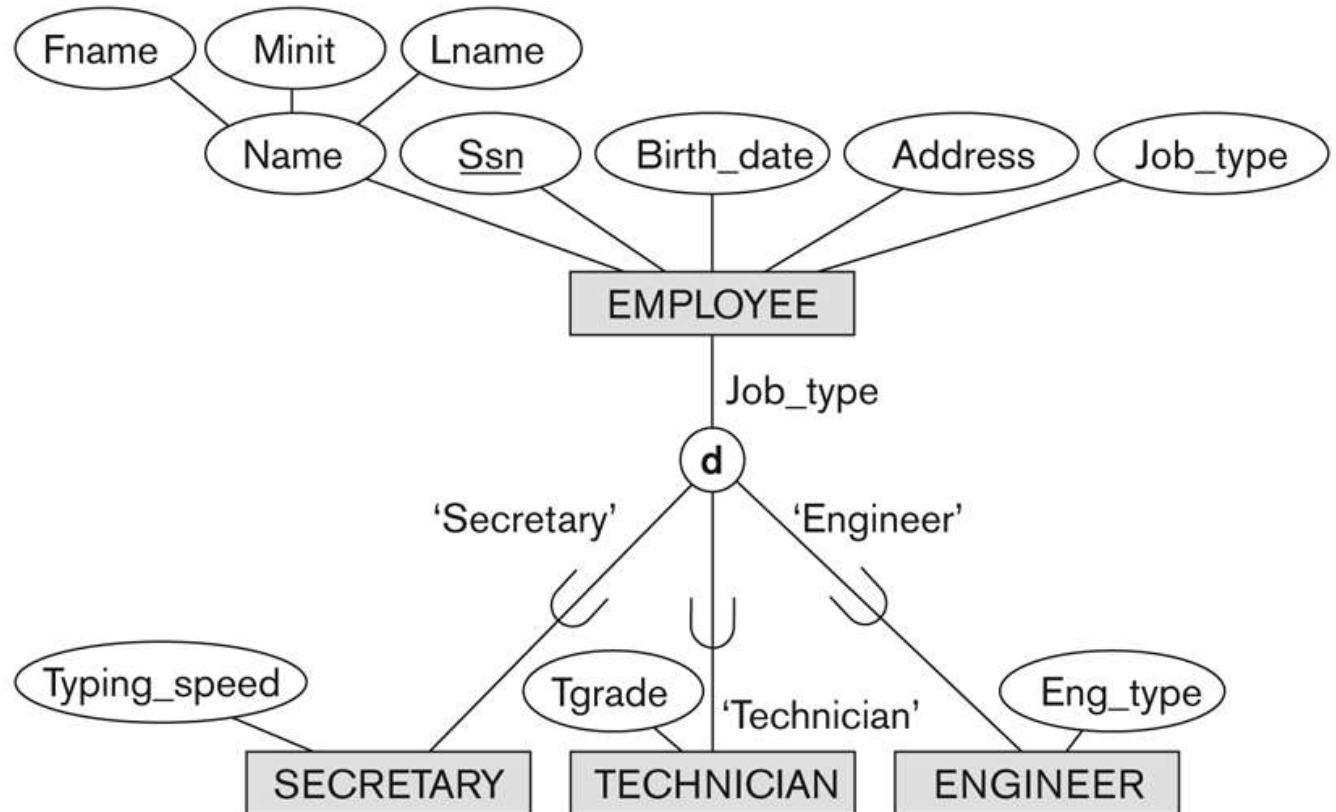
Constraints on Specialization and Generalization (6)

- Hence, we have four types of specialization/generalization:
 - Disjoint, total
 - Disjoint, partial
 - Overlapping, total
 - Overlapping, partial
- Note: Generalization usually is total because the superclass is derived from the subclasses.

Example of disjoint partial Specialization

Figure 4.4

EER diagram notation for an attribute-defined specialization on Job_type.



Example of overlapping total Specialization

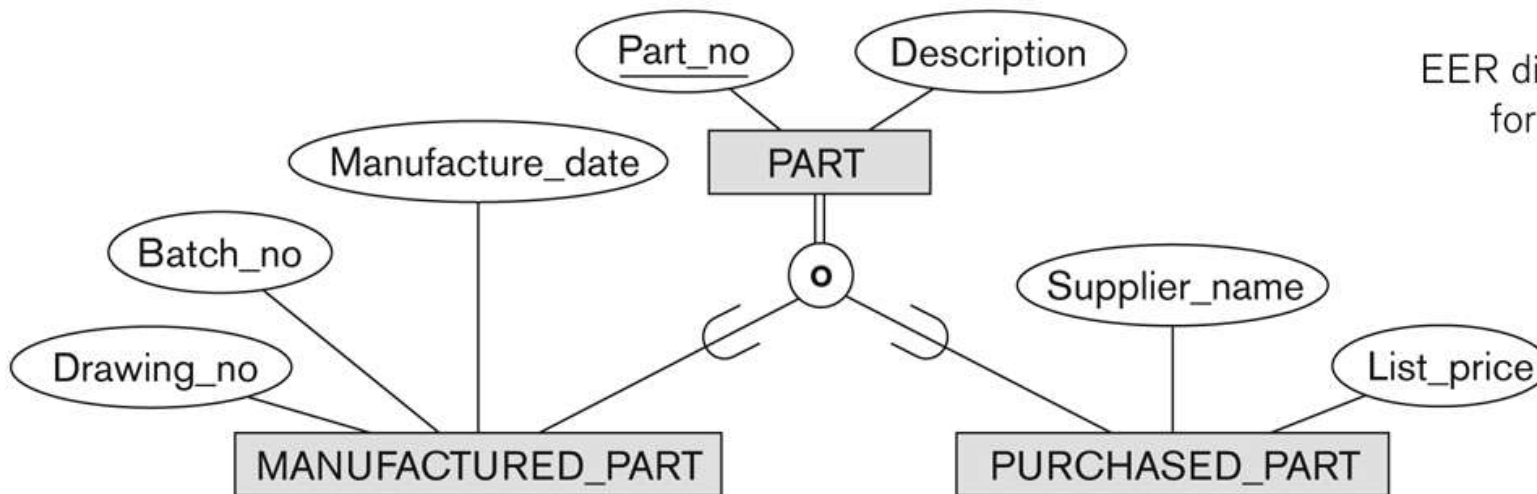


Figure 4.5
EER diagram notation
for an overlapping
(nondisjoint)
specialization.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (1)

- A subclass may itself have further subclasses specified on it
 - forms a hierarchy or a lattice
- **Hierarchy** has a constraint that every subclass has only one superclass (called **single inheritance**); this is basically a **tree structure**
- In a **lattice**, a subclass can be subclass of more than one superclass (called **multiple inheritance**)

Shared Subclass “Engineering_Manager”

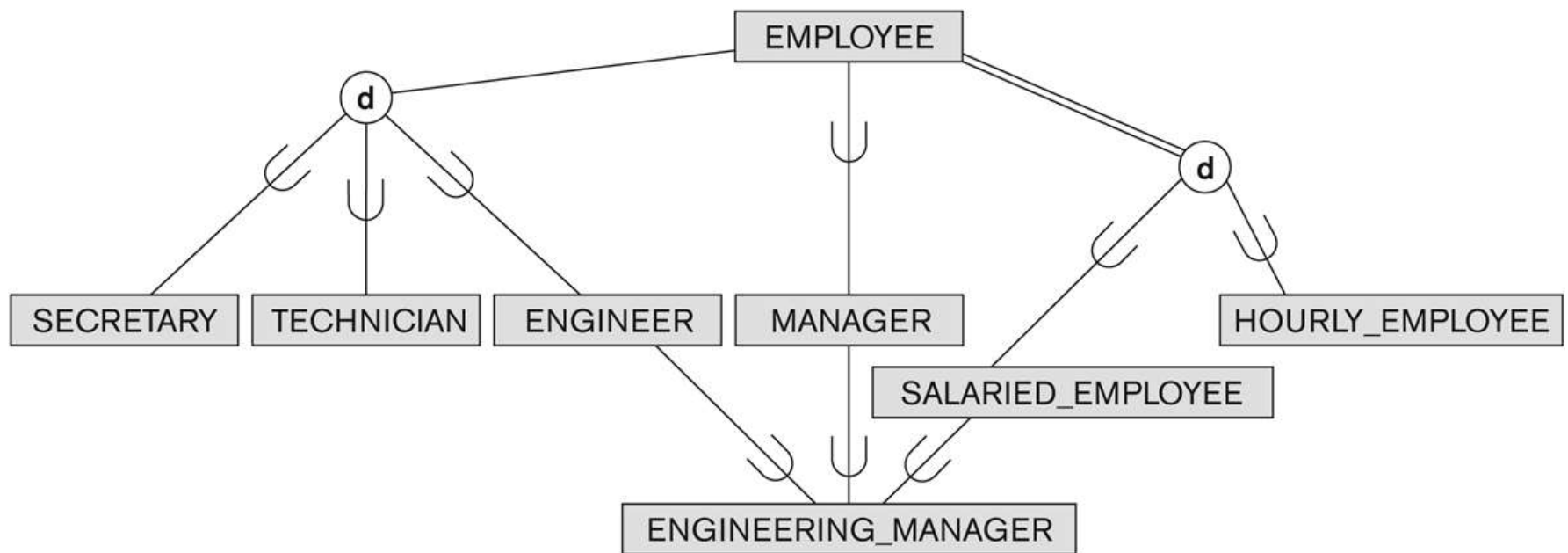


Figure 4.6

A specialization lattice with shared subclass ENGINEERING_MANAGER.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (2)

- In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses
- A subclass with more than one superclass is called a shared subclass (multiple inheritance)
- Can have:
 - *specialization* hierarchies or lattices, or
 - *generalization* hierarchies or lattices,
 - depending on how they were *derived*
- We just use *specialization* (to stand for the end result of either specialization or generalization)

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (3)

- In *specialization*, start with an entity type and then define subclasses of the entity type by successive specialization
 - called a *top down* conceptual refinement process
- In *generalization*, start with many entity types and generalize those that have common properties
 - Called a *bottom up* conceptual synthesis process
- In practice, a *combination of both processes* is usually employed

Specialization / Generalization Lattice Example (UNIVERSITY)

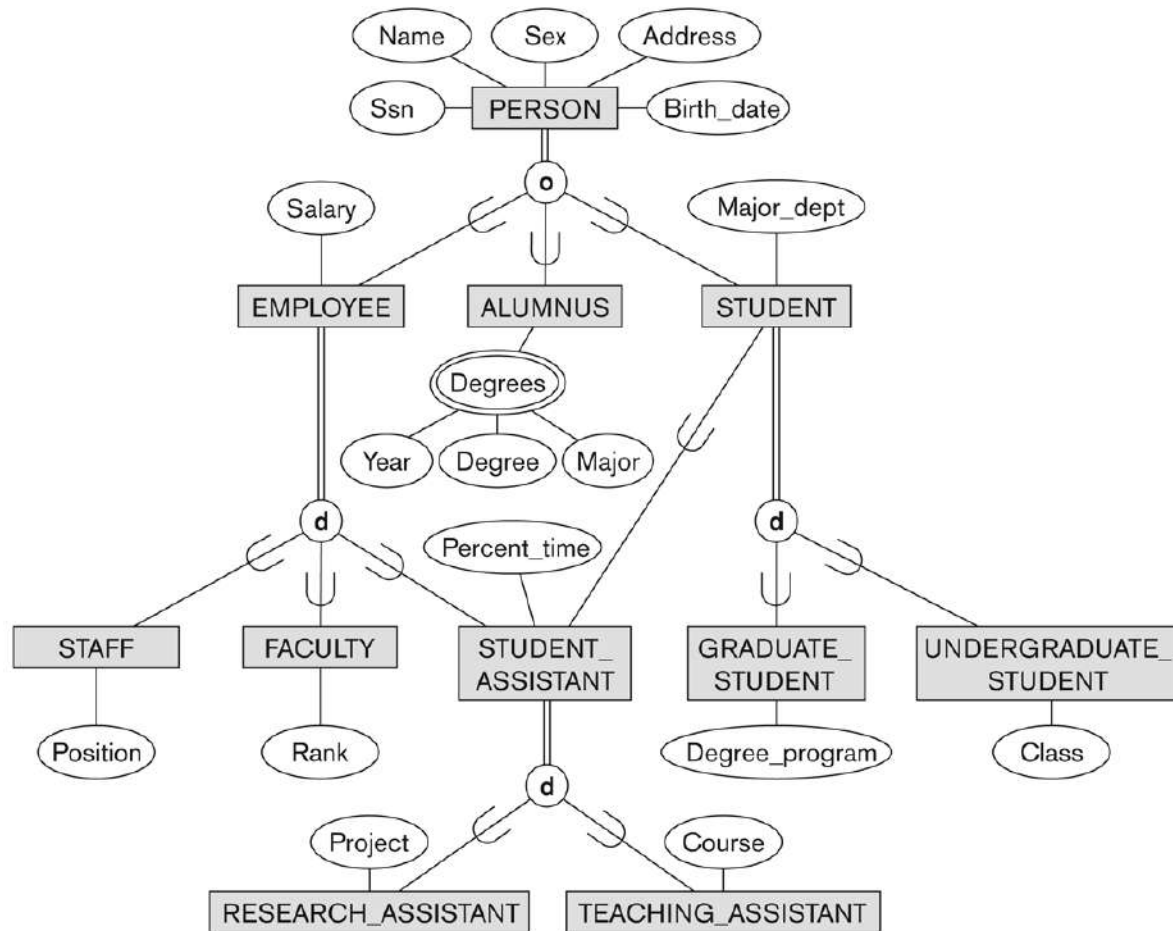


Figure 4.7

A specialization lattice with multiple inheritance for a UNIVERSITY database.

Categories (UNION TYPES) (1)

- All of the *superclass/subclass relationships* we have seen thus far have a single superclass
- A shared subclass is a subclass in:
 - *more than one* distinct superclass/subclass relationships
 - each relationships has a *single* superclass
 - shared subclass leads to multiple inheritance
- In some cases, we need to model a *single superclass/subclass relationship* with more than one superclass
- Superclasses can represent different entity types
- Such a subclass is called a category or UNION TYPE

Categories (UNION TYPES) (2)

- Example: In a database for vehicle registration, a vehicle owner can be a PERSON, a BANK (holding a lien on a vehicle) or a COMPANY.
 - A *category* (UNION type) called OWNER is created to represent a subset of the *union* of the three superclasses COMPANY, BANK, and PERSON
 - A category member must exist in ***at least one (typically just one)*** of its superclasses
- Difference from *shared subclass*, which is a:
 - subset of the *intersection* of its superclasses
 - shared subclass member must exist in ***all*** of its superclasses

Two categories (UNION types): OWNER, REGISTERED_VEHICLE

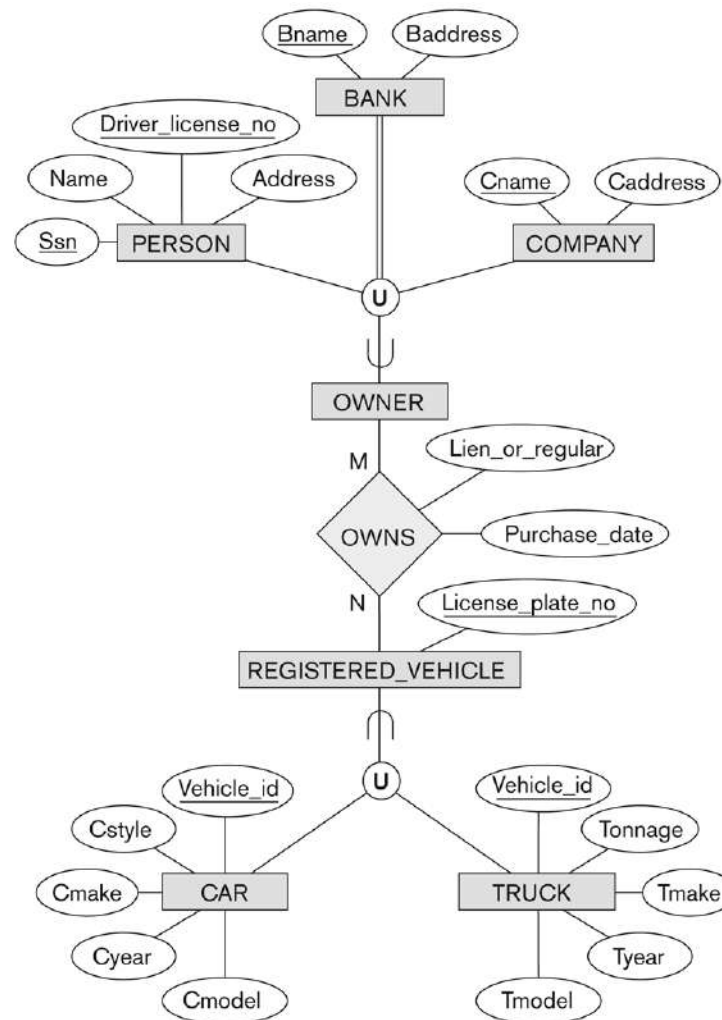


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Formal Definitions of EER Model (1)

- Class C:
 - A type of entity with a corresponding set of entities:
 - could be entity type, subclass, superclass, or category
- Note: The definition of *relationship type* in ER/EER should have 'entity type' replaced with 'class' to allow relationships among classes in general
- Subclass S is a class whose:
 - Type inherits all the attributes and relationship of a class C
 - Set of entities must always be a subset of the set of entities of the other class C
 - $S \subseteq C$
 - C is called the superclass of S
 - A superclass/subclass relationship exists between S and C

Formal Definitions of EER Model (2)

- Specialization Z: $Z = \{S_1, S_2, \dots, S_n\}$ is a set of subclasses with same superclass G; hence, G/S_i is a superclass relationship for $i = 1, \dots, n$.
 - G is called a generalization of the subclasses $\{S_1, S_2, \dots, S_n\}$
 - Z is total if we always have:
 - $S_1 \cup S_2 \cup \dots \cup S_n = G$;
 - Otherwise, Z is partial.
 - Z is disjoint if we always have:
 - $S_i \cap S_j$ empty-set for $i \neq j$;
 - Otherwise, Z is overlapping.

Formal Definitions of EER Model (3)

- Subclass S of C is predicate defined if predicate (condition) p on attributes of C is used to specify membership in S ;
 - that is, $S = C[p]$, where $C[p]$ is the set of entities in C that satisfy condition p
- A subclass not defined by a predicate is called user-defined
- Attribute-defined specialization: if a predicate $A = c_i$ (where A is an attribute of G and c_i is a constant value from the domain of A) is used to specify membership in each subclass S_i in Z
 - Note: If $c_i \neq c_j$ for $i \neq j$, and A is single-valued, then the attribute-defined specialization will be disjoint.

Formal Definitions of EER Model (4)

- Category or UNION type T
 - A class that is a subset of the *union* of n defining superclasses D_1, D_2, \dots, D_n , $n > 1$:
 - $T \subseteq (D_1 \cup D_2 \cup \dots \cup D_n)$
 - Can have a predicate p_i on the attributes of D_i to specify entities of D_i that are members of T.
 - If a predicate is specified on every D_i : $T = (D_1[p_1] \cup D_2[p_2] \cup \dots \cup D_n[p_n])$

Alternative diagrammatic notations

- ER/EER diagrams are a specific notation for displaying the concepts of the model diagrammatically
- DB design tools use many alternative notations for the same or similar concepts
- One popular alternative notation uses *UML class diagrams*
- see next slides for UML class diagrams and other alternative notations

UML Example for Displaying Specialization / Generalization

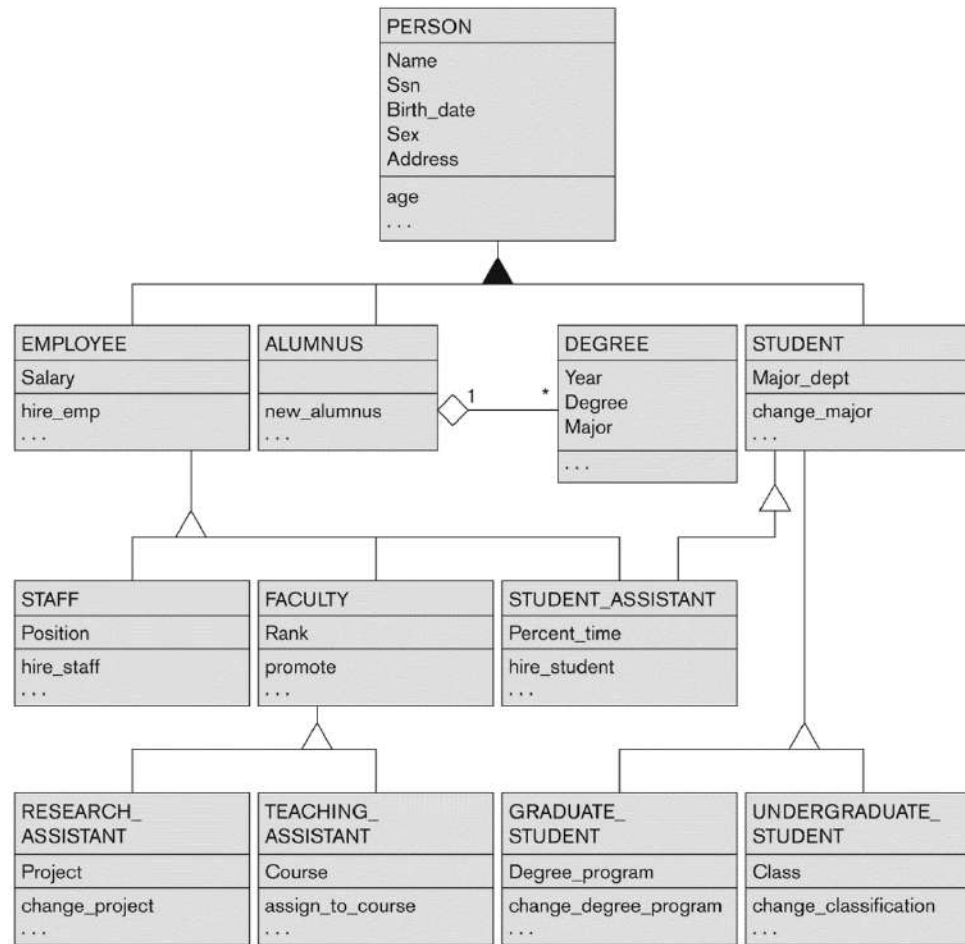


Figure 4.10

A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

Alternative Diagrammatic Notations

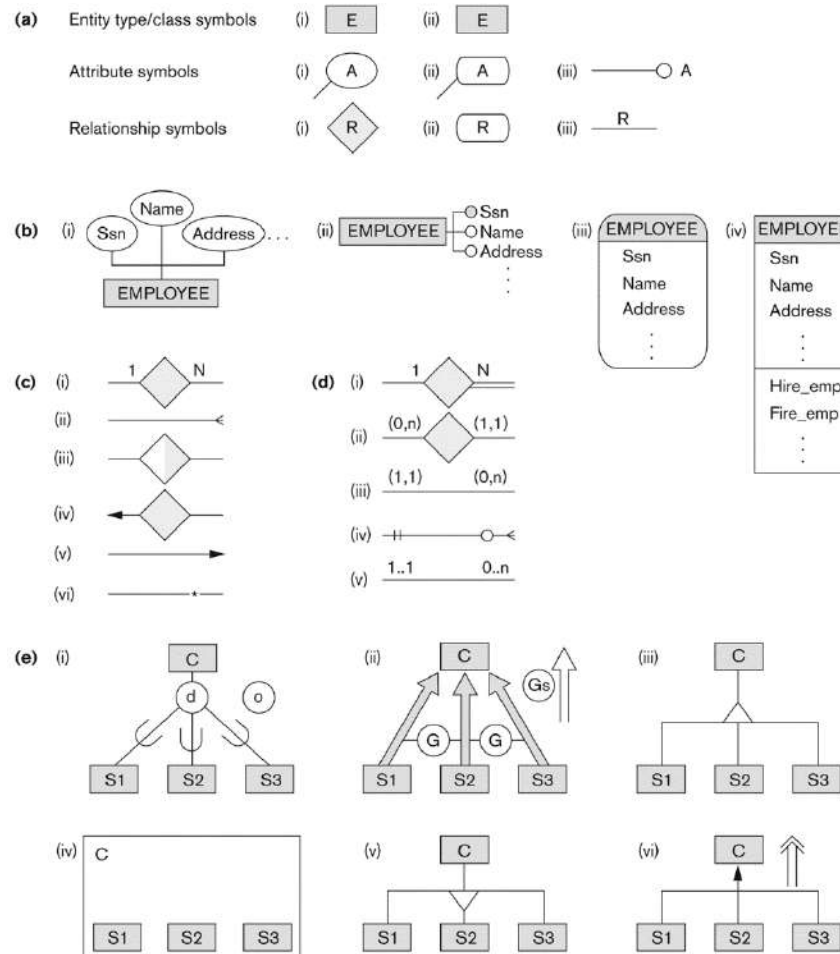


Figure A.1

Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

DBMS (DJ)

Knowledge Representation (KR)-1

- Deals with modeling and representing a certain domain of knowledge.
- Typically done by using some formal model of representation and by creating an Ontology
- An ontology for a specific domain of interest describes a set of concepts and interrelationships among those concepts
- An Ontology serves as a “schema” which enables interpretation of the knowledge in a “knowledge-base”

Knowledge Representation (KR)-2

COMMON FEATURES between KR and Data Models:

- Both use similar set of abstractions – classification, aggregation, generalization, and identification.
- Both provide concepts, relationships, constraints, operations and languages to represent knowledge and model data

DIFFERENCES:

- KR has broader scope: tries to deal with missing and incomplete knowledge, default and common-sense knowledge etc.

Knowledge Representation (KR)-3

DIFFERENCES (continued):

- KR schemes typically include rules and reasoning mechanisms for inferencing
- Most KR techniques involve data and metadata. In data modeling, these are treated separately
- KR is used in conjunction with artificial intelligence systems to do decision support applications

For more details on spatial, temporal and multimedia data modeling, see Chapter 26. For details on use of Ontologies see Sections 27.4.3 and 27.7.4.

General Basis for Conceptual Modeling

- TYPES OF DATA ABSTRACTIONS
 - CLASSIFICATION and INSTANTIATION
 - AGGREGATION and ASSOCIATION (relationships)
 - GENERALIZATION and SPECIALIZATION
 - IDENTIFICATION
- CONSTRAINTS
 - CARDINALITY (Min and Max)
 - COVERAGE (Total vs. Partial, and Exclusive (Disjoint) vs. Overlapping)

Ontologies

- Use conceptual modeling and other tools to develop “a specification of a conceptualization”
 - **Specification** refers to the language and vocabulary (data model concepts) used
 - **Conceptualization** refers to the description (schema) of the concepts of a particular field of knowledge and the relationships among these concepts
- Many medical, scientific, and engineering ontologies are being developed as a means of standardizing concepts and terminology

CONSTRAINTS

Constraints determine which values are permissible and which are not in the database.

They are of three main types:

1. **Inherent or Implicit Constraints:** These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
2. **Schema-based or Explicit Constraints:** They are expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
3. **Application based or semantic constraints:** These are beyond the expressive power of the model and must be specified and enforced by the application programs.

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of (explicit schema-based) constraints that can be expressed in the relational model:
 - **Key** constraints
 - **Entity integrity** constraints
 - **Referential integrity** constraints
- Another schema-based constraint is the **domain** constraint
 - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

Key Constraints

- **Superkey** of R:
 - Is a set of attributes SK of R with the following condition:
 - No two tuples in any valid relation state $r(R)$ will have the same value for SK
 - That is, for any distinct tuples $t1$ and $t2$ in $r(R)$, $t1[SK] \neq t2[SK]$
 - This condition must hold in *any valid state* $r(R)$
- **Key** of R:
 - A "minimal" superkey
 - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)
- A Key is a Superkey but not vice versa

Key Constraints (continued)

- Example: Consider the CAR relation schema:
 - CAR(State, Reg#, SerialNo, Make, Model, Year)
 - CAR has two keys:
 - Key1 = {State, Reg#}
 - Key2 = {SerialNo}
 - Both are also superkeys of CAR
 - {SerialNo, Make} is a superkey but *not* a key.
- In general:
 - Any *key* is a *superkey* (but not vice versa)
 - Any set of attributes that *includes a key* is a *superkey*
 - A *minimal* superkey is also a key

Key Constraints (continued)

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
 - The primary key attributes are underlined.
- Example: Consider the CAR relation schema:
 - CAR(State, Reg#, SerialNo, Make, Model, Year)
 - We chose SerialNo as the primary key
- The primary key value is used to *uniquely identify* each tuple in a relation
 - Provides the tuple identity
- Also used to *reference* the tuple from another tuple
 - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
 - Not always applicable – choice is sometimes subjective

CAR table with two candidate keys – LicenseNumber chosen as Primary Key

CAR

<u>License_number</u>	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.

Relational Database Schema

- **Relational Database Schema:**

- A set S of relation schemas that belong to the same database.
 - S is the name of the whole **database schema**
 - $S = \{R_1, R_2, \dots, R_n\}$ and a set IC of integrity constraints.
 - R_1, R_2, \dots, R_n are the names of the individual **relation schemas** within the database S
- Following slide shows a COMPANY database schema with 6 relation schemas

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

PROJECT

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

WORKS_ON

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

DEPENDENT

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

Figure 5.5
Schema diagram for
the COMPANY
relational database
schema.

Relational Database State

- 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 relational database *state* is sometimes called a relational database *snapshot* or *instance*.
- We will not use the term *instance* since it also applies to single tuples.
- A database state that does not meet the constraints is an invalid state

Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
 - INSERT a new tuple in a relation
 - DELETE an existing tuple from a relation
 - MODIFY an attribute of an existing tuple
- Next slide (Fig. 5.6) shows an example state for the COMPANY database schema shown in Fig. 5.5.

Populated database state for COMPANY

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

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

Entity Integrity

- **Entity Integrity:**

- The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of $r(R)$.
 - 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
- Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- 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

- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the **referenced relation** R2.
 - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if $t1[FK] = t2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity (or foreign key) Constraint

- Statement of the constraint
 - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
 - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
 - (2) a **null**.
- In case (2), the FK in R1 should **not** be a part of its own primary key.

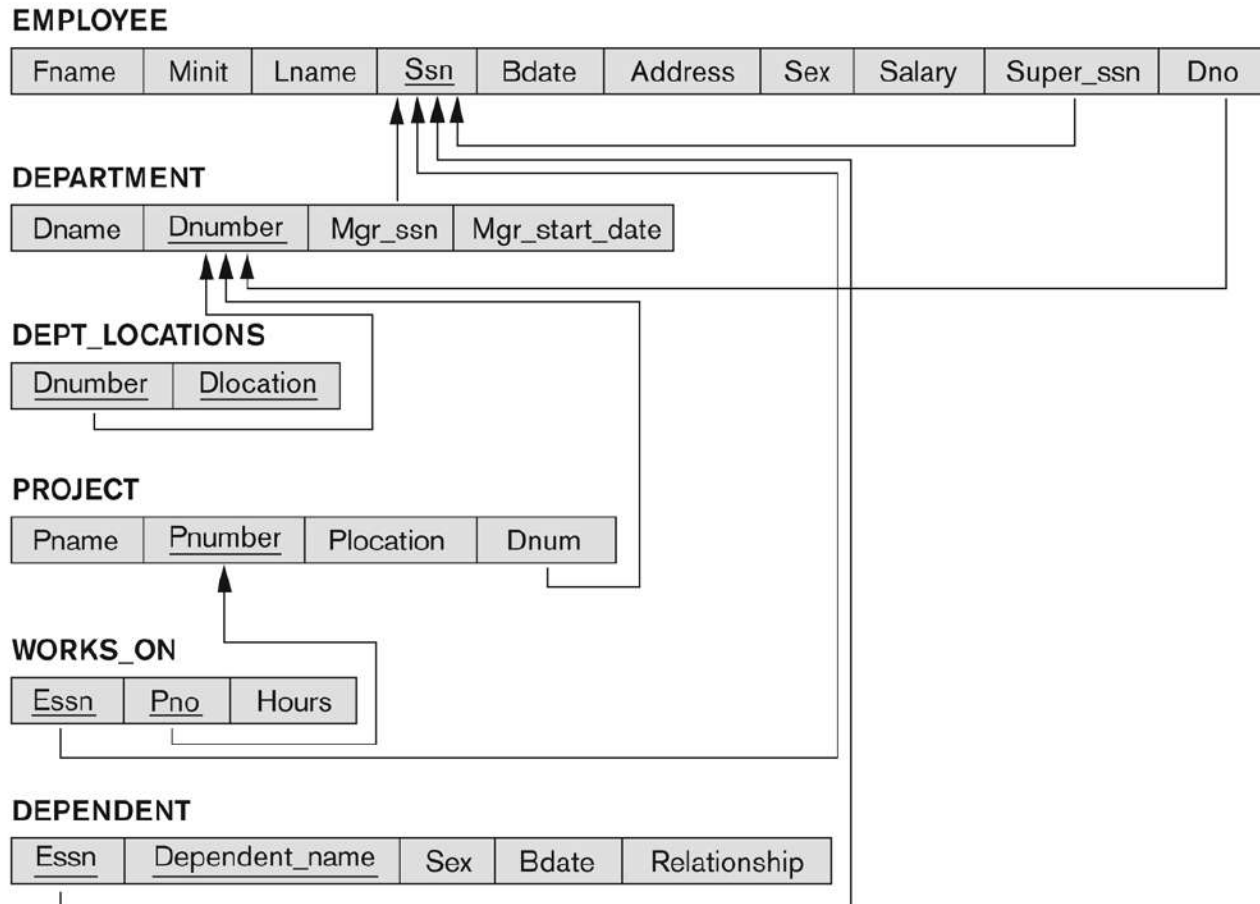
Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
 - Can also point the the primary key of the referenced relation for clarity
- Next slide shows the COMPANY **relational schema diagram with referential integrity constraints**

Referential Integrity Constraints for COMPANY database

Figure 5.7

Referential integrity constraints displayed on the COMPANY relational database schema.



Other Types of Constraints

- Semantic Integrity Constraints:
 - based on application semantics and cannot be expressed by the model per se
 - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A **constraint specification** language may have to be used to express these
- SQL-99 allows **CREATE TRIGGER** and **CREATE ASSERTION** to express some of these semantic constraints
- Keys, Permissibility of Null values, Candidate Keys (Unique in SQL), Foreign Keys, Referential Integrity etc. are expressed by the **CREATE TABLE** statement in SQL.

Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
 - Cancel the operation that causes the violation (RESTRICT or REJECT option)
 - Perform the operation but inform the user of the violation
 - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
 - Execute a user-specified error-correction routine

Possible violations for each operation

- INSERT may violate any of the constraints:
 - Domain constraint:
 - if one of the attribute values provided for the new tuple is not of the specified attribute domain
 - Key constraint:
 - if the value of a key attribute in the new tuple already exists in another tuple in the relation
 - Referential integrity:
 - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
 - Entity integrity:
 - if the primary key value is null in the new tuple

Possible violations for each operation

- DELETE may violate only referential integrity:
 - If the primary key value of the tuple being deleted is referenced from other tuples in the database
 - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
 - RESTRICT option: reject the deletion
 - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
 - SET NULL option: set the foreign keys of the referencing tuples to NULL
 - One of the above options must be specified during database design for each foreign key constraint

Possible violations for each operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- 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
 - Updating an ordinary attribute (neither PK nor FK):
 - Can only violate domain constraints

In-Class Exercise

(Taken from Exercise 5.15)

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)

Draw a relational schema diagram specifying the foreign keys for this schema.

Semantics of the Relational Attributes must be clear

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
 - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
 - Only foreign keys should be used to refer to other entities
 - Entity and relationship attributes should be kept apart as much as possible.
- Bottom Line: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

Figure 14.1 A simplified COMPANY relational database schema

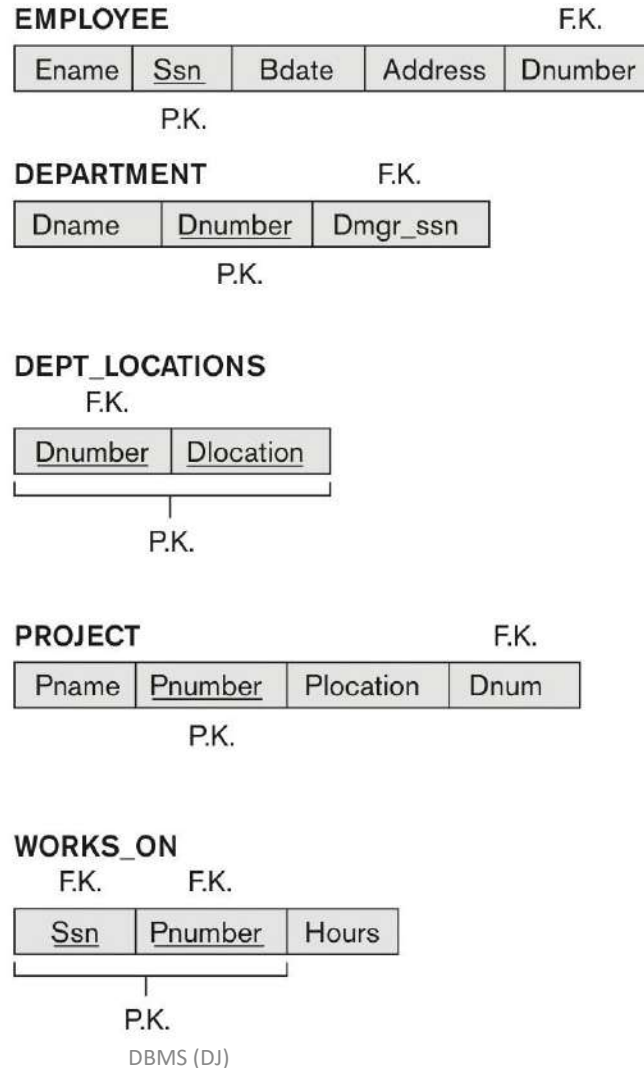


Figure 14.1 A simplified COMPANY relational database schema.

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
 - Wastes storage
 - Causes problems with update anomalies
 - Insertion anomalies
 - Deletion anomalies
 - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY

- Consider the relation:
 - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
 - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

EXAMPLE OF AN INSERT ANOMALY

- Consider the relation:
 - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert Anomaly:
 - Cannot insert a project unless an employee is assigned to it.
- Conversely
 - Cannot insert an employee unless an he/she is assigned to a project.

EXAMPLE OF A DELETE ANOMALY

- Consider the relation:
 - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
 - When a project is deleted, it will result in deleting all the employees who work on that project.
 - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

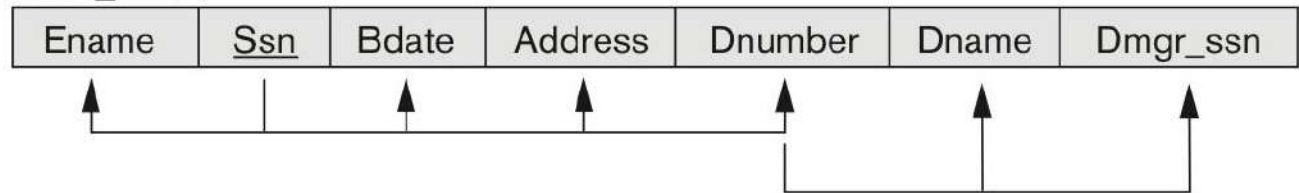
Figure 14.3 Two relation schemas suffering from update anomalies

Figure 14.3

Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

(a)

EMP_DEPT



(b)

EMP_PROJ

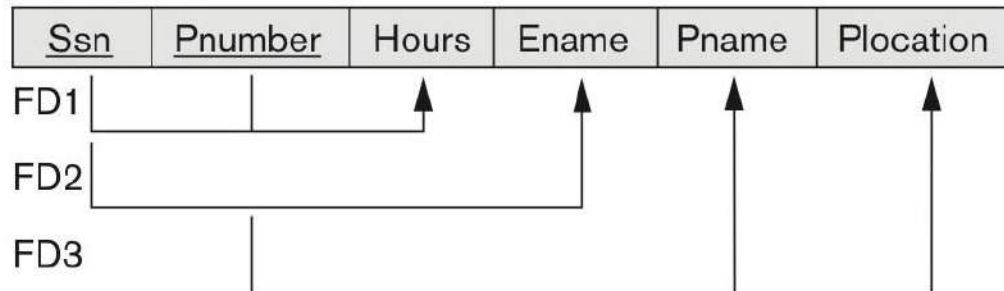


Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ

Figure 14.4

Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

Redundancy

EMP_DEPT

<u>Ename</u>	<u>Ssn</u>	Bdate	Address	Dnumber	Dname	Dmgr_ssn
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 Redundancy

EMP_PROJ

<u>Ssn</u>	<u>Pnumber</u>	Hours	<u>Ename</u>	<u>Pname</u>	<u>Plocation</u>
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Guideline for Redundant Information in Tuples and Update Anomalies

- **GUIDELINE 2:**
 - Design a schema that does not suffer from the insertion, deletion and update anomalies.
 - If there are any anomalies present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

- GUIDELINE 3:
 - Relations should be designed such that their tuples will have as few NULL values as possible
 - Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- Reasons for nulls:
 - Attribute not applicable or invalid
 - Attribute value unknown (may exist)
 - Value known to exist, but unavailable

1.4 Generation of Spurious Tuples – avoid at any cost

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations
- GUIDELINE 4:
 - The relations should be designed to satisfy the lossless join condition.
 - No spurious tuples should be generated by doing a natural-join of any relations.

Spurious Tuples (2)

- There are two important properties of decompositions:
 - a) Non-additive or losslessness of the corresponding join
 - b) Preservation of the functional dependencies.
- Note that:
 - Property (a) is extremely important and cannot be sacrificed.
 - Property (b) is less stringent and may be sacrificed. (See Chapter 15).

2. Functional Dependencies

- Functional dependencies (FDs)
 - Are used to specify *formal measures* of the "goodness" of relational designs
 - And keys are used to define **normal forms** for relations
 - 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

2.1 Defining Functional Dependencies

- $X \rightarrow Y$ holds if whenever two tuples have the same value for X , they *must have* the same value for Y
 - For any two tuples t_1 and t_2 in any relation instance $r(R)$: If $t_1[X]=t_2[X]$, *then* $t_1[Y]=t_2[Y]$
- $X \rightarrow Y$ in R specifies a *constraint* on all relation instances $r(R)$
- Written as $X \rightarrow Y$; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Examples of FD constraints (1)

- Social security number determines employee name
 - $SSN \rightarrow ENAME$
- Project number determines project name and location
 - $PNUMBER \rightarrow \{PNAME, PLOCATION\}$
- Employee ssn and project number determines the hours per week that the employee works on the project
 - $\{SSN, PNUMBER\} \rightarrow HOURS$

Examples of FD constraints (2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance $r(R)$
- If K is a key of R , then K functionally determines all attributes in R
 - (since we never have two distinct tuples with $t_1[K]=t_2[K]$)

Defining FDs from instances

- Note that in order to define the FDs, we need to understand the meaning of the attributes involved and the relationship between them.
- An FD is a property of the attributes in the schema R
- Given the instance (population) of a relation, all we can conclude is that an FD may exist between certain attributes.
- What we can definitely conclude is – that certain FDs do not exist because there are tuples that show a violation of those dependencies.

Figure 14.7 Ruling Out FDs

Note that given the state of the TEACH relation, we can say that the FD: Text \rightarrow Course may exist. However, the FDs Teacher \rightarrow Course, Teacher \rightarrow Text and Course \rightarrow Text are ruled out.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.8 What FDs may exist?

- A relation $R(A, B, C, D)$ with its extension.
- Which FDs may exist in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Figure 14.9 Normalization into 1NF

properties: Atomic (single) Values in Each column. Unique Column Names to avoid ambiguity and confusion. Unique Rows having a primary key column

(a)

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	Dlocations

Figure 14.9

Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

(b)

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	<u>Dlocation</u>
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 14.10 Normalizing nested relations into 1NF

(a)

EMP_PROJ		Projs	
Ssn	Ename	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

EMP_PROJ2		
Ssn	Pnumber	Hours

Figure 14.10

Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a nested relation attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

3.5 Second Normal Form (1)

- Uses the concepts of **FDs, primary key**
- Definitions
 - **Prime attribute:** An attribute that is member of the primary key K
 - **Full functional dependency:** a FD $Y \rightarrow Z$ where removal of any attribute from Y means the FD does not hold any more
- Examples:
 - $\{SSN, PNUMBER\} \rightarrow HOURS$ is a full FD since neither $SSN \rightarrow HOURS$ nor $PNUMBER \rightarrow HOURS$ hold
 - $\{SSN, PNUMBER\} \rightarrow ENAME$ is not a full FD (it is called a partial dependency) since $SSN \rightarrow ENAME$ also holds

Second Normal Form (2)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key
- R can be decomposed into 2NF relations via the process of 2NF normalization or “second normalization”

Figure 14.11 Normalizing into 2NF and 3NF

eliminate redundant data and improve data integrity. Removal of partial dependencies: 1NF + All non-key attributes (columns) must be functionally dependent on the entire primary key. There should be no partial dependencies, meaning no attribute is dependent on only a portion of the primary key.

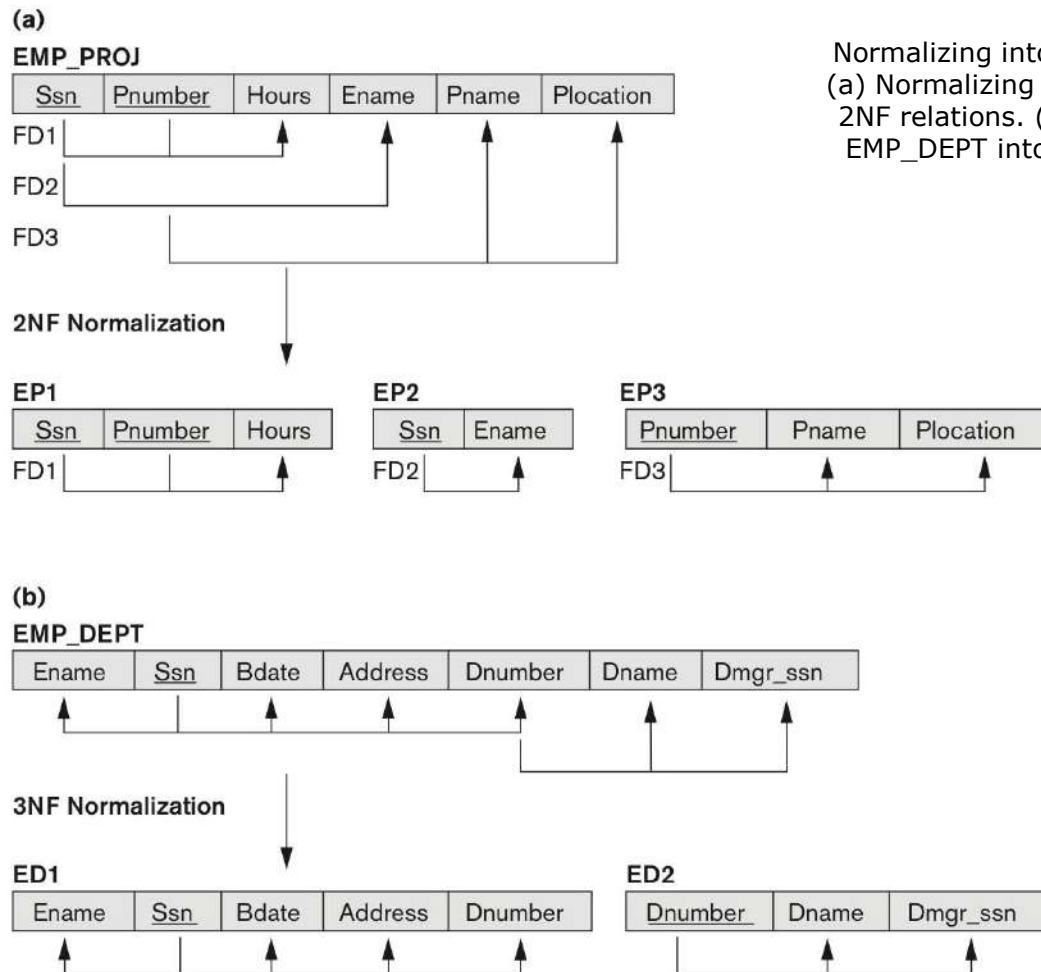
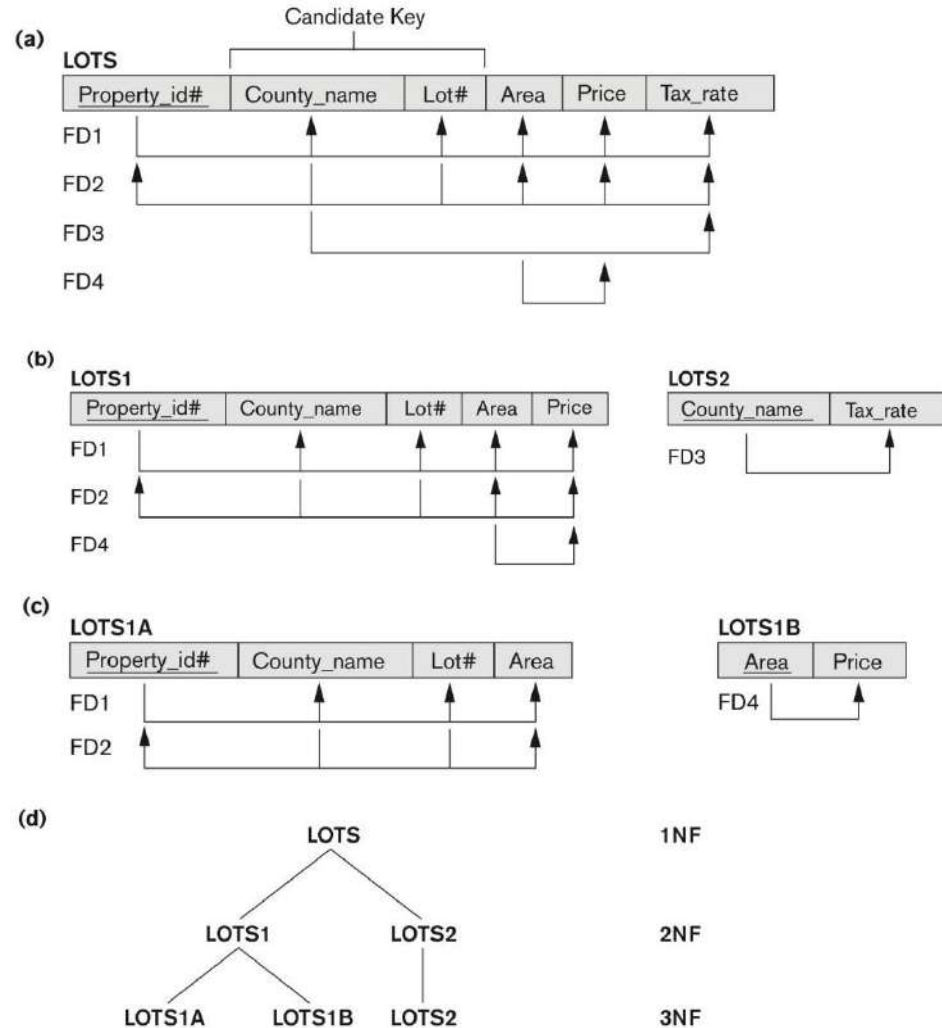


Figure 14.11
Normalizing into 2NF and 3NF.
(a) Normalizing EMP_PROJ into 2NF relations.
(b) Normalizing EMP_DEPT into 3NF relations.

Figure 14.12 Normalization into 2NF and 3NF

Figure 14.12

Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.



3.6 Third Normal Form (1)

eliminate redundant data and improve data integrity. Removal of transitive dependencies: A table should not have any transitive dependencies, where non-key attributes depend on other non-key attributes.

- Definition:
 - **Transitive functional dependency:** a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$
- Examples:
 - $SSN \rightarrow DMGRSSN$ is a **transitive** FD
 - Since $SSN \rightarrow DNUMBER$ and $DNUMBER \rightarrow DMGRSSN$ hold
 - $SSN \rightarrow ENAME$ is **non-transitive**
 - Since there is no set of attributes X where $SSN \rightarrow X$ and $X \rightarrow ENAME$

Third Normal Form (2)

- A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key
- R can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
 - In $X \rightarrow Y$ and $Y \rightarrow Z$, with X as the primary key, we consider this a problem only if Y is not a candidate key.
 - When Y is a candidate key, there is no problem with the transitive dependency .
 - E.g., Consider EMP (SSN, Emp#, Salary).
 - Here, $SSN \rightarrow Emp\# \rightarrow Salary$ and Emp# is a candidate key.

Normal Forms Defined Informally

- 1st normal form
 - All attributes depend on **the key**
- 2nd normal form
 - All attributes depend on **the whole key**
- 3rd normal form
 - All attributes depend on **nothing but the key**

4. General Normal Form Definitions (For Multiple Keys) (1)

- The above definitions consider the primary key only
- The following more general definitions take into account relations with multiple candidate keys
- Any attribute involved in a candidate key is a *prime attribute*
- All other attributes are called *non-prime attributes*.

4.1 General Definition of 2NF (For Multiple Candidate Keys)

- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on *every* key of R
- In Figure 14.12 the FD
County_name \rightarrow Tax_rate violates 2NF.

So second normalization converts LOTS into

LOTS1 (Property_id#, County_name, Lot#, Area, Price)

LOTS2 (County_name, Tax_rate)

4.2 General Definition of Third Normal Form

- Definition:
 - **Superkey** of relation schema R - a set of attributes S of R that contains a key of R
 - A relation schema R is in **third normal form (3NF)** if whenever a FD $X \rightarrow A$ holds in R, then either:
 - (a) X is a superkey of R, or
 - (b) A is a prime attribute of R
- LOTS1 relation violates 3NF because
Area \rightarrow Price ; and Area is not a superkey in LOTS1. (see Figure 14.12).

4.3 Interpreting the General Definition of Third Normal Form

- Consider the 2 conditions in the Definition of 3NF:

A relation schema R is in **third normal form (3NF)** if whenever a FD $X \rightarrow A$ holds in R, then either:

- (a) X is a superkey of R, or
- (b) A is a prime attribute of R

- Condition (a) catches two types of violations :

- one where a prime attribute functionally determines a non-prime attribute. This catches 2NF violations due to non-full functional dependencies.

-second, where a non-prime attribute functionally determines a non-prime attribute. This catches 3NF violations due to a transitive dependency.

4.3 Interpreting the General Definition of Third Normal Form (2)

- **ALTERNATIVE DEFINITION of 3NF:** We can restate the definition as:

A relation schema R is in **third normal form (3NF)** if every non-prime attribute in R meets both of these conditions:

- It is fully functionally dependent on every key of R
- It is non-transitively dependent on every key of R

Note that stated this way, a relation in 3NF also meets the requirements for 2NF.

- The condition (b) from the last slide takes care of the dependencies that “slip through” (are allowable to) 3NF but are “caught by” BCNF which we discuss next.

5. BCNF (Boyce-Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD $X \rightarrow A$** holds in R, then **X is a superkey** of R
- Each normal form is strictly stronger than the previous one
 - Every 2NF relation is in 1NF
 - Every 3NF relation is in 2NF
 - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- Hence BCNF is considered a **stronger form of 3NF**
- The goal is to have each relation in BCNF (or 3NF)

Figure 14.13 Boyce-Codd normal form

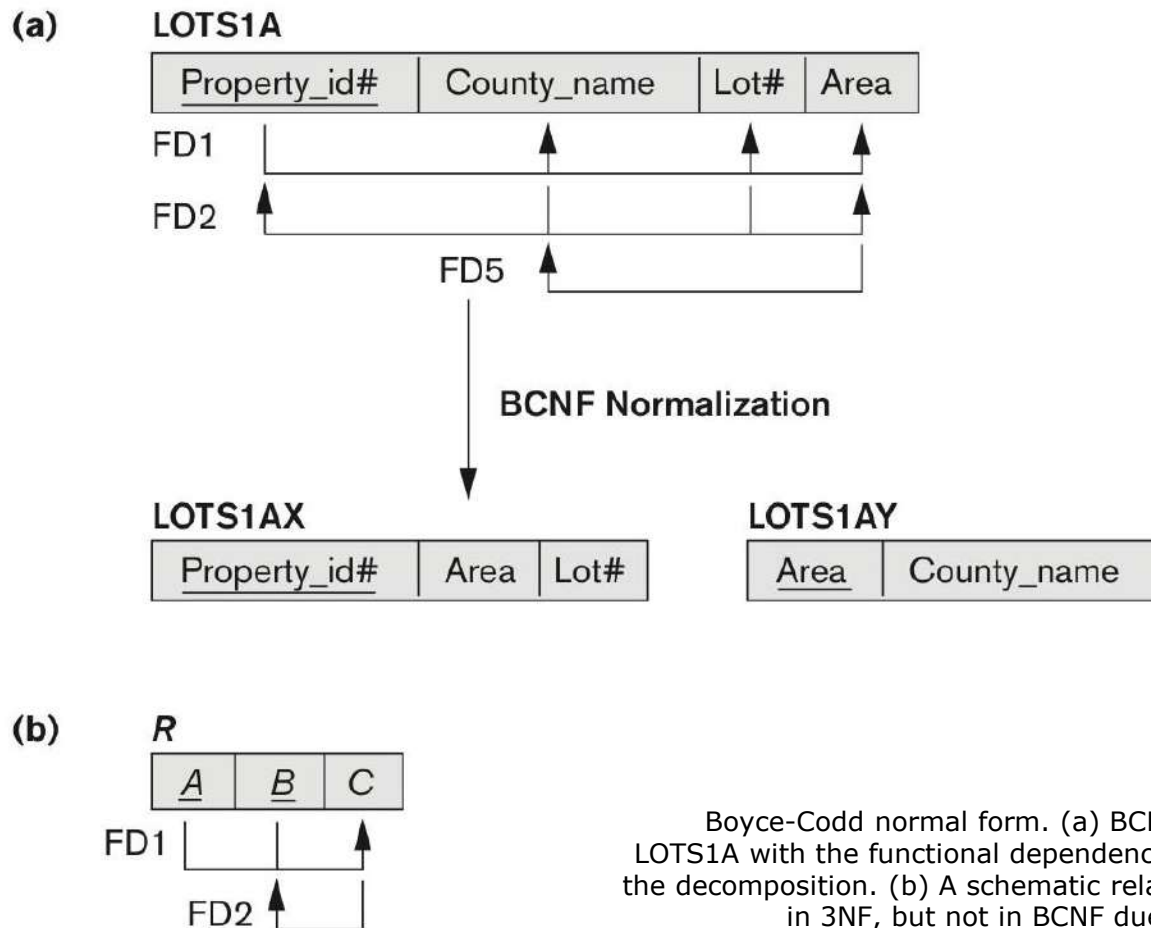


Figure 14.13
 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. $C \rightarrow B$.

Figure 14.14 A relation TEACH that is in 3NF but not in BCNF

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 14.14
A relation TEACH that is in 3NF
but not BCNF.

Achieving the BCNF by Decomposition (1)

- Two FDs exist in the relation TEACH:
 - fd1: { student, course} \rightarrow instructor
 - fd2: instructor \rightarrow course
- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 14.13 (b).
 - So this relation is in 3NF *but not in* BCNF
- A relation **NOT** in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.
 - (See Algorithm 15.3)

Achieving the BCNF by Decomposition (2)

- Three possible decompositions for relation TEACH
 - D1: {student, instructor} and {student, course}
 - D2: {course, instructor} and {course, student}
 - D3: {instructor, course} and {instructor, student} ✓
- All three decompositions will lose fd1.
 - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).
- A test to determine whether a binary decomposition (decomposition into two relations) is non-additive (lossless) is discussed under Property NJB on the next slide. We then show how the third decomposition above meets the property.

Test for checking non-additivity of Binary Relational Decompositions

- **Testing Binary Decompositions for Lossless Join (Non-additive Join) Property**
 - **Binary Decomposition:** Decomposition of a relation R into two relations.
 - **PROPERTY NJB (non-additive join test for binary decompositions):** A decomposition $D = \{R_1, R_2\}$ of R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
 - The f.d. $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+ , or
 - The f.d. $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+ .

Test for checking non-additivity of Binary Relational Decompositions

If you apply the NJB test to the 3 decompositions of the TEACH relation:

- D1 gives **Student** \rightarrow Instructor or **Student** \rightarrow Course, none of which is true.
- D2 gives **Course** \rightarrow Instructor or **Course** \rightarrow Student, none of which is true.
- However, in D3 we get **Instructor** \rightarrow Course or **Instructor** \rightarrow Student.

Since **Instructor** \rightarrow Course is indeed true, the NJB property is satisfied and D3 is determined as a non-additive (good) decomposition.

General Procedure for achieving BCNF when a relation fails BCNF

Here we make use the algorithm from Chapter 15 (Algorithm 15.5):

- Let R be the relation not in BCNF, let X be a subset-of R , and let $X \rightarrow A$ be the FD that causes a violation of BCNF. Then R may be decomposed into two relations:
- (i) $R - A$ and (ii) $X \cup A$.
- If either $R - A$ or $X \cup A$ is not in BCNF, repeat the process.

Note that the f.d. that violated BCNF in TEACH was $\text{Instructor} \rightarrow \text{Course}$. Hence its BCNF decomposition would be :

(TEACH – COURSE) and (Instructor \cup Course), which gives

the relations: (Instructor, Student) and (Instructor, Course) that we obtained before in decomposition D3.

5. Multivalued Dependencies and Fourth Normal Form (1)

Definition:

- A **multivalued dependency (MVD)** $X \twoheadrightarrow Y$ specified on relation schema R , where X and Y are both subsets of R , specifies the following constraint on any relation state r of R : If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $R - (X \cup Y)$:
 - $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
 - $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
 - $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.
- An MVD $X \twoheadrightarrow Y$ in R is called a **trivial MVD** if (a) Y is a subset of X , or (b) $X \cup Y = R$.

Multivalued Dependencies and Fourth Normal Form (3)

Definition:

- A relation schema R is in **4NF** with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^+ , X is a superkey for R .
- Note: F^+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every relation state r of R that satisfies F . It is also called the **closure** of F .

Figure 14.15 Fourth and fifth normal forms.

(a) EMP

<u>Ename</u>	<u>Pname</u>	<u>Dname</u>
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(c) SUPPLY

<u>Sname</u>	<u>Part_name</u>	<u>Proj_name</u>
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(b) EMP_PROJECTS

<u>Ename</u>	<u>Pname</u>
Smith	X
Smith	Y

EMP_DEPENDENTS

<u>Ename</u>	<u>Dname</u>
Smith	John
Smith	Anna

(d) R_1

<u>Sname</u>	<u>Part_name</u>
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

<u>Sname</u>	<u>Proj_name</u>
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

<u>Part_name</u>	<u>Proj_name</u>
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Figure 14.15

Fourth and fifth normal forms. (a) The EMP relation with two MVDs: Ename \twoheadrightarrow Pname and Ename \twoheadrightarrow Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R_1, R_2, R_3). (d) Decomposing the relation SUPPLY into the 5NF relations R_1, R_2, R_3 .

6. Join Dependencies and Fifth Normal Form (1)

Definition:

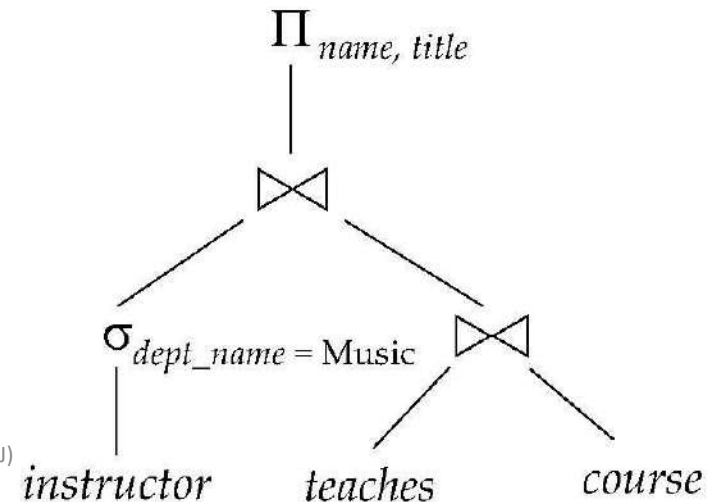
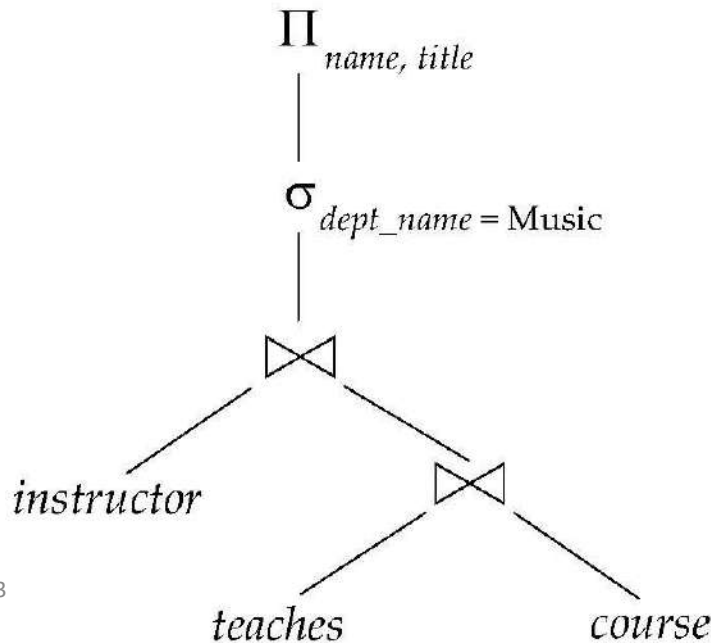
- A **join dependency (JD)**, denoted by $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , specifies a constraint on the states r of R .
 - The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, \dots, R_n ; that is, for every such r we have
 - $$* (\pi_{R_1}(r), \pi_{R_2}(r), \dots, \pi_{R_n}(r)) = r$$

Note: an MVD is a special case of a JD where $n = 2$.

- A join dependency $JD(R_1, R_2, \dots, R_n)$, specified on relation schema R , is a **trivial JD** if one of the relation schemas R_i in $JD(R_1, R_2, \dots, R_n)$ is equal to R .

Query Optimization: Introduction

- Conducted by a query optimizer in a DBMS
- Goal: select best available strategy for executing query
 - Based on information available
- Most RDBMSs use a tree as the internal representation of a query
- Alternative ways of evaluating a given query
 - Equivalent expressions
 - Different algorithms for each operation



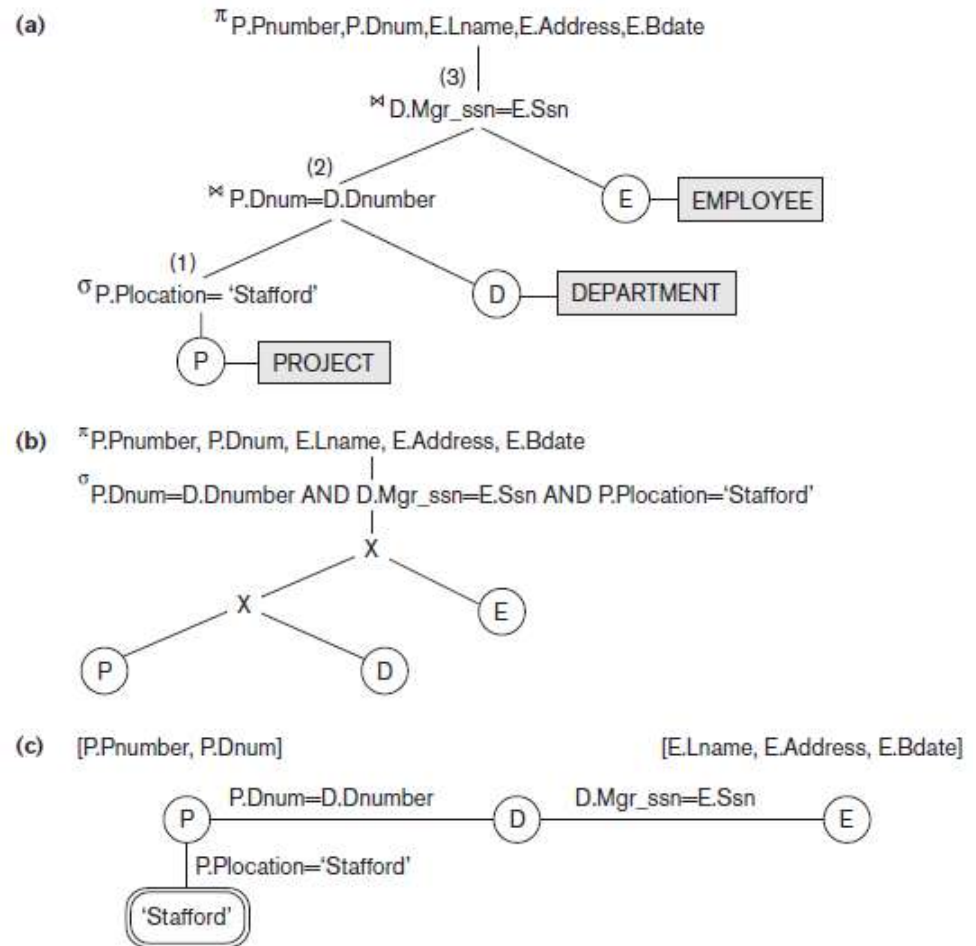
Query Trees and Heuristics for Query Optimization

- Step 1: scanner and parser generate initial query representation
- Step 2: representation is optimized according to heuristic rules
- Step 3: query execution plan is developed
 - Execute groups of operations based on access paths available and files involved
- Example heuristic rule
 - Apply SELECT and PROJECT before JOIN
 - Reduces size of files to be joined
- Query tree
 - Represents relational algebra expression
- Query graph
 - Represents relational calculus expression

Query Trees and Query Graph Corresponding to Q2

Q2: **SELECT** P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE P.Dnum=D.Dnumber **AND** D.Mgr_ssn=E.Ssn **AND**
P.Plocation= 'Stafford';

Figure 19.1 Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.



Query Trees and Heuristics for Query Optimization (cont'd.)

- Query tree represents a specific order of operations for executing a query
 - Preferred to query graph for this reason
- Query graph
 - Relation nodes displayed as single circles
 - Constants represented by constant nodes
 - Double circles or ovals
 - Selection or join conditions represented as edges
 - Attributes to be retrieved displayed in square brackets

Heuristic Optimization of Query Trees

- Many different query trees can be used to represent the query and get the same results
- Figure 19.1b shows initial tree for Q2
 - Very inefficient - will never be executed
 - Optimizer will transform into equivalent final query tree

Query Transformation Example

Q: **SELECT** E.Lname
 FROM EMPLOYEE E, WORKS_ON W, PROJECT P
 WHERE P.Pname='Aquarius' **AND** P.Pnumber=W.Pno **AND** E.Essn=W.Ssn
 AND E.Bdate > '1957-12-31';

(a)

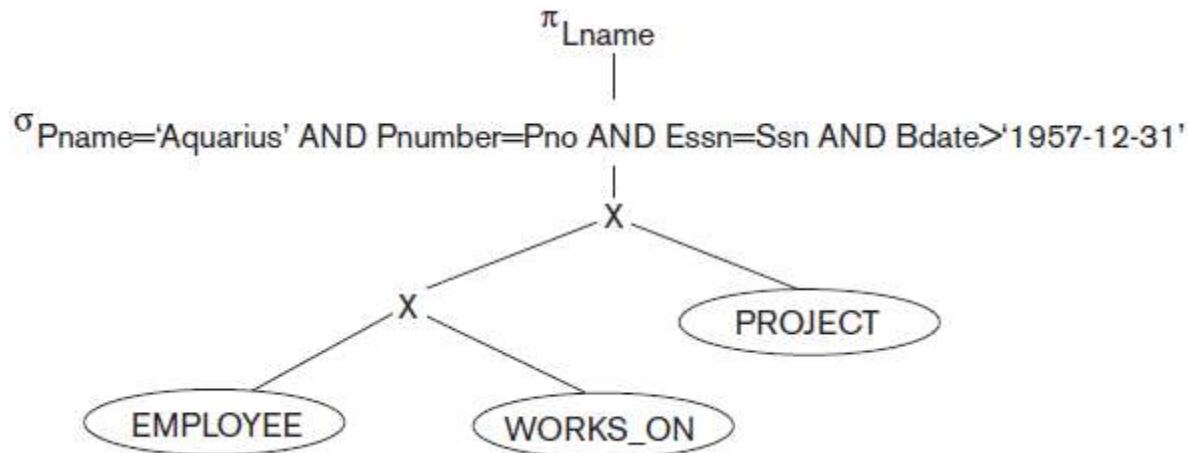


Figure 19.2 Steps in converting a query tree during heuristic optimization. (a) Initial (canonical) query tree for SQL query Q.

Query Transformation Example (cont'd.)

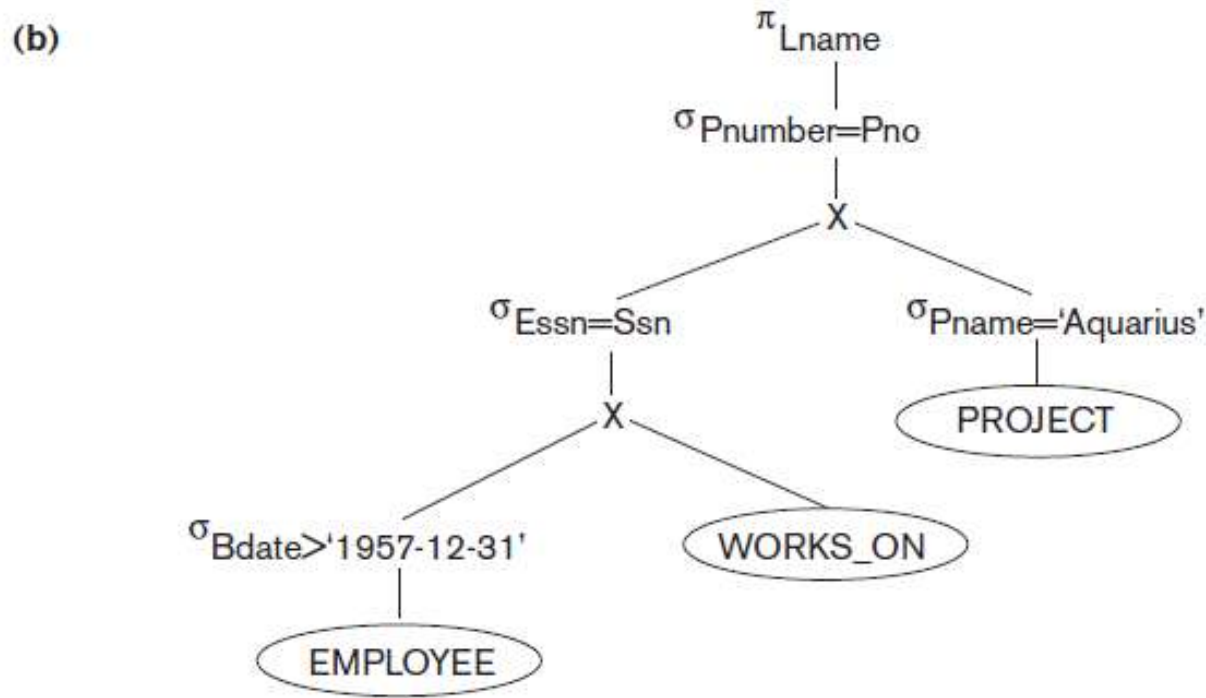


Figure 19.2 Steps in converting a query tree during heuristic optimization
(b) Moving SELECT operations down the query tree.

Query Transformation Example (cont'd.)

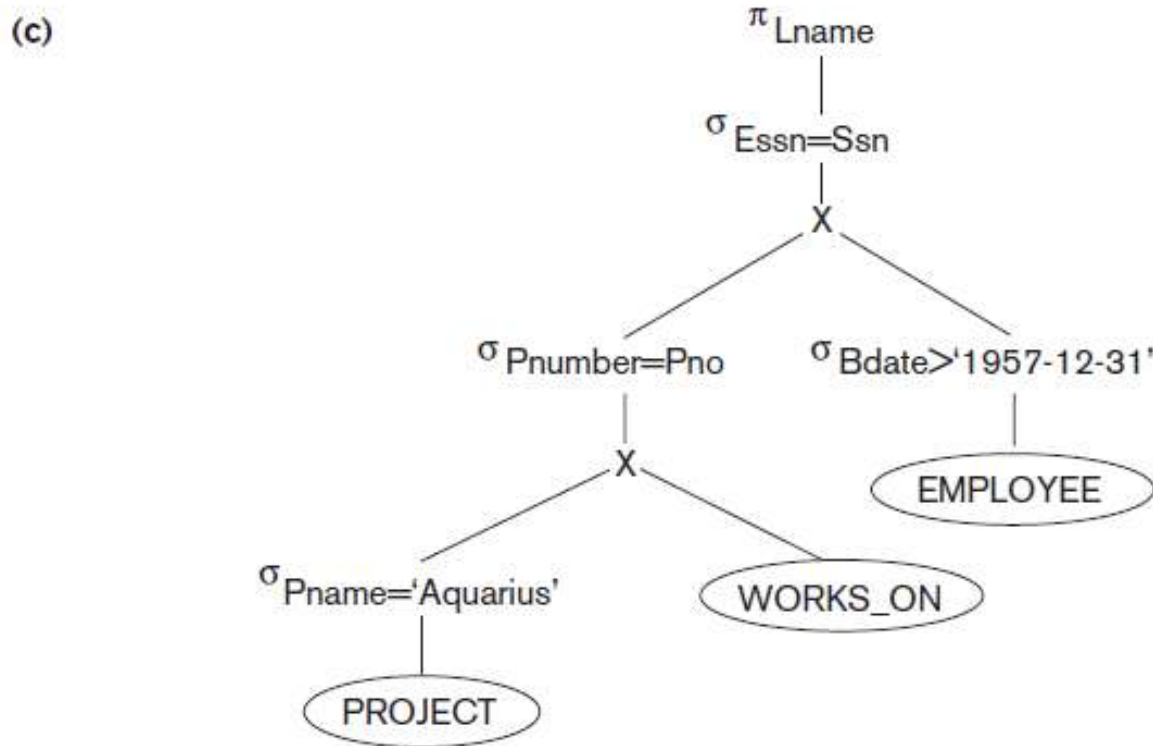


Figure 19.2 Steps in converting a query tree during heuristic optimization
(c) Applying the more restrictive SELECT operation first.

Query Transformation Example (cont'd.)

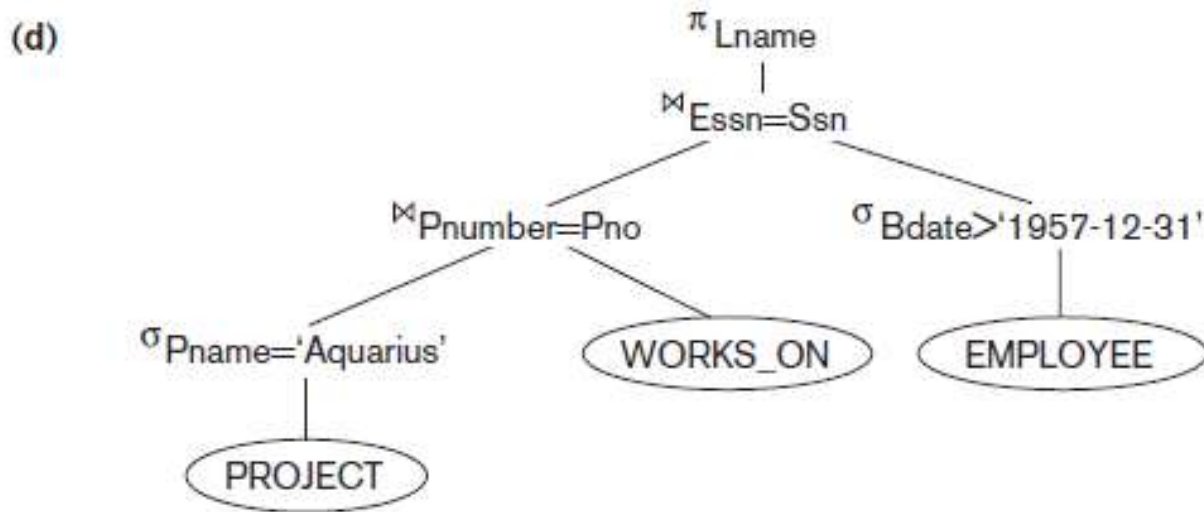


Figure 19.2 Steps in converting a query tree during heuristic optimization
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.

Query Transformation Example (cont'd.)

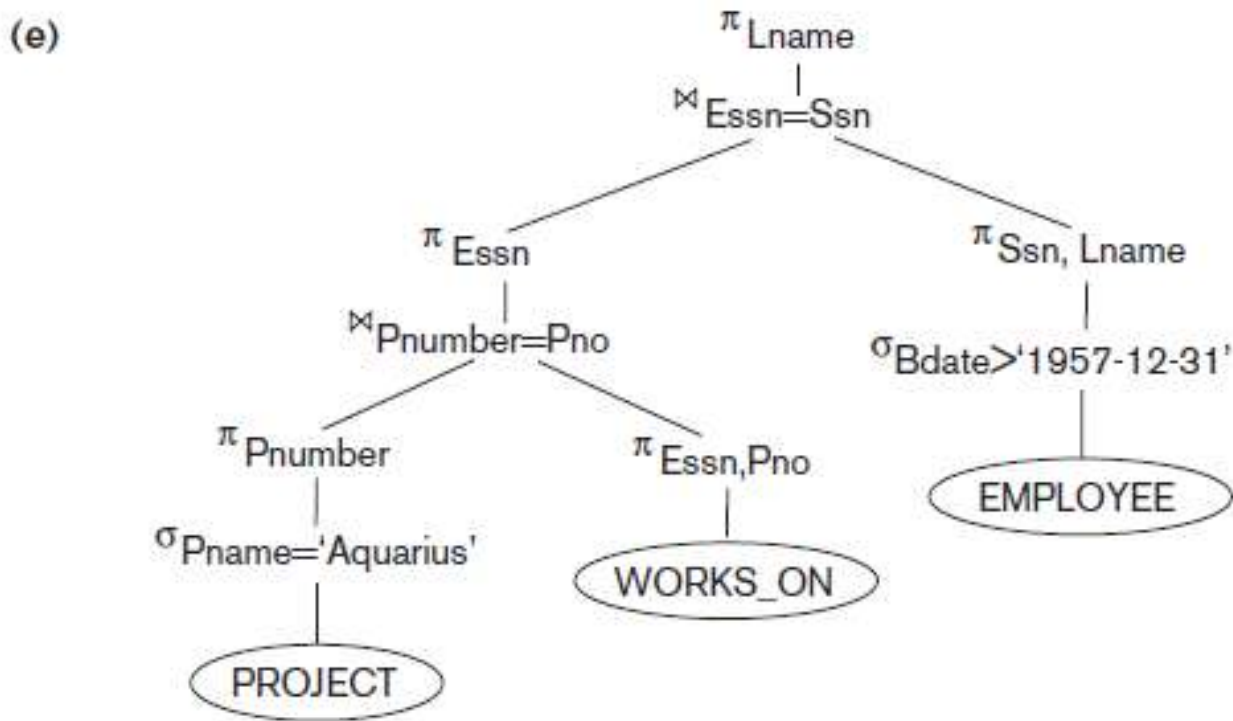


Figure 19.2 Steps in converting a query tree during heuristic optimization
(e) Moving PROJECT operations down the query tree.

General Transformation Rules for Rational Algebra Equations

- Some transformation rules useful in query optimization
 - Cascade of σ
 - Conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations
 - Commutativity of σ
 - Cascade of π
 - In a sequence of π operations, all but the last one can be ignored
 - Commuting σ with π

Summary of Heuristics for Algebraic Optimization

- Apply first the operations that reduce the size of intermediate results
 - Perform SELECT and PROJECT operations as early as possible to reduce the number of tuples and attributes
 - The SELECT and JOIN operations that are most restrictive should be executed before other similar operations

19.2 Choice of Query Execution Plans

- Materialized evaluation
 - Result of an operation stored as temporary relation
- Pipelined evaluation
 - Operation results forwarded directly to the next operation in the query sequence

Nested Subquery Optimization

- Unnesting
 - Process of removing the nested query and converting the inner and outer query into one block
- Queries involving a nested subquery connected by IN or ANY connector can always be converted into a single block query
- Alternate technique
 - Creating temporary result tables from subqueries and using them in joins

Subquery (View) Merging Transformation

- Inline view
 - FROM clause subquery
- View merging operation
 - Merges the tables in the view with the tables from the outer query block
 - Views containing select-project-join operations are considered simple views
 - Can always be subjected to this type of view-merging

Subquery (View) Merging Transformation (cont'd.)

- Group-By view-merging
 - Delaying the Group By operation after performing joins may reduce the data subjected to grouping in case the joins have low join selectivity
 - Alternately, performing Group By early may reduce the amount of data subjected to subsequent joins
 - Optimizer determines whether to merge GROUP-BY views based on estimated costs

Materialized Views

- View defined in database as a query
 - Materialized view stores results of that query
 - May be stored temporarily or permanently
- Optimization technique
 - Using materialized views to avoid some of the computation involved in the query
 - Easier to read it when needed than recompute from scratch

Incremental View Maintenance

- Update view incrementally by accounting for changes that occurred since last update
 - Join
 - Selection
 - Projection
 - Intersection
 - Aggregation

19.3 Use of Selectives in Cost-Based Optimization

- Query optimizer estimates and compares costs of query execution using different strategies
 - Chooses lowest cost estimate strategy
 - Process suited to compiled queries
- Interpreted queries
 - Entire process occurs at runtime
 - Cost estimate may slow down response time

Use of Selectives in Cost-Based Optimization (cont'd.)

- Cost-based query optimization approach
 - For a given query subexpression, multiple equivalence rules may apply
 - Quantitative measure for evaluating alternatives
 - Cost metric includes space and time requirements
 - Design appropriate search strategies by keeping cheapest alternatives and pruning costlier alternatives
 - Scope of query optimization is a query block
 - Global query optimization involves multiple query blocks

Use of Selectives in Cost-Based Optimization (cont'd.)

- Cost components for query execution
 - Access cost to secondary storage
 - Disk storage cost
 - Computation cost
 - Memory usage cost
 - Communication cost

Catalog Information Used in Cost Functions

- Information stored in DBMS catalog and used by optimizer
 - File size
 - Organization
 - Number of levels of each multilevel index
 - Number of distinct values of an attribute
 - Attribute selectivity
 - Allows calculation of selection cardinality
 - Average number of records that satisfy equality selection condition on that attribute

Histograms

- Tables or data structures that record information about the distribution of data
- RDBMS stores histograms for most important attributes

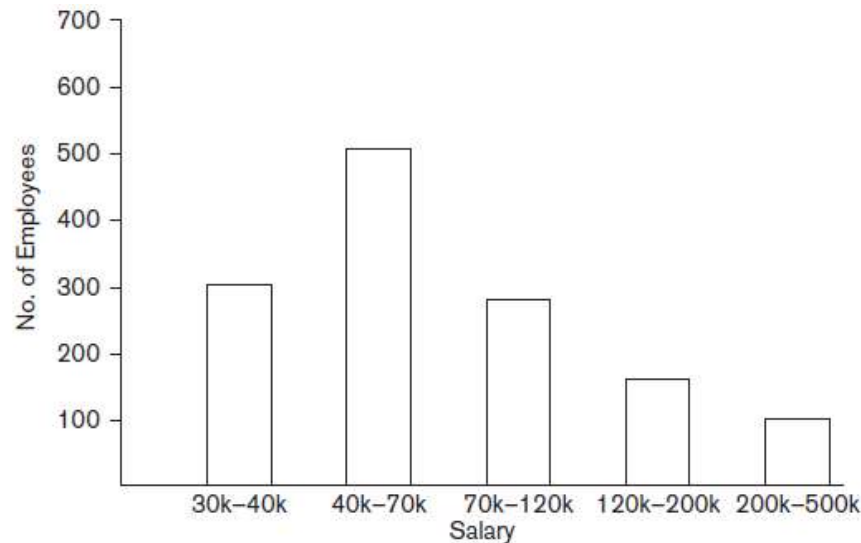


Figure 19.4 Histogram of salary in the relation EMPLOYEE

19.4 Cost Functions for SELECT Operation

- Notation used in cost formulas

C_{Si} : Cost for method Si in block accesses

r_X : Number of records (tuples) in a relation X

b_X : Number of blocks occupied by relation X (also referred to as b)

bfr_X : Blocking factor (i.e., number of records per block) in relation X

sl_A : Selectivity of an attribute A for a given condition

sA : Selection cardinality of the attribute being selected ($= sl_A * r$)

xA : Number of levels of the index for attribute A

$b_{11}A$: Number of first-level blocks of the index on attribute A

$NDV(A, X)$: Number of distinct values of attribute A in relation X

Cost Function for SELECT Operation (cont'd.)

- S1: Linear search (brute force approach)

- Search all file blocks to retrieve all records

$$C_{S1a}=b$$

- For equality condition on a key attribute, on average one-half the records are searched

$$C_{S1b}=\frac{b}{2}$$

- S2: Binary search

$$C_{S2}=\log_2 b + \left[\frac{s}{bfr}\right] - 1$$

- Reduces to $\log_2 b$ if equality condition is on a key attribute

Cost Function for SELECT Operation (cont'd.)

- S3a: Using a primary index to retrieve a single record

$$C_{S3a} = x + 1$$

- S3b: Using a hash key to retrieve a single record

$$C_{S3b} = 1$$

- S4: Using an ordering index to retrieve multiple records

$$C_{S4} = x + \frac{b}{2}$$

Cost Functions for SELECT Operation (cont'd.)

- S5: Using a clustering index to retrieve multiple records

$$C_{S5} = x + \left\lceil \frac{s}{bfr} \right\rceil$$

- S6: Using a secondary (B+ tree) index

$$C_{S6a} = x + 1 + s \quad (\text{worst case})$$

$$C_{S6b} = x + \frac{b_{I1}}{2} + \frac{r}{2} \quad (\text{for range queries})$$

Cost Functions for SELECT Operation (cont'd.)

- Dynamic programming
 - Cost-based optimization approach
 - Subproblems are solved only once
 - Applies when a problem has subproblems that themselves have subproblems

19.5 Cost Functions for the JOIN Operation

- Cost functions involve estimate of file size that results from the JOIN operation
- Join selectivity
 - Ratio of the size of resulting file to size of the CARTESIAN PRODUCT file
 - Simple formula for join selectivity
- Join cardinality

$$js = 1 / \max (NDV (A, R), NDV (B,S))$$

$$jc = |(R_c S)| = js * |R| * |S|$$

Cost Functions for the JOIN Operation (cont'd.)

- J1: Nested-loop join
 - For three memory buffer blocks:

- For n_B memory buffer blocks:

$$C_{J1} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$$

- J2: Index-based nested-loop join

- $C_{J1} = b_R + (\lceil b_R/(n_B - 2) \rceil * b_S) + ((js * |R| * |S|)/bfr_{RS})$ in

$$C_{J2a} = b_R + (|R| * (x_B + 1 + s_B)) + ((js * |R| * |S|)/bfr_{RS})$$

Cost Functions for the JOIN Operation (cont'd.)

- J3: Sort-merge join
 - For files already sorted on the join attributes
 - Cost of sorting must be added if sorting needed
- J4 $C_{J3a} = b_R + b_S + ((js * |R| * |S|)/bfr_{RS})$

$$C_{J4} = 3 * (b_R + b_S) + ((js * |R| * |S|)/bfr_{RS})$$

Cost Functions for the JOIN Operation (cont'd.)

- Join selectivity and cardinality for semi-join

- Unnesting q

```
SELECT COUNT(*)  
FROM T1  
WHERE T1.X IN (SELECT T2.Y  
               FROM T2);
```

- Join selectivity

```
SELECT COUNT(*)  
FROM T1, T2  
WHERE T1.X S= T2.Y;
```

- Join cardinality

$$js = \text{MIN}(1, \text{NDV}(Y, T2) / \text{NDV}(X, T1))$$

$$jc = |T1| * js$$

Cost Functions for the JOIN Operation (cont'd.)

- Join selectivity and cardinality for anti-join

- Unnesting q

```
SELECT COUNT (*)  
FROM T1  
WHERE T1.X NOT IN (SELECT T2.Y  
FROM T2);
```

- Join selectivity

```
SELECT COUNT(*)  
FROM T1, T2  
WHERE T1.X A= T2.Y;
```

- Join cardinality

$$js = 1 - \text{MIN}(1, \text{NDV}(\text{T2.y}) / \text{NDV}(\text{T1.x}))$$

$$jc = |T1| * js$$

Cost Functions for the JOIN Operation (cont'd.)

- Multirelation queries and JOIN ordering choices
 - Left-deep join tree
 - Right-deep join tree
 - Bushy join tree

No. of Relations N	No. of Left-Deep Trees $N!$	No. of Bushy Shapes $S(N)$	No. of Bushy Trees $(2N - 2)! / (N - 1)!$
2	2	1	2
3	6	2	12
4	24	5	120
5	120	14	1,680
6	720	42	30,240
7	5,040	132	665,280

Table 19.1 Number of permutations of left-deep and bushy join trees of n relations

Cost Functions for the JOIN Operation (cont'd.)

- Physical optimization involves execution decision at the physical level
 - Cost-based physical optimization
 - Top-down approach
 - Bottom-up approach
- Certain physical level heuristics make cost optimizations unnecessary
 - Example: for selections, use index scans whenever possible

Cost Functions for the JOIN Operation (cont'd.)

- Left-deep trees generally preferred
 - Work well for common algorithms for join
 - Able to generate fully pipelined plans
- Characteristics of dynamic programming algorithm
 - Optimal solution structure is developed
 - Value of the optimal solution is recursively defined
 - Optimal solution is computed and its value developed in a bottom-up fashion

19.6 Example to Illustrate Cost-Based Query Optimization

- Example: Consider Q2 below and query tree from Figure 19.1(a) on slide 6

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

- Informative slide)

Figure 19.6 (next slide)

Figure 19.6 Sample statistical information for relations in Q2.

(a) Column information

(b) Table information

(c) Index information

(a)

Table_name	Column_name	Num_distinct	Low_value	High_value
PROJECT	Plocation	200	1	200
PROJECT	Pnumber	2000	1	2000
PROJECT	Dnum	50	1	50
DEPARTMENT	Dnumber	50	1	50
DEPARTMENT	Mgr_ssn	50	1	50
EMPLOYEE	Ssn	10000	1	10000
EMPLOYEE	Dno	50	1	50
EMPLOYEE	Salary	500	1	500

(b)

Table_name	Num_rows	Blocks
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

(c)

Index_name	Uniqueness	Blevel*	Leaf_blocks	Distinct_keys
PROJ_PLOC	NONUNIQUE	1	4	200
EMP_SSN	UNIQUE	1	50	10000
EMP_SAL	NONUNIQUE	1	50	500

*Blevel is the number of levels without the leaf level.

Example to Illustrate Cost-Based Query Optimization (cont'd.)

- Assume optimizer considers only left-deep trees
- Evaluate potential join orders
 - PROJECT DEPARTMENT EMPLOYEE
 - DEPARTMENT PROJECT EMPLOYEE
 - DEPARTMENT ⋈ EMPLOYEE PROJECT ⋈
 - EMPLOYEE DEPARTMENT PROJECT ⋈



19.7 Additional Issues Related to Query Optimization

- Displaying the system's query execution plan
 - Oracle syntax
 - EXPLAIN PLAN FOR <SQL query>
 - IBM DB2 syntax
 - EXPLAIN PLAN SELECTION [additional options] FOR <SQL-query>
 - SQL server syntax
 - SET SHOWPLAN_TEXT ON or SET SHOWPLAN_XML ON or SET SHOWPLAN_ALL ON

Additional Issues Related to Query Optimization (cont'd.)

- Size estimation of other operations
 - Projection
 - Set operations
 - Aggregation
 - Outer join
- Plan caching
 - Plan stored by query optimizer for later use by same queries with different parameters
- Top k-results optimization
 - Limits strategy generation

19.8 An Example of Query Optimization in Data Warehouses

- Star transformation optimization
 - Goal: access a reduced set of data from the fact table and avoid using a full table scan on it
 - Classic star transformation
 - Bitmap index star transformation
- Joining back

19.9 Overview of Query Optimization in Oracle

- Physical optimizer is cost-based
- Scope is a single query block
- Calculates cost based on object statistics, estimated resource use and memory needed
- Global query optimizer
 - Integrates logical transformation and physical optimization phases to generate optimal plan for entire query tree
- Adaptive optimization
 - Feedback loop used to improve on previous decisions

Overview of Query Optimization in Oracle (cont'd.)

- Array processing
- Hints
 - Specified by application developer
 - Embedded in text of SQL statement
 - Types: access path, join order, join method, enabling or disabling a transformation
- Outlines used to preserve execution plans
- SQL plan management

19.10 Semantic Query Optimization

- Uses constraints specified on the database schema
- Goal: modify one query into another that is more efficient to execute

19.11 Summary

- Query trees
- Heuristic approaches used to improve efficiency of query execution
- Reorganization of query trees
- Pipelining and materialized evaluation
- Cost-based optimization approach
- Oracle query optimizer
- Semantic query optimization

