

CHAPTER 4

Enhanced Entity-Relationship (EER) Modeling

Chapter Outline

- EER stands for Enhanced ER or Extended ER
- EER Model Concepts
 - Includes all modeling concepts of basic ER
 - Additional concepts:
 - subclasses/superclasses
 - specialization/generalization
 - categories (UNION types)
 - attribute and relationship inheritance
 - Constraints on Specialization/Generalization
- The additional EER concepts are used to model applications more completely and more accurately
 - EER includes some object-oriented concepts, such as inheritance
- Knowledge Representation and Ontology Concepts

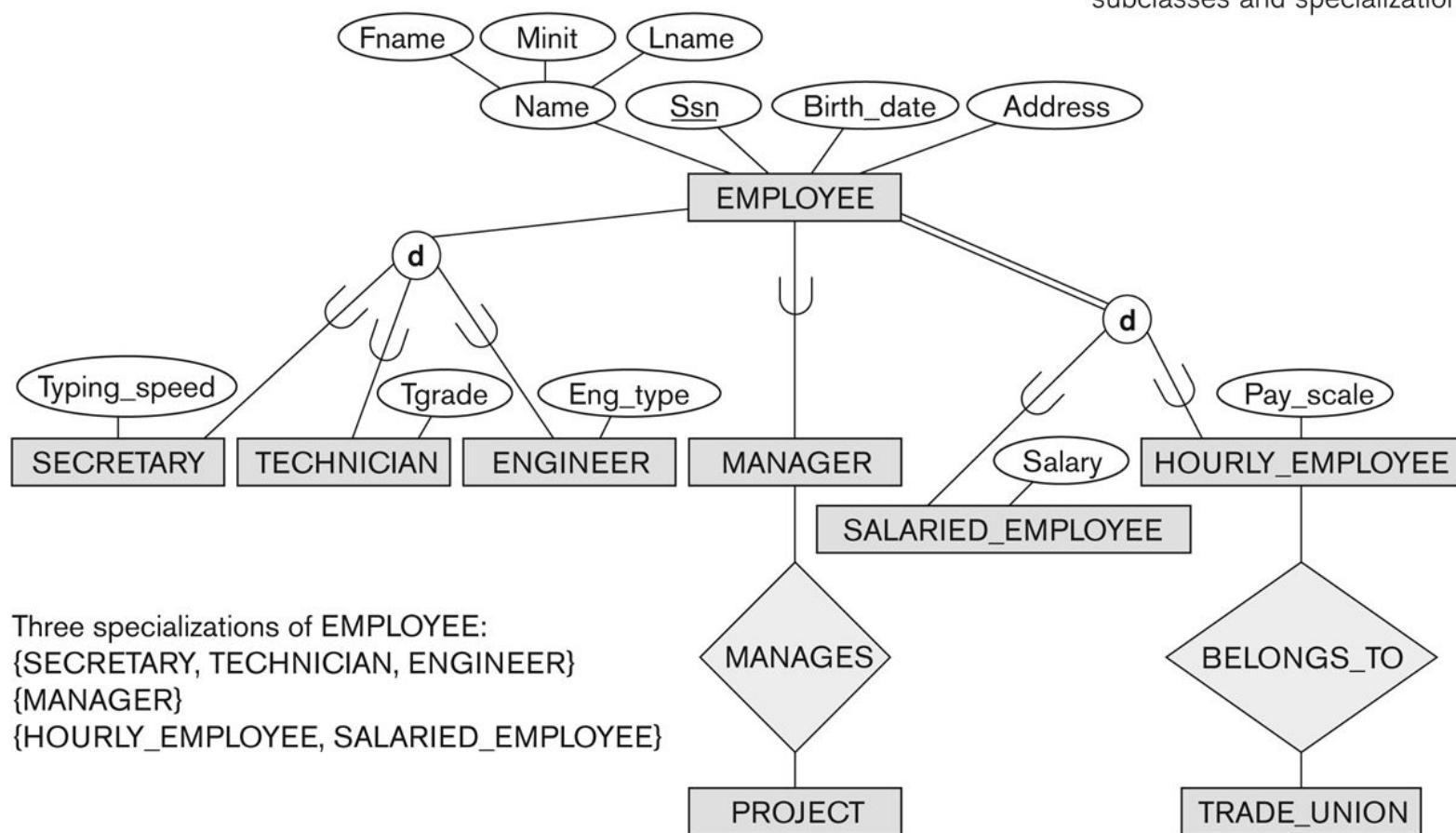
Subclasses and Superclasses (1)

- 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

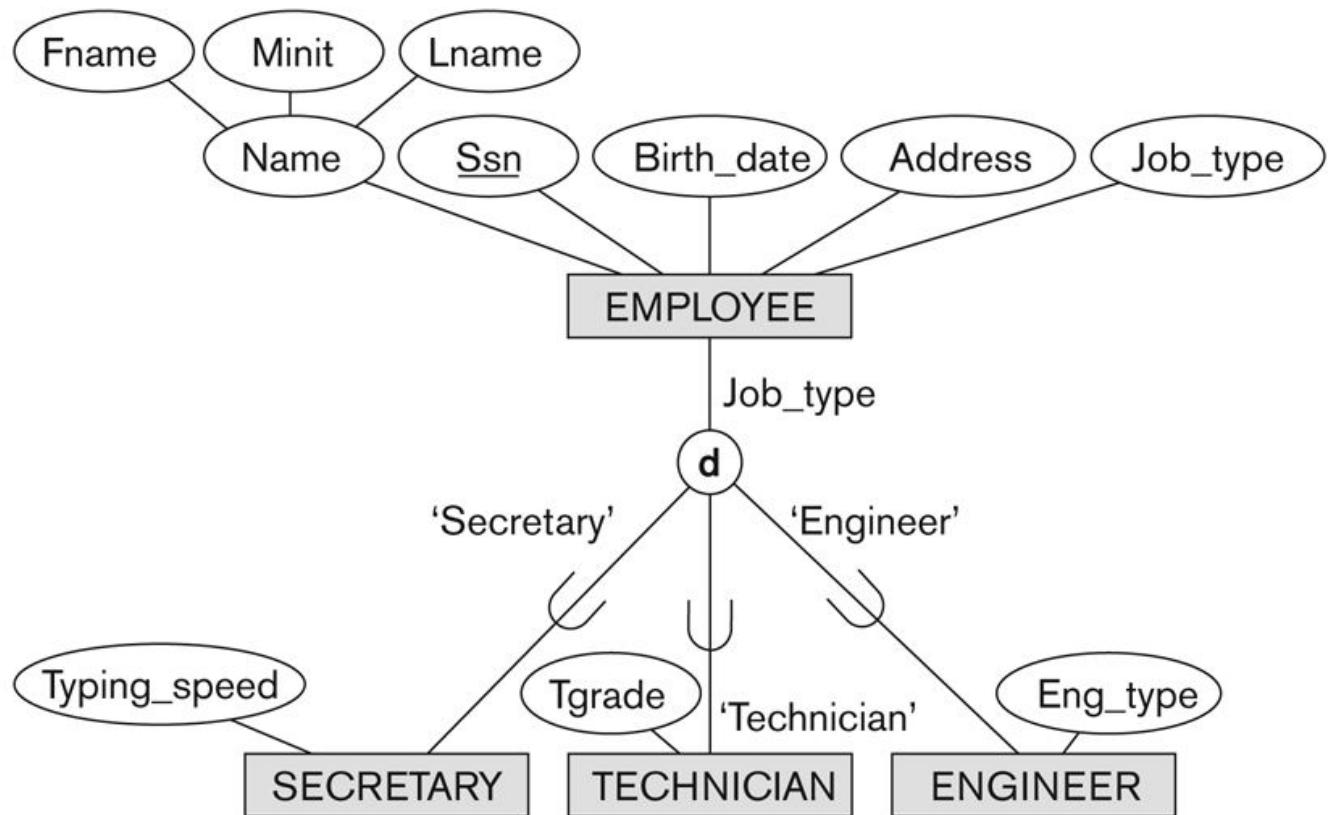
Subclasses and Superclasses (4)

- 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

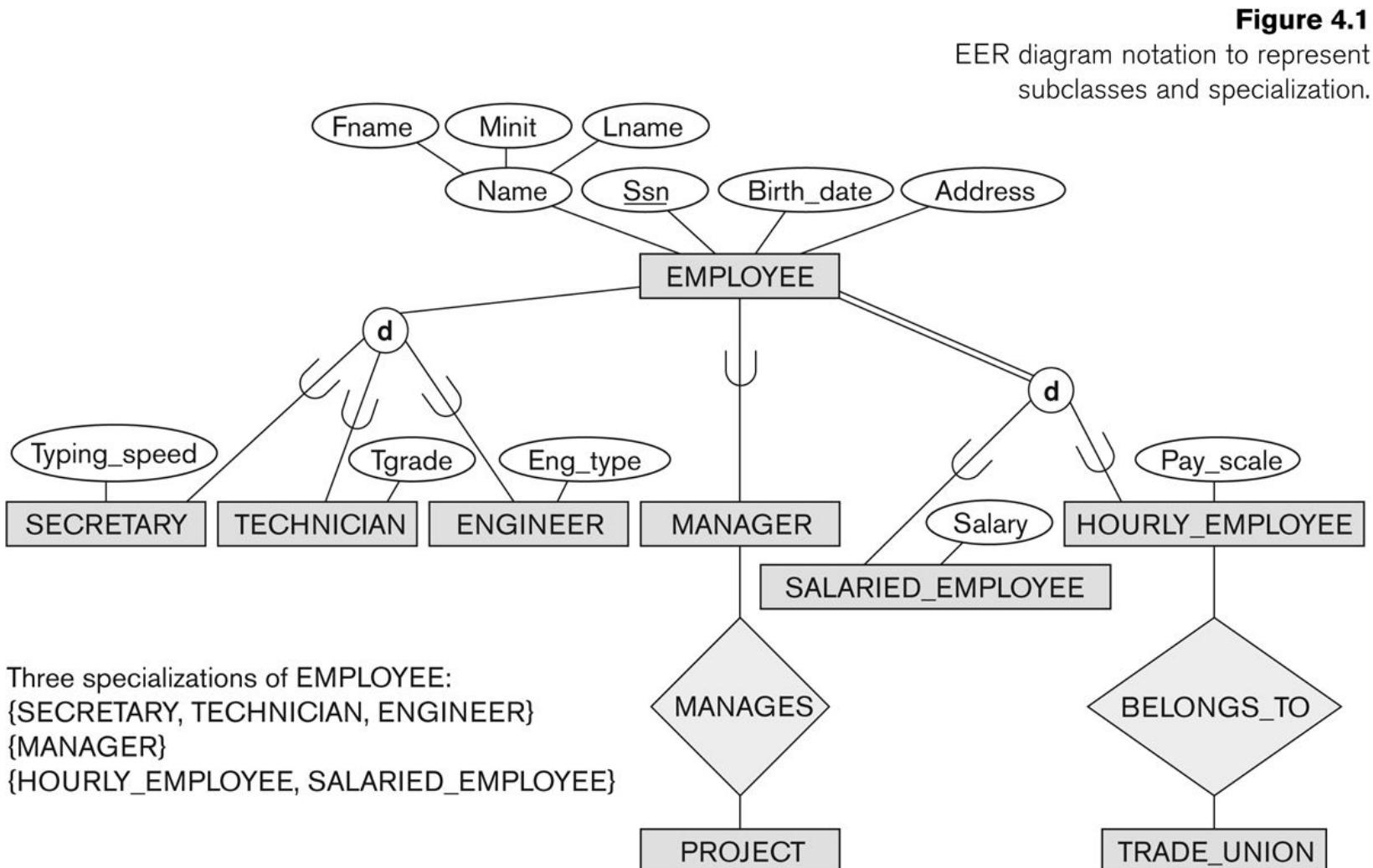
Specialization (1)

- 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)

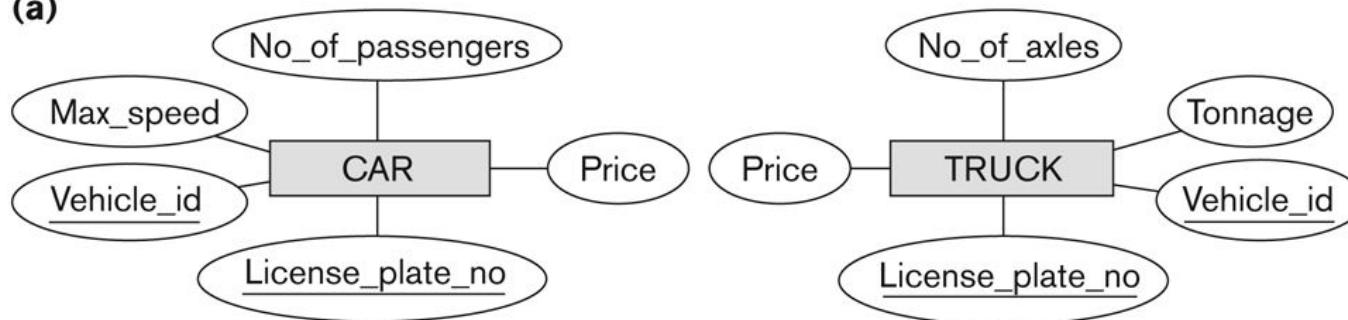


Generalization

- 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)

(a)



(b)

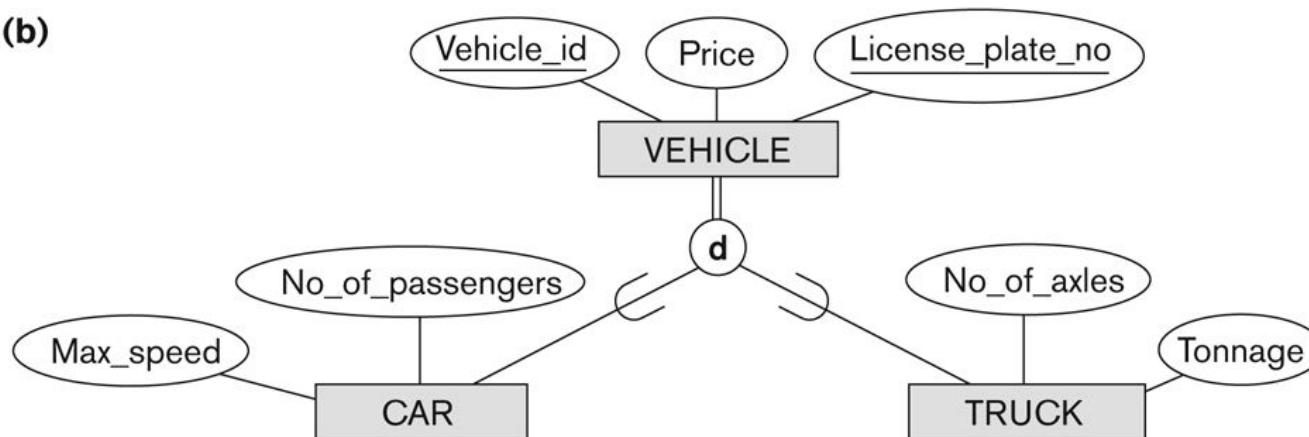


Figure 4.3

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

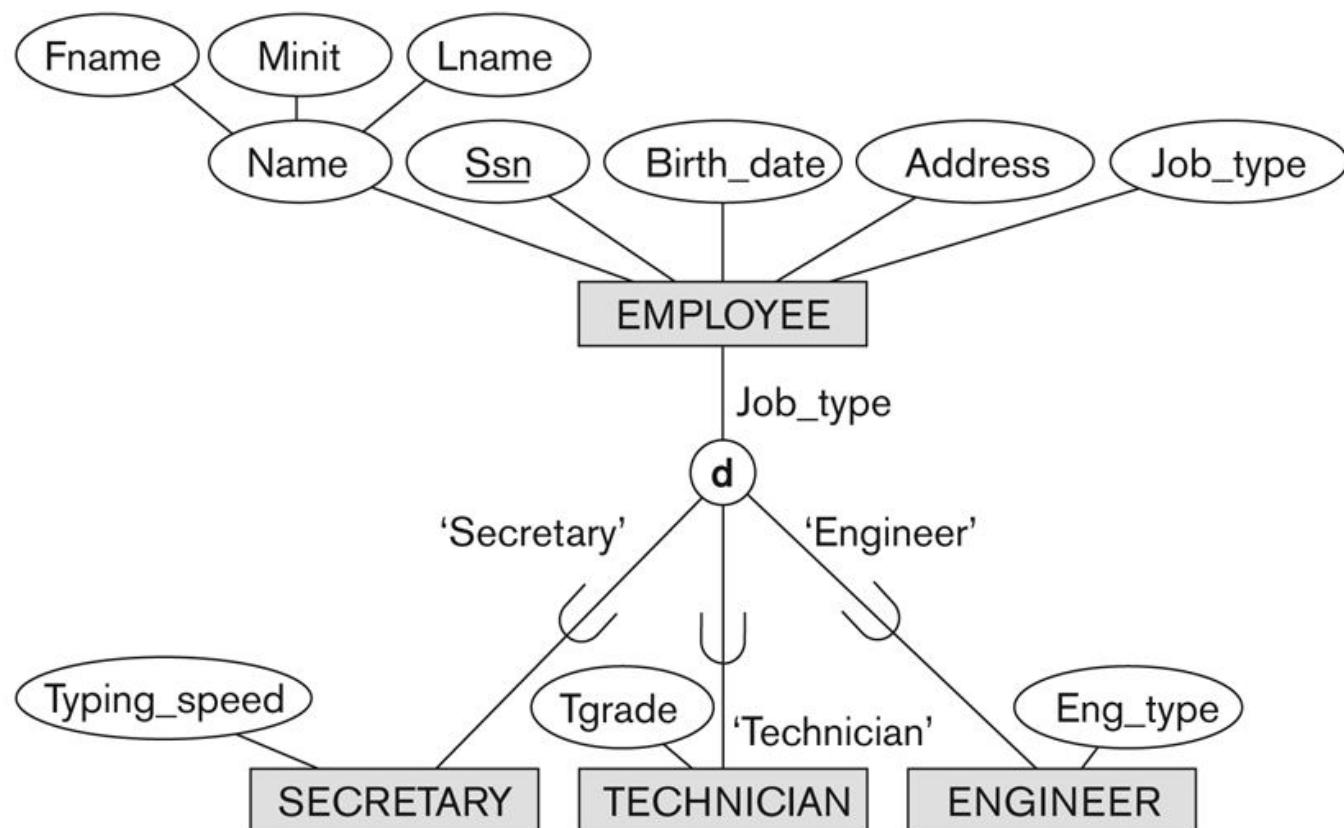
Constraints on Specialization and Generalization (2)

- 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

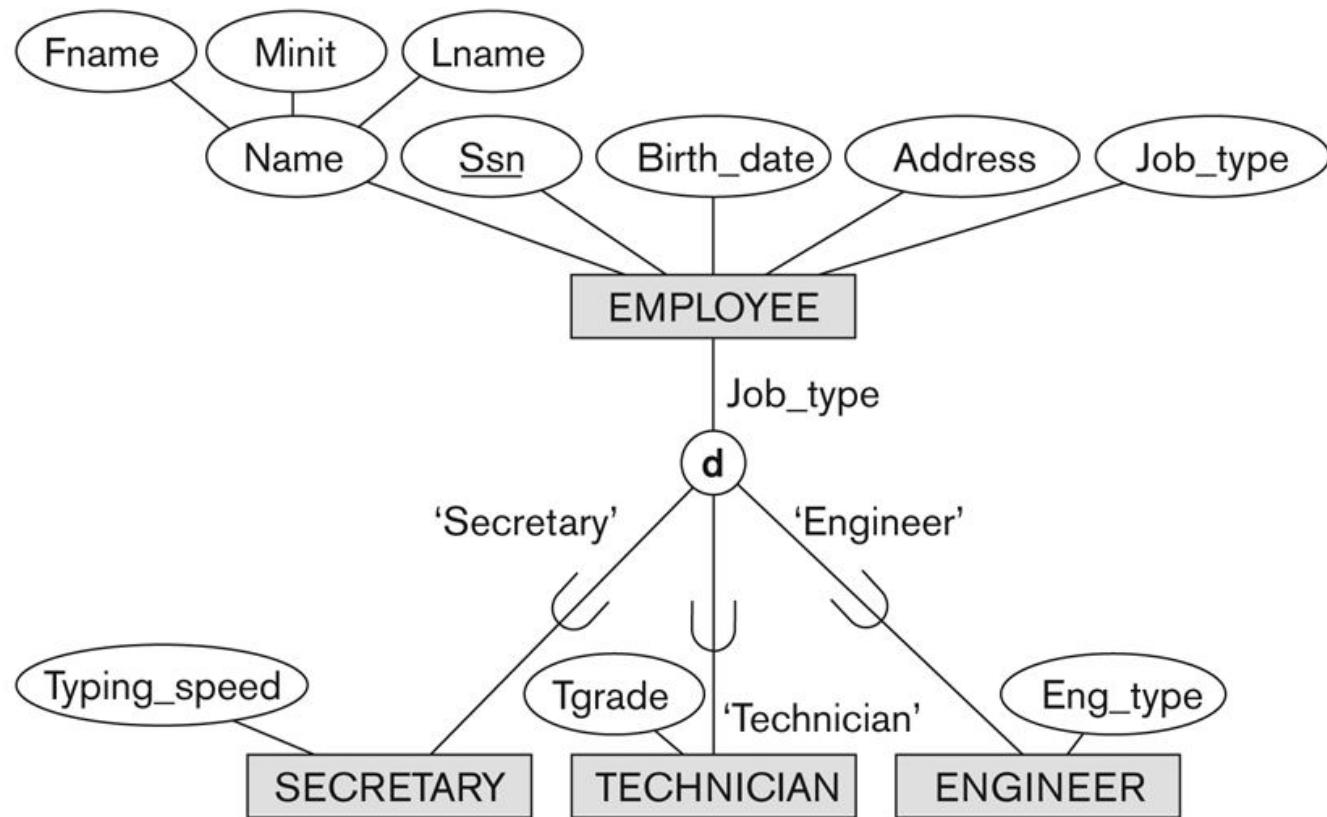
Constraints on Specialization and Generalization (6)

- 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

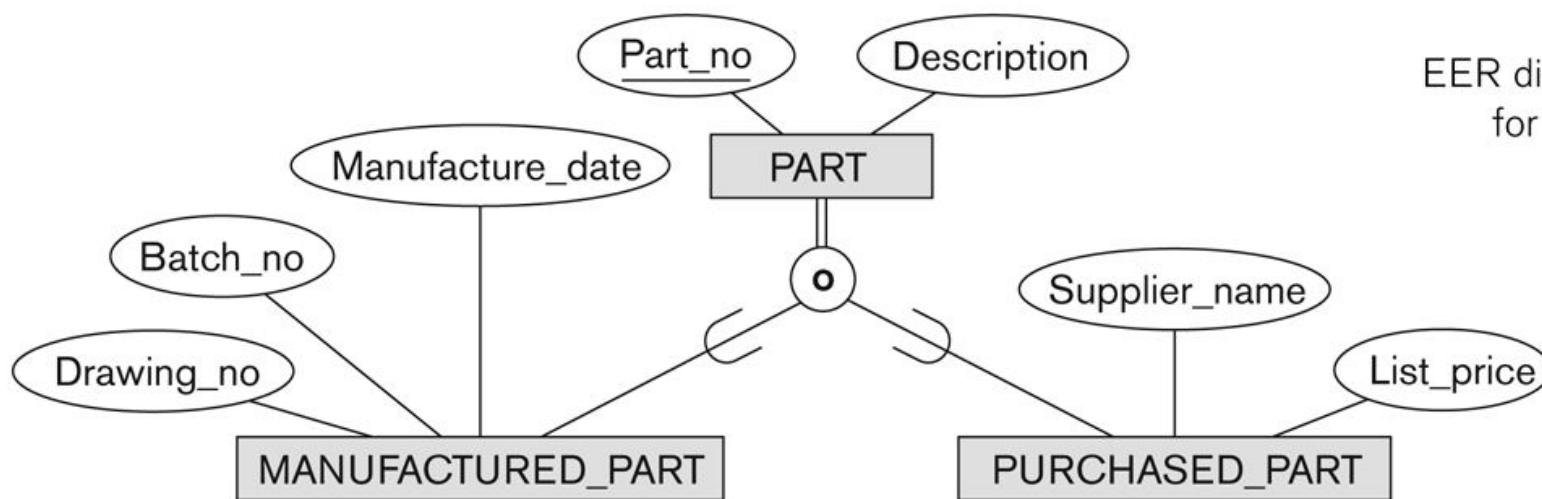


Figure 4.5
EER diagram notation
for an overlapping
(nondisjoint)
specialization.

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”

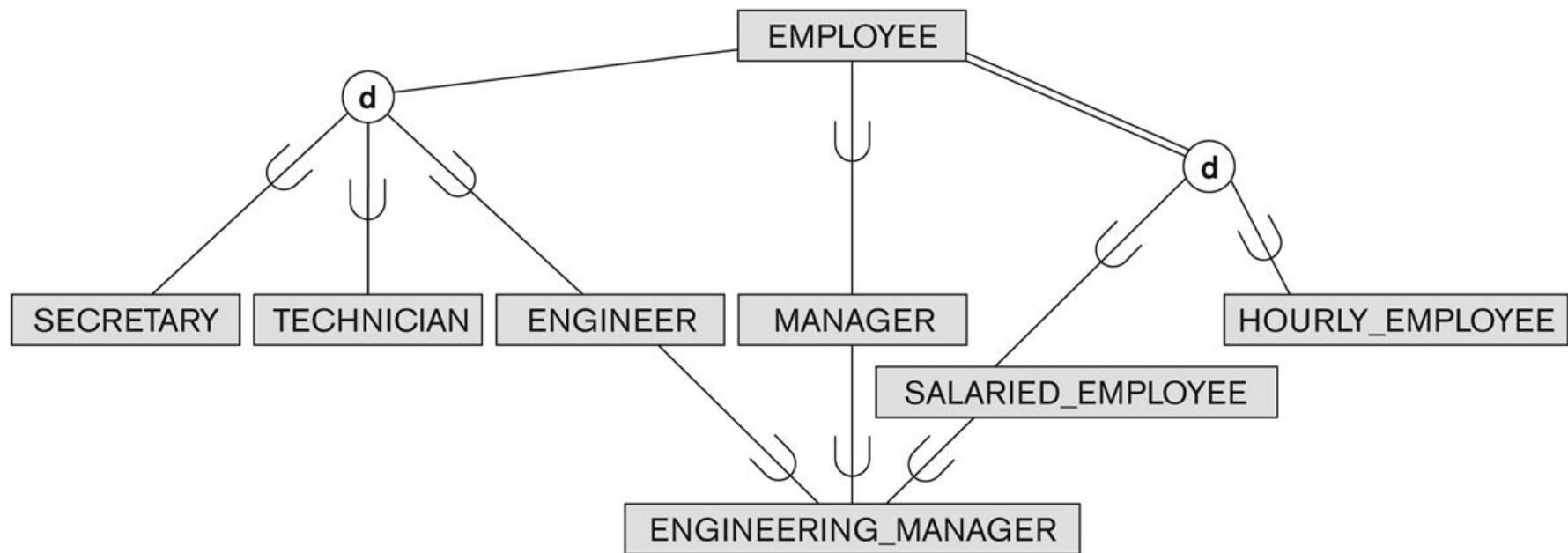


Figure 4.6
A specialization lattice with shared subclass **ENGINEERING_MANAGER**.

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)

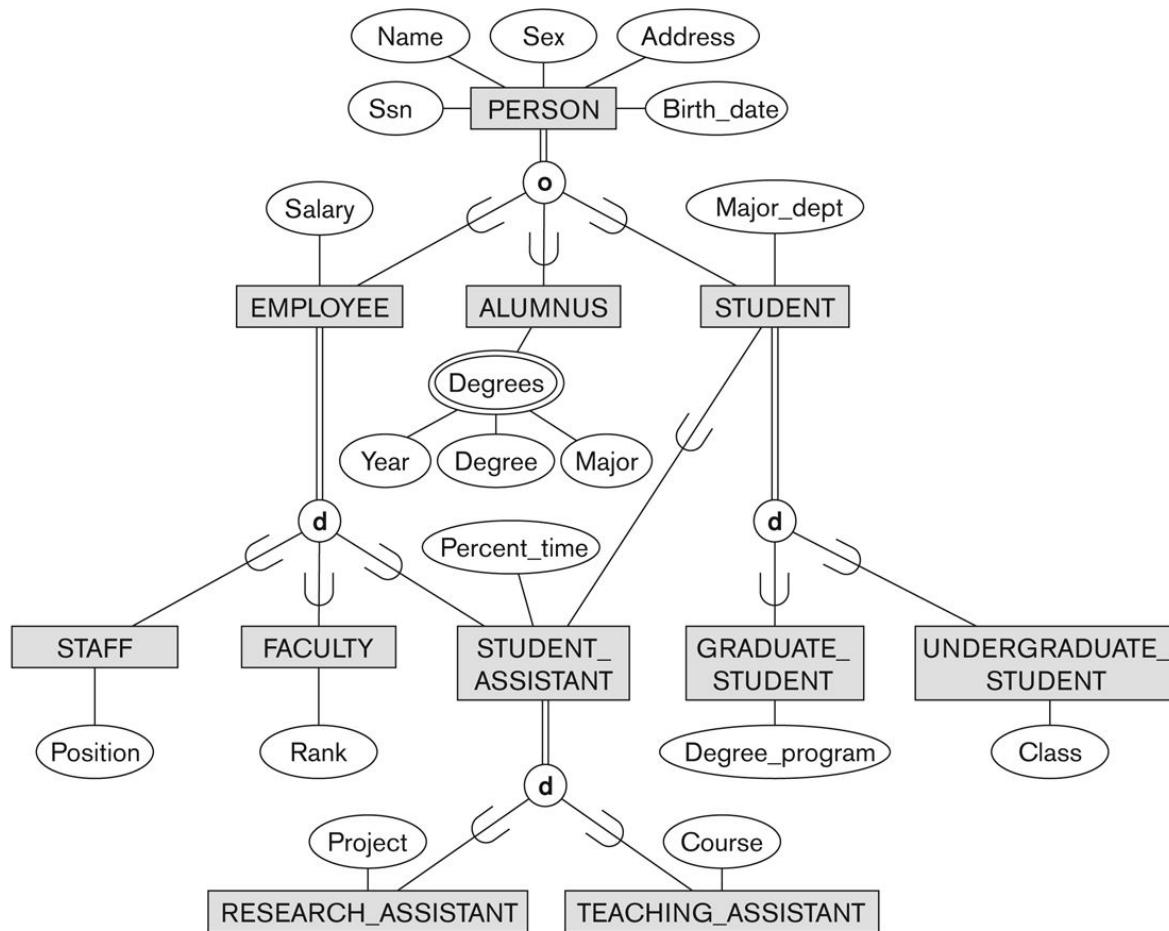


Figure 4.7

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

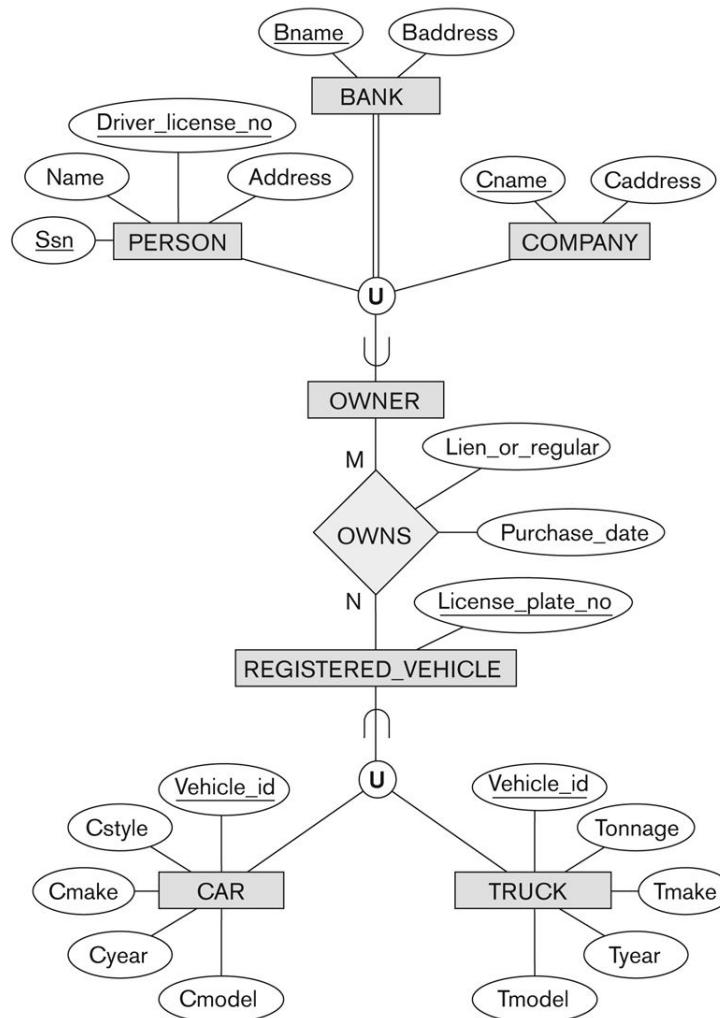


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

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 $A = ci$ (where A is an attribute of G and ci is a constant value from the domain of A) is used to specify membership in each subclass Si in Z
 - Note: If $ci \neq cj$ 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

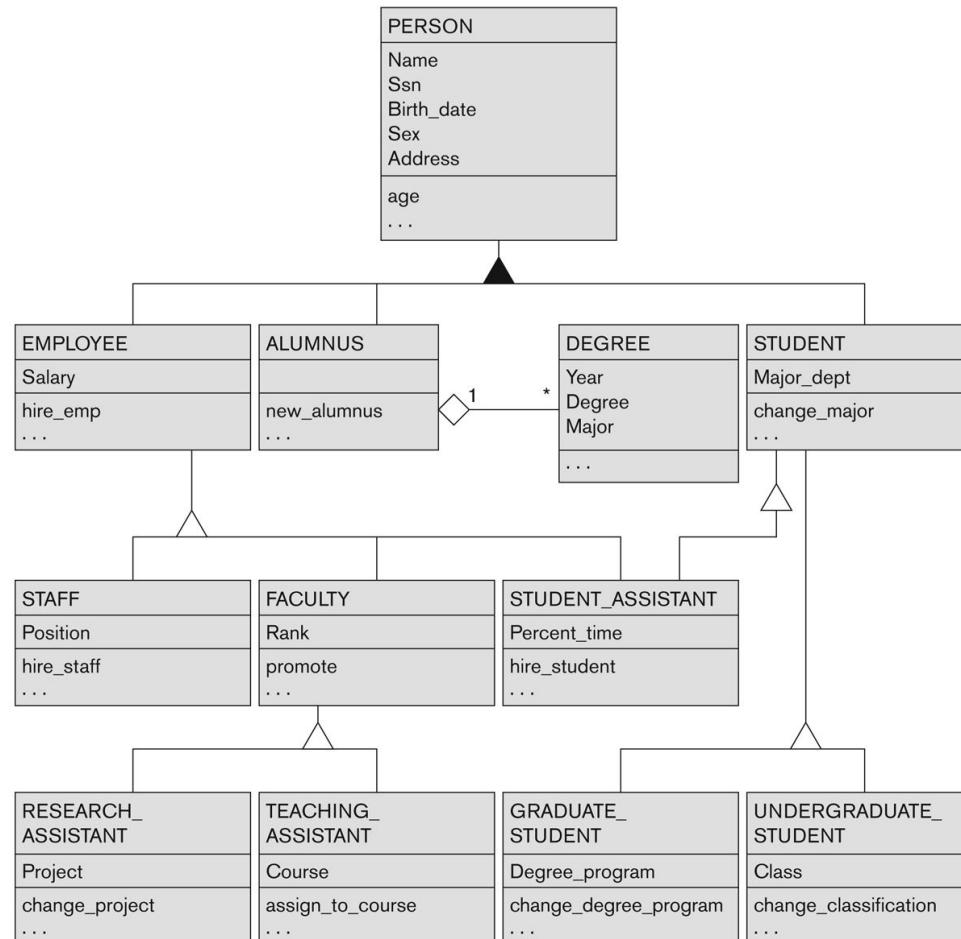


Figure 4.10

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

Alternative Diagrammatic Notations

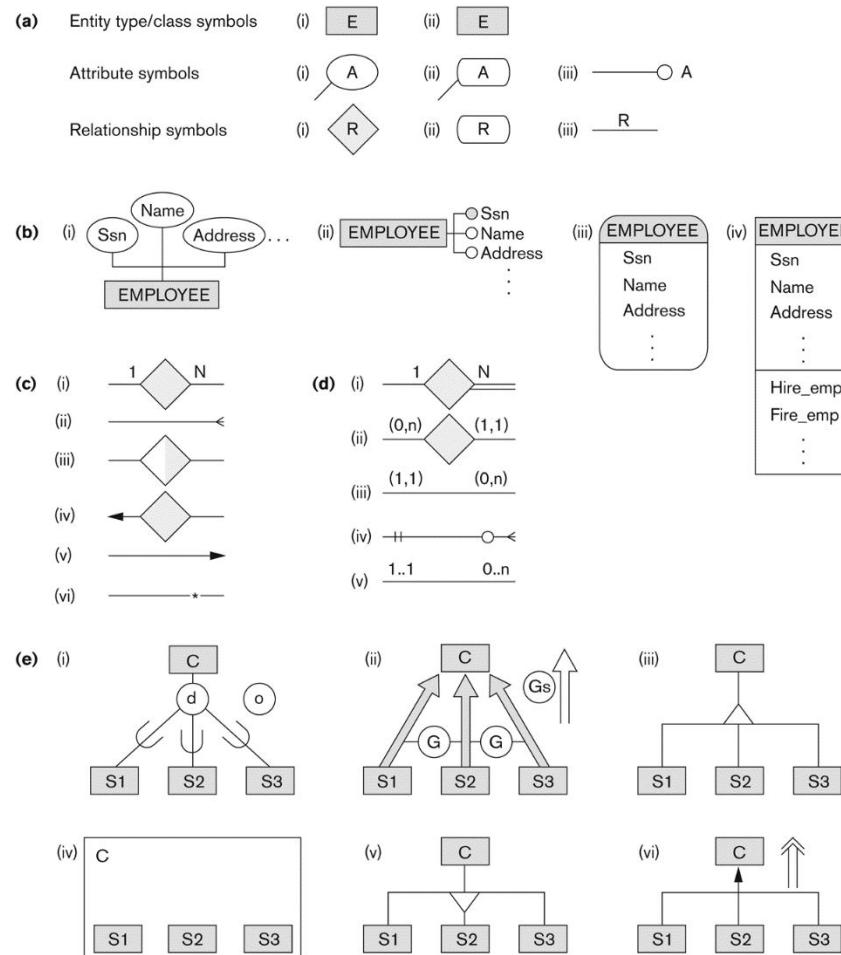


Figure A.1

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.

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

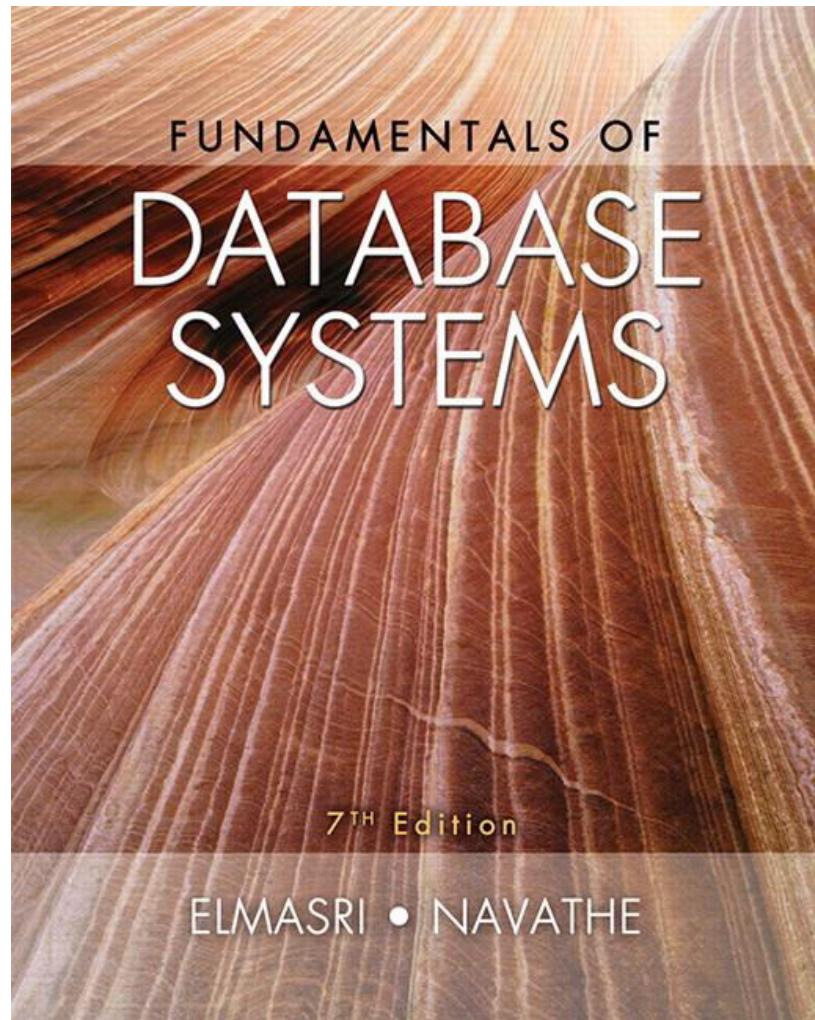
- TYPES OF DATA ABSTRACTIONS
 - CLASSIFICATION and INSTANTIATION
 - AGGREGATION and ASSOCIATION
(relationships)
 - GENERALIZATION and SPECIALIZATION
 - IDENTIFICATION
- CONSTRAINTS
 - CARDINALITY (Min and Max)
 - COVERAGE (Total vs. Partial, and Exclusive
(Disjoint) vs. Overlapping)

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

Summary

- Introduced the EER model concepts
 - Class/subclass relationships
 - Specialization and generalization
 - Inheritance
- Constraints on EER schemas
- These augment the basic ER model concepts introduced in Chapter 3
- EER diagrams and alternative notations were presented
- Knowledge Representation and Ontologies were introduced and compared with Data Modeling



CHAPTER 5

The Relational Data Model and Relational Database Constraints

Chapter Outline

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

Relational Model Concepts

- The relational Model of Data is based on the concept of a *Relation*
 - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations
- We review the essentials of the *formal relational model* in this chapter
- In *practice*, there is a *standard model* based on SQL – this is described in Chapters 6 and 7 as a language
- Note: There are several important differences between the *formal* model and the *practical* model, as we shall see

Relational Model Concepts

- A Relation is a mathematical concept based on the ideas of sets
- The model was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper:
 - "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

Informal Definitions

- Informally, a **relation** looks like a **table** of values.
- A relation typically contains a **set of rows**.
- The data elements in each **row** represent certain facts that correspond to a real-world **entity** or **relationship**
 - In the formal model, rows are called **tuples**
- Each **column** has a column header that gives an indication of the meaning of the data items in that column
 - In the formal model, the column header is called an **attribute name** (or just **attribute**)

Example of a Relation

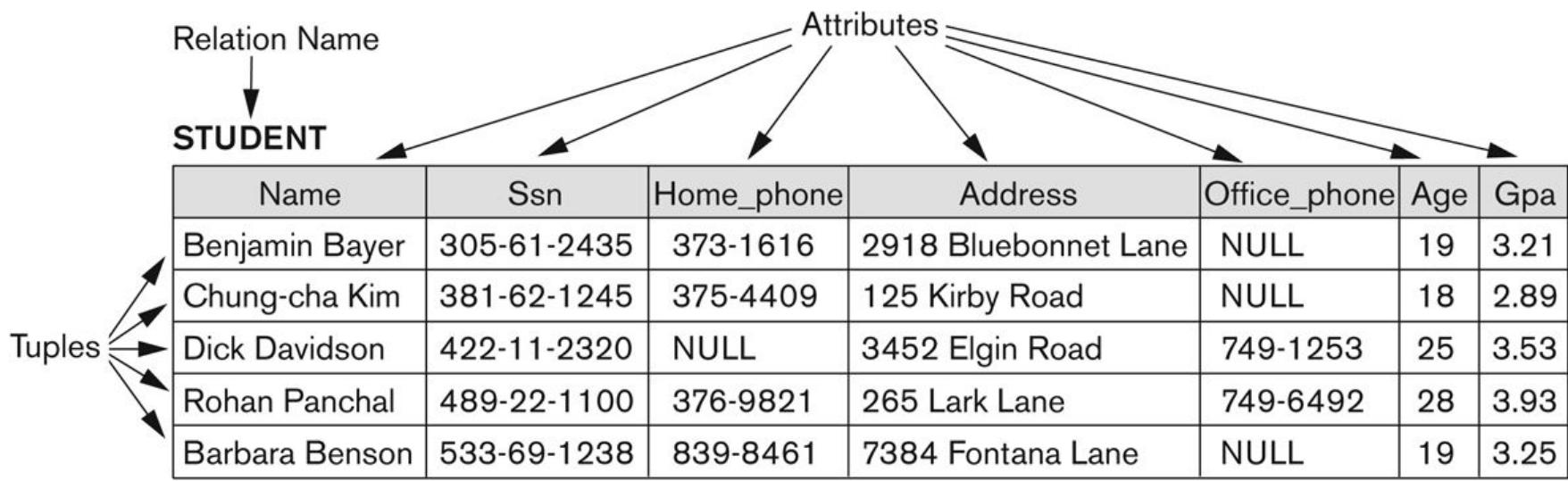


Figure 5.1
The attributes and tuples of a relation STUDENT.

Informal Definitions

■ Key of a Relation:

- Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
 - Called the *key*
- In the STUDENT table, SSN is the key
- Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table
 - Called *artificial key* or *surrogate key*

Formal Definitions - Schema

- The **Schema** (or description) of a Relation:
 - Denoted by $R(A_1, A_2, \dots, A_n)$
 - R is the **name** of the relation
 - The **attributes** of the relation are A_1, A_2, \dots, A_n
- Example:

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

 - **CUSTOMER** is the relation name
 - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
 - For example, the domain of Cust-id is 6 digit numbers.

Formal Definitions - Tuple

- A **tuple** is an ordered set of values (enclosed in angled brackets '< ... >')
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
 - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
 - This is called a 4-tuple as it has 4 values
 - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows)

Formal Definitions - Domain

- A **domain** has a logical definition:
 - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
 - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
 - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.
- The attribute name designates the role played by a domain in a relation:
 - Used to interpret the meaning of the data elements corresponding to that attribute
 - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings

Formal Definitions - State

- The **relation state** is a subset of the Cartesian product of the domains of its attributes
 - each domain contains the set of all possible values the attribute can take.
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
 - $\text{dom}(\text{Cust-name})$ is $\text{varchar}(25)$
- The role these strings play in the CUSTOMER relation is that of the *name of a customer*.

Formal Definitions - Summary

- Formally,
 - Given $R(A_1, A_2, \dots, A_n)$
 - $r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$
- $R(A_1, A_2, \dots, A_n)$ is the **schema** of the relation
- R is the **name** of the relation
- A_1, A_2, \dots, A_n are the **attributes** of the relation
- $r(R)$: a specific **state** (or "value" or "population") of relation R – this is a *set of tuples* (rows)
 - $r(R) = \{t_1, t_2, \dots, t_n\}$ where each t_i is an n -tuple
 - $t_i = \langle v_1, v_2, \dots, v_n \rangle$ where each v_j element-of $\text{dom}(A_j)$

Formal Definitions - Example

- Let $R(A_1, A_2)$ be a relation schema:
 - Let $\text{dom}(A_1) = \{0,1\}$
 - Let $\text{dom}(A_2) = \{a,b,c\}$
- Then: $\text{dom}(A_1) \times \text{dom}(A_2)$ is all possible combinations:
 $\{\langle 0,a \rangle, \langle 0,b \rangle, \langle 0,c \rangle, \langle 1,a \rangle, \langle 1,b \rangle, \langle 1,c \rangle\}$
- The relation state $r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2)$
- For example: $r(R)$ could be $\{\langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle\}$
 - this is one possible state (or “population” or “extension”) r of the relation R , defined over A_1 and A_2 .
 - It has three 2-tuples: $\langle 0,a \rangle, \langle 0,b \rangle, \langle 1,c \rangle$

Definition Summary

<u>Informal Terms</u>	<u>Formal Terms</u>
Table	Relation
Column Header	Attribute
All possible Column Values	Domain
Row	Tuple
Table Definition	Schema of a Relation
Populated Table	State of the Relation

Example – A relation STUDENT

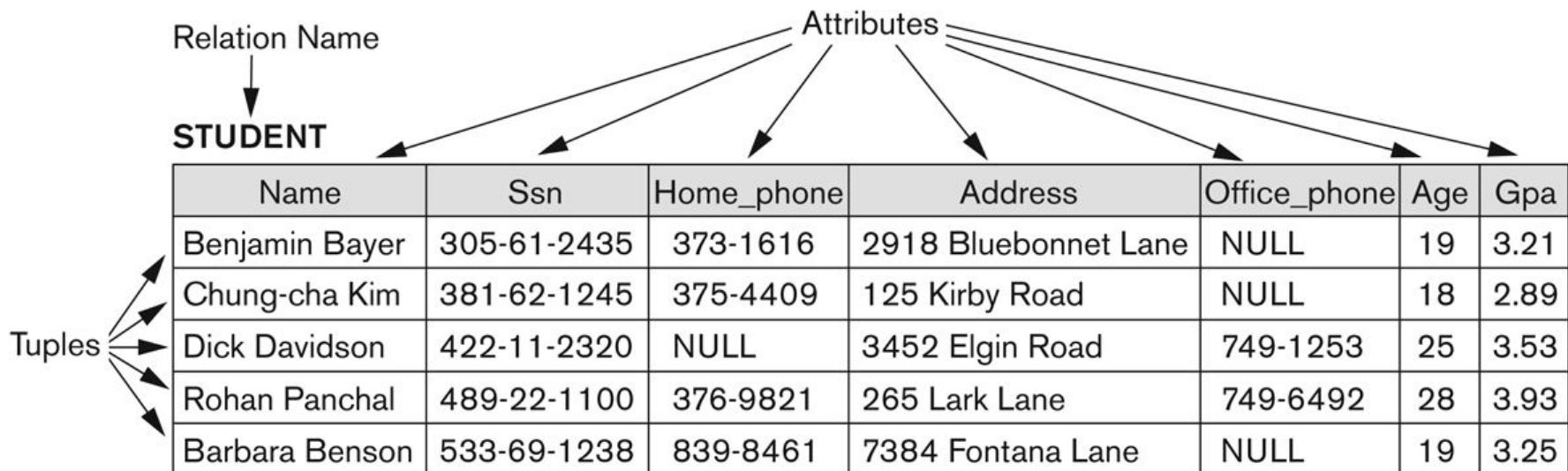


Figure 5.1

The attributes and tuples of a relation STUDENT.

Characteristics Of Relations

- Ordering of tuples in a relation $r(R)$:
 - The tuples are *not considered to be ordered*, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple):
 - We will consider the attributes in $R(A_1, A_2, \dots, A_n)$ and the values in $t = \langle v_1, v_2, \dots, v_n \rangle$ to be ordered .
 - (However, a more general alternative definition of relation does not require this ordering. It includes both the name and the value for each of the attributes).
 - Example: $t = \{ \langle \text{name}, \text{"John"} \rangle, \langle \text{SSN}, 123456789 \rangle \}$
 - This representation may be called as “self-describing”.

Same state as previous Figure (but with different order of tuples)

Figure 5.2

The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

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

Characteristics Of Relations

- Values in a tuple:
 - All values are considered atomic (indivisible).
 - Each value in a tuple must be from the domain of the attribute for that column
 - If tuple $t = \langle v_1, v_2, \dots, v_n \rangle$ is a tuple (row) in the relation state r of $R(A_1, A_2, \dots, A_n)$
 - Then each v_i must be a value from $\text{dom}(A_i)$
 - A special **null** value is used to represent values that are unknown or not available or inapplicable in certain tuples.

Characteristics Of Relations

- Notation:
 - We refer to **component values** of a tuple t by:
 - $t[A_i]$ or $t.A_i$
 - This is the value v_i of attribute A_i for tuple t
 - Similarly, $t[A_u, A_v, \dots, A_w]$ refers to the subtuple of t containing the values of attributes A_u, A_v, \dots, A_w , respectively in t

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 t_1 and t_2 in $r(R)$, $t_1[SK] \neq t_2[SK]$
 - 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

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

Figure 5.4

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

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>	Dlocation
----------------	-----------

PROJECT

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

WORKS_ON

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

DEPENDENT

<u>Essn</u>	Dependent_name	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 t_1 in R1 is said to **reference** a tuple t_2 in R2 if $t_1[FK] = t_2[PK]$.
- 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.

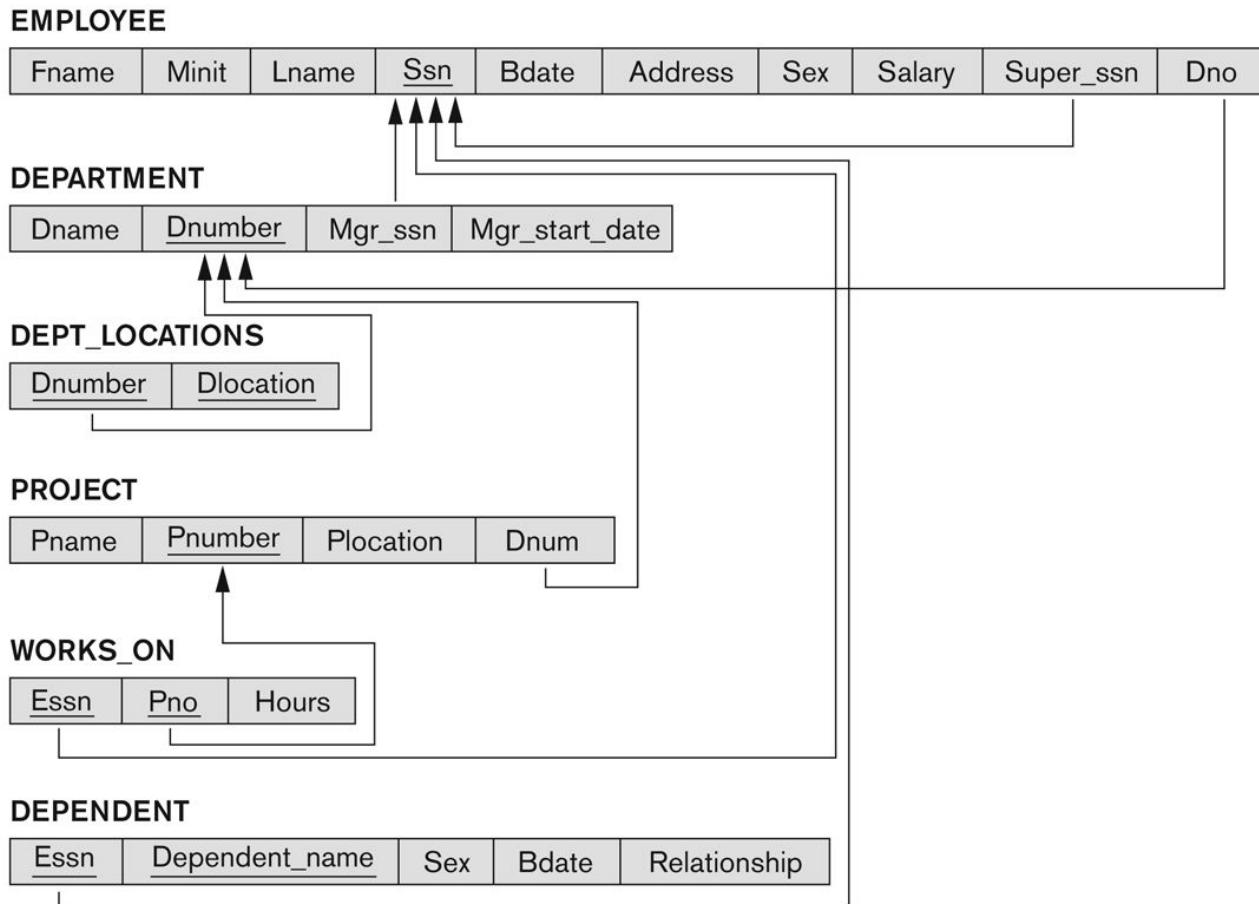
Displaying a relational database schema and its constraints

- 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

Summary

- Presented Relational Model Concepts
 - Definitions
 - Characteristics of relations
- Discussed Relational Model Constraints and Relational Database Schemas
 - Domain constraints
 - Key constraints
 - Entity integrity
 - Referential integrity
- Described the Relational Update Operations and Dealing with Constraint Violations

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)

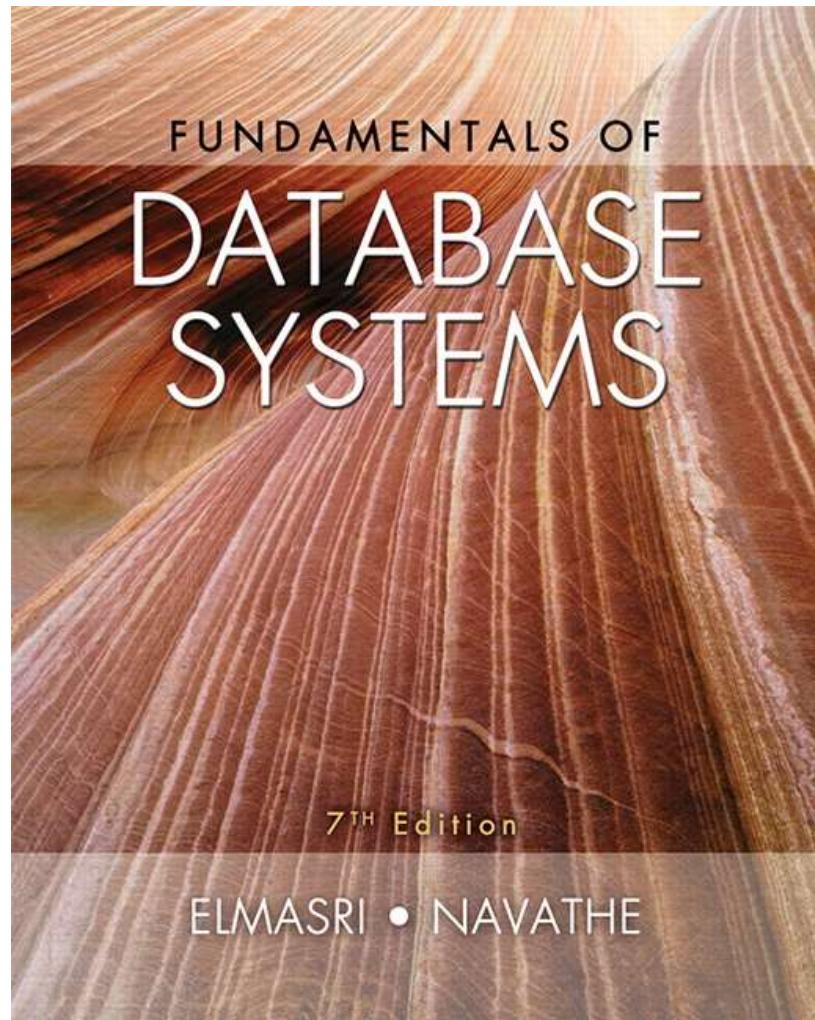
COURSE(Course#, Cname, Dept)

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.



CHAPTER 8

The Relational Algebra and The Relational Calculus (plus QBE- Appendix C)

Chapter Outline

- Relational Algebra
 - Unary Relational Operations
 - Relational Algebra Operations From Set Theory
 - Binary Relational Operations
 - Additional Relational Operations
 - Examples of Queries in Relational Algebra
- Relational Calculus
 - Tuple Relational Calculus
 - Domain Relational Calculus
- Example Database Application (COMPANY)
- Overview of the QBE language (appendix D)

Relational Algebra Overview

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
 - This property makes the algebra “closed” (all objects in relational algebra are relations)

Relational Algebra Overview (continued)

- The **algebra operations** thus produce new relations
 - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
 - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

Brief History of Origins of Algebra

- Muhammad ibn Musa al-Khwarizmi (800-847 CE) – from Morocco wrote a book titled al-jabr about arithmetic of variables
 - Book was translated into Latin.
 - Its title (al-jabr) gave Algebra its name.
- Al-Khwarizmi called variables “shay”
 - “Shay” is Arabic for “thing”.
 - Spanish transliterated “shay” as “xay” (“x” was “sh” in Spain).
 - In time this word was abbreviated as x.
- Where does the word Algorithm come from?
 - Algorithm originates from “al-Khwarizmi”
 - Reference: PBS (<http://www.pbs.org/empires/islam/innoalgebra.html>)

Relational Algebra Overview

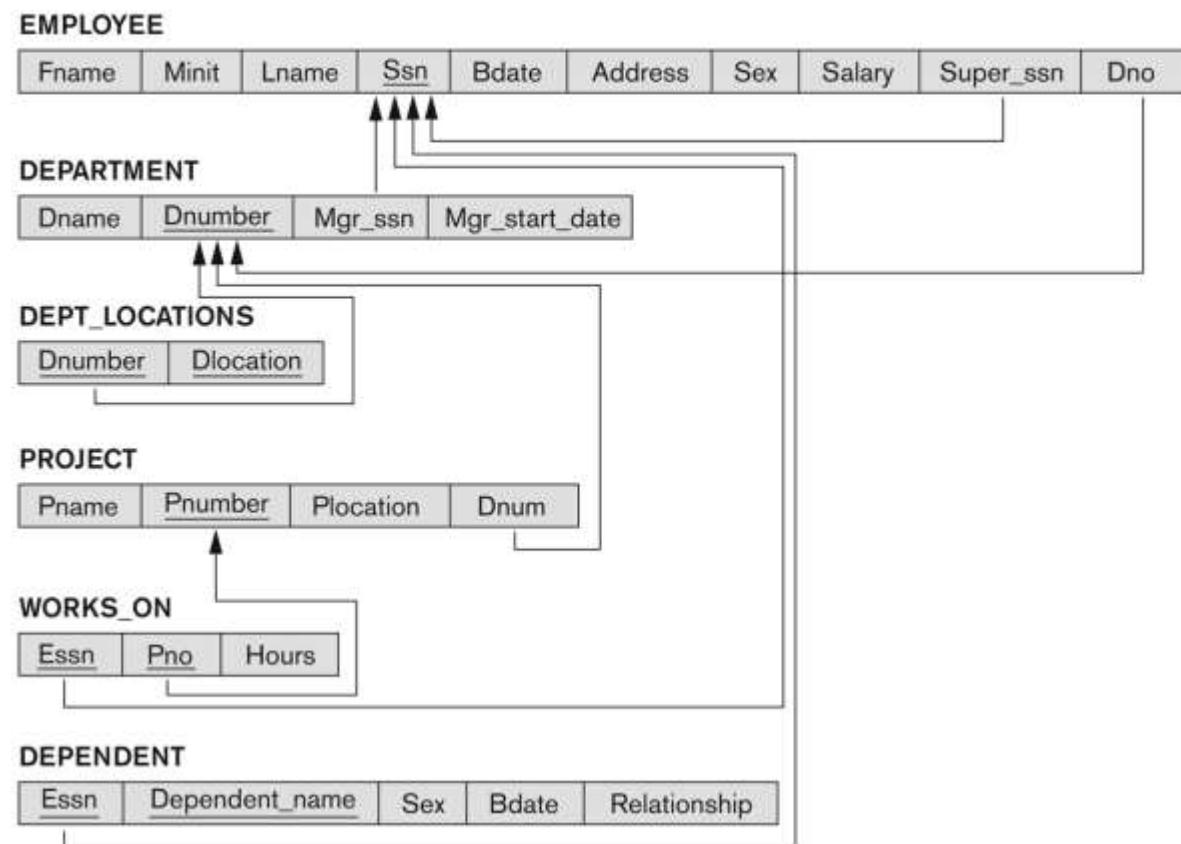
- Relational Algebra consists of several groups of operations
 - Unary Relational Operations
 - SELECT (symbol: σ (sigma))
 - PROJECT (symbol: π (pi))
 - RENAME (symbol: ρ (rho))
 - Relational Algebra Operations From Set Theory
 - UNION (\cup), INTERSECTION (\cap), DIFFERENCE (or MINUS, $-$)
 - CARTESIAN PRODUCT (\times)
 - Binary Relational Operations
 - JOIN (several variations of JOIN exist)
 - DIVISION
 - Additional Relational Operations
 - OUTER JOINS, OUTER UNION
 - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Database State for COMPANY

- All examples discussed below refer to the COMPANY database shown here.

Figure 5.7

Referential integrity constraints displayed on the COMPANY relational database schema.



Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**.
 - The selection condition acts as a **filter**
 - Keeps only those tuples that satisfy the qualifying condition
 - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
 - Select the EMPLOYEE tuples whose department number is 4:
$$\sigma_{DNO = 4} (\text{EMPLOYEE})$$
 - Select the employee tuples whose salary is greater than \$30,000:
$$\sigma_{\text{SALARY} > 30,000} (\text{EMPLOYEE})$$

Unary Relational Operations: SELECT

- In general, the *select* operation is denoted by
 $\sigma_{<\text{selection condition}>}(R)$ where
 - the symbol σ (sigma) is used to denote the *select* operator
 - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
 - tuples that make the condition **true** are selected
 - appear in the result of the operation
 - tuples that make the condition **false** are filtered out
 - discarded from the result of the operation

Unary Relational Operations: SELECT (continued)

■ SELECT Operation Properties

- The SELECT operation $\sigma_{<\text{selection condition}>} (R)$ produces a relation S that has the same schema (same attributes) as R
- SELECT σ is commutative:
 - $\sigma_{<\text{condition1}>} (\sigma_{<\text{condition2}>} (R)) = \sigma_{<\text{condition2}>} (\sigma_{<\text{condition1}>} (R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
 - $\sigma_{<\text{cond1}>} (\sigma_{<\text{cond2}>} (\sigma_{<\text{cond3}>} (R))) = \sigma_{<\text{cond2}>} (\sigma_{<\text{cond3}>} (\sigma_{<\text{cond1}>} (R)))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
 - $\sigma_{<\text{cond1}>} (\sigma_{<\text{cond2}>} (\sigma_{<\text{cond3}>} (R))) = \sigma_{<\text{cond1}> \text{ AND } <\text{cond2}> \text{ AND } <\text{cond3}>} (R))$
- The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R

The following query results refer to this database state

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	1941-08-20	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	1981-08-19

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	1987-05-05	Spouse

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by π (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
 - PROJECT creates a vertical partitioning
 - The list of specified columns (attributes) is kept in each tuple
 - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$$

Unary Relational Operations: PROJECT (cont.)

- The general form of the *project* operation is:

$$\pi_{\langle \text{attribute list} \rangle}(R)$$

- π (pi) is the symbol used to represent the *project* operation
- $\langle \text{attribute list} \rangle$ is the desired list of attributes from relation R.
- The project operation *removes any duplicate tuples*
 - This is because the result of the *project* operation must be a *set of tuples*
 - Mathematical sets *do not allow* duplicate elements.

Unary Relational Operations: PROJECT (contd.)

- PROJECT Operation Properties
 - The number of tuples in the result of projection $\pi_{<\text{list}>}(\text{R})$ is always less or equal to the number of tuples in R
 - If the list of attributes includes a key of R, then the number of tuples in the result of PROJECT is equal to the number of tuples in R
 - PROJECT is *not* commutative
 - $\pi_{<\text{list1}>}(\pi_{<\text{list2}>}(\text{R})) = \pi_{<\text{list1}>}(\text{R})$ as long as $<\text{list2}>$ contains the attributes in $<\text{list1}>$

Examples of applying SELECT and PROJECT operations

Figure 8.1 Results of SELECT and PROJECT operations. (a) $\sigma_{Dno=4 \text{ AND } Salary > 25000} \text{ OR } (Dno=5 \text{ AND } Salary > 30000)$ (EMPLOYEE). (b) $\pi_{Lname, Fname, Salary}$ (EMPLOYEE). (c) $\pi_{Sex, Salary}$ (EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
 - Either we can write the operations as a single **relational algebra expression** by nesting the operations, or
 - We can apply one operation at a time and create **intermediate result relations**.
- In the latter case, we must give names to the relations that hold the intermediate results.

Single expression versus sequence of relational operations (Example)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a *single relational algebra expression* as follows:
 - $\pi_{\text{FNAME}, \text{LNAME}, \text{SALARY}}(\sigma_{\text{DNO}=5}(\text{EMPLOYEE}))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
 - $\text{DEP5_EMPS} \leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE})$
 - $\text{RESULT} \leftarrow \pi_{\text{FNAME}, \text{LNAME}, \text{SALARY}} (\text{DEP5_EMPS})$

Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
 - Useful when a query requires multiple operations
 - Necessary in some cases (see JOIN operation later)

Unary Relational Operations: RENAME (continued)

- The general RENAME operation ρ can be expressed by any of the following forms:
 - $\rho_{S(B_1, B_2, \dots, B_n)}(R)$ changes both:
 - the relation name to S , and
 - the column (attribute) names to B_1, B_1, \dots, B_n
 - $\rho_S(R)$ changes:
 - the *relation name* only to S
 - $\rho_{(B_1, B_2, \dots, B_n)}(R)$ changes:
 - the *column (attribute) names* only to B_1, B_1, \dots, B_n

Unary Relational Operations: RENAME (continued)

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
 - If we write:
 - $\text{RESULT} \leftarrow \pi_{\text{FNAME}, \text{LNAME}, \text{SALARY}}(\text{DEP5_EMPS})$
 - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
 - If we write:
 - $\text{RESULT}(F, M, L, S, B, A, SX, SAL, SU, DNO) \leftarrow \rho_{\text{RESULT}(F.M.L.S.B,A,SX,SAL,SU,DNO)}(\text{DEP5_EMPS})$
 - The 10 attributes of DEP5_EMPS are *renamed* to $F, M, L, S, B, A, SX, SAL, SU, DNO$, respectively

Note: the \leftarrow symbol is an assignment operator

Example of applying multiple operations and RENAME

Figure 8.2 Results of a sequence of operations. (a) $\pi_{\text{Fname}, \text{Lname}, \text{Salary}}(\sigma_{\text{Dno}=3}(\text{EMPLOYEE}))$.
(b) Using intermediate relations and renaming of attributes.

(a)

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

(b)

TEMP

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1985-01-09	731 Fendren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1982-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

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

Relational Algebra Operations from Set Theory: UNION

■ UNION Operation

- Binary operation, denoted by \cup
- The result of $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S
- Duplicate tuples are eliminated
- The two operand relations R and S must be “type compatible” (or UNION compatible)
 - R and S must have same number of attributes
 - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

Relational Algebra Operations from Set Theory: UNION

■ Example:

- To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
- We can use the UNION operation as follows:

$$\text{DEP5_EMPS} \leftarrow \sigma_{\text{DNO}=5}(\text{EMPLOYEE})$$
$$\text{RESULT1} \leftarrow \pi_{\text{SSN}}(\text{DEP5_EMPS})$$
$$\text{RESULT2(SSN)} \leftarrow \pi_{\text{SUPERSSN}}(\text{DEP5_EMPS})$$
$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

- The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Figure 8.3 Result of the UNION operation $\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra Operations from Set Theory

- Type Compatibility of operands is required for the binary set operation UNION \cup , (also for INTERSECTION \cap , and SET DIFFERENCE $-$, see next slides)
- $R1(A1, A2, \dots, An)$ and $R2(B1, B2, \dots, Bn)$ are type compatible if:
 - they have the same number of attributes, and
 - the domains of corresponding attributes are type compatible (i.e. $\text{dom}(Ai)=\text{dom}(Bi)$ for $i=1, 2, \dots, n$).
- The resulting relation for $R1 \cup R2$ (also for $R1 \cap R2$, or $R1 - R2$, see next slides) has the same attribute names as the *first* operand relation $R1$ (by convention)

Relational Algebra Operations from Set Theory: INTERSECTION

- INTERSECTION is denoted by \cap
- The result of the operation $R \cap S$, is a relation that includes all tuples that are in both R and S
 - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be “type compatible”

Relational Algebra Operations from Set Theory: SET DIFFERENCE (cont.)

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of $R - S$, is a relation that includes all tuples that are in R but not in S
 - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be “type compatible”

Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

Figure 8.4 The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT \cup INSTRUCTOR. (c) STUDENT \cap INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

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

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Some properties of UNION, INTERSECT, and DIFFERENCE

- Notice that both union and intersection are *commutative* operations; that is
 - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
 - $R \cup (S \cup T) = (R \cup S) \cup T$
 - $(R \cap S) \cap T = R \cap (S \cap T)$
- The minus operation is not commutative; that is, in general
 - $R - S \neq S - R$

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
 - This operation is used to combine tuples from two relations in a combinatorial fashion.
 - Denoted by $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$
 - Result is a relation Q with degree $n + m$ attributes:
 - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, in that order.
 - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
 - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples.
 - The two operands do NOT have to be "type compatible"

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- Generally, CROSS PRODUCT is not a meaningful operation
 - Can become meaningful when followed by other operations
- Example (not meaningful):
 - $\text{FEMALE_EMPS} \leftarrow \sigma_{\text{SEX}='F'}(\text{EMPLOYEE})$
 - $\text{EMP NAMES} \leftarrow \pi_{\text{FNAME, LNAME, SSN}}(\text{FEMALE_EMPS})$
 - $\text{EMP_DEPENDENTS} \leftarrow \text{EMP NAMES} \times \text{DEPENDENT}$
- EMP_DEPENDENTS will contain every combination of EMP NAMES and DEPENDENT
 - whether or not they are actually related

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows
- Example (meaningful):
 - $\text{FEMALE_EMPS} \leftarrow \sigma_{\text{SEX}='F'}(\text{EMPLOYEE})$
 - $\text{EMPNAMES} \leftarrow \pi_{\text{FNAME, LNAME, SSN}}(\text{FEMALE_EMPS})$
 - $\text{EMP_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$
 - $\text{ACTUAL_DEPS} \leftarrow \sigma_{\text{SSN}=\text{ESSN}}(\text{EMP_DEPENDENTS})$
 - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, DEPENDENT_NAME}}(\text{ACTUAL_DEPS})$
- RESULT will now contain the name of female employees and their dependents

Figure 8.5 The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAME

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

continued on next slide

Figure 8.5 (continued) The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

continued on next slide

Figure 8.5 (continued) The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Binary Relational Operations: JOIN

- JOIN Operation (denoted by \bowtie)
 - The sequence of CARTESIAN PRODUCT followed by SELECT is used quite commonly to identify and select related tuples from two relations
 - A special operation, called JOIN combines this sequence into a single operation
 - This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
 - The general form of a join operation on two relations R(A₁, A₂, . . . , A_n) and S(B₁, B₂, . . . , B_m) is:
$$R \bowtie_{\text{join condition}} S$$
 - where R and S can be any relations that result from general *relational algebra expressions*.

Binary Relational Operations: JOIN (cont.)

- Example: Suppose that we want to retrieve the name of the manager of each department.
 - To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
 - We do this by using the join  operation.
- DEPT_MGR \leftarrow DEPARTMENT  MGRSSN=SSN EMPLOYEE
- MGRSSN=SSN is the join condition
 - Combines each department record with the employee who manages the department
 - The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

Figure 8.6 Result of the JOIN operation

$\text{DEPT_MGR} \leftarrow \text{DEPARTMENT}^{\text{IXI}} \text{ Mgr_ssn} = \text{Ssn} \text{EMPLOYEE}.$

DEPT_MGR

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

Some properties of JOIN

- Consider the following JOIN operation:
 - $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$
 $R.A_i=S.B_j$
 - Result is a relation Q with degree $n + m$ attributes:
 - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, in that order.
 - The resulting relation state has one tuple for each combination of tuples—r from R and s from S, but *only if they satisfy the join condition $r[A_i]=s[B_j]$*
 - Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than $n_R * n_S$* tuples.
 - Only related tuples (based on the join condition) will appear in the result

Some properties of JOIN

- The general case of JOIN operation is called a Theta-join: R  S
theta
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S; for example:
 - $R.A_i < S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p < S.B_q)$
- Most join conditions involve one or more equality conditions “AND”ed together; for example:
 - $R.A_i = S.B_j \text{ AND } R.A_k = S.B_l \text{ AND } R.A_p = S.B_q$

Binary Relational Operations: EQUIJOIN

- EQUIJOIN Operation
- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is $=$, is called an EQUIJOIN.
 - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
 - The JOIN seen in the previous example was an EQUIJOIN.

Binary Relational Operations: NATURAL JOIN Operation

- NATURAL JOIN Operation
 - Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
 - because one of each pair of attributes with identical values is superfluous
 - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
 - If this is not the case, a renaming operation is applied first.

Binary Relational Operations

NATURAL JOIN (continued)

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
 - $\text{DEPT_LOCS} \leftarrow \text{DEPARTMENT} * \text{DEPT_LOCATIONS}$
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:
 $\text{DEPARTMENT.DNUMBER}=\text{DEPT_LOCATIONS.DNUMBER}$

- Another example: $Q \leftarrow R(A,B,C,D) * S(C,D,E)$
 - The implicit join condition includes *each pair* of attributes with the same name, “AND”ed together:
 - $R.C=S.C \text{ AND } R.D=S.D$
 - Result keeps only one attribute of each such pair:
 - $Q(A,B,C,D,E)$

Example of NATURAL JOIN operation

Figure 8.7 Results of two natural join operations. (a) proj_dept ← project * dept. (b) dept_locs ← department * dept_locations.

(a)

PROJ_DEPT

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

(b)

DEPT_LOCS

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

Complete Set of Relational Operations

- The set of operations including SELECT σ , PROJECT π , UNION \cup , DIFFERENCE $-$, RENAME ρ , and CARTESIAN PRODUCT \times is called a *complete* set because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
 - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
 - $R \bowtie_{\text{join condition}} S = \sigma_{\text{join condition}} (R \times S)$

Binary Relational Operations: DIVISION

- DIVISION Operation
 - The division operation is applied to two relations
 - $R(Z) \div S(X)$, where X subset Z . Let $Y = Z - X$ (and hence $Z = X \cup Y$); that is, let Y be the set of attributes of R that are not attributes of S .
 - The result of DIVISION is a relation $T(Y)$ that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with
 - $t_R[X] = t_s$ for every tuple t_s in S .
 - For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S .

Example of DIVISION

Figure 8.8 The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS		SMITH_PNOS	
Essn	Pno	Pno	
123456789	1	1	
123456789	2	2	
6668844444	3		
453453453	1		
453453453	2		
333445555	2		
333445555	3		
333445555	10		
333445555	20		
999887777	30		
999887777	10		
987987987	10		
987987987	30		
987654321	30		
987654321	20		
888665555	20		

(b)

R		S	
A	B	A	
a1	b1	a1	
a2	b1	a2	
a3	b1		
a4	b1		
a1	b2	a3	
a3	b2		
a2	b3	a2	
a3	b3		
a4	b3		
a1	b4	a1	
a2	b4	a2	
a3	b4		

SSNS

Ssn
123456789
453453453

T

B
b1
b4

Table 8.1 Operations of Relational Algebra

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{join attributes 1} \rangle, \langle \text{join attributes 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\langle \text{join condition} \rangle} R_2$, OR $R_1 *_{(\langle \text{join attributes 1} \rangle, \langle \text{join attributes 2} \rangle)} R_2$ $R_2 \text{ OR } R_1 * R_2$

continued on next slide

Table 8.1 Operations of Relational Algebra (continued)

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

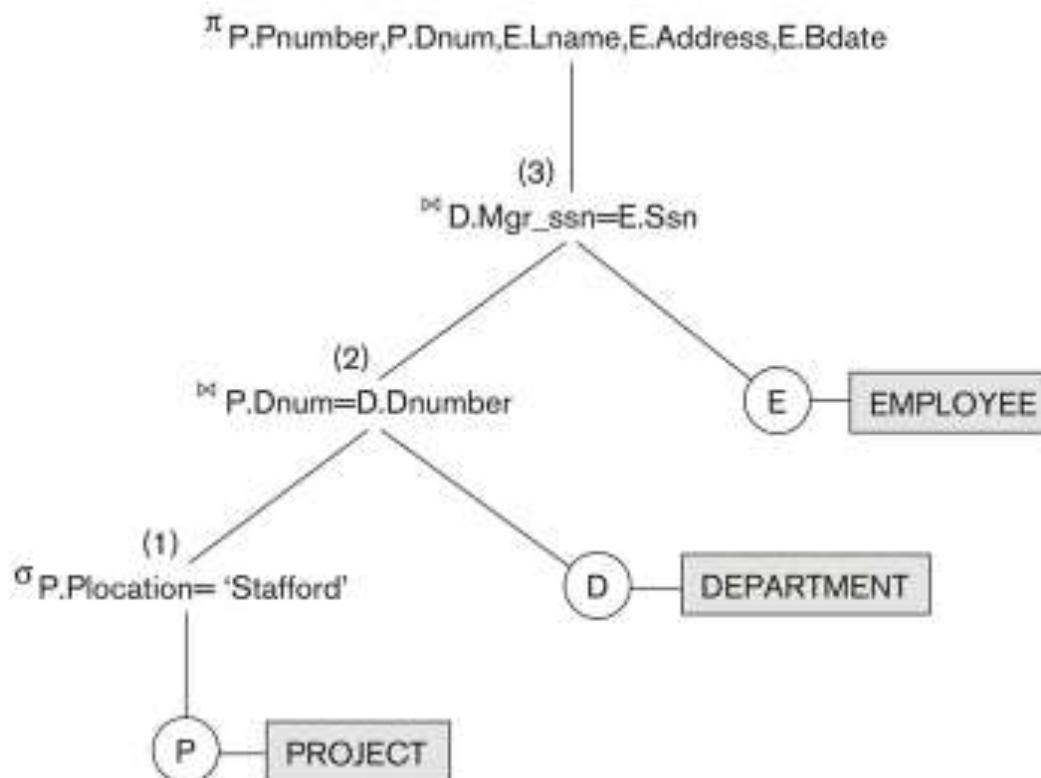
Query Tree Notation

- Query Tree
 - An internal data structure to represent a query
 - Standard technique for estimating the work involved in executing the query, the generation of intermediate results, and the optimization of execution
 - Nodes stand for operations like selection, projection, join, renaming, division,
 - Leaf nodes represent base relations
 - A tree gives a good visual feel of the complexity of the query and the operations involved
 - Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

(see Chapter 15)

Example of Query Tree

Figure 8.9 Query tree corresponding to the relational algebra expression for Q2.



Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
 - These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include
 - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Aggregate Function Operation

- Use of the Aggregate Functional operation \mathcal{F}
 - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
 - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
 - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
 - $\mathcal{F}_{\text{COUNT SSN}, \text{AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
 - Note: count just counts the number of rows, without removing duplicates

Using Grouping with Aggregation

- The previous examples all summarized one or more attributes for a set of tuples
 - Maximum Salary or Count (number of) Ssn
- Grouping can be combined with Aggregate Functions
- Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY
- A variation of aggregate operation \mathcal{F} allows this:
 - Grouping attribute placed to left of symbol
 - Aggregate functions to right of symbol
 - DNO $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}(\text{EMPLOYEE})$
- Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Figure 8.10 The aggregate function operation.

- a. $\rho_R(Dno, No_of_employees, Average_sal)(Dno \Sigma COUNT Ssn, AVERAGE Salary (EMPLOYEE)).$
- b. $Dno \Sigma alary(EMPLOYEE).$
- c. $\Sigma COUNT Ssn, AVERAGE Salary(EMPLOYEE).$

R

(a)

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

(b)

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

(c)

Count_ssn	Average_salary
8	35125

Figure 7.1a Results of GROUP BY and HAVING (in SQL). Q24.

The diagram illustrates the process of generating the result of query Q24 from the EMPLOYEE table. On the left, the EMPLOYEE table is shown with columns: Fname, Minit, Lname, Ssn, ..., Salary, Super_ssn, and Dno. The rows represent individual employees. A vertical ellipsis between the 5th and 6th columns indicates that many more rows exist, all sharing the same value for Dno (5). An arrow points from this ellipsis to the first row of a summary table on the right. This summary table has three columns: Dno, Count (*), and Avg (Salary). It contains three rows corresponding to Dno values 5, 4, and 1. The label "Result of Q24" is placed below this summary table.

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno	Dno	Count (*)	Avg (Salary)
John	B	Smith	123456789	...	30000	333445555	5	5	4	33250
Franklin	T	Wong	333445555		40000	888665555	5		3	31000
Ramesh	K	Narayan	666884444		38000	333445555	5		1	55000
Joyce	A	English	453453453		25000	333445555	5	4	Result of Q24	
Alicia	J	Zelaya	999887777		25000	987654321	4			
Jennifer	S	Wallace	987654321		43000	888665555	4			
Ahmad	V	Jabbar	987987987		25000	987654321	4			
James	E	Bong	888665555		55000	NULL	1			

Grouping EMPLOYEE tuples by the value of Dno

continued on next slide

Additional Relational Operations (continued)

- Recursive Closure Operations
 - Another type of operation that, in general, cannot be specified in the basic original relational algebra is **recursive closure**.
 - This operation is applied to a **recursive relationship**.
 - An example of a recursive operation is to retrieve all SUPERVISEES of an EMPLOYEE e at all levels — that is, all EMPLOYEE e' directly supervised by e ; all employees e'' directly supervised by each employee e' ; all employees e''' directly supervised by each employee e'' ; and so on.

Additional Relational Operations (continued)

- Although it is possible to retrieve employees at each level and then take their union, we cannot, in general, specify a query such as “retrieve the supervisees of ‘James Borg’ at all levels” without utilizing a looping mechanism.
 - The SQL3 standard includes syntax for recursive closure.

Figure 8.11 A two-level recursive query.

SUPERVISION	
(Borg's Ssn is 888665555)	
(Ssn)	(Super_ssn)
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1	
Ssn	
333445555	
987654321	

(Supervised by Borg)

RESULT2	
Ssn	
123456789	
999887777	
666884444	
453453453	
987987987	

(Supervised by
Borg's subordinates)

RESULT	
Ssn	
123456789	
999887777	
666884444	
453453453	
987987987	
333445555	
987654321	

(RESULT1 \cup RESULT2)

Additional Relational Operations (continued)

- The OUTER JOIN Operation
 - In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result
 - Tuples with null in the join attributes are also eliminated
 - This amounts to loss of information.
 - A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.

Additional Relational Operations (continued)

- The left outer join operation keeps every tuple in the first or left relation R in $R \bowtie S$; if no matching tuple is found in S, then the attributes of S in the join result are filled or “padded” with null values.
- A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of $R \ltimes S$.
- A third operation, full outer join, denoted by $\bowtie\bowtie$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Figure 8.12 The result of a LEFT OUTER JOIN operation.

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Additional Relational Operations (continued)

■ OUTER UNION Operations

- The outer union operation was developed to take the union of tuples from two relations if the relations are *not type compatible*.
- This operation will take the union of tuples in two relations $R(X, Y)$ and $S(X, Z)$ that are **partially compatible**, meaning that only some of their attributes, say X , are type compatible.
- The attributes that are type compatible are represented only once in the result, and those attributes that are not type compatible from either relation are also kept in the result relation $T(X, Y, Z)$.

Additional Relational Operations (continued)

- Example: An outer union can be applied to two relations whose schemas are STUDENT(Name, SSN, Department, Advisor) and INSTRUCTOR(Name, SSN, Department, Rank).
 - Tuples from the two relations are matched based on having the same combination of values of the shared attributes— Name, SSN, Department.
 - If a student is also an instructor, both Advisor and Rank will have a value; otherwise, one of these two attributes will be null.
 - The result relation STUDENT_OR_INSTRUCTOR will have the following attributes:

STUDENT_OR_INSTRUCTOR (Name, SSN, Department, Advisor, Rank)

Examples of Queries in Relational Algebra : Procedural Form

- **Q1: Retrieve the name and address of all employees who work for the ‘Research’ department.**

$\text{RESEARCH_DEPT} \leftarrow \sigma_{\text{DNAME}=\text{'Research'}}(\text{DEPARTMENT})$

$\text{RESEARCH_EMPS} \leftarrow (\text{RESEARCH_DEPT} \bowtie_{\text{DNUMBER} = \text{DNOEMPLOYEE}} \text{EMPLOYEE})$

$\text{RESULT} \leftarrow \pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}}(\text{RESEARCH_EMPS})$

- **Q6: Retrieve the names of employees who have no dependents.**

$\text{ALL_EMPS} \leftarrow \pi_{\text{SSN}}(\text{EMPLOYEE})$

$\text{EMPS_WITH_DEPS(SSN)} \leftarrow \pi_{\text{ESSN}}(\text{DEPENDENT})$

$\text{EMPS_WITHOUT_DEPS} \leftarrow (\text{ALL_EMPS} - \text{EMPS_WITH_DEPS})$

$\text{RESULT} \leftarrow \pi_{\text{LNAME}, \text{FNAME}}(\text{EMPS_WITHOUT_DEPS} * \text{EMPLOYEE})$

Examples of Queries in Relational Algebra – Single expressions

As a single expression, these queries become:

- **Q1: Retrieve the name and address of all employees who work for the ‘Research’ department.**

$$\pi_{\text{Fname, Lname, Address}} (\sigma_{\text{Dname} = \text{'Research'}}$$
$$(\text{DEPARTMENT} \bowtie_{\text{Dnumber}=\text{Dno}} (\text{EMPLOYEE}))$$

- **Q6: Retrieve the names of employees who have no dependents.**

$$\pi_{\text{Lname, Fname}} ((\pi_{\text{Ssn}} (\text{EMPLOYEE}) - \rho_{\text{Ssn}} (\pi_{\text{Essn}} (\text{DEPENDENT}))) * \text{EMPLOYEE})$$

Relational Calculus

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

Relational Calculus (continued)

- Relational calculus is considered to be a **nonprocedural** or **declarative** language.
- This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form

$$\{t \mid \text{COND}(t)\}$$

- where t is a tuple variable and $\text{COND }(t)$ is a conditional expression involving t .
- The result of such a query is the set of all tuples t that satisfy $\text{COND }(t)$.

Tuple Relational Calculus (continued)

- Example: To find the first and last names of all employees whose salary is above \$50,000, we can write the following tuple calculus expression:

$$\{t.\text{FNAME}, t.\text{LNAME} \mid \text{EMPLOYEE}(t) \text{ AND } t.\text{SALARY} > 50000\}$$

- The condition $\text{EMPLOYEE}(t)$ specifies that the **range relation** of tuple variable t is EMPLOYEE .
- The first and last name (PROJECTION $\pi_{\text{FNAME}, \text{LNAME}}$) of each EMPLOYEE tuple t that satisfies the condition $t.\text{SALARY} > 50000$ (SELECTION $\sigma_{\text{SALARY} > 50000}$) will be retrieved.

The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (\forall) and the existential quantifier (\exists).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an $(\forall t)$ or $(\exists t)$ clause; otherwise, it is free.
- If F is a formula, then so are $(\exists t)(F)$ and $(\forall t)(F)$, where t is a tuple variable.
 - The formula $(\exists t)(F)$ is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F ; otherwise $(\exists t)(F)$ is false.
 - The formula $(\forall t)(F)$ is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F ; otherwise $(\forall t)(F)$ is false.

The Existential and Universal Quantifiers (continued)

- \forall is called the universal or “for all” quantifier because every tuple in “the universe of” tuples must make F true to make the quantified formula true.
- \exists is called the existential or “there exists” quantifier because any tuple that exists in “the universe of” tuples may make F true to make the quantified formula true.

Example Query Using Existential Quantifier

- Retrieve the name and address of all employees who work for the ‘Research’ department. The query can be expressed as :
$$\{t.FNAME, t.LNAME, t.ADDRESS \mid EMPLOYEE(t) \text{ and } (\exists d) (DEPARTMENT(d) \text{ and } d.DNAME='Research' \text{ and } d.DNUMBER=t.DNO) \}$$
- The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|).
 - In above query, t is the only free variable; it is then *bound successively* to each tuple.
- If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.
 - The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d.
 - The condition d.DNAME = ‘Research’ is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

Example Query Using Universal Quantifier

- Find the names of employees who work on *all* the projects controlled by department number 5. The query can be:
$$\{e.\text{LNAME}, e.\text{FNAME} \mid \text{EMPLOYEE}(e) \text{ and } ((\forall x)(\text{not}(\text{PROJECT}(x)) \text{ or } \text{not}(x.\text{DNUM}=5))$$

$$\text{OR } ((\exists w)(\text{WORKS_ON}(w) \text{ and } w.\text{ESSN}=e.\text{SSN} \text{ and } x.\text{PNUMBER}=w.\text{PNO})))\}$$
- Exclude from the universal quantification all tuples that we are not interested in by making the condition true *for all such tuples*.
 - The first tuples to exclude (by making them evaluate automatically to true) are those that are not in the relation R of interest.
- In query above, using the expression **not(PROJECT(x))** inside the universally quantified formula evaluates to true all tuples x that are not in the PROJECT relation.
 - Then we exclude the tuples we are not interested in from R itself. The expression **not(x.DNUM=5)** evaluates to true all tuples x that are in the project relation but are not controlled by department 5.
- Finally, we specify a condition that must hold on all the remaining tuples in R.
$$((\exists w)(\text{WORKS_ON}(w) \text{ and } w.\text{ESSN}=e.\text{SSN} \text{ and } x.\text{PNUMBER}=w.\text{PNO})$$

Languages Based on Tuple Relational Calculus

- The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:
 - SELECT <list of attributes>
 - FROM <list of relations>
 - WHERE <conditions>
- SELECT clause mentions the attributes being projected, the FROM clause mentions the relations needed in the query, and the WHERE clause mentions the selection as well as the join conditions.
 - SQL syntax is expanded further to accommodate other operations. (See Chapter 8).

Languages Based on Tuple Relational Calculus (continued)

- Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:
 - RANGE OF <variable name> IS <relation name>
- Then it uses
 - RETRIEVE <list of attributes from range variables>
 - WHERE <conditions>
- This language was proposed in the relational DBMS INGRES. (system is currently still supported by Computer Associates – but the QUEL language is no longer there).

The Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
 - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
 - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.

The Domain Relational Calculus (continued)

- An expression of the domain calculus is of the form

$$\{ x_1, x_2, \dots, x_n \mid \text{COND}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}) \}$$

- where $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$ are domain variables that range over domains (of attributes)
- and COND is a condition or formula of the domain relational calculus.

Example Query Using Domain Calculus

Retrieve the birthdate and address of the employee whose name is ‘John B. Smith’.

- Query :

$\{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$
 $(\text{EMPLOYEE}(qrstuvwxyz) \text{ and } q='John' \text{ and } r='B' \text{ and } s='Smith')\}$

- Abbreviated notation **EMPLOYEE(qrstuvwxyz)** uses the variables without the separating commas: **EMPLOYEE(q,r,s,t,u,v,w,x,y,z)**
- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.
 - Of the ten variables q, r, s, . . . , z, only u and v are free.
- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.
- Specify the condition for selecting a tuple following the bar (|)—
 - namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be ‘John’, ‘B’, and ‘Smith’, respectively.

QBE: A Query Language Based on Domain Calculus (Appendix C)

- This language is based on the idea of giving an example of a query using “example elements” which are nothing but domain variables.
- Notation: An example element stands for a domain variable and is specified as an example value preceded by the underscore character.
- P. (called **P dot**) operator (for “print”) is placed in those columns which are requested for the result of the query.
- A user may initially start giving actual values as examples, but later can get used to providing a minimum number of variables as example elements.

QBE: A Query Language Based on Domain Calculus (Appendix C)

- The language is very user-friendly, because it uses minimal syntax.
- QBE was fully developed further with facilities for grouping, aggregation, updating etc. and is shown to be equivalent to SQL.
- The language is available under QMF (Query Management Facility) of DB2 of IBM and has been used in various ways by other products like ACCESS of Microsoft, and PARADOX.
- For details, see **Appendix C** in the text.

QBE Examples

- QBE initially presents a relational schema as a “blank schema” in which the user fills in the query as an example:

Example Schema as a QBE Query Interface

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure C.1

The relational schema of Figure 5.5 as it may be displayed by QBE.

QBE Examples

- The following domain calculus query can be successively minimized by the user as shown:
- Query :

{uv | ($\exists q$) ($\exists r$) ($\exists s$) ($\exists t$) ($\exists w$) ($\exists x$) ($\exists y$) ($\exists z$)
**(EMPLOYEE(qrstuvwxyz) and q='John' and r='B' and
s='Smith')**}

Four Successive Ways to Specify a QBE Query

(a) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	_123456789	P_9/1/60	P_100 Main, Houston, TX	_M	_25000	_123456789	_3

(b) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P_9/1/60	P_100 Main, Houston, TX				

(c) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P_X	P_Y				

(d) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P.	P.				

Figure C.2

Four ways to specify the query Q0 in QBE.

QBE Examples

- Specifying complex conditions in QBE:
- A technique called the “condition box” is used in QBE to state more involved Boolean expressions as conditions.
- The C.4(a) gives employees who work on either project 1 or 2, whereas the query in C.4(b) gives those who work on both the projects.

Complex Conditions with and without a condition box as a part of QBE Query

WORKS_ON		
(a)	Essn	Pno
	P.	> 20

WORKS_ON		
(b)	Essn	Pno
	P.	_PX

CONDITIONS

_HX > 20 and (PX = 1 or PX = 2)

WORKS_ON		
(c)	Essn	Pno
	P.	1
	P.	2

Figure C.3

Specifying complex conditions in QBE. (a) The query Q0A. (b) The query Q0B with a condition box. (c) The query Q0B without a condition box.

Handling AND conditions in a QBE Query

WORKS_ON

(a)

Essn	Pno	Hours
P_ES	1	
P_ES	2	

WORKS_ON

(b)

Essn	Pno	Hours
P_EX	1	
P_EY	2	

CONDITIONS

$_EX = _EY$

Figure C.4

Specifying EMPLOYEES who work on both projects. (a) Incorrect specification of an AND condition. (b) Correct specification.

JOIN in QBE : Examples

- The join is simply accomplished by using the same example element (variable with underscore) in the columns being joined from different (or same as in C.5 (b)) relation.
- Note that the Result is set us as an independent table to show variables from multiple relations placed in the result.

Performing Join with common example elements and use of a RESULT relation

Figure C.5

Illustrating JOIN and result relations in QBE. (a) The query Q1. (b) The query Q8.

(a) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
_FN		_LN			_Addr				_DX

DEPARTMENT

Dname	Dnumber	Mgrssn	Mgr_start_date
Research	_DX		

RESULT			
P.	_FN	_LN	_Addr

(b) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
_E1		_E2						_Xssn	
_S1		_S2	_Xssn						

RESULT				
P.	_E1	_E2	_S1	_S2

AGGREGATION in QBE

- Aggregation is accomplished by using .CNT for count,.MAX, .MIN, .AVG for the corresponding aggregation functions
- Grouping is accomplished by .G operator.
- Condition Box may use conditions on groups (similar to HAVING clause in SQL – see Section 8.5.8)

AGGREGATION in QBE : Examples

(a) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
							P.CNT.		

(b) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
							P.CNT.ALL		

(c) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
			P.CNT.ALL				P.AVG.ALL		P.G.

(d) PROJECT

Pname	Pnumber	Plocation	Dnum
P.	_PX		

WORKS_ON

Essn	Pno	Hours
P.CNT.EX	G._PX	

CONDITIONS

CNT_EX > 2

Figure C.6

Functions and grouping in QBE. (a) The query Q23.
(b) The query Q23A. (c) The query Q24. (d) The query Q26.

NEGATION in QBE : Example

Figure C.7

Illustrating negation by the query Q6.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
P.		P.	_SX						

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
_SX				

UPDATING in QBE : Examples

(a)

EMPLOYEE

I.	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
I.	Richard	K	Marini	653298653	30-Dec-52	98 Oak Forest, Katy, TX	M	37000	987654321	4

(b)

EMPLOYEE

D.	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
D.				653298653						

(c)

EMPLOYEE

	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
	John		Smith					U._S*1.1		U.4

Figure C.8

Modifying the database in QBE. (a) Insertion. (b) Deletion. (c) Update in QBE.

Chapter Summary

- Relational Algebra
 - Unary Relational Operations
 - Relational Algebra Operations From Set Theory
 - Binary Relational Operations
 - Additional Relational Operations
 - Examples of Queries in Relational Algebra
- Relational Calculus
 - Tuple Relational Calculus
 - Domain Relational Calculus
- Overview of the QBE language (appendix C)