Database Management System (DBMS)

Avinash V. Gondal

email: avinash.gondal@gmail.com

Good database design will get you through poor programming **better than good programming** will get you through poor database design….

Module 3

Relational Model

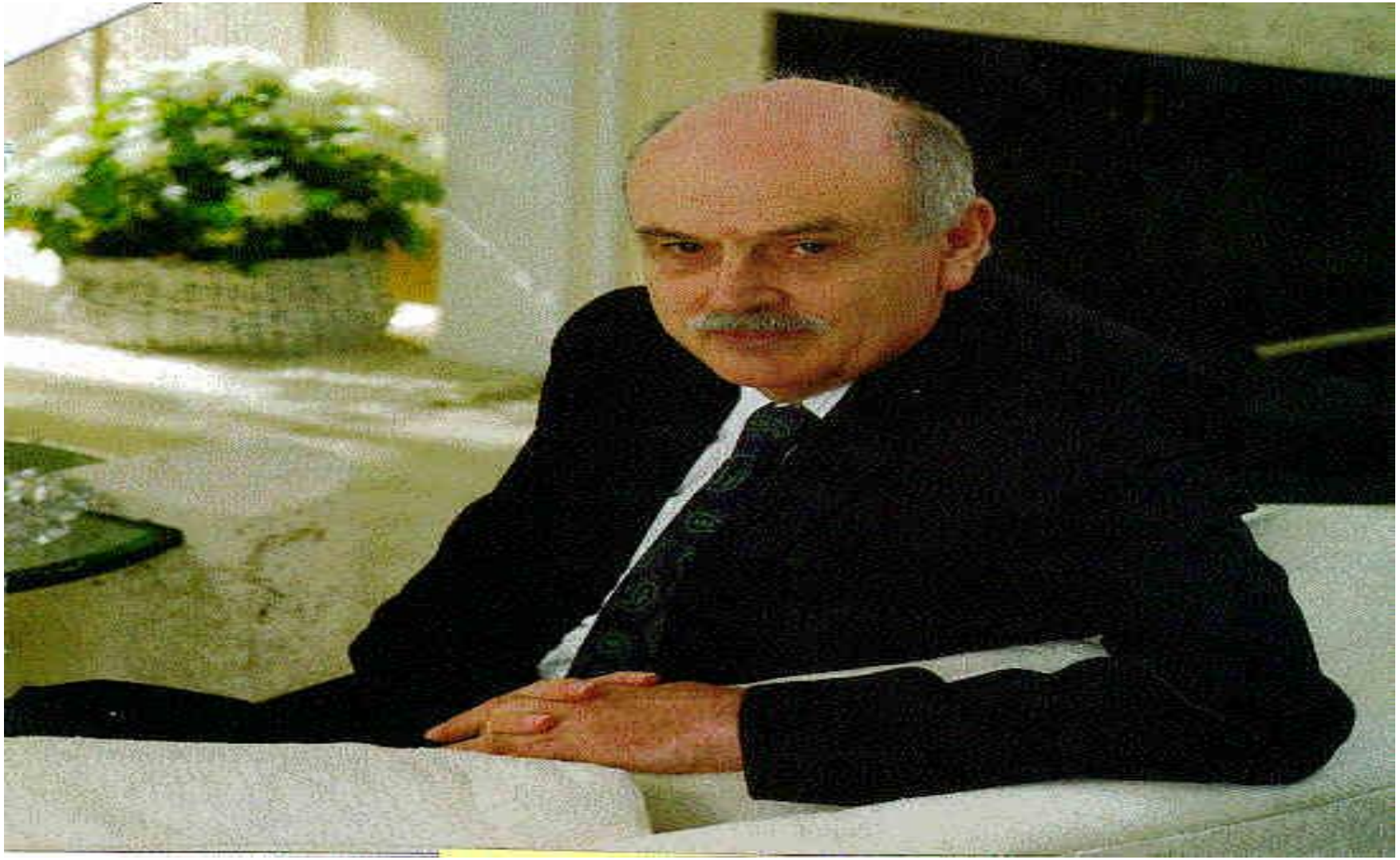
3. Relational Model

Contents :

- Introduction
- Mapping the ER and EER Model to the Relational Model
- Data Manipulation
- Data Integrity
- Advantages of the Relational Model
- Relational Algebra
- Relational Algebra Queries
- Relational Calculus

Introduction

E. F. Codd Our Hero



Introduction

- The **relational data model** was first introduced by E.F. Codd of IBM research at 1970. The relation model is based on the concept of mathematical relation. It uses set theory and first-order predicate logic as theoretical basis.
- The relational model represents both data and the relationships among those data using relation.
- A relation is used to represent information about any entity (such as book, author etc.) and its relationship with other entities in the form of attributes (or columns) and tuples (or rows).

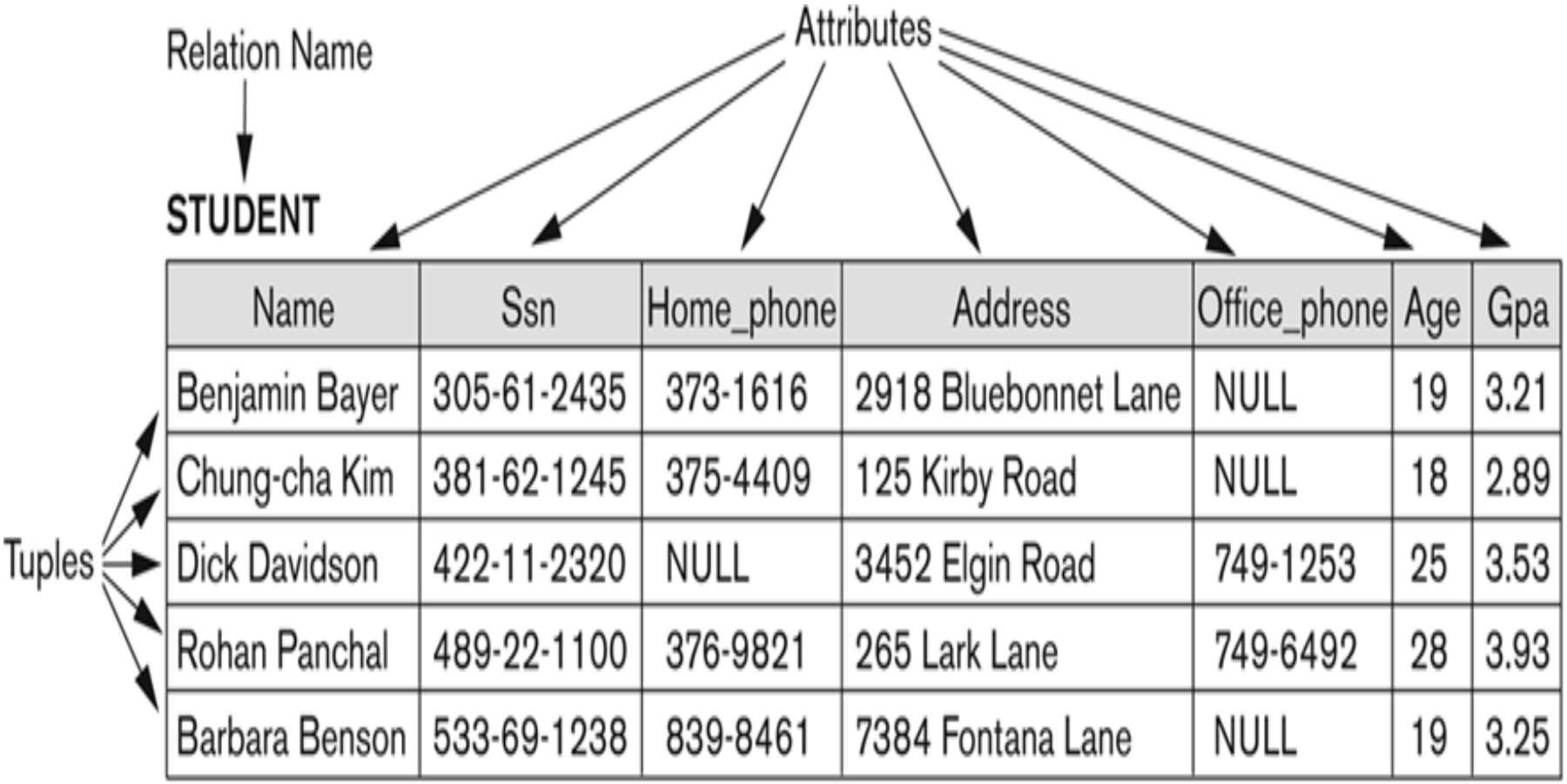
Introduction

- A relation schema (also termed as relation intension) depicts the attributes of the table.
- A relation instance (also termed as relation extension) is a two dimensional table with a time-varying set of tuples.

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



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

Figure

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(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An
- 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
  - dom(Cust-name) is 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(A1, A2, ..., An)
  - $r(R) \subset dom(A1) \times dom(A2) \times ....X\ dom(An)$
- R(A1, A2, …, An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, …, An 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) = {t1, t2, …, tn} where each ti is an n-tuple
  - ti = <v1, v2, …, vn> where each vj *element-of* dom(Aj)

Definition Summary

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

Basic Concepts

Concepts of Relational Databases

- A relational database is represented as a collection of **relations.**
- A relation is physically expressed as a **table** in which each row corresponds to individual **record** and each column corresponds to individual **attribute,** where attributes can appear in any order within the table.
- In formal relational data model terminology, a row is called a **tuple**, a column header is called an attribute and the table is called a relation.
- The **degree** (or **arity**) of a relation is represented by the number of attributes it contains.

Concepts of Relational Databases

- The **cardinality** of a relation is represented by the number of tuples it contains.
- The set of permitted values for each attribute is known as the **domain** of the attribute.
- A relation R defined over a set of n attributes A1, A2, …., An with domains D1, D2, ……, Dn is a set of n-tuples denoted by <d1, d2, ……., dn>, such that d1∈ D1, d2∈ D2, ……., dn∈ Dn. Hence, the set {A1:D1, A2:D2, ……., An:Dn) represents the relational schema.

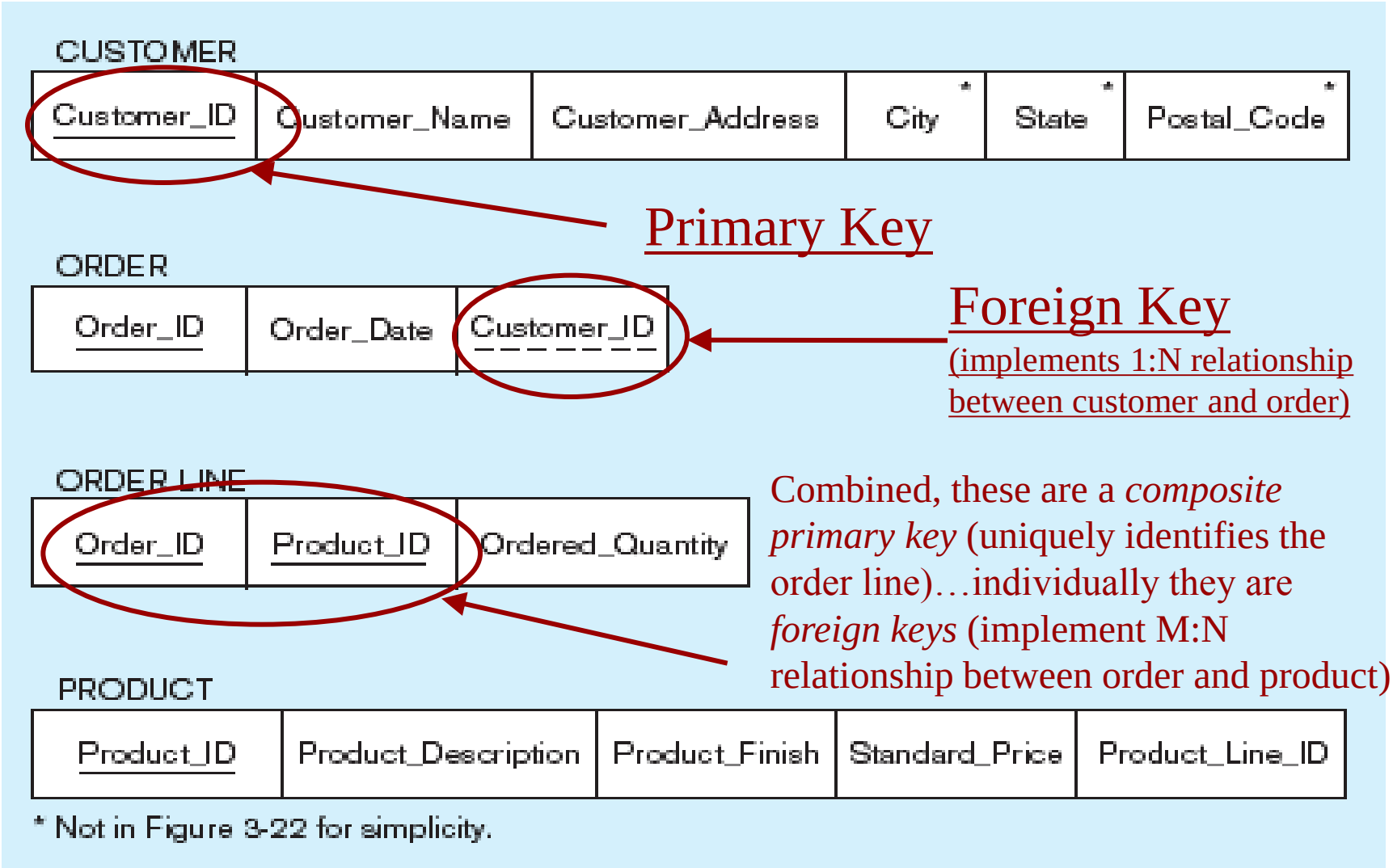
Concepts of Relational Databases

- The **key** of a relation is the nonempty subset of its attributes, which is used to identify each tuple uniquely from a relation.
- The attributes of a relation that make up the key are called **prime attributes** or **key attributes**.
- The attributes of a relation that do not participate in key formation are called **non-prime attributes** or **non-key attributes**.

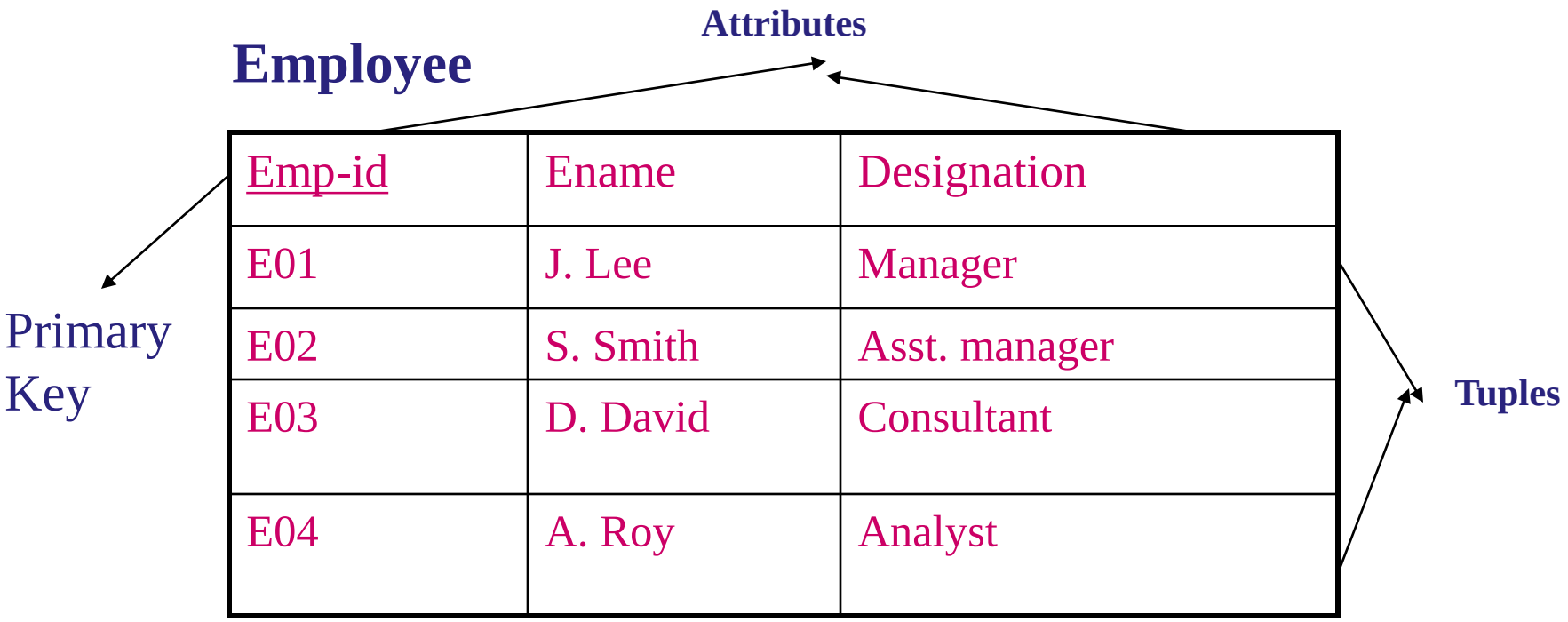
Keys in Relational Data Model

- The superset of a key is usually known as **superkey**.
- A **candidate key** is a superkey such that no proper subset is a superkey within the relation. Thus, a minimal superkey is called candidate key.
- A relational schema can have more than one candidate key among which one is chosen as **primary key** during design time.
- The candidate keys that are not selected as primary key are known as **alternate keys**.
- A **foreign key** is an attribute or a set of attributes within one relation that matches the candidate key of some other relation (possibly the same relation).

Schema for four relations (Furniture Company)



An Example



Emp-id	Ename	Designation
E01	J. Lee	Manager
E02	S. Smith	Asst. manager
E03	D. David	Consultant
E04	A. Roy	Analyst

The Relation Employee

Characteristics of Relations

Characteristics of Relations

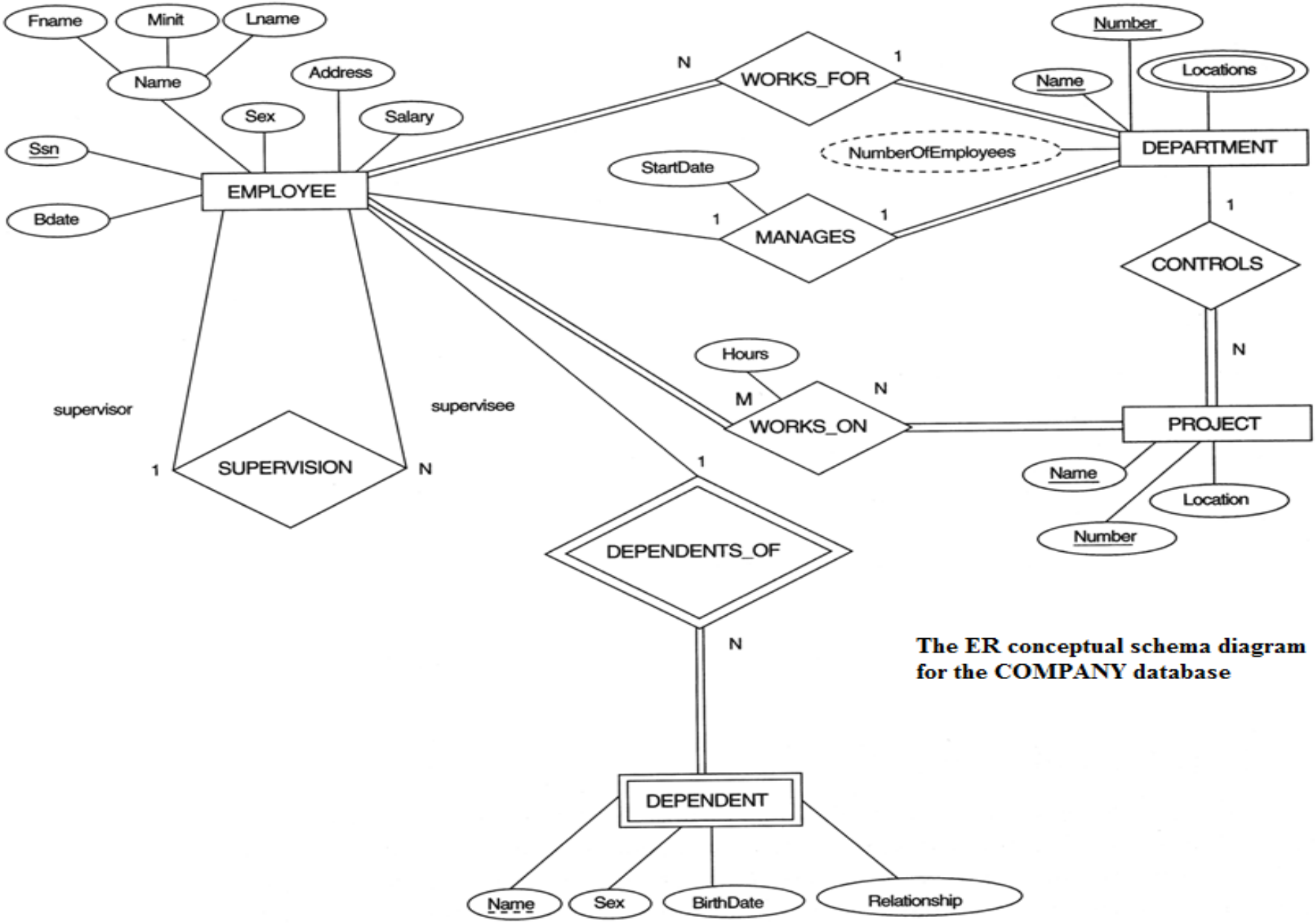
- Table name is distinct from all other table names in the database.
- Each cell of table contains exactly one atomic (single) value.
- Each column has a distinct name.
- Values of a column are all from the same domain.
- Each record is distinct; there are no duplicate records.
- Order of columns has no significance.
- Order of records has no significance, theoretically.

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

Outline

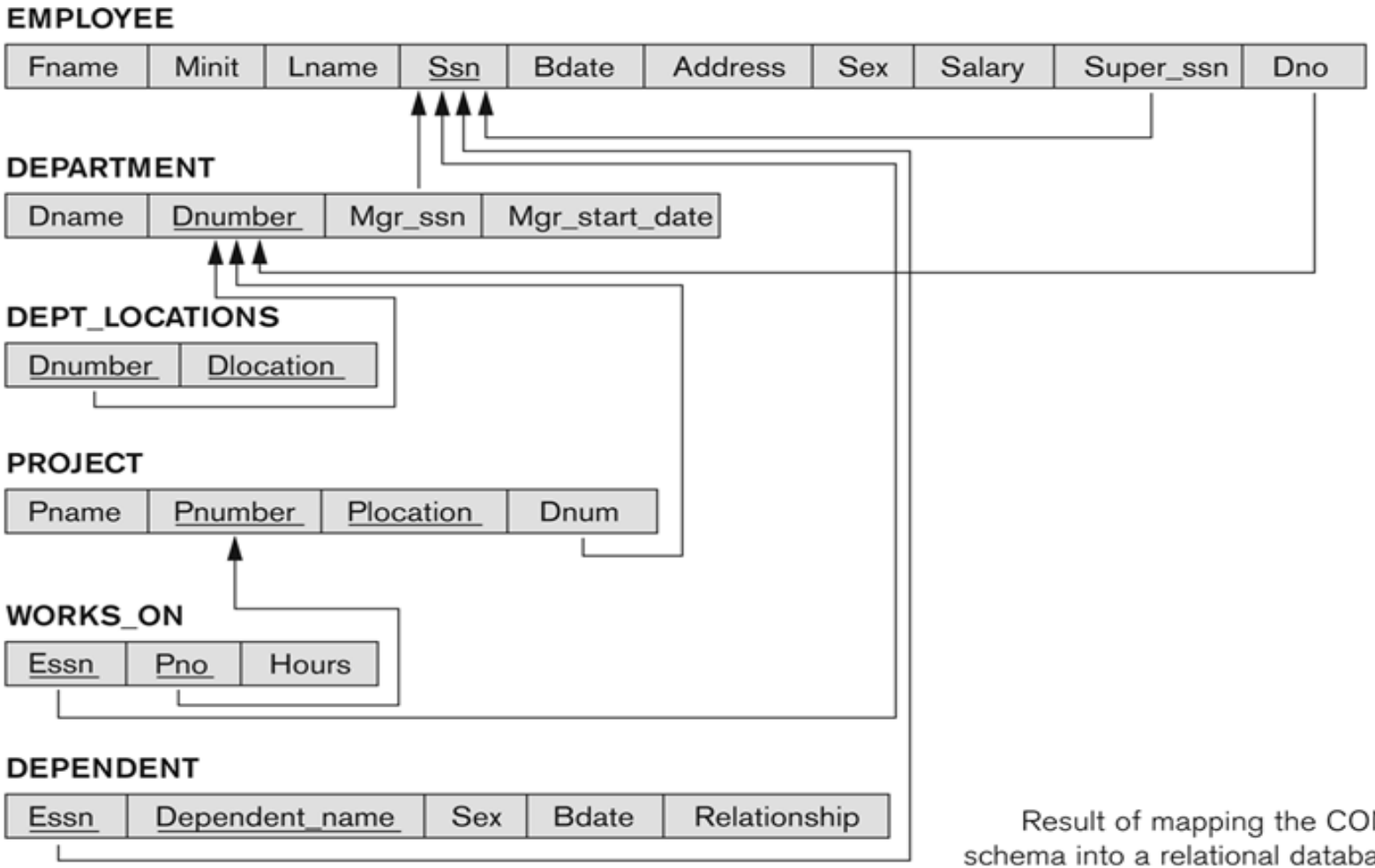
- **ER-to-Relational Mapping Algorithm**
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relation Types
  - Step 4: Mapping of Binary 1:N Relationship Types.
  - Step 5: Mapping of Binary M:N Relationship Types.
  - Step 6: Mapping of Multivalued attributes.
  - Step 7: Mapping of N-ary Relationship Types.

- **Mapping EER Model Constructs to Relations**
  - Step 8: Options for Mapping Specialization or Generalization.
  - Step 9: Mapping of Union Types (Categories).



The ER conceptual schema diagram for the COMPANY database

Result of mapping the COMPANY ER schema into a relational schema.



Result of mapping the COMPANY ER schema into a relational database schema.

Step 1 : Mapping of Regular Entity Types

- Step 1: Mapping of Regular Entity Types.
  - For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - Choose one of the key attributes of E as the primary key for R.
  - If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Step 1: Mapping of Regular Entity Types

- Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
  - SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.

- Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).

- The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Step 2: Mapping of Weak Entity Types

- **Example:** Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.

  – Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).

  – The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT

Step 3: Mapping of Binary 1:1 Relation Types

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

Step 3: Mapping of Binary 1:1 Relation Types

– **Foreign Key approach:** Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with **total** participation in R in the role of S.

- Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.

- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.

- Include any simple attributes of the 1:N relation type as attributes of S.

Step 4: Mapping of Binary 1:N Relationship Types

- Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure.

  – For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

Step 5: Mapping of Binary M:N Relationship Types

- For each regular binary M:N relationship type R, *create a new relation* S to represent R.

- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; *their combination will form the primary key* of S.

- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Step 5: Mapping of Binary M:N Relationship Types

- Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema.
  - The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
  - Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

Step 6: Mapping of Multivalued attributes

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

Step 6: Mapping of Multivalued attributes

- **Example:** The relation DEPT_LOCATIONS is created.
  - The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation.
  - The primary key of R is the combination of {DNUMBER, DLOCATION}.

DEPT_LOCATIONS

Dnumber	Dlocation

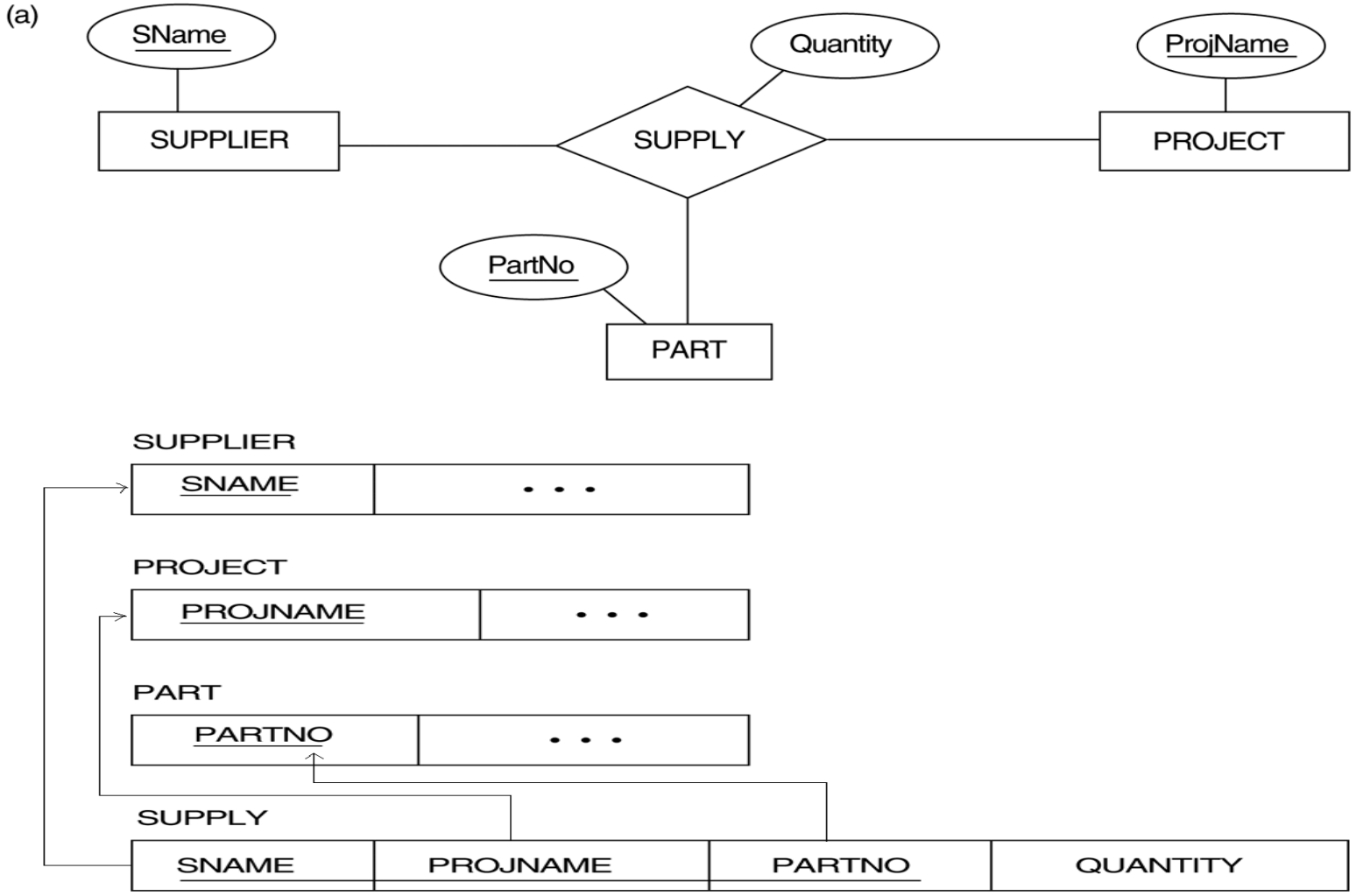
Step 7: Mapping of N-ary Relationship Types

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

Step 7: Mapping of N-ary Relationship Types

- **Example:** The relationship type SUPPY in the ER on the next slide.
  - This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

Step 7: Mapping of N-ary Relationship Types



Summary of Mapping Constructs and Constraints

Table : Correspondence between ER and Relational Models

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

Mapping EER Model Constructs to Relations

- Step 8: Options for Mapping Specialization or Generalization.
- Step 9: Mapping of Union Types (Categories).

Step8: Options for Mapping Specialization or Generalization

- Option 8A: Multiple relations-Superclass and subclasses
- Option 8B: Multiple relations-Subclass relations only
- Option 8C: Single relation with one type attribute
- Option 8D: Single relation with multiple type attributes

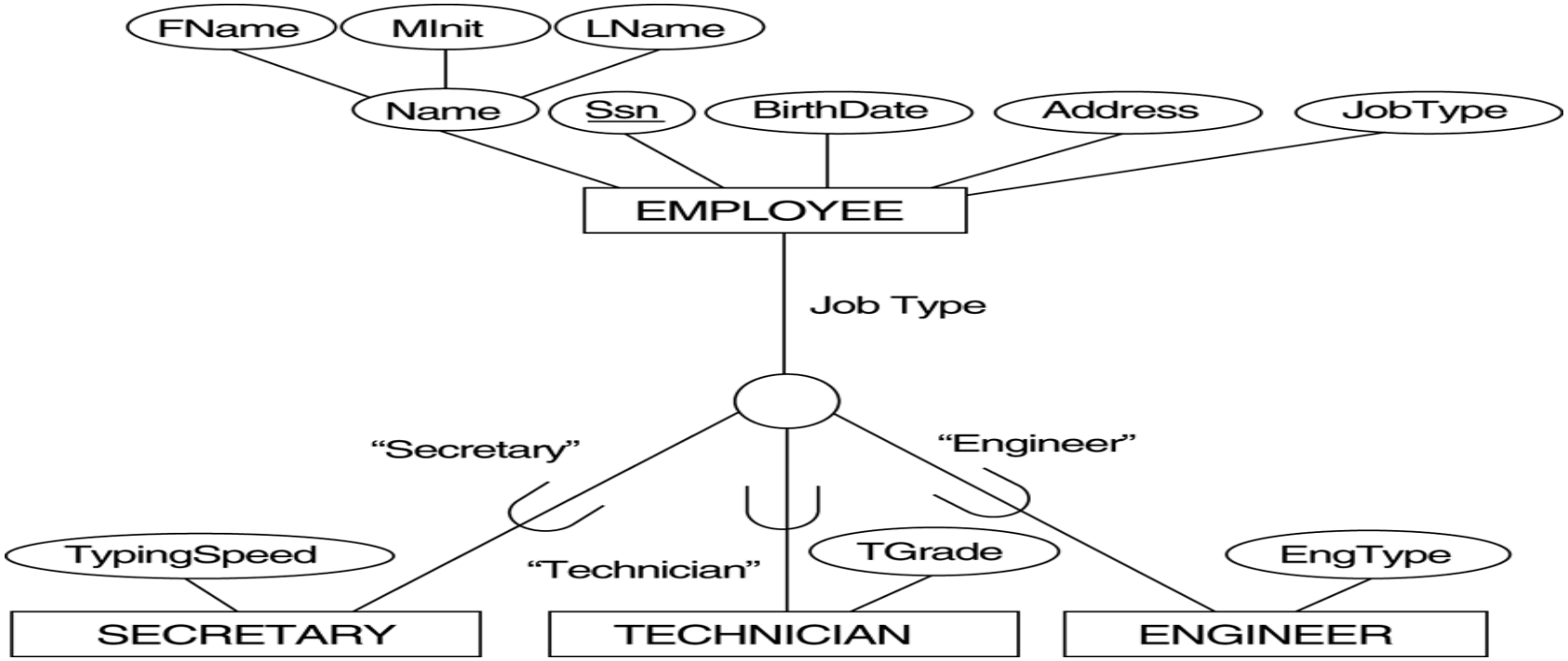
Step8: Options for Mapping Specialization or Generalization

- Convert each specialization with *m* subclasses {*S*1, *S*2, ..., *Sm*} and (generalized) superclass *C*, where the attributes of *C* are {*k*, *a*1, ...*an*} and *k* is the (primary) key, into relation schemas using one of the following options:

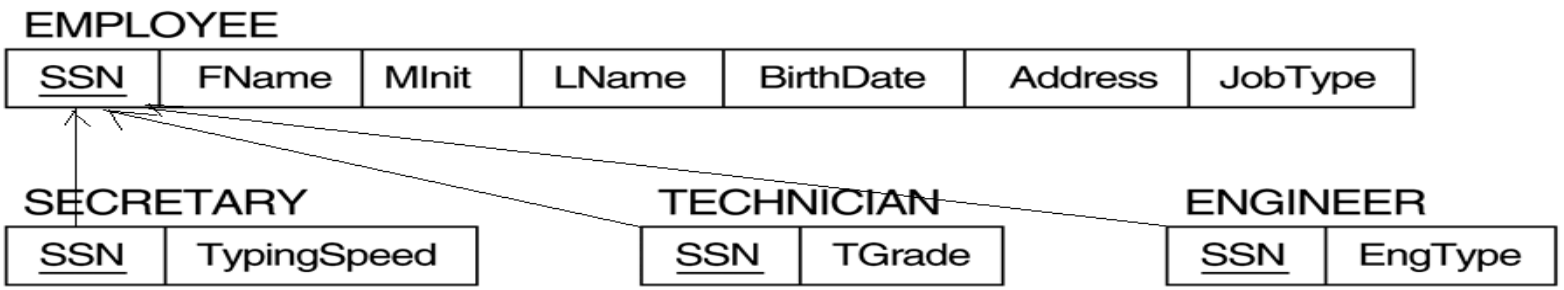
Step8: Options for Mapping Specialization or Generalization

- **Option 8A: Multiple relations-Superclass and subclasses**
  - Create a relation L for C with attributes Attrs(L) = {k,a1,…an} and PK(L) = k. Create a relation Li for each subclass Si, $1 \leq i \leq m$, with the attributes Attrs(Li) = {k} U {attributes of Si} and PK(Li)=k.
  - This option works for any specialization (total or partial, disjoint or overlapping).

Option 8A: Multiple relations- Superclass and subclasses



(a)

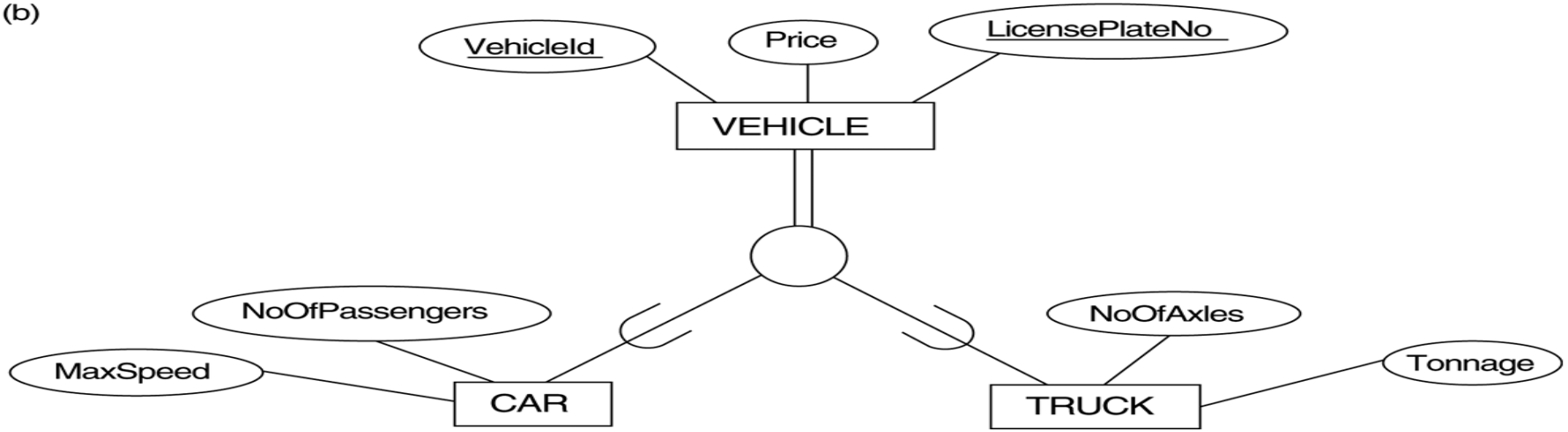


Step8: Options for Mapping Specialization or Generalization

- **Option 8B: Multiple relations-Subclass relations only**
  - Create a relation Li for each subclass Si, $1 \leq i \leq m$, with the attributes Attr(Li) = {attributes of Si} U {k,a1…,an} and PK(Li) = k.
  - This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses.

Option 8B: Multiple relations- Subclass relations only

(b)



(b)

CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

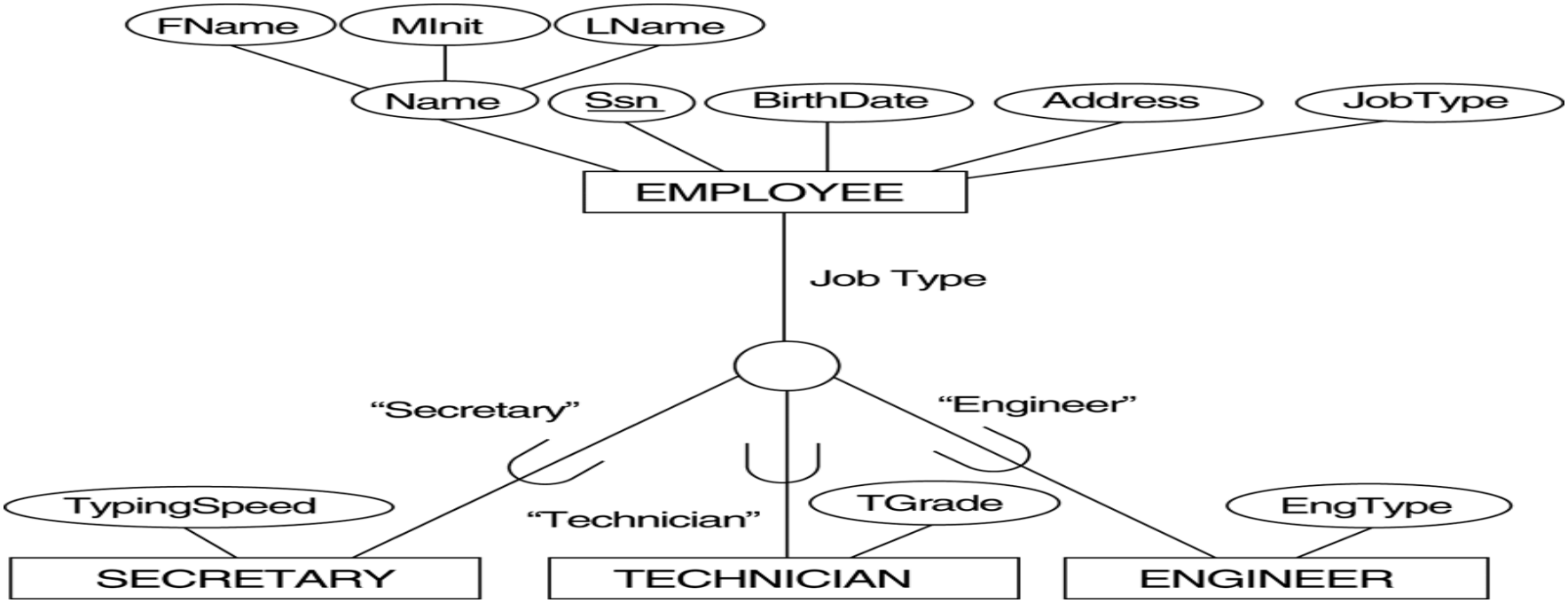
TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	Tonnage

Step8: Options for Mapping Specialization or Generalization

- **Option 8C: Single relation with one type attribute**
  - Create a single relation L with attributes Attrs(L) = $\{k,a_1,\ldots a_n\}$ U {attributes of S_1} U…U {attributes of S_m} U {t} and PK(L) = k.
  - The attribute t is called a type (or **image** or **discriminating**) attribute that indicates the subclass to which each tuple belongs, if any.
  - This option works only for a specialization whose subclasses are disjoint.

Option 8C: Single relation with one type attribute



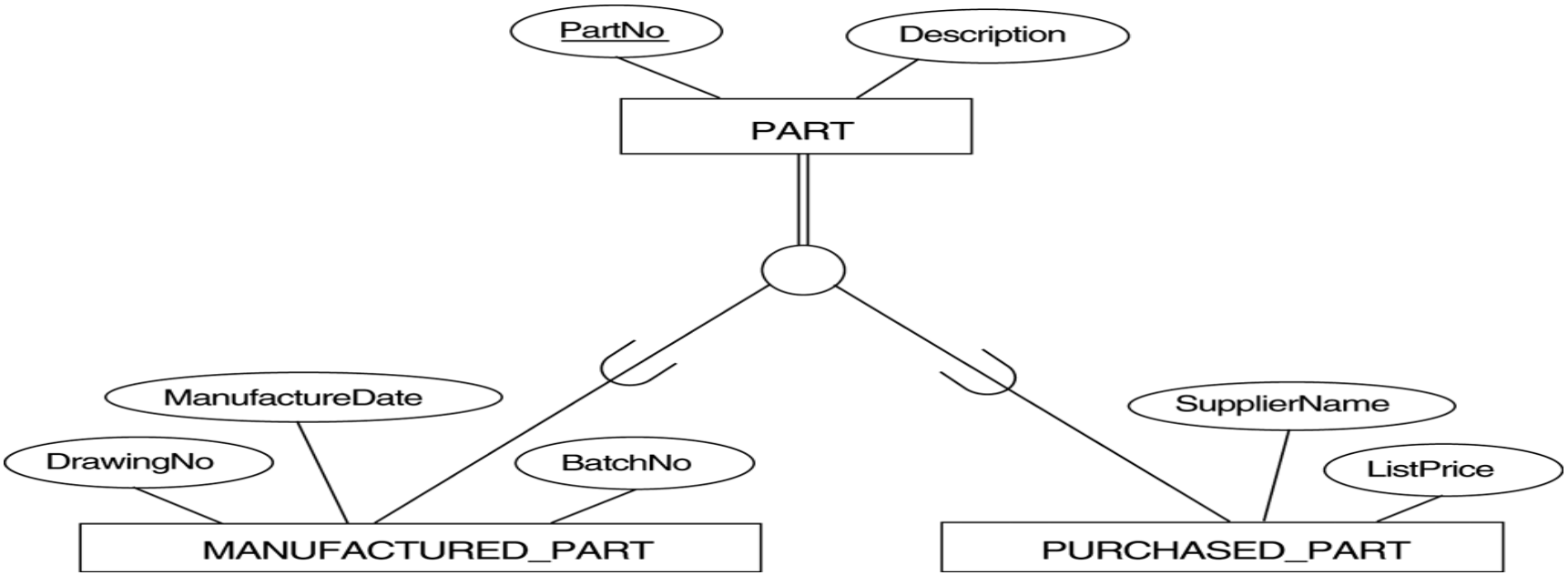
EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	Eng_type

Step8: Options for Mapping Specialization or Generalization

- **Option 8D: Single relation with multiple type attributes**
  - Create a single relation schema L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t_1, t_2,…,t_m} and PK(L) = k.
  - Each t_i, $1 \leq i \leq m$, is a **Boolean type** (or **flag**) attribute indicating whether a tuple belongs to the subclass S_i.
  - This option is used for a specialization whose subclasses are overlapping (but will also work for a disjoint specialization).

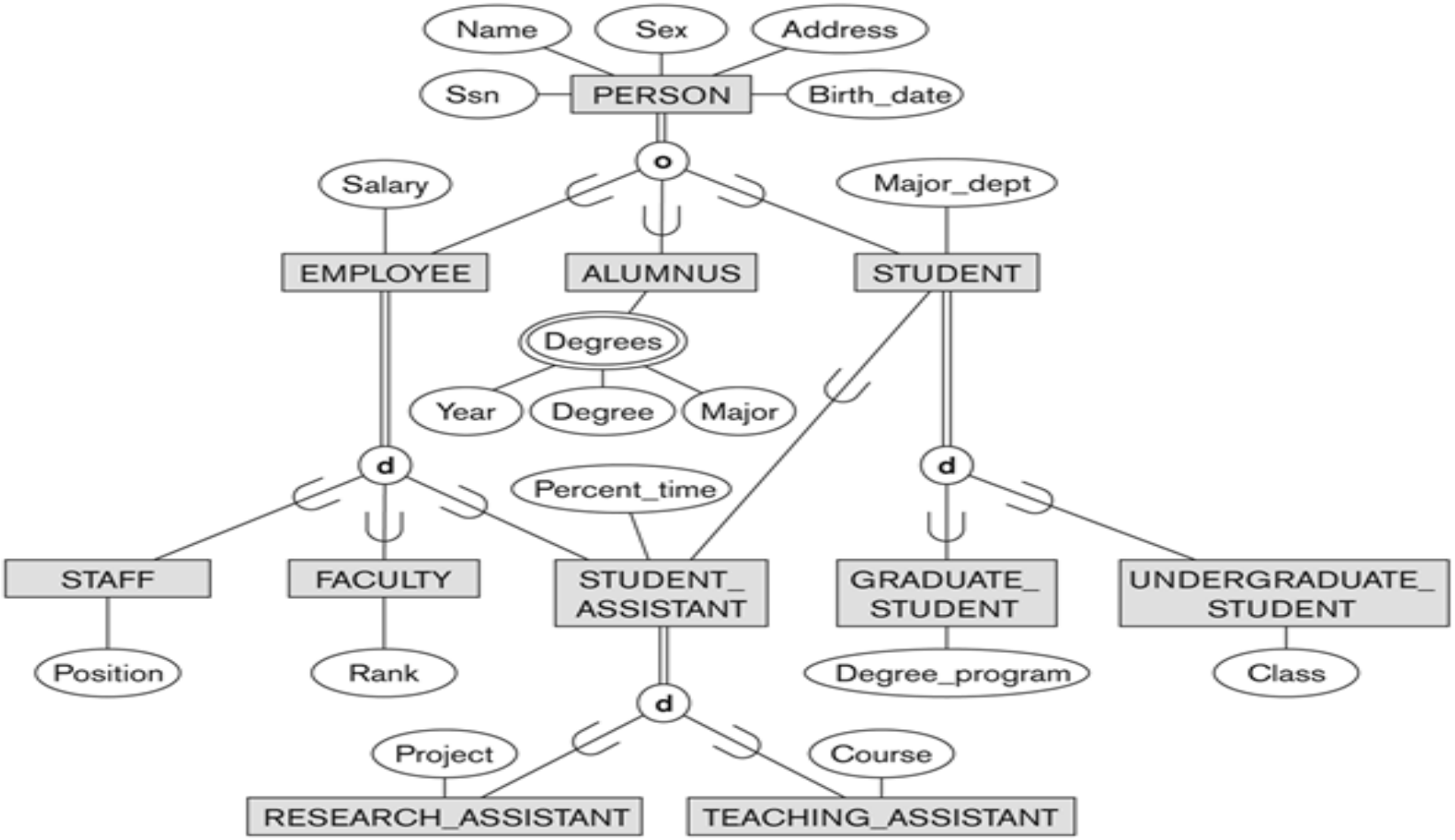
Option 8D: Single relation with multiple type attributes



(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice

A specialization lattice with multiple inheritance for a UNIVERSITY database.



A specialization lattice with multiple inheritance for a UNIVERSITY database.

Mapping the EER specialization lattice in Previous Figure using multiple options.

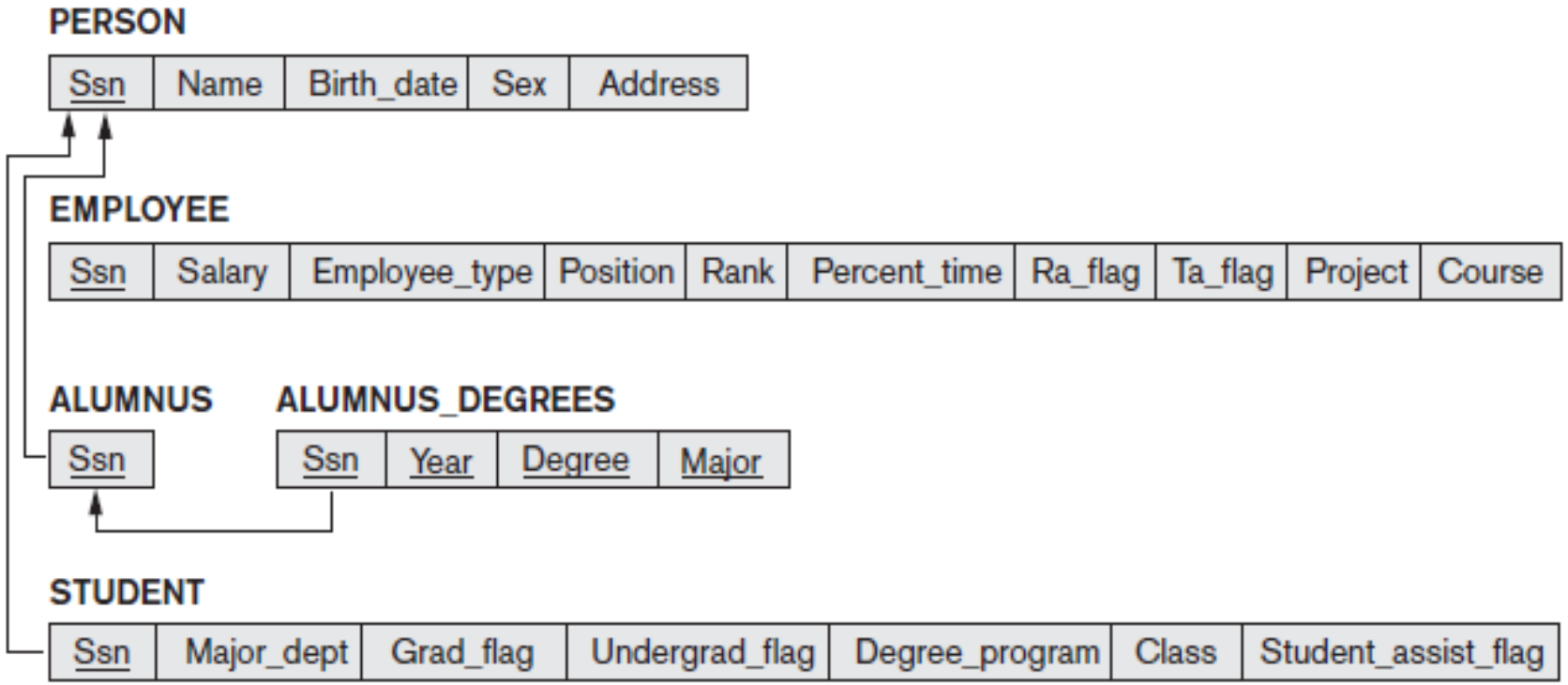
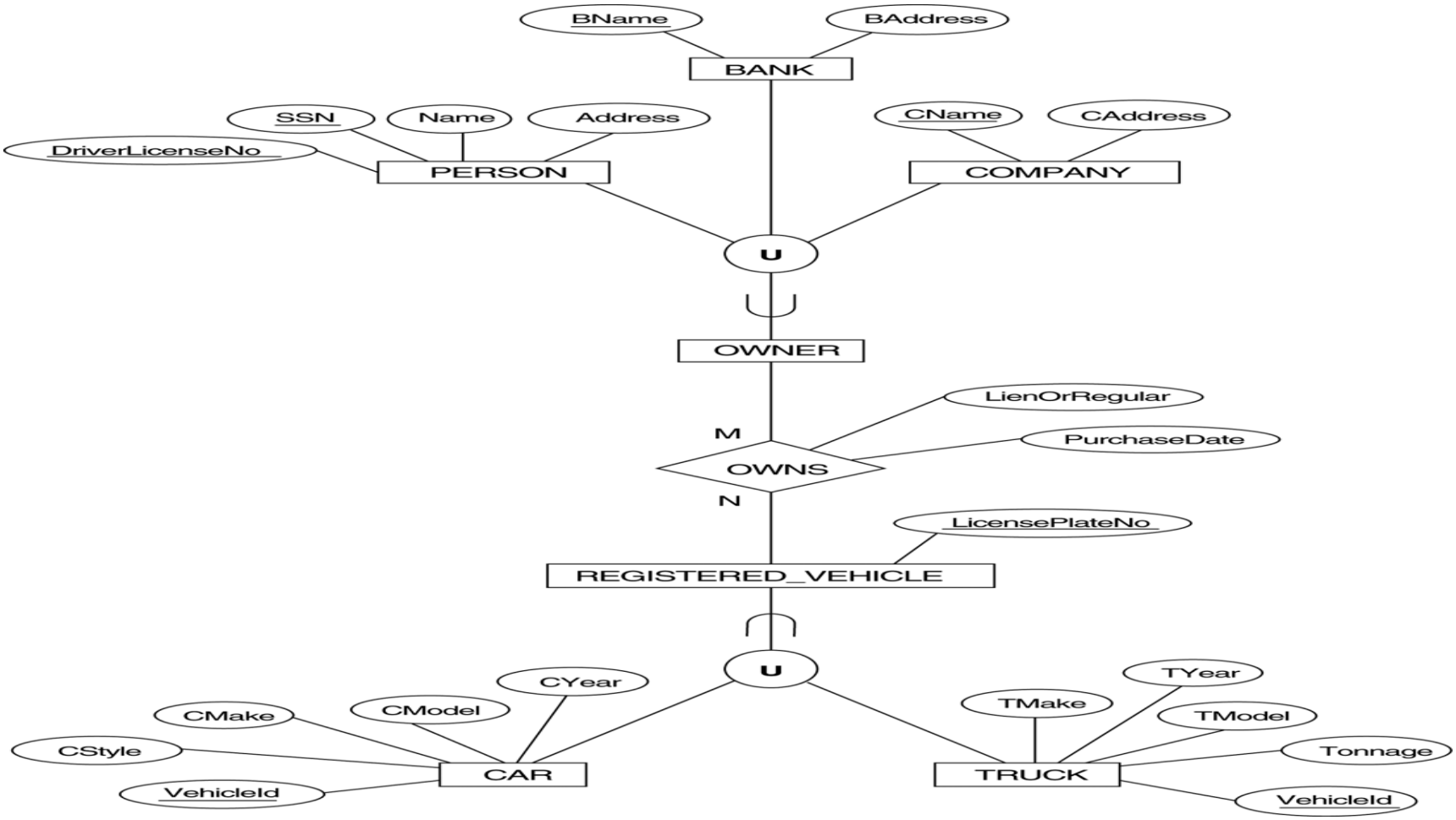


Figure
Mapping the EER specialization lattice in Figure using multiple options.

Step 9: Mapping of Union Types (Categories)

- For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the category.
- For a category whose superclasses have the same key, there is no need for a surrogate key.

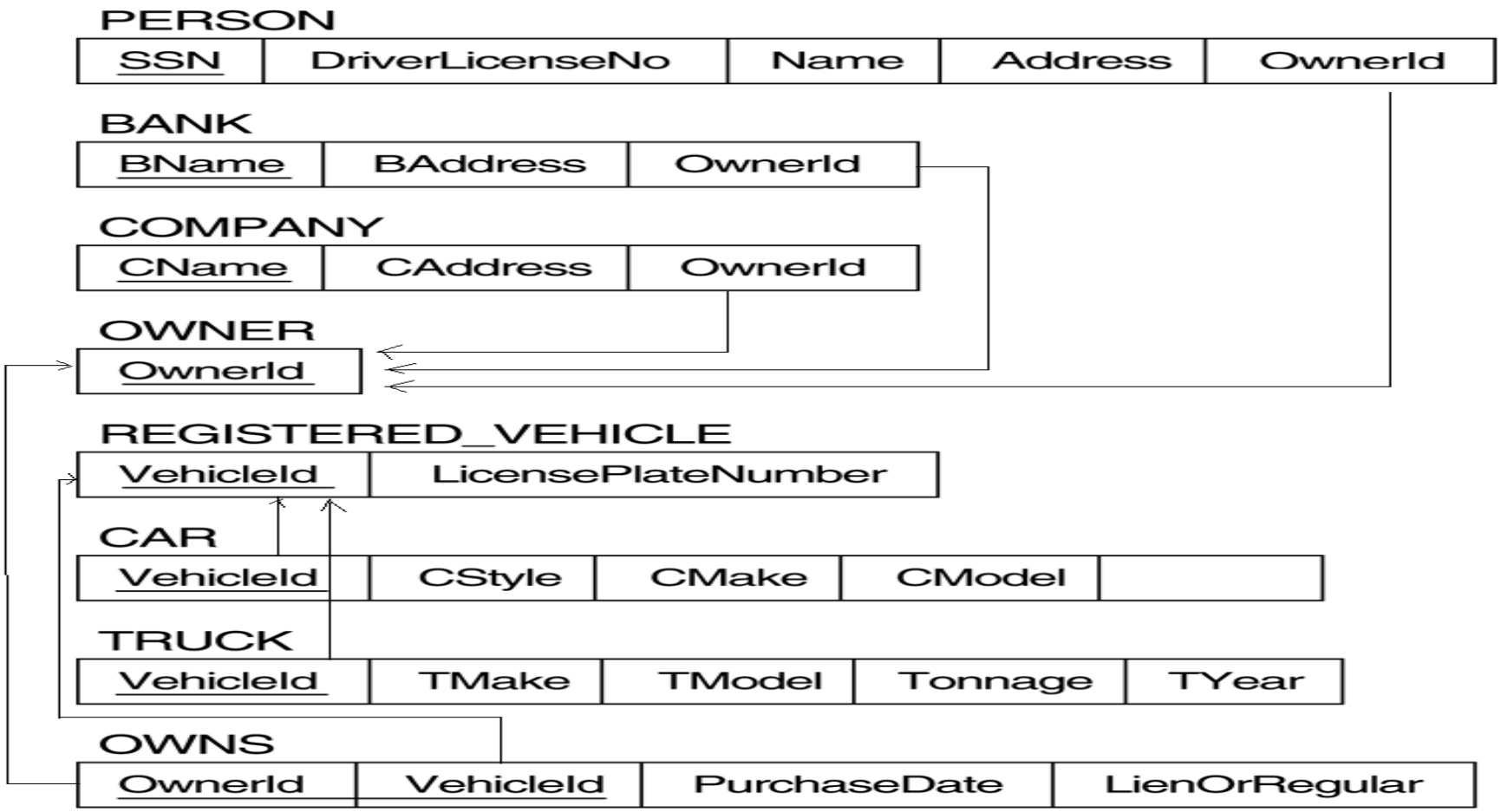
Step 9: Mapping of Union Types (Categories)



Step 9: Mapping of Union Types (Categories)

– In the example below we can create a relation OWNER to correspond to the OWNER category and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called OwnerId.

Step 9: Mapping of Union Types (Categories)



Advantages of the Relational Model

Advantages of the Relational Model

- Structural Independence :
  - In relational database, changes in the database structure do not affect the data access. So the relational database has structural independence.
- Conceptual Simplicity:
  - The Relational database model is simpler at the conceptual level. Since the relational data model frees the designer from the physical data storage details, the designers can concentrate on the logical view of the database.
- Set-Processing:
  - Relational database model provides facilities for manipulating a set of records at a time so that programmers are not operating on the database record by record.

Advantages of the Relational Model

- Sound Theoretical Background:
  - Relational database model provides a theoretical background for the database management field.
- It is simpler to navigate.
- Provides greater flexibility.

Query Languages

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Relational Algebra

Relational Algebra

- The relational algebra is a *procedural* query language.
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations

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

- 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) 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"

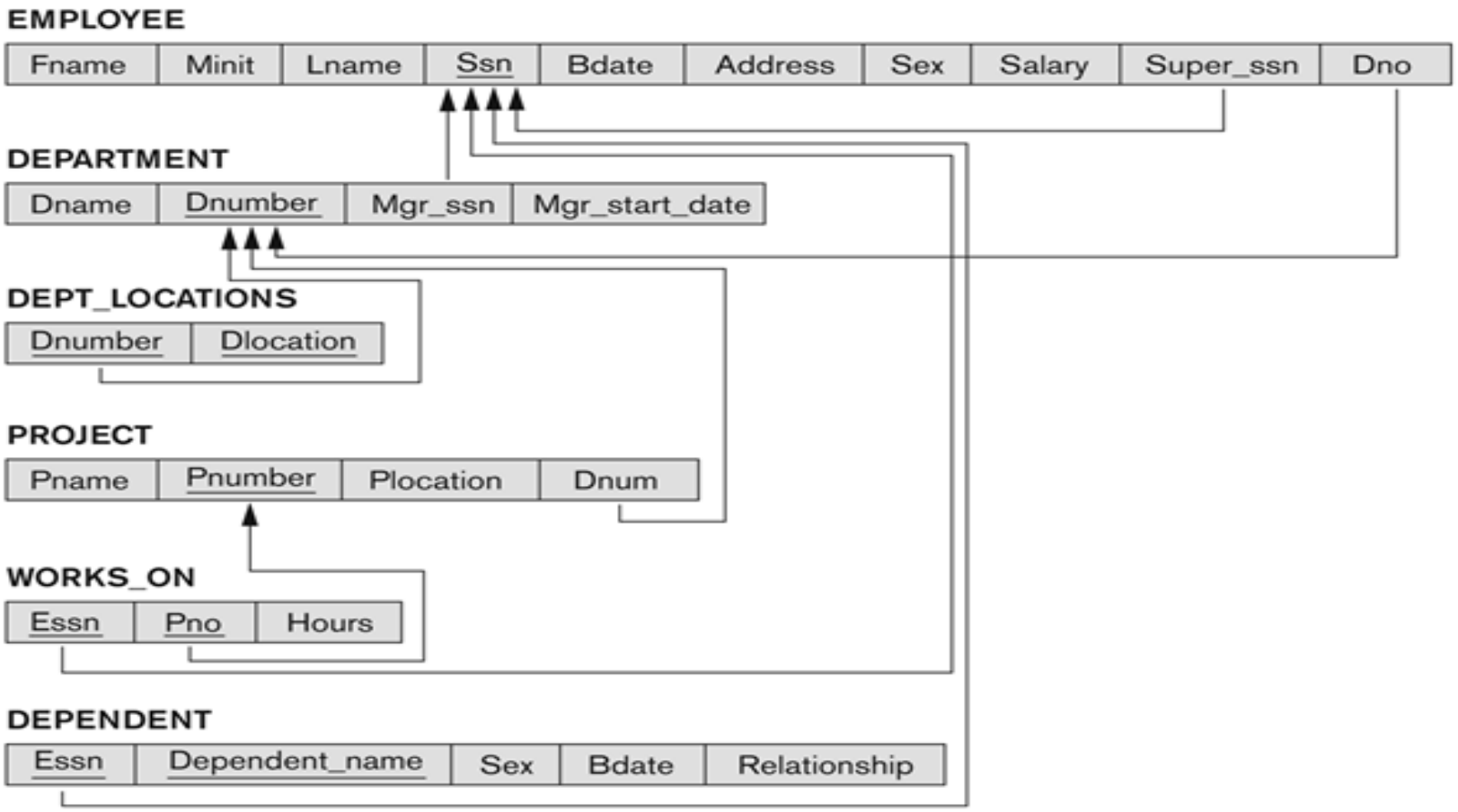
Relational Algebra Overview

- 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 (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, –)
    - CARTESIAN PRODUCT (**x**)
  - 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
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:

$$\sigma_{DNO = 4} (EMPLOYEE)$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{SALARY > 30,000} (EMPLOYEE)$$

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

- 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	1941-06-20	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

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

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

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

  - π (pi) is the symbol used to represent the project operation
  - <attribute list> 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

- PROJECT Operation Properties
  - The number of tuples in the result of projection $\pi_{<list>}(R)$ 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
    - $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

Examples of applying SELECT and PROJECT operations

Figure

Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$. (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:
  - $\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
  - $\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$

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

- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S\,(B1,\,B2,\,\ldots,\,Bn\,)}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..Bn
  - $\rho_{S}(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1,\,B2,\,\ldots,\,Bn\,)}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, …..Bn

Unary Relational Operations: RENAME

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

Example of applying multiple operations and RENAME

(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	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

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

Figure

Results of a sequence of operations.
(a) $\pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$.
(b) Using intermediate relations and renaming of attributes.

oo

Relational Algebra Operations from Set Theory: UNION

- UNION Operation
  - Binary operation, denoted by ∪
  - The result of R ∪ 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

Example of the result of a UNION operation

- UNION Example :

Figure
Result of the UNION operation RESULT ← RESULT1 ∪ 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, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., 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 ∩
- The result of the operation R ∩ 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

- 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

(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

Figure
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

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 or CROSS Join) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), 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 x 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

- Generally, CROSS PRODUCT is not a meaningful operation
  - Can become meaningful when followed by other operations
- Example (not meaningful): For example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:
  - FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}(EMPLOYEE)$
  - EMPNAMES $\leftarrow \pi_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES x DEPENDENT
- EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT
  - whether or not they are actually related

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

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

Example of applying CARTESIAN PRODUCT

Figure
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	9999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	9999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	9999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	9999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	9999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	9999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	9999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	9999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	9999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

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

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Additional Operations

Additional Operations

- The fundamental operations of the relational algebra are sufficient to express any relational-algebra query.
- However, if we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express.
- Therefore, we define additional operations that do not add any power to the algebra, but simplify common queries.
- For each new operation, we give an equivalent expression that uses only the fundamental operations. We define additional operations that do not add any power relational algebra, but that simplify common queries.

Additional Operations

- Set intersection
- Natural join
- Division
- Assignment

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Set-Intersection Operation - Example

- Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∩ s

A	B
α	2

Set-Intersection Operation - Example

- Suppose that we wish to find all customers who have both a loan and an account. Using set intersection, we can write

  $\Pi_{customer\text{-}name}(borrower) \cap \Pi_{customer\text{-}name}(depositor)$

- Note that we can rewrite any relational algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as:

  $$r \cap s = r - (r - s)$$

- Thus, set intersection is not a fundamental operation and does not add any power to the relational algebra. It is simply more convenient to write $r \cap s$ than to write $r - (r - s)$.

Set-Intersection Operation - Example

- List the names of managers who have at least one dependent.

$$MGRS(Ssn) \leftarrow \pi_{Mgr\_ssn}(DEPARTMENT)$$

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

$$MGRS\_WITH\_DEPS \leftarrow (MGRS \cap EMPS\_WITH\_DEPS)$$

$$RESULT \leftarrow \pi_{Lname,\ Fname}(MGRS\_WITH\_DEPS * EMPLOYEE)$$

Binary Relational Operations: JOIN

- JOIN Operation (denoted by⋈)
  - The sequence of CARTESIAN PRODECT 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, . . ., An) and S(B1, B2, . . ., Bm) 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

- 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 $\bowtie$ operation.
  - DEPT_MGR $\leftarrow$ DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ 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

Example of applying the JOIN operation

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	...

Figure
Result of the JOIN operation

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE}$$

Some properties of JOIN

- Consider the following JOIN operation:
  - $R(A1, A2, \ldots, An) \bowtie_{R.Ai=S.Bj} S(B1, B2, \ldots, Bm)$
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), 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[Ai]=s[Bj]
  - 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 $\bowtie_{theta}$ S
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S; for example:
  - R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)
- Most join conditions involve one or more equality conditions "AND"ed together; for example:
  - R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

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 Operation

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

- Another example: Q ← 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 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

(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

Figure

Results of two NATURAL JOIN operations.
(a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Complete Set of Relational Operations

- The set of operations including SELECT σ, PROJECT π , UNION $\cup$, DIFFERENCE – , RENAME ρ, and CARTESIAN PRODUCT X is called a *complete set* because any other relational algebra expression can be expressed by a combination of these 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.

Binary Relational Operations: DIVISION

- The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications.
- Example : *Retrieve the names of employees who work on **all** the projects that 'John Smith' works on*.
- To express this query using the DIVISION operation, proceed as follows.
- First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation

  - SMITH_PNOS: SMITH ← σ Fname='John' **AND** Lname='Smith' (EMPLOYEE)

  - SMITH_PNOS ← πPno(WORKS_ON ⋈ Essn=Ssn SMITH)

Binary Relational Operations: DIVISION

- Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:
  - SSN_PNOS ← $\pi_{Essn, Pno}$(WORKS_ON)
- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:
  - SSNS(Ssn) ← SSN_PNOS ÷ SMITH_PNOS
  - RESULT ← $\pi_{Fname, Lname}$(SSNS * EMPLOYEE)
- The preceding operations are shown in Figure (a).

Example of DIVISION Operation

(a)

SSN_PNOS

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

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

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

S

A
a1
a2
a3

T

B
b1
b4

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

Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The **assignment** operation, denoted by ←, works like assignment in a programming language.
- Example : We could write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$

$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$

$$result = temp1 - temp2$$

- The evaluation of an assignment does not result in any relation being displayed to the user.

Assignment Operation

- Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the←. This relation variable may be used in subsequent expressions.
- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.
- For relational-algebra queries, assignment must always be made to a temporary relation variable. Assignments to permanent relations constitute a database modification.
- Note that the assignment operation does not provide any additional power to the algebra. It is, however, a convenient way to express complex queries.

Recap of Relational Algebra Operations

Table
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$
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.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$
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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Extended Relational-Algebra Operations

Extended Relational-Algebra Operations

- Generalized Projection
- Outer Join
- Aggregate Functions and Grouping

Generalized Projection

- **Generalized Projection Operation :**
- The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list.
- The generalized form can be expressed as:

$$\pi_{F1,\, F2,\, \ldots,\, Fn}(R)$$

where $F1$, $F2$, ..., Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.

- This operation is helpful when developing reports where computed values have to be produced in the columns of a query result.

Generalized Projection

- **Generalized Projection Operation (contd.) :**
- Example : consider the relation

  EMPLOYEE (Ssn, Salary, Deduction, Years_service)
- A report may be required to show

  Net Salary = Salary – Deduction,

  Bonus = 2000 * Years_service, and

  Tax = 0.25 * Salary.
- Then a generalized projection combined with renaming may be used as follows:

  $REPORT \leftarrow \rho(Ssn, Net\_salary, Bonus, Tax)(\pi\ Ssn, Salary - Deduction, 2000 * Years\_service, 0.25 * Salary(EMPLOYEE))$.

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
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (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
  - $_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
- Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Examples of applying aggregate functions and grouping

- **Example 1 :** To retrieve each department number, the number of employees in the department, and their average salary, renaming the resulting attributes. The result of this operation on the EMPLOYEE relation is shown in Figure (a).
- **Example 2 :** If no renaming is applied, Figure (b) shows the result .
- **Example 3 :** If no grouping attributes are specified, the functions are applied to *all the tuples* in the relation, so the resulting relation has a *single tuple only*. Figure (c) shows the result .

Examples of applying aggregate functions and grouping

Figure

The aggregate function operation.

(a) $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}\ (_{\text{Dno}}\ \mathfrak{I}_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE}))$.

(b) $_{\text{Dno}}\ \mathfrak{I}_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE})$.

(c) $\mathfrak{I}_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE})$.

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

Inner Join

- The JOIN operations described earlier match tuples that satisfy the join condition.
- For example, for a NATURAL JOIN operation *R* * *S*, only tuples from *R* that have matching tuples in *S*—and vice versa—appear in the result.
- Hence, tuples without a *matching* (or *related*) tuple are eliminated from the JOIN result. Tuples with NULL values in the join attributes are also eliminated.
- This type of join, where tuples with no match are eliminated, is known as an **inner join**. The join operations we described earlier are all inner joins.

Outer Join

- Inner Join amounts to the loss of information if the user wants the result of the JOIN to include all the tuples in one or more of the component relations.
- A set of operations, called **outer joins**, were developed for the case where the user wants 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.
- This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.

Outer Join

- The **LEFT OUTER JOIN** operation keeps every tuple in the *first*, or *left*, relation *R* in *R* ⟕ *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.
- Example : suppose that we want a list of all employee names as well as the name of the departments they manage *if they happen to manage a department*; if they do not manage one, we can indicate it with a NULL value.
- We can apply an operation **LEFT OUTER JOIN**, denoted by , to retrieve the result as follows:

Outer Join

TEMP ← (EMPLOYEE ⟕ Ssn=Mgr_ssn DEPARTMENT)

RESULT ← πFname, Minit, Lname, Dname(TEMP)

- The result of these operations is shown in Figure :

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

Figure
The result of a LEFT OUTER JOIN operation.

Outer Join

- A similar operation, **right outer join**, keeps every tuple in the second or right relation S in the result of R ⟖ S.
- A third operation, **full outer join**, denoted by ⟗ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Examples of Queries in Relational Algebra : Procedural Form

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

  RESEARCH_DEPT ← σ DNAME='Research' (DEPARTMENT)

  RESEARCH_EMPS ← (RESEARCH_DEPT ⋈ DNUMBER= DNO EMPLOYEE)

  RESULT ← π FNAME, LNAME, ADDRESS (RESEARCH_EMPS)

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

  ALL_EMPS ← π SSN (EMPLOYEE)

  EMPS_WITH_DEPS(SSN) ← π ESSN (DEPENDENT)

  EMPS_WITHOUT_DEPS ← (ALL_EMPS - EMPS_WITH_DEPS)

  RESULT ← π LNAME, FNAME (EMPS_WITHOUT_DEPS * 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.**

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

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

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

Modification of the Database

Modification of the Database

- In this section, we address how to add, remove, or change information in the database.
- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Example

- Delete an employee named John Smith from the employee table.

$$\textit{Employee} \leftarrow \textit{Employee} - \sigma_{\text{Fname='John' } \textbf{AND} \text{ Lname='Smith'}}(\textit{Employee})$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- The attribute values for inserted tuples must be members of the attribute's domain. Similarly, tuples inserted must be of the correct arity.
- In relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Insertion Example

- Insert an employee named Richard K Bob to employee table:

Employee←*Employee* ∪{(‘Richard’, ‘K’, ‘Marini’, ‘653298653’,
‘1962-12-30’, ‘98 Oak Forest, Katy, TX’,
‘M’, 37000, ‘653298653’, 4)}

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F1,\, F2,\, \ldots,\, FI,} (r)$$

- Each F_i is either
  - the *i*th attribute of *r*, if the *i*th attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute

Update Example

- Example : Consider the relation

  EMPLOYEE (Ssn, Salary, Deduction, Years_service)

- Increase the salary of all employees by 25 percent:

  $\text{Employee} \leftarrow \prod_{\text{Ssn, Salary*1.25, Deduction, Years\_service}} (\text{Employee})$

Views

Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.
- A **view** is a single table that is derived from other tables. These other tables can be *base tables* or previously defined views.
- A view does not necessarily exist in physical form; it is considered to be a **virtual table**, in contrast to **base tables**, whose tuples are always physically stored in the database.
- This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.

Views

- A view is defined using the **create view** statement which has the form

  create view *v* **as** <query expression >

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v*.

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

View Example

- Consider the relation

  Faculty (id, name, sex, salary, address, major_subject, dname)

- WE may want to let a Head of Departments see only the department faculty rows for own department.

  CREATE VIEW Comp_faculty AS

  $\Pi_{\text{id, name, major\_subject, address}}(\sigma_{\text{dname='Computer'}}(\text{Faculty}))$

Updates Through View

- There are some situations in which some rows will be inserted, updated or deleted from view these operations need to be translated into equivalent operations against the base table of the view.
- Example : In the above created view we can write

Comp_faculty←*Comp_faculty* ∪{(Comp23,'Richard', 'DBMS', '98 Oak Forest, Katy, TX',)}

- Modifications are generally not permitted on view relations, except in limited cases.

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be *recursive* if it depends on itself.

Views Defined Using Other Views

- Example : We can define the view DBMS_Comp_faculty from above created Comp_faculty view as follows

CREATE VIEW DBMS_Comp_Faculty AS

Π name ($\sigma_{\text{major\_subject='DBMS'}}$ (Comp_Faculty))

University Questions

- Explain following relational algebra operations with examples:-
- Generalized Projection (8T- 3M)
- Set Difference (4T-3M)
- Assignment (5T-3M)
- Aggregate. (1T-3M)
- Set Intersection (4T-3M)
- Cartesian product (1T-3M)
- Rename (3T-3M)
- Outer Join (3T-3M)
- Natural Join (6T-3M)
- Division (2T-3M)
- Project (1T-2M)

University Questions

- What is view? How it is defined and stored? What are the benefits and limitations of view? (8T- 10M)
- Explain five relational algebra operators (2T-10M)