Relational Model Concepts

- Relational model represents the database as a collection of relation. The term '**relation**' refers to '**table**'
- Important concepts of relational model are:
    1. Relation Schema (R)
    2. Relation or Relation instance or Relation State
    3. Degree of Relation
    4. Tuple (t)
    5. Attribute (A)
    6. Domain (Dom)
    7. Keys or Key Constraints
- **Relation Schema (R)** : - Schema of a relation refers to its definition or structure. It includes a set of **column names**, the **data type** of each column, **constraints** of each column and the name of the relation.

    Eg: A student relation schema with 5 columns

    Student(regno:integer, name:string. Age:integer, grade:character, gradepoint:real)
- **Relation Instance** : - Instance refers to a set of tuples present in a relation at a particular moment of time. It is expressed as r(R) = { $\mathbf{t_1, t_2, \ldots, t_n}$} where each tuple $\mathbf{t_i}$ is a set of ordered values < $\mathbf{v_1, v_2, v_3, \ldots, v_n}$ > where each $\mathbf{v_j}$ belongs to Dom($\mathbf{A_j}$)

    Eg: An instance of STUDENT relation schema

Regno	Name	Age	grade	Gradepoint
101	Vipin	18	A	9
102	Veena	19	S	10

- **Degree of a relation** : Number of columns or fields or attributes in a relation
- **Tuple (t)** : - The term tuple is used to refer to a row (or record) in a relation. Each tuple t is a set of ordered values < $\mathbf{v_1, v_2, v_3, \ldots, v_n}$ >

    Eg: - In the above STUDENT relation, (101, 'Vipin', 18, 'A' , 9) is a tuple
- **Attribute** : - Attribute refers to a column (or field) in a relation.

    Eg :- In the above STUDENT relation, attributes are REGNO, NAME, AGE, GRADE & GRADEPOINT
- **Domain (Value set)** : The term domain refers to the set of permitted values for an attribute in a relation. Domain is defined by data types and constraints in SQL.

    Eg: In the above STUDENT relation, domain of the attribute GRADE = { S, A, B, C, D, E, F }
- **Relational Database Schema** : - It is a collection of relation schemas describing one or more relations and their integrity constraints

    Eg : - A COMPANY database schema includes the relation schemas
    - EMPLOYEE(empid, empname, deptid, salary)
    - DEPARTMENT(deptid, deptname)
    - PROJECT(prjid, prjTitle, location)

    In general, it can be expressed as S = { $\mathbf{R_1, R_2, \ldots, R_n}$ }
- **Relational Database Instance** : - It is a set of relation states { $\mathbf{r_1, r_2, r_3, \ldots, r_n}$ } , each $\mathbf{r_i}$ is a state of $\mathbf{R_i}$ satisfying the required integrity constraint
- **Key Constraints : -** The important key constraints are :***Super key, candidate key, composite key, primary key, foreign key, alternate key***

Different types of Keys

Consider the relations (or tables)

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key:

- The minimal set of attribute which can uniquely identify a tuple (row or record) is known as candidate key. For Example, **STUD_NO** in **STUDENT** relation.
- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT.
- It is the minimal Super Key.

Super Key:

- The set of attributes which can uniquely identify a tuple (row or record) is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.
- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Primary Key:

- There can be more than one candidate key in a relation out of which one can be chosen as primary key.
- The entity integrity constraint says that the value of primary key can never be NULL.
- For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT but STUD_NO can be chosen as primary key (only one out of many candidate keys).

Alternate Key:

- The candidate key other than primary key is called as alternate key. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

Composite Key

- A *composite key* is a candidate key with a combination of two or more attributes.
- The candidate key can be simple (having only one attribute) or composite as well.
- Eg1: {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.
- Eg2 : In the relation schema **WORKS_ON(<u>empid, prjid</u>, hours_worked)** in company database, the composite key (**empid** , **prjid**) forms the candidate key.

<u>Foreign Keys</u>

- Foreign Keys are constraints used to specify the referential integrity between two relations
- Foreign keys help to maintain the consistency of data among the tuples in both relations.
- Foreign keys are a set of one or more attributes in a relation schema R1 which references the primary key of relation schema R2 while inserting or updating its values.
- Foreign key & the referencing primary key must have the same domain (data type)

- **Eg 1:-** STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.
- **Eg2 :- deptno** in EMPLOYEE relation can be designated as a FOREIGN KEY referencing the PRIMARY KEY **deptno** of DEPARTMENT relation
- **In SQL**, one way for defining this FOREIGN KEY using DDL command is :

ALTER TABLE EMPLOYEE ADD CONSTRAINT fkeydept FOREIGN KEY(deptno) REFERENCES DEPARTMENT(deptno);

Entity Relationship Model Concepts

ER-Model is used in the conceptual or high level design phase of database application. The diagrammatic representation of ER-Model is known as ER diagrams.

Major Concepts are :

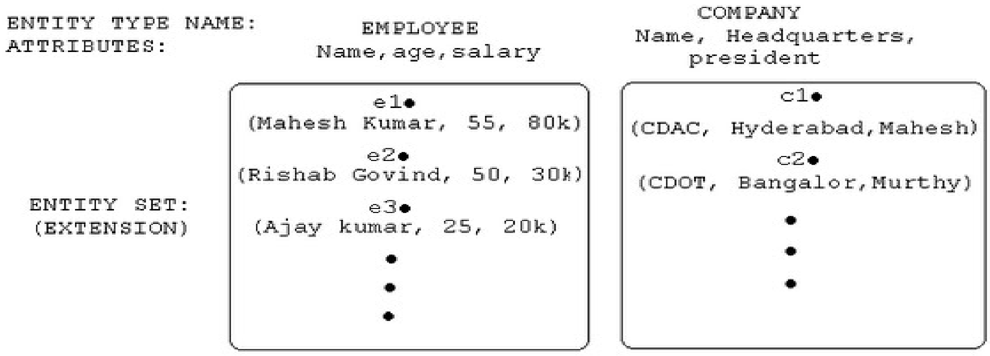
- **Entity type & Entity set**
- **Relationship type & Relationship set**
- **Attributes** (Simple Vs. Composite , Single Valued Vs. Multi Valued , Stored Vs. Derived)
- **Constraints of Relationship types** (Cardinality ratio & Participation Constraint)

ENTITY TYPES & ENTITY SETS

An **entity type** defines a set or collection of entities that have same attributes. Each entity type in a database is described by its name and attributes

A collection of all entities of a particular entity type in the database at any point in time is called an **entity set**

The **entity type** describes the **schema** or **intension** for a set of entities that share the same structure. The **entity set** is called the **extension** of the entity type.



Entity is a real world object with independent existence

Examples :

Entity with *physical existence* like **person, car, house, employee, student** etc.

Entity with a *conceptual existence* like **company, job, course** etc.

Attributes: It is the properties that describe the entity. For example, **employee** entity may be described by the attributes employee's **name, age, salary and job**. Each entity will have a value for its attribute.

Example : An entity & its attributes

Employee Entity	**e1**
Attributes of e1 & their values	
Attribute	**Value**
Name	Latha
Age	40
Salary	12000
Job	Clerk

TYPES OF ATTRIBUTES

1. Simple & Composite

2. Single valued & Multivalued
3. Stored & Derived

1. Composite & Simple Attributes

Attributes that can be divided into smaller subparts are called **Composite attributes**.

Eg: **Address** which can be divided as **housename, street, city, state, zipcode** etc.

Attributes that are not divisible are called **simple** or **atomic attributes.** Eg**: age, designation**

2. Single-valued & Multi-valued attributes

Attributes having a single value for a particular entity are called **single valued.** Eg: **age** is a single valued attribute

If an attribute has more than one value for the same entity, it is called **multi-valued** attribute. Eg: **college_degree** attribute of a **person** (a person can have multiple degree).

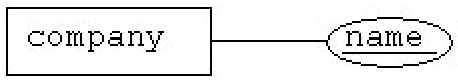
3. Stored & Derived Attributes

Sometimes two or more attributes are related. In such cases, one of the related attributes can be derived from the other.

For example, **age & date_of_birth** are related. Here **age** can be derived from **date_of_birth**. So **age** is a **derived attribute** and **date_of_birth** is a **stored attribute.**

KEY ATTRIBUTES

An entity usually has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. In diagrammatic notations each key attribute is represented by underlined attribute inside the oval. For example in the following figure name is the key attribute of the company.



RELATIONSHIP TYPE & RELATIONSHIP SET

- A relationship type **R** among **n** entity types E1, E2, ..., En defines a set of associations among entities from these entity types.
- A relationship set **R** is a collection of relationship instances that exists between the entities in the participation entity types.

Example : WORKS_FOR is a relationship type between the entity types EMPLOYEE & DEPARTMENT

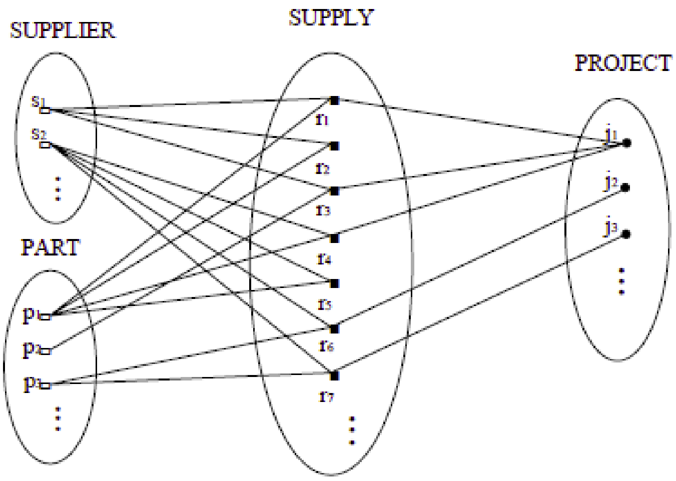
<u>Properties of relationship type</u>

1. Degree
2. Role
3. Constraints on relationship type
    a) Cardinality Ratio
    b) Participation Constraints

<u>1. Degree of a relationship type</u>

It is the number of participating entity types. WORKS_FOR relationship between the entity types EMPLOYEE & DEPARTMENT is of degree **two**. A relationship of degree two is also called **binary** relationship.

A relationship of degree three is called **ternary** relationship. An example for ternary relationship **SUPPLY** is shown below.

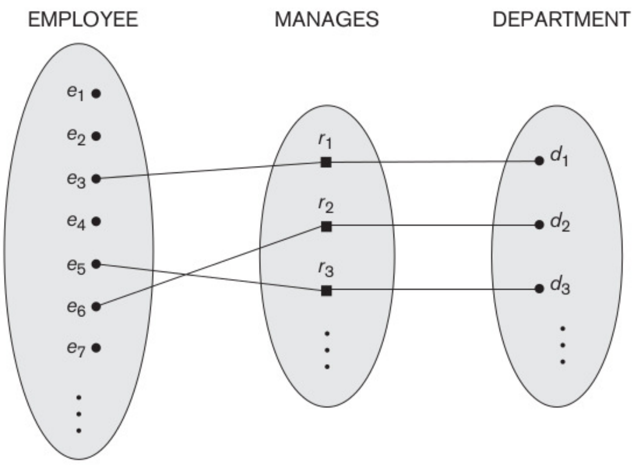


2) Cardinality Ratios for Binary relationships

- The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.
- For example, in the WORKS_FOR relationship type, **DEPARTMENT:EMPLOYEE** is of cardinality ration **1:N,** meaning that each department can have any number of employees. N indicates there is no maximum number.
- The possible cardinality ratios are **1:1 , 1:N, N:1, M:N**

Examples : MANAGES relationship type between EMPLOYEE and DEPARTMENT is of cardinality 1:1 , meaning an employee can manage only one department and a department can have only one manager.

Figure 7.12
A 1:1 relationship, MANAGES.



- WORKS_ON relationship type between EMPLOYEE and PROJECT is of cardinality M:N, meaning an employee can work on several projects and a project can have several employees.

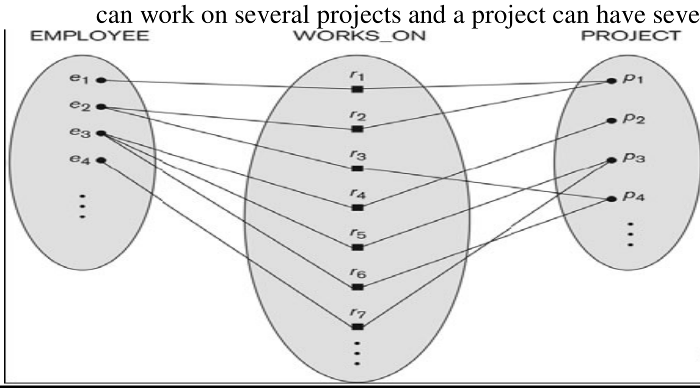


Figure 3.13
An M:N relationship, WORKS_ON.

4. Participation Constraints & existence dependencies

- It is the minimum number of relationship instances that each entity can participate in. It is also called the **minimum cardinality constraint**.

- There are two types of participation constraints :
  1) Total participation constraint
  2) Partial participation constraint

Example 1 : If the company policy states that every employee must work for a department, then the participation of EMPLOYEE entity type in **WORKS_FOR** relationship type is called **total participation.** Total participation is also called **existence dependency**.

Example 2: We can't expect that every employee manages a department. So the participation of EMPLOYEE entity type in **MANAGES** relationship type is **partial**.

Structural constraints (min , max) : - The cardinality ration and participation constraint taken together, is referred to as structural constraints of a relationship type.

Draw a sample ER diagram

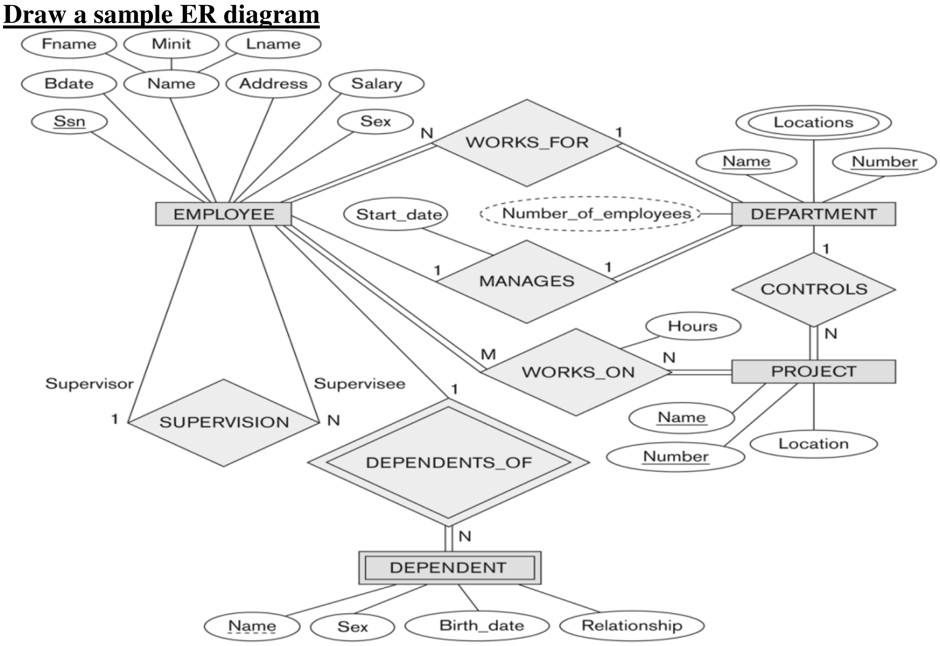


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Detailed explanation of Weak entity type & strong entity type

- Entity types that do not have key attributes of their own are called **weak entity types.**
- Regular entity types that do have a key attribute are called **strong entity types.**
- For example, **EMPLOYEE** entity type of company database is a **strong entity** because each employee entity can be uniquely identified using the key attribute **empid**.
- However, the **DEPENDENT** entity type of company database is a **weak entity**. DEPENDENT entity has existence only if there is an **EMPLOYEE** entity.
- **EMPLOYEE** entity is called the **identifying** or **owner entity type**.
- The relationship type which connects a weak entity to its owner entity is called **identifying relationship**.
- A weak entity always has a total participation (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- A weak entity normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In **DEPENDENT** entity type, **name** can be considered as partial key, provided no two dependents of the same owner have the same name.
- In ER diagram, weak entity type uses double lined rectangle, identifying relationship uses double lined diamonds and partial key is underlined with dashed or dotted line.

EER Model Concepts

EER model includes all modelling concepts of ER model and the following additional concepts :

- **subclass** , **super class & Inheritance**
- **specialization** & **generalization**
- **Category** or **Union type (Not explained here)**

Diagrammatic representation of EER Model is called EER Diagram.

Subclass , Super class, Inheritance

- In some cases, an entity type has several subtypes or sub groupings. For example, the entities that are members of EMPLOYEE entity type may be further grouped as SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE
- Every entity that is a member of one of these sub groupings is also an Employee. The sub groupings are called **subclass** of EMPLOYEE entity type. EMPLOYEE entity type is called **super class**.
- The relationship between a super class and any one of its subclasses is called **super class/subclass** , or **class/subclass** relationship.
- **Inheritance** : - A member entity of subclass inherits all the attributes of its super class and all the relationships in which its super class entities are involved. This is known as **inheritance** In addition, a subclass can have its own attributes and relationships.

Specialization

- Specialization is the process of defining a set of subclasses of an entity type. The entity type is called the **super class** of specialization. Specialization is based on some distinguishing characteristics of the entities in the super class.
- **Example :-** The set of subclasses {SECRETARY, ENGINEER, TECHNICIAN } is a specialization of the super class EMPLOYEE with the distinguishable characteristics , **job_type**.

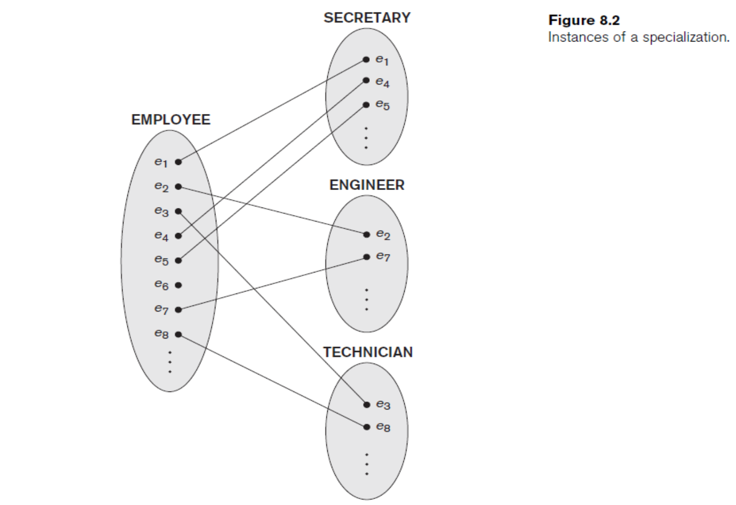


Figure 8.2
Instances of a specialization.

- The set of subclasses { HOURLY_EMPLOYEE, MONTHLY_EMPLOYEE} is another specialization of EMPLOYEE with the distinguishable characteristics, **pay_method**.
- Figure shows how we represent a specialization diagrammatically in an EER diagram.

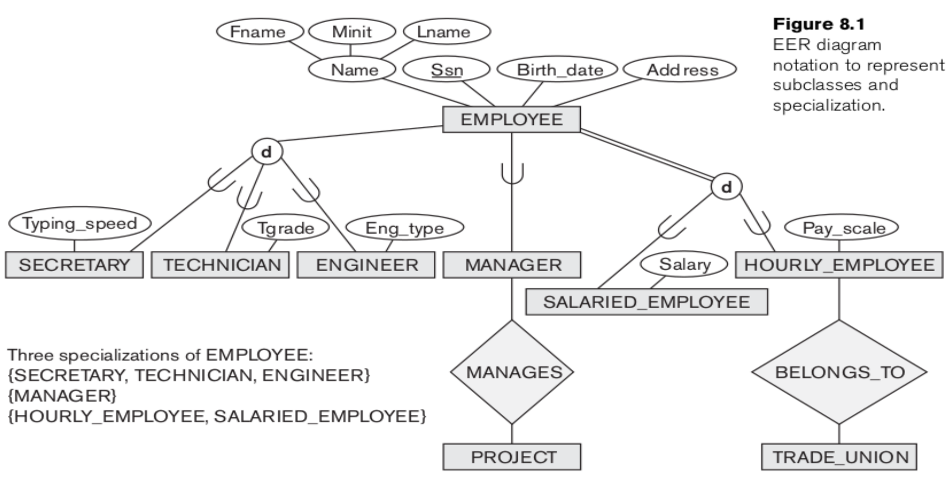


Figure 8.1
EER diagram notation to represent subclasses and specialization.

- In summary, Specialization process allows to do the following:
  - define a set of subclasses of an entity type

- define additional attributes with each subclass
- Define additional relationship types between each subclass and other entity types or other subclasses.

Generalization

- It is the reverse process of abstraction in which we identify the common features of several entity types and generalize them into a single super class of which the original entity types are special subclasses.
- Example :- Consider two entity types – CAR & TRUCK. Both of them have several common features or attributes. Those common features can be generalized into the entity type VEHICLE. VEHICLE is the generalized super class.

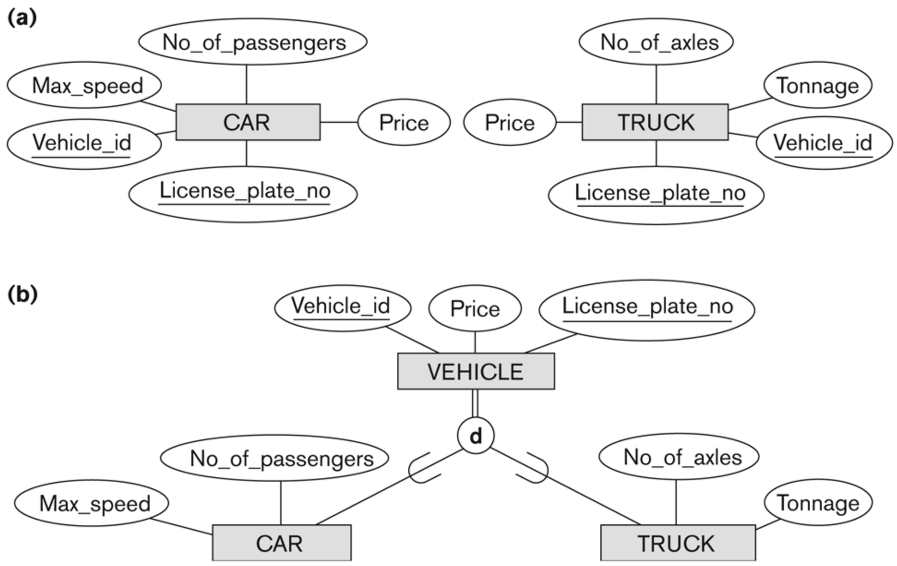


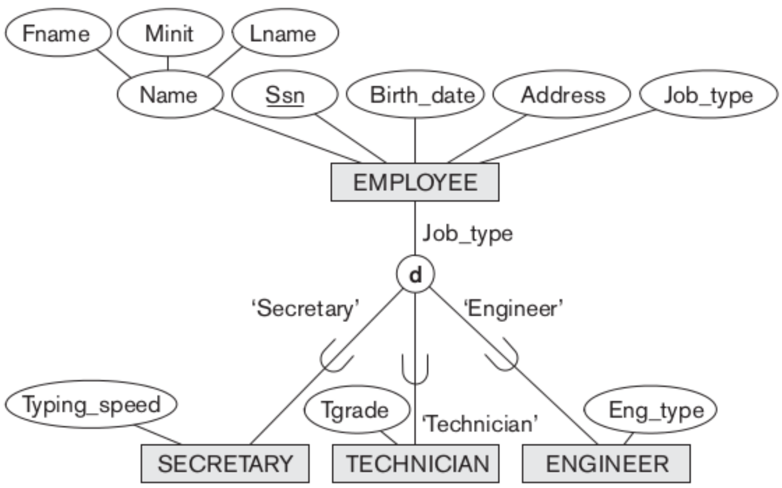
Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Generalization process can be viewed as being functionally the inverse of specialization process.

Constraints of Specialization and Generalization

1. **Attribute-defined specialization : -** If all subclasses in a specialization have their membership condition on the same attribute of the super class, that specialization is called **attribute-defined specialization**, and the attribute is called the **defining-attribute**.

Figure 8.4
EER diagram notation for an attribute-defined specialization on Job_type.



2. User-defined specialization : - Here, there is no specific condition or predicate to determine the membership of entities in subclasses. Those subclasses are called **user-defined.** Membership is specified individually by the database users, not by any automatic conditions.

3. Disjointness vs. Overlapping Constraint

- **Disjointness** specifies that the subclasses of the specialization must be disjoint. That means an entity can be a member of ***at most*** one of the subclasses of the specialization. In EER diagram, the notation **d** in the circle indicate disjointness
- The specializations {SECRETARY, TECHNICIAN,ENGINEER} and {HOURLY_EMPLOYEE, SALARIED_EMPLOYEE} are disjoint.
- If the subclasses are not disjoint, their entities may be **overlapping**. This is indicated by placing an **o** in the circle as shown below. Entities of super class can be member of more than one sub classes of the same specialization

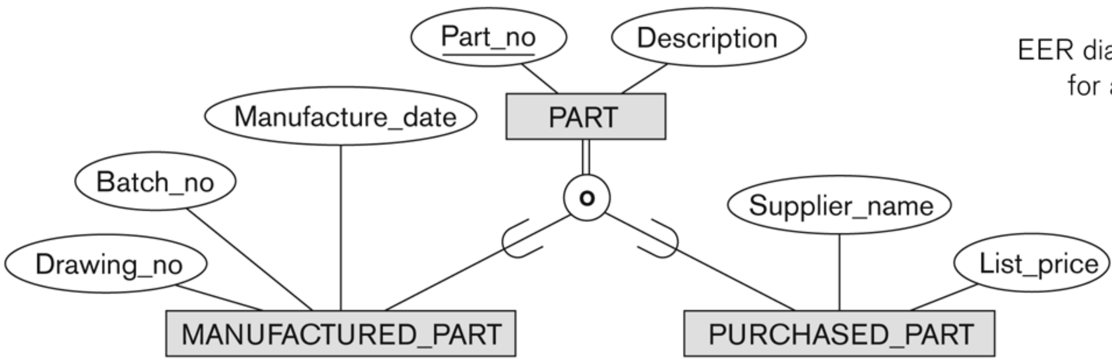


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

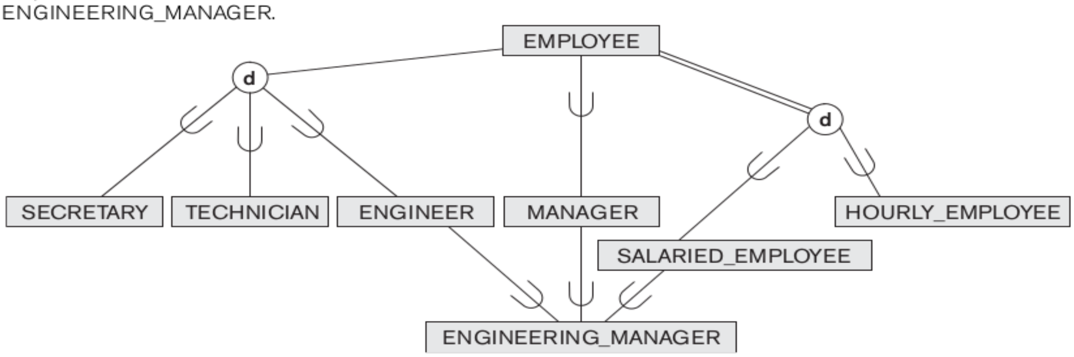
4. Completeness Constraint: partial vs. total

- The **total** specialization constraint requires that every entity of a super class be a member of *at least one* of its (immediate) subclasses. A **partial** constraint is simply the absence of the total constraint (and hence is no constraint at all).
- The specialization {HOURLY_EMPLOYEE, SALARIED_EMPLOYEE} of EMPLOYEE is total because every employee must either a hourly_employee or a salaried_employee.
- The specialization {SECRETARY, TECHNICIAN,ENGINEER} of EMPLOYEE is partial because some employees do not belong to any of the subclasses.
- In EER diagram, total specialization is represented by double line and partial by single line
- Note that the disjointness and completeness constraints are independent of one another, giving rise to four possible combinations: ***disjoint-total , disjoint-partial , overlapping-total, overlapping-partial***

<u>Specialization Lattices</u>

In a specialization **lattice**, a subclass may have more than one (immediate) super class. Having more than one super class gives rise to the concept of **multiple inheritances**.

Figure 8.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.



<u>Specialization Hierarchy</u>

In a specialization **hierarchy** (i.e., tree), each subclass has exactly one (immediate) super class. Concept is **single inheritance.**

Mapping ER Diagram to Relational Model

1. **Mapping of Regular entity type** (Create new relation , set a suitable primary key)
2. **Mapping of weak entity type** (Create new relation by adding the primary key of owner entity type as foreign key)

3. **Mapping of 1:1 relationship type** (Choose total participation side, then add foreign key referencing primary key of other relation)
4. **Mapping of 1:N relationship type** (Add Foreign key in N-side relation referencing primary key of other relation)
5. **Mapping of M:N relationship type** (Create new table with two foreign keys referencing primary keys of both participating relations)
6. **Mapping of multi-valued attributes (** Create new table with foreign key referencing primary key of parent table)
7. **Mapping of n-ary relationships** (Create a new table with n foreign keys referencing the primary key of all participating entity types)

Explain with an example (Discussed in class)

RELATIONAL ALGEBRA

Unary Operations (SELECT, PROJECT RENAME)

SELECT (σ) is a unary relational algebra operation used to choose a subset of tuples from a relation that satisfies the specified selection criteria.

Syntax

σ < selection-criteria> (R)

Example : σ dept=4 (EMPLOYEE) -- selects all employee tuples with deptno=4

Degree of the relation resulting from a SELECT operation is same as the degree of R

PROJECT (π) is another unary operation which selects the specified columns or attributes from a relation

Syntax

π < attribute-list> (R)

Example : π < name, gender > (EMPNAME) -- Retrieves only name & gender of all employees

The result set of PROJECT operation is a set of distinct tuples eliminating duplicates.

Degree of the result set is equal to the number of attributes in <attribute-list>

RENAME (ρ) : - Intermediate results of relational algebra expressions can be renamed to refer them later . Even relations and their attributes can be renamed temporarily while using in expressions.

General Syntax of RENAME operation

Form 1: $\rho_{S(B1,B2, \ldots ,Bn)}(R)$ ----> Renames both relation and attributes

Form 2: $\rho_{S}(R)$ ----> Renames only the relation

Form 3: $\rho_{(B1,B2, \ldots ,Bn)}(R)$ ----> Renames only attributes

, where ρ (rho) is the rename operator , S is the new name of relation and B1,B2, ... ,Bn are the new names of attributes.

Eg: - To retrieve empname & deptname from relations EMPLOYEE & DEPARTMENT

$\sigma_{a.deptno = b.deptno}$ (ρ_a EMPLOYEE x ρ_b DEPARTMENT)

Operations from SET Theory (UNION , INTERSECTION & MINUS)

UNION : - It is a binary relational operation applied on two relations, say R and S, and denoted as **R U S**. The result set includes all tuples that is either in R or in S or in both R and S eliminating duplicates.

INTERSECTION : - This is another binary relational operation , denoted by **R ∩ S**. The result set of intersection include all tuples that are present in both R and S.

SET DIFFERENCE (or MINUS) : - It is another binary relational operation denoted as denoted as **R – S**. The result set includes all tuples that are present in R but not in S.

Note :-

- The resulting relation has the same attribute names as the first relation R.
- Also, both R and S must be **type compatible** or **union compatible**. For R(A1, A2, ... , An) and S(B1, B2, ... , Bn) to be union compatible, they must have the same degree **n** and dom(Ai)=dom(Bi), $1 \leq i \leq n$

- R U S = S U R and vice versa, set operations are commutative
- set operations are associative R U (S U T) = (R U S) U T
- INTERSECTION can be expressed in terms of UNION & set difference
  - **R ∩ S = ((R U S) – (R – S)) - (S – R)**

Example

(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

STUDENT ∪ INSTRUCTOR

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

STUDENT ∩ INSTRUCTOR

STUDENT – INSTRUCTOR

(d)

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

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

INSTRUCTOR – STUDENT

‹#›

ADVANCED RELATIONAL OPERATIONS

CARTESIAN PRODUCT (x)

It is also known as **CROSS PRODUCT or CROSS JOIN**

- The Cartesian product returns all the rows in all tables listed in the query. Each row in the first relation is paired with all the rows in the second relation. This happens when there is no relationship between relations
- Syntax
- R x S , WHERE R & S are two relations
- The result set of **R(A1, A2, ... , An) x S(B1,B2, ... , Bm)** is a relation **Q(A1, A2, ... , An, B1,B2, ... , Bm)**
  - R is of degree **n** and S is of degree **m**, then R x S is of degree **n+m**

(Write an example instance for **STUDENT x DEPARTMENT)**

JOIN operation

- The JOIN operation , denoted by ⋈, is used to combine related tuples from two relations into single longer tuples. JOIN operation is very important as it allows us to process relationships among relations.

Syntax

R ⋈ **<Join Condition> S**

Where, R is a relation **R(A1, A2, ... , An)** of degree **n**

and S is a relation **S(B1, B2, ..., Bm)** of degree **m**

- The result of join is a another relation **Q(A1, A2, ... , An, B1, B2, ..., Bm)** of degree **m+n**
- **Q** has one tuple for each combination of tuples – **one from R and one from S** – whenever the combination **satisfies the join condition**
- JOIN operation is equivalent to CARTESIAN PRODUCT operation followed by a SELECT operation

Example : To retrieve the name of manager of each department

$\prod_{empname}$ (DEPARTMENT $\bowtie_{\textbf{empid=mgrid}}$ EMPLOYEE)

Various types of JOIN

a) **EQUIJOIN**

The JOIN operation, where the only comparison operator used is =, is called an EQUIJOIN

b) **NATURAL JOIN** , denoted by *

It is same as EQUIJOIN except that join attributes of second relation are not included in the resulting relation.

Syntax

$\mathbf{R} *_{\textbf{<Join Condition>}} \mathbf{S}$

If the join attributes have the same names , they do not have to be specified at all.

Syntax

R * S

Eg : To Lists employee tuples with details of their departments, provided the relationship attribute **deptno** has the same name in both relations.

EMPLOYEE * DEPARTMENT

c) **OUTER JOIN**

The result set of OUTER JOIN includes all tuples in relation R , or all tuples in relation S, or all in both relations, regardless of whether or not they have matching tuples in other relations.

There are three types of outer join :

LEFT OUTER JOIN *denoted by* ⟕

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 with NULL values.

RIGHT OUTER JOIN *denoted by* ⟖

The RIGHT OUTER JOIN operation keeps every tuple in the second, or right relation S in **R ⟖ S**

FULL OUTER JOIN *denoted*⟗

The FULL OUTER JOIN keeps all tuples in both the left and the right relations. When no matching tuples are found, padding them with NULL values as needed.

Eg: - Given two relations **EMPLOYEE & DEPARTMENT** with join attribute **deptno**

Various JOIN operations can be expressed as :

a) JOIN (INNER JOIN) : ρ_a **(DEPARTMENT)** $\bowtie_{\textbf{a.deptno=b.deptno}}$ ρ_b**(EMPLOYEE)**
b) LEFT JOIN : ρ_a **(DEPARTMENT)** ⟕$_{\textbf{a.deptno=b.deptno}}$ ρ_b**(EMPLOYEE)**
c) RIGHT JOIN : ρ_a **(DEPARTMENT)** ⟖$_{\textbf{a.deptno=b.deptno}}$ ρ_b**(EMPLOYEE)**
d) FULL OUTER JOIN : ρ_a **(DEPARTMENT)** ⟗$_{\textbf{a.deptno=b.deptno}}$ ρ_b**(EMPLOYEE)**
e) NATURAL JOIN : **DEPARTMENT * EMPLOYEE**

(Write an example instance)