EER to Relations Mapping

Source:
Fundamentals of Database Systems, 5th ed.
Ramez Elmasri and Shamkant B. Navathe
Chapter-7

PM Jat
pm_jat@daiict.ac.in

DBMS support for Inheritance

- OODBMS and ORDBMS support inheritance. You can create a relation that inherits from other; for example

```
CREATE TABLE cities (
   name text,
   population float,
   altitude int -- in feet
);

CREATE TABLE capitals (
state char(2)
) INHERITS (cities);
```

Source: http://intranet.daiict.ac.in/~pm_jat/postgres/html/ddl-inherit.html

EER to Relations

- However, classic relational model does not support inheritance
- In that case certain strategies can be used to represent the generalization and specialization in relational schema.
- Here we discuss approaches suggested by Elmasri and Navathe are presented here. (Chapter 7)

Generalization/Specialization to Relation

- Two type of approaches have been suggested here
  - Multiple Relation Approach
  - Single relation approach

- The Context to explain approaches:
  - Let us say we have a super-class C with attributes {<u>k</u>,a1,a2,…. an}, k is PK here.
  - C has m specialization subclasses {S1,S2,S3, …. Sm}.

Generalization/Specialization to Relation

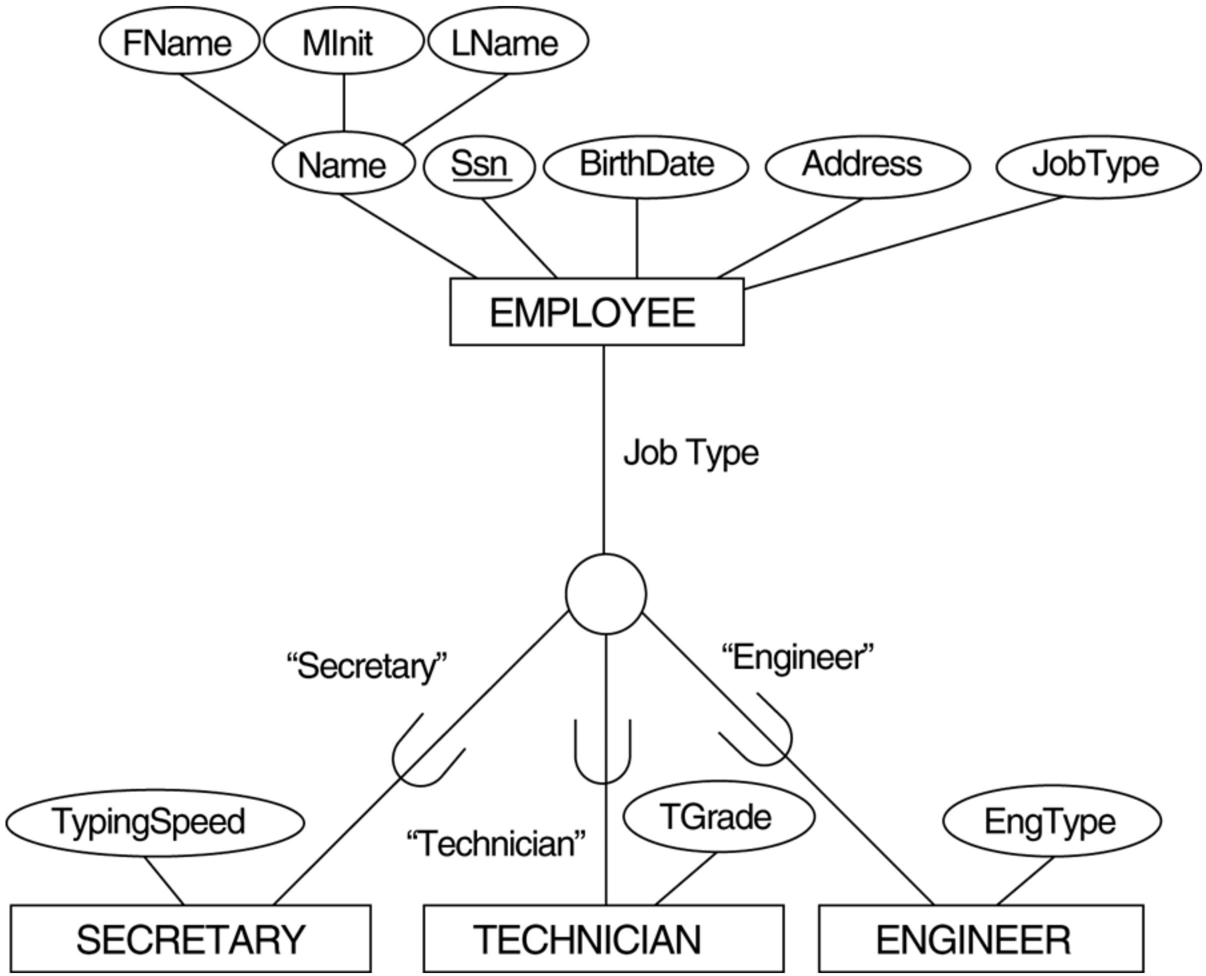
- Option 1: Relations for both sub and super classes

- Create a relation L for C with attributes Attrs(L) = {k,a1,a2,…. an} and make k as PK.

- Create relation Li for each subclass Si, 1 <= i <= m, with the attributes Attrs(Li)={k} U {attributes of Si}, and PK(Li)=k. (Example Next)

- The approach works well for total or partial and disjoint and overlapping specializations. Only the thing is you may have to JOIN Li with L to get details of super-class of tuples in sub-class relation

Example using option 1



Example using option 1

(a) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType

SECRETARY

SSN	TypingSpeed

TECHNICIAN

SSN	TGrade

ENGINEER

SSN	EngType

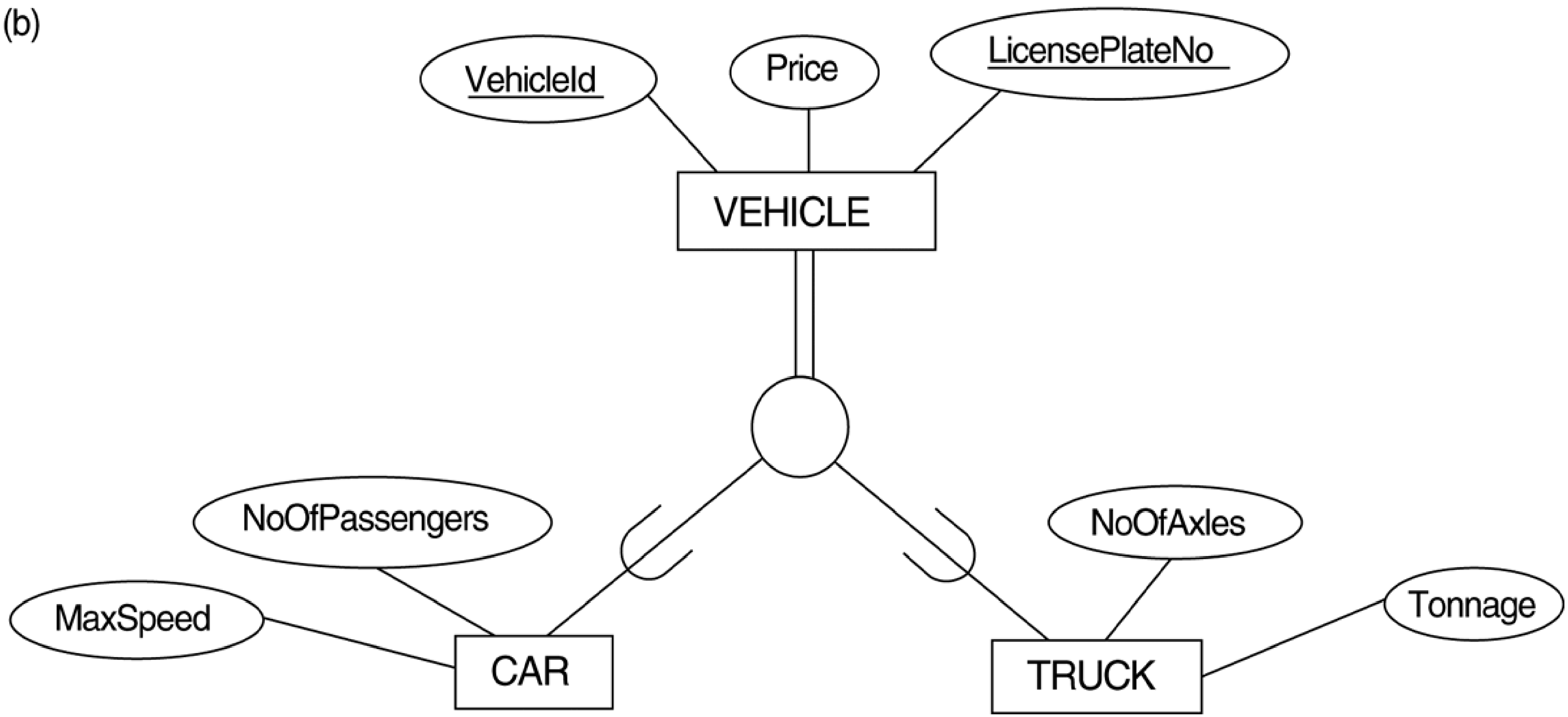
Generalization/Specialization to Relation

- Option 2: Relations for Subclasses only

- Create a relation Li for each subclass Si, 1 <= i <= m, with the attributes Attrs(Li) = {attributes of Si} U {k,a1,a2,…. an} and k is PK.

- This option works well when both the disjoint and total participation holds.

Example using option 2



Example using option 2

(b) CAR

<u>VehicleId</u>	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

TRUCK

<u>VehicleId</u>	LicensePlateNo	Price	NoOfAxles	

Generalization/Specialization to Relation

- In option 2, if specialization is not total, and an entity not belonging to any of sub-class then it is lost? Where do you place such entity?

- Also, when specialization is overlapping then entity C will be duplicated in several relations

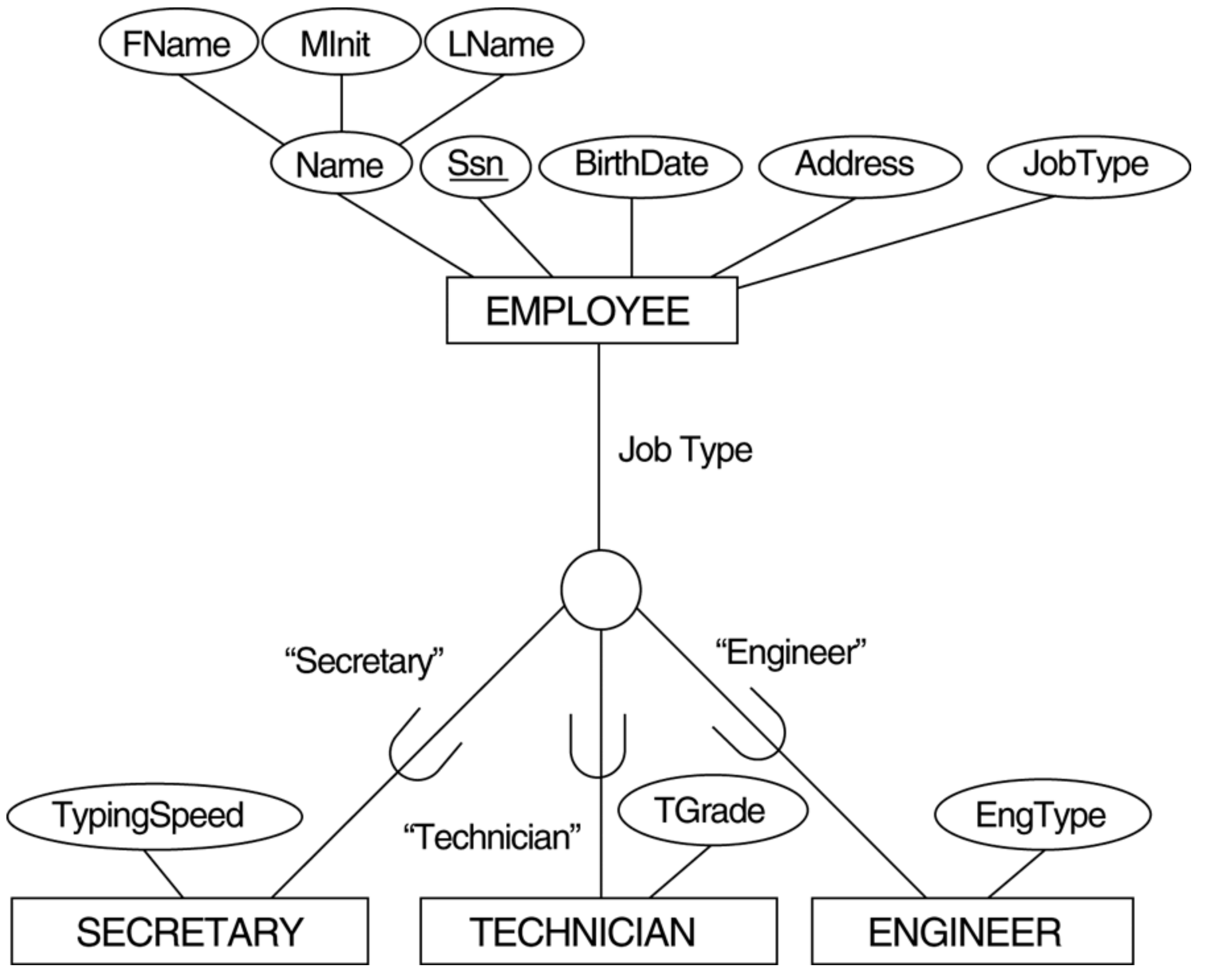
Generalization/Specialization to Relation

- Option 3: Single relation with one type attribute

- Create a single relation L with attributes Attrs(L)= {k,a1,a2,…. an} U {attributes of S1} U {attributes of S2} U …. {attributes of Sm} U {t} and PK(L)=k.

- The attribute t is called a type (or discriminating) attribute that indicates the subclass name of subclass to which the tuple belongs, if any

Example using option 3



Example using option 3

(c) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	

<-- t and other attributes of sub-classes-->

- <u>This option is good for specializations whose subclasses are *disjoint*</u>

- However, this option has potential for generating a large number of null values, if many attributes appear in sub-classes (tuple can belong to only one sub-class and attributes of others will be null)

- This option (and option 4 too) hence is not recommended if many attributes are defined for the subclasses.

- If few attributes, however are specified, these options can be preferred because in these case we don’t have to specify JOIN or UNION operations and can provide more efficient implementation.

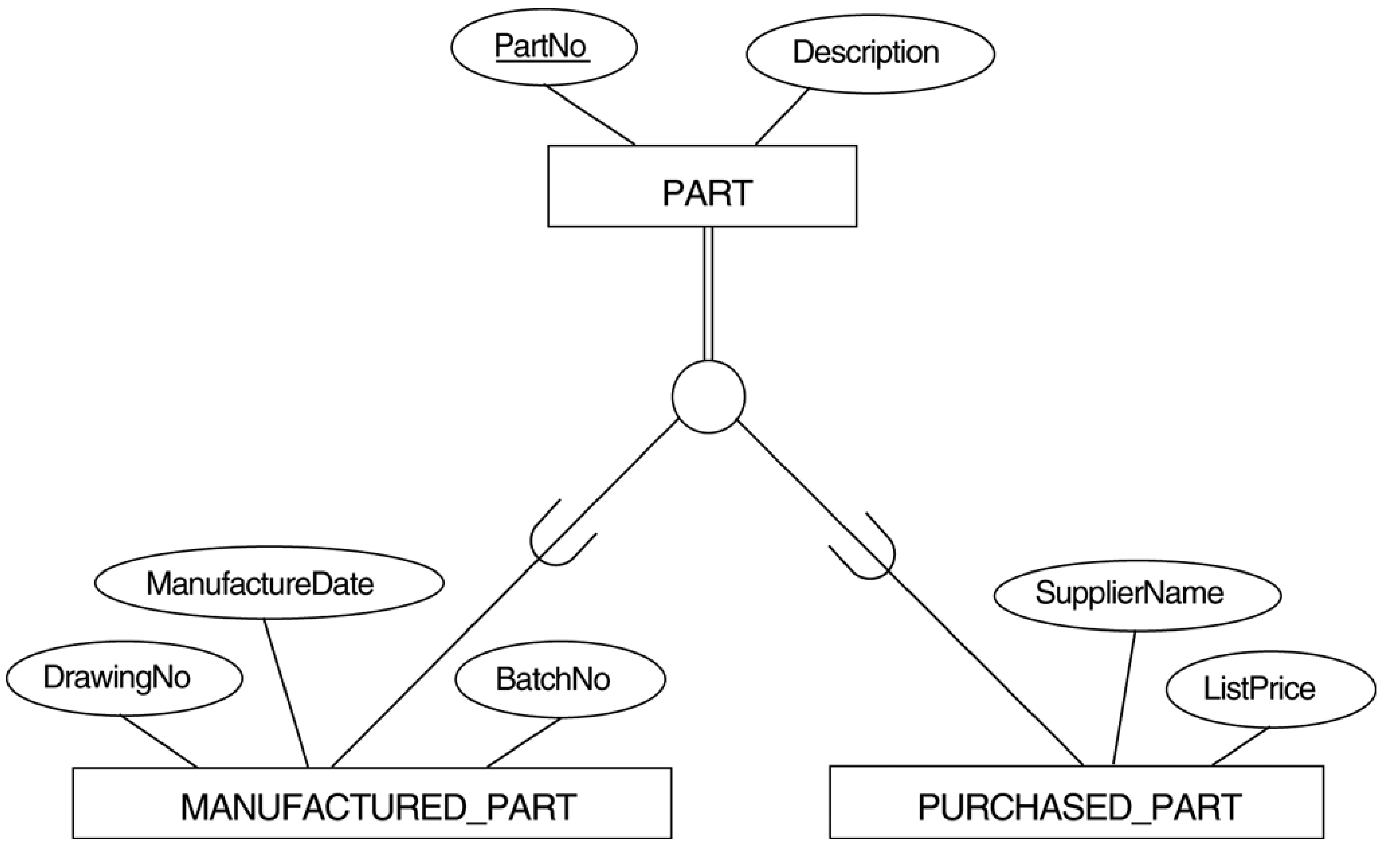
Generalization/Specialization to Relation

- Option 4: Single relation with multiple type attributes

- Create a single relation L with attributes Attrs(L)= {k,a1,a2,…. an} U {attributes of S1} U {attributes of S2} U …. {attributes of Sm} U {t1,t2,…. tm} and PK(L) = k.

- This option is for specializations whose subclasses are *overlapping*, but will also work for disjoint specialization, and each ti, $1<=i<=m$, is a Boolean attribute indicating whether a tuple belong to a specialization class Si.

Example using option 4



Example using option 4

- Option 4 is used to handle overlapping subclasses by including m Boolean type fields, one for each subclass

- Following is schema for ERD on previous slide with two boolean type fields MFlag(manufactured) and PFlag (purchase)

(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice

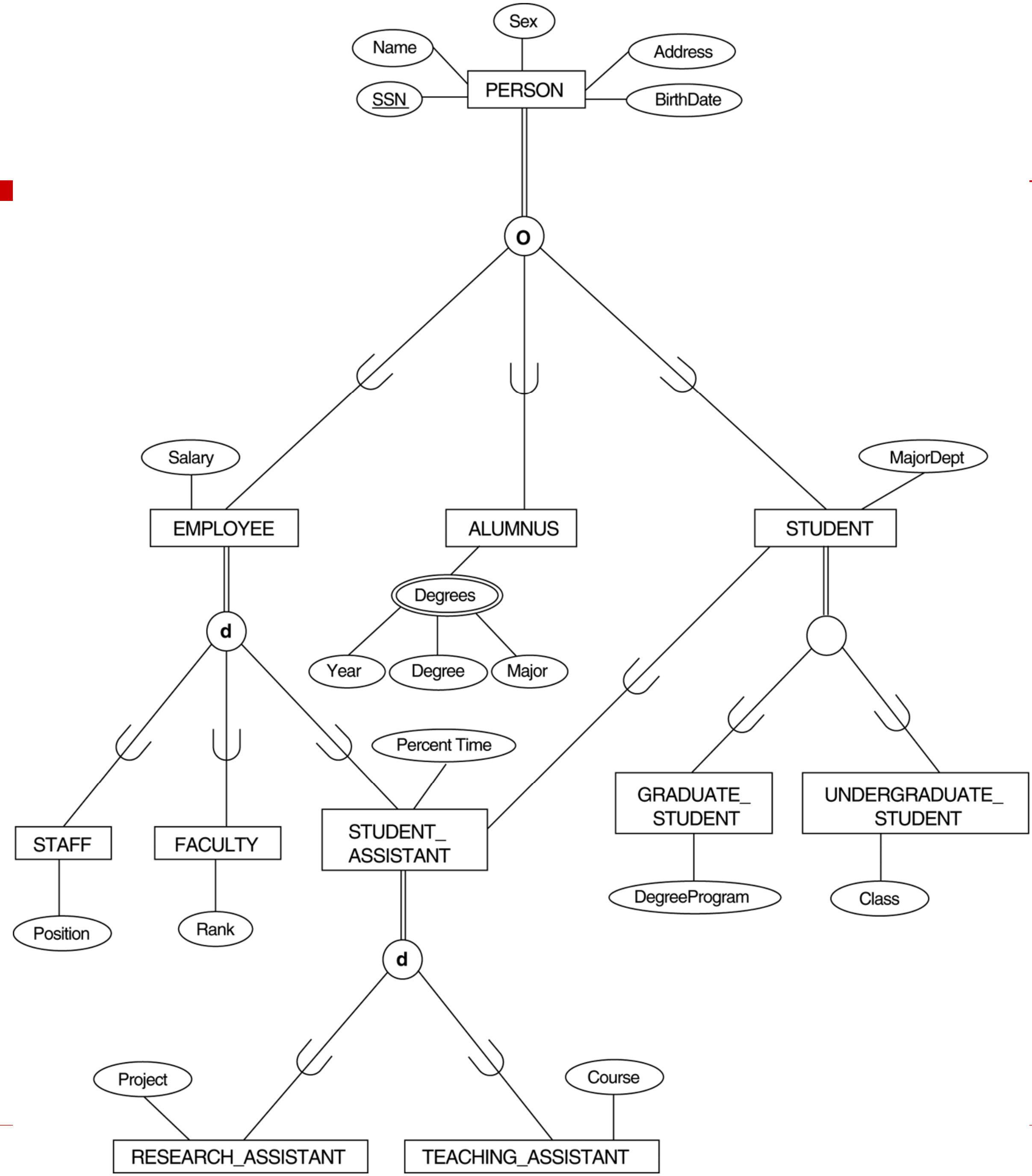
<---------Manufactured Part --------> <---- Purchased Part --->

Conclusion

- **In most cases option-1 is good.**
- However, to avoid join, we have go for option-2, but we may require to have UNION for searching generation (part) of entities
- If there are less number of attributes in sub-classes then option3 (for disjoint) and option 4 for overlapping specializations.
- Option 3 and 4 avoids JOIN and UNION operations to be performed.

Mapping of Multiple Inheritance

- In case of Multiple Inheritance, we can use one or combination of approaches (1 to 4) discussed.
- Let us consider the example (given on next slide) from the book.
- And consider the multiple inheritance of STUDENT_ASSISTANT sub-class



Sex
Name
Address
SSN
PERSON
BirthDate
O
Salary
MajorDept
EMPLOYEE
ALUMNUS
STUDENT
Degrees
d
Year
Degree
Major
Percent Time
STAFF
FACULTY
STUDENT_
ASSISTANT
GRADUATE_
STUDENT
UNDERGRADUATE_
STUDENT
Position
Rank
DegreeProgram
Class
d
Project
Course
RESEARCH_ASSISTANT
TEACHING_ASSISTANT

Mapping of Multiple Inheritance

- We use option 3 for EMPLOYEE and its sub-classes.
- We use option 4 for STUDENT and its sub-classes (so that we may not have to store attributes of STUD-ASSISTANT-ships.

Example-Mapping of Multiple Inheritance

PERSON

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

EMPLOYEE

					<Stud-Assist>		<TeachAssist>		<TeachAssist>
<u>SSN</u>	Salary	EmployeeType	Position	Rank	PercentTime	RAFlag	TAFlag	Project	Course
		<Discriminator>	<Staff>	<Faculty>		<Res-Assist>		<Res-Assist>	

ALUMNUS

<u>SSN</u>

ALUMNUS_DEGREES

<u>SSN</u>	<u>Year</u>	<u>Degree</u>	

STUDENT

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

Example (contd.)

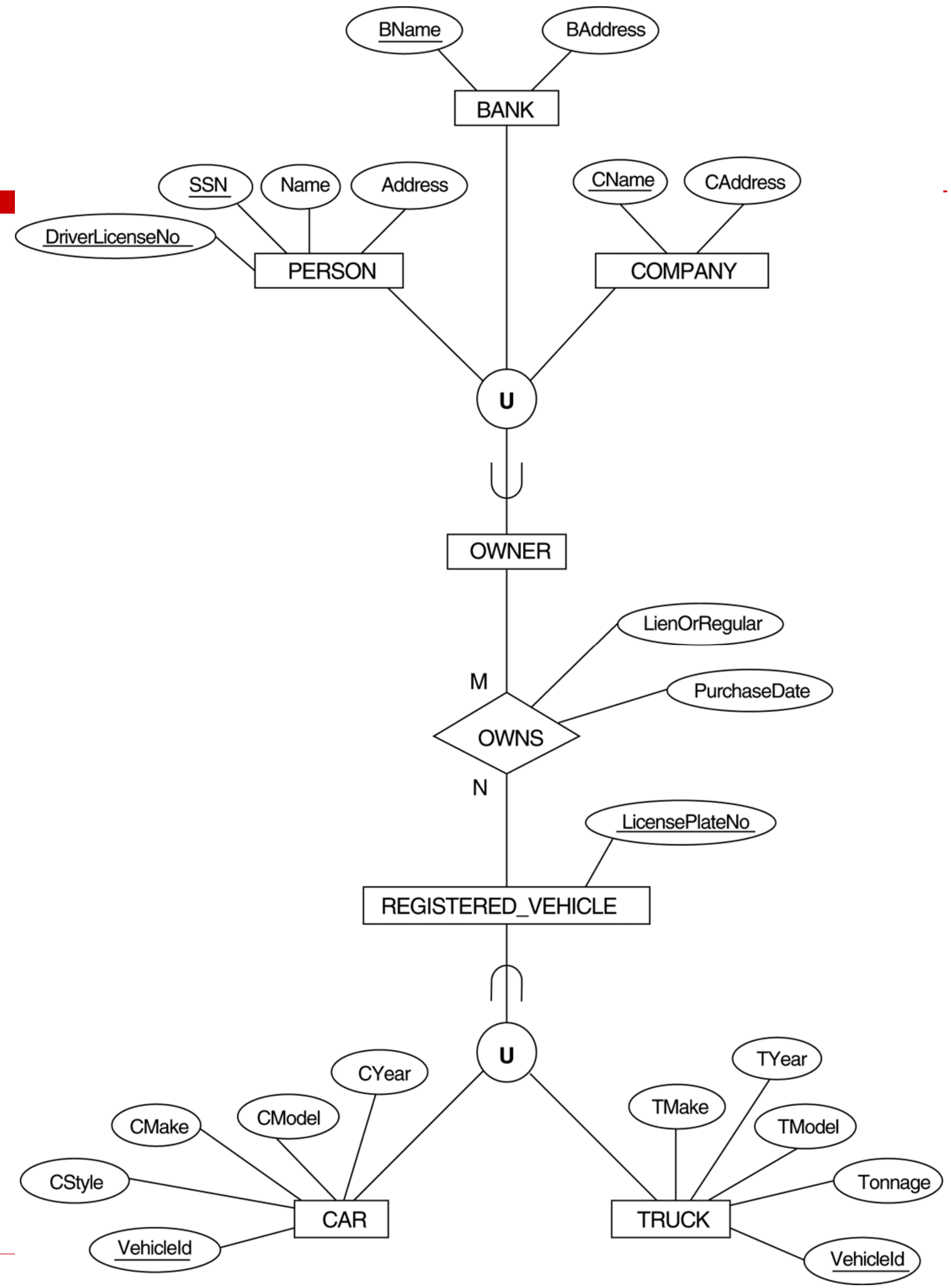
- Note that, Person-Employee, Person-Alumnus, and Person-Student sub-classing have been mapped using option 1.

Mapping of Union Types

- Term Category is also used for UNION relationship type (Possibly avoids confusion between union operations and union relationship type)

- 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.

Example



BName
BAddress
BANK
SSN
Name
Address
CName
CAddress
DriverLicenseNo
PERSON
COMPANY
U
OWNER
LienOrRegular
M
PurchaseDate
OWNS
N
LicensePlateNo
REGISTERED_VEHICLE
U
CYear
TYear
CModel
CMake
TMake
TModel
CStyle
Tonnage
CAR
TRUCK
VehicleId
VehicleId

Example: Mapping of Union

- We have relation OWNER for OWNER category.
- The primary key of the OWNER relation is the OwnerId -surrogate key, and VehicleID for Registered_Vehicle

PERSON

SSN	DriverLicenseNo	Name	Address	

BANK

BName	BAddress	OwnerId

COMPANY

CName	CAddress	OwnerId

OWNER

OwnerId

REGISTERED_VEHICLE

VehicleId	LicensePlateNumber

CAR

VehicleId	CStyle	CMake	CModel	

TRUCK

VehicleId	TMake	TModel	Tonnage	TYear

OWNS

OwnerId	VehicleId	PurchaseDate	LienOrRegular