Advanced Database

Course Code - ST2BDA

Specialty : IT

Course Credits: Jean-Charles Huet & Hanen OCHI

Dr. Lilia Sfaxi
Assistant professor

CM1 – Object Relational Database

Databases for IT Engineers

Dr. Lilia Sfaxi
Assistant Professor

01

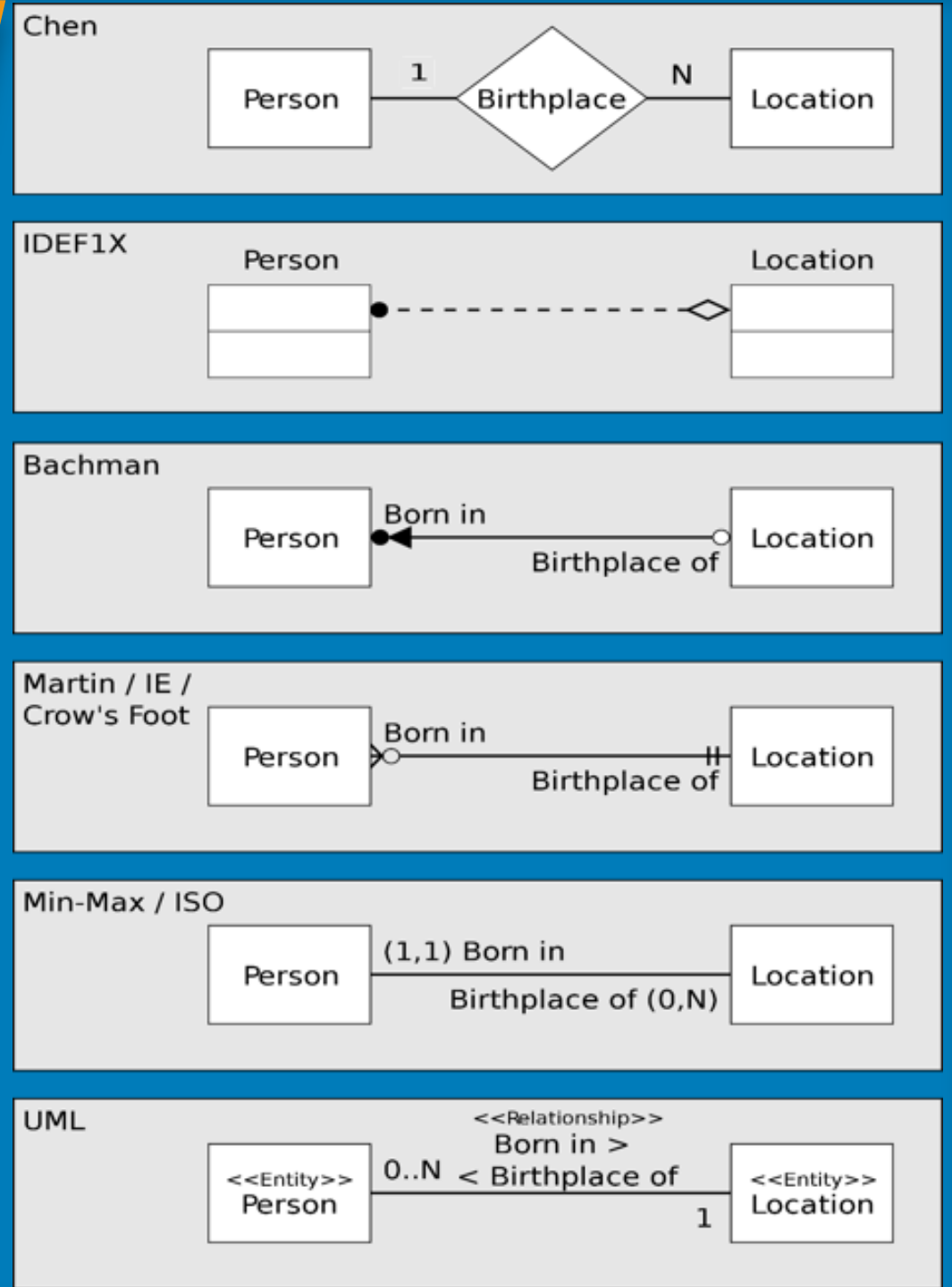
Conceptual Model to Logical Model

01

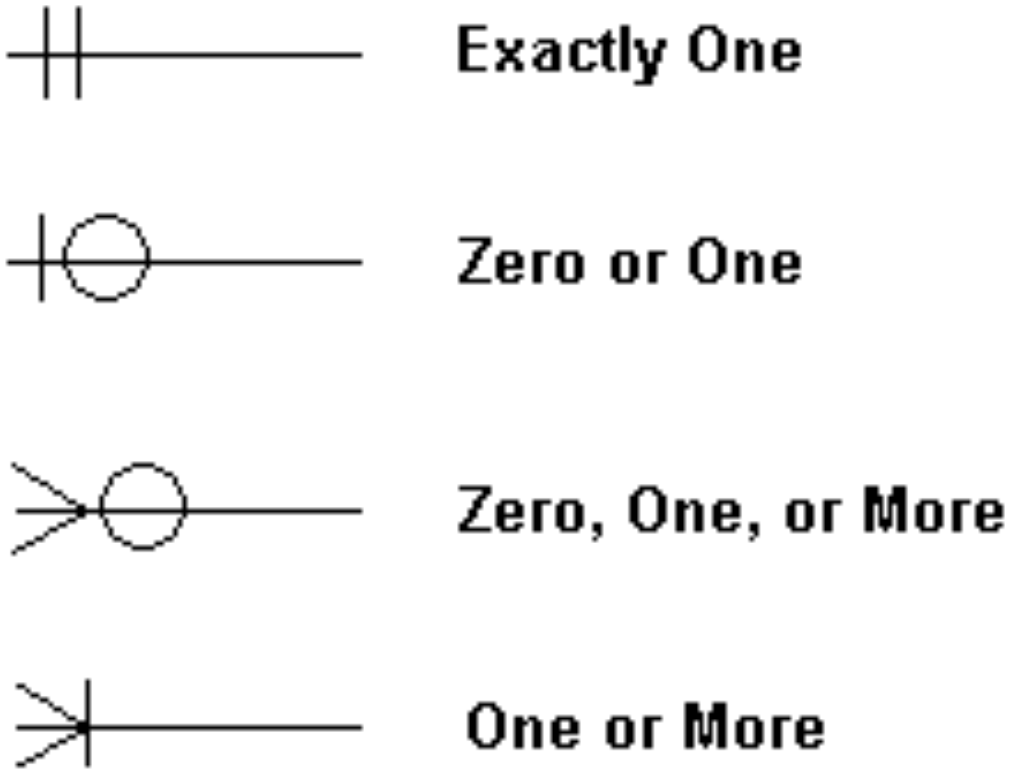
Developing a Conceptual Model

- Overall view of the database that integrates all the needed information discovered during the requirements analysis.
- Elements of the Conceptual Model are represented by diagrams, Entity-Relationship or ER Diagrams, that show the meanings and relationships of those elements independently of any particular database systems or implementation details.
- Can also be represented using other modeling tools (such as UML)

Some ER Diagram Styles



Crow's Feet Notation



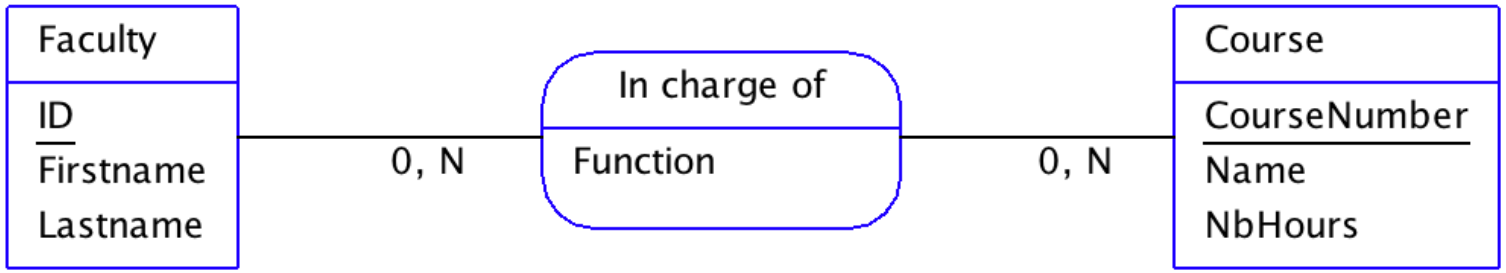
Logical Database Design

- The process of deciding how to arrange the attributes of the entities in the business environment into database structures, such as the tables of a relational database.

- The goal is to create well structured tables that properly reflect the company's business environment.

Entity-Relationship Model

- Entity: Any concrete or abstract concept that can be individualized
- Class or type of entities: group of entities with the same characteristics (generic level)
- Association: relation between multiple entities
- Class or type of association: group of associations with the same characteristics



Logical Design of Relational Database Systems

1. The conversion of E-R diagrams into relational tables.

2. The data normalization technique.

3. The use of the data normalization technique to test the tables resulting from the E-R diagram conversions.

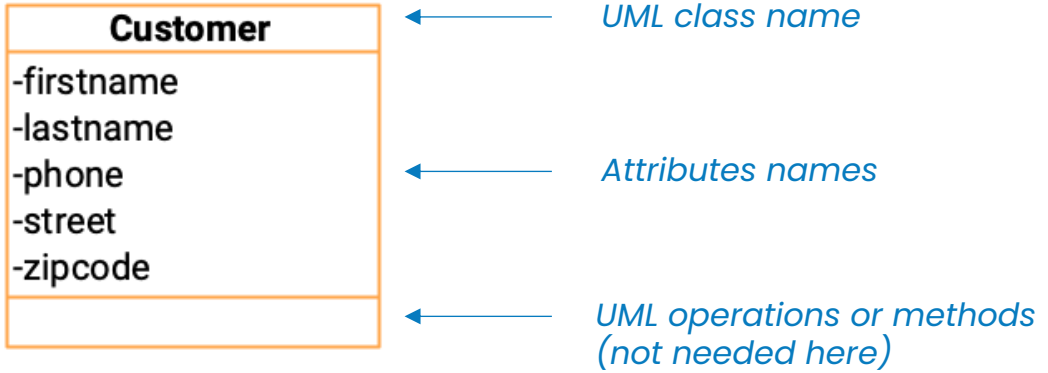
Basic Structures: Classes and Schemas

UML Class

- UML class (ER term: entity) is anything in the enterprise that is to be represented in our database.
  - The first step in modeling a class is to describe it in natural language.
    - *Example*: build a sales database. Let's start by defining customer using natural language.
- Attribute (properties) is a piece of information that characterizes each member of a class.
  - Descriptive attributes (natural attribute) are those which actually provide real-world information about the class.
  - UML only uses descriptive attributes
  - ID number are not descriptive attributes

Basic Structures: Classes and Schemas
Class Diagram

- A class diagram shows the class name and list of attributes that identify data elements we need to know about each *member* (*instance*, occurrence of a class)
- The Customer class represents any person who has done business with us or who we think might do business with us in the future. Its attributes are:
  - Customer first name.
  - Customer last name.
  - Customer phone.
  - Customer street.
  - Customer zip code.



Basic Structures: Classes and Schemas

Relation Schema

- In an OO programming language, each class is **instantiated** with **objects** of that class. In building a relational database, each class is first translated into a relation model schema. The scheme starts with all of the attributes from the class diagram.
- *Example:* **Customers**(cFirstName, cLastName, cPhone, cSteet, cZipCode)
  - Schema name: Customers
  - Attributes:
    - Customer first name, a person's first name.
    - Customer last name, a person's last name.
    - Customer phone, a valid telephone number.
    - Customer street, a street address.
    - Customer zip code, a zip code designated by the United States Postal Service.

Basic Structures: Classes and Schemas

Table Structure

- When we actually build the database, each relation schema becomes one table.

```
CREATE TABLE Customers (
        firstname VARCHAR(20) NOT NULL,
        lastname VARCHAR(20) NOT NULL,
        phone VARCHAR(20) NOT NULL,
        street VARCHAR(50),
        zipcode VARCHAR(5)
);
```

Converting E-R Diagrams into Relational Tables

- Each entity will become a table.
- Each many-to-many relationship or associative entity will become a table.
- During the conversion, certain rules must be followed to ensure that foreign keys appear in their proper places in the tables.

Converting a Simple Entity

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire

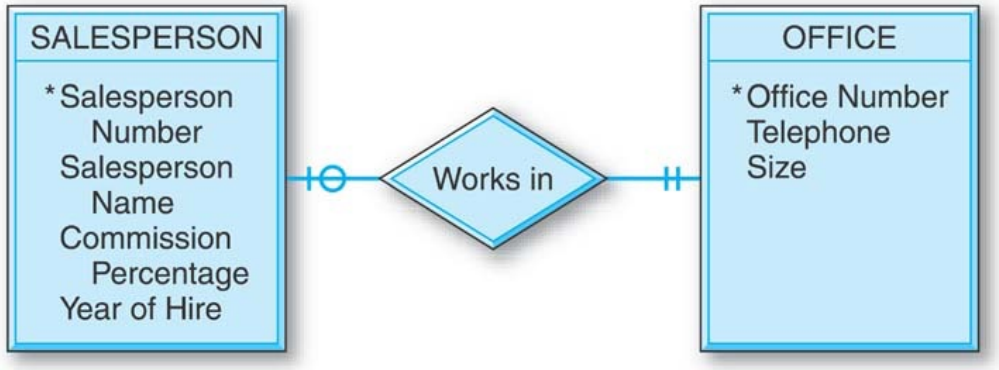
SALESPERSON

- The table simply contains the attributes that were specified in the entity box.

- *Salesperson Number* is underlined to indicate that it is the unique identifier of the entity and the primary key of the table.

Converting Entities in Binary Relationships

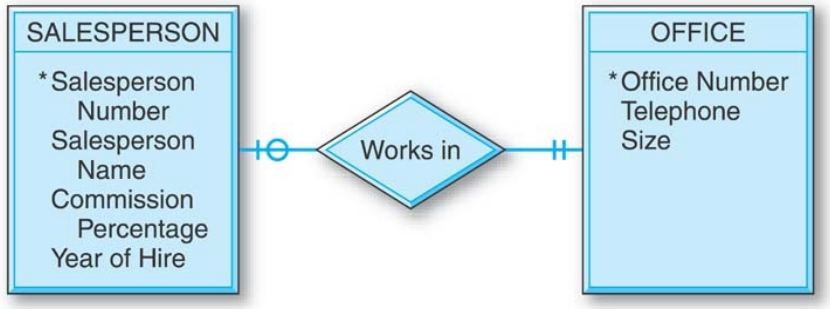
One-to-One



There are three options for designing tables to represent this data.

One-to-One: Option #1

- The two entities are combined into one relational table.



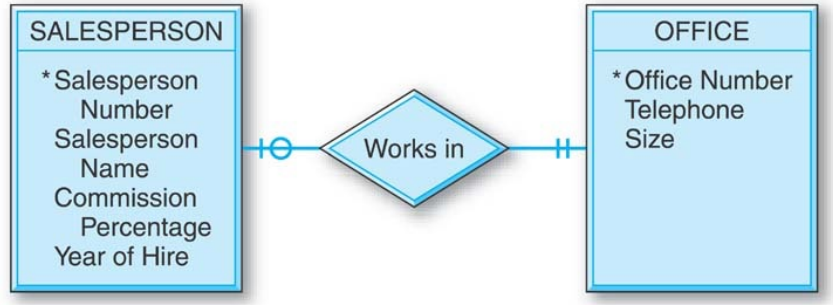
Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number	Telephone	Size
SALESPERSON/OFFICE						

One-to-One: Option #2

- Separate tables for the SALESPERSON and OFFICE entities, with Office Number as a foreign key in the SALESPERSON table.

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number
SALESPERSON				

Office Number	Telephone	Size
OFFICE		



One-to-One: Option #3

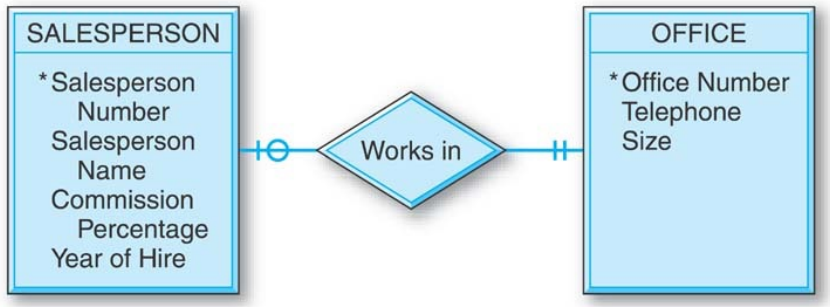
- Separate tables for the SALESPERSON and OFFICE entities, with Salesperson Number as a foreign key in the OFFICE table.

Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire

SALESPERSON

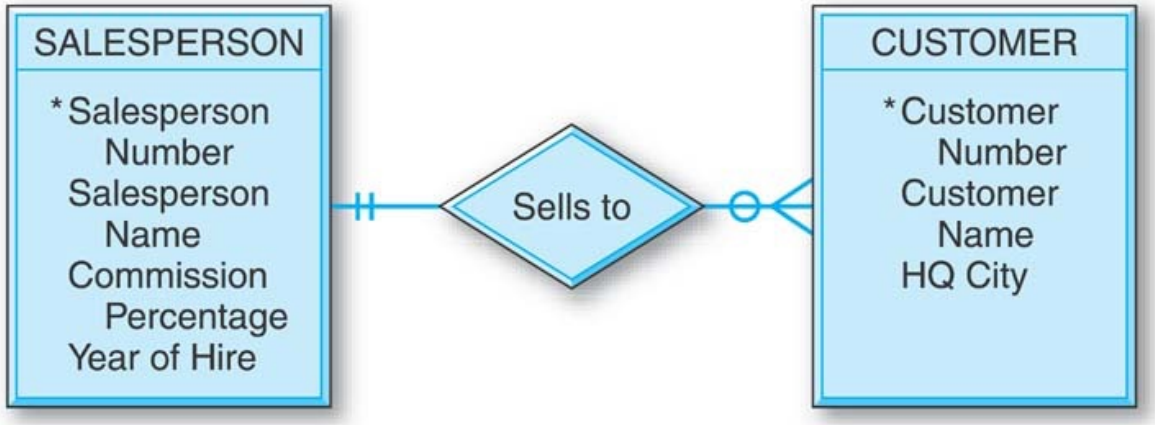
Office Number	Telephone	Salesperson Number	Size

OFFICE



Converting Entities in Binary Relationships

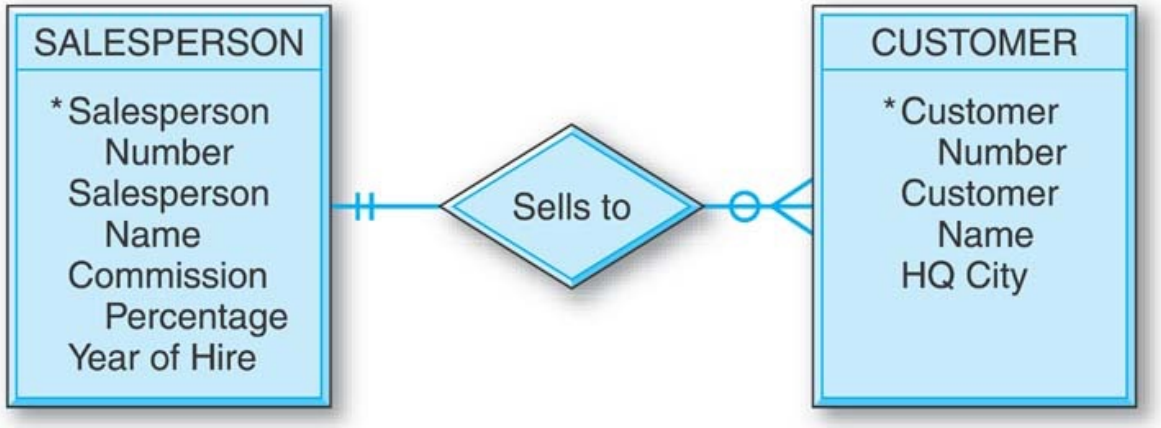
One-to-Many



- The unique identifier of the entity on the “one side” of the one-to-many relationship is placed as a foreign key in the table representing the entity on the “many side.”
- So, the Salesperson Number attribute is placed in the CUSTOMER table as a foreign key.

Converting Entities in Binary Relationships

One-to-Many



Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire
SALESPERSON			

Customer Number	Customer Name	HQ City	Salesperson Number
CUSTOMER			

Converting Entities in Binary Relationships

Many-to-Many

E-R diagram with the many-to-many binary relationship and the equivalent diagram using an associative entity.

Converting Entities in Binary Relationships
Many-to-Many

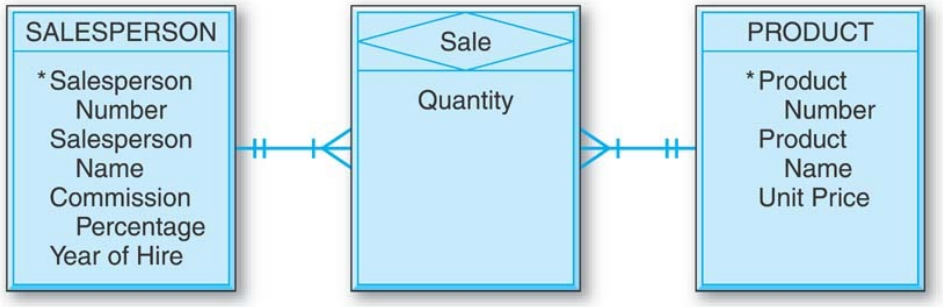
- An E-R diagram with two entities in a many-to-many relationship converts to three relational tables.

- Each of the two entities converts to a table with its own attributes but with no foreign keys (regarding this relationship).

- In addition, there must be a third table for the many-to-many relationship.

Converting Entities in Binary Relationships
Many-to-Many

- The primary key of SALE is the combination of the unique identifiers of the two entities in the many-to-many relationship. Additional attributes are the intersection data.



Product Number	Product Name	Unit Price

PRODUCT

Salesperson Number	Product Number	Quantity

SALE

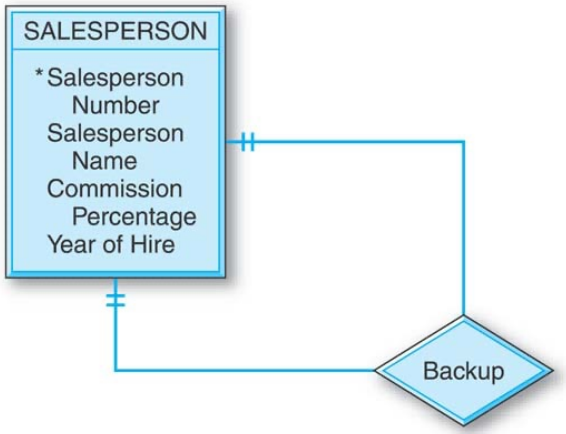
Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire

SALESPERSON

Converting Entities in Unary Relationships

One-to-One

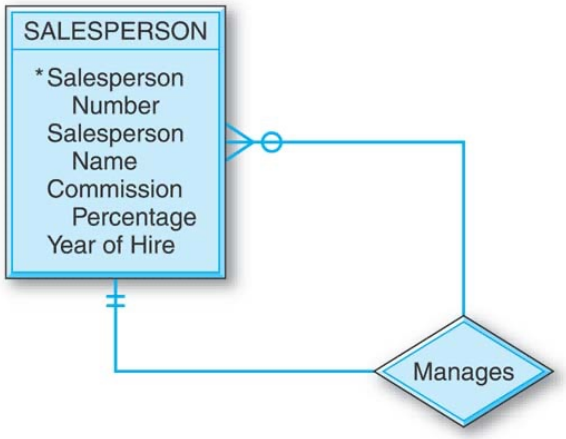
- With only one entity type involved and with a one-to-one relationship, the conversion requires only one table.



Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Backup Number
SALESPERSON				

Converting Entities in Unary Relationships
One-to-Many

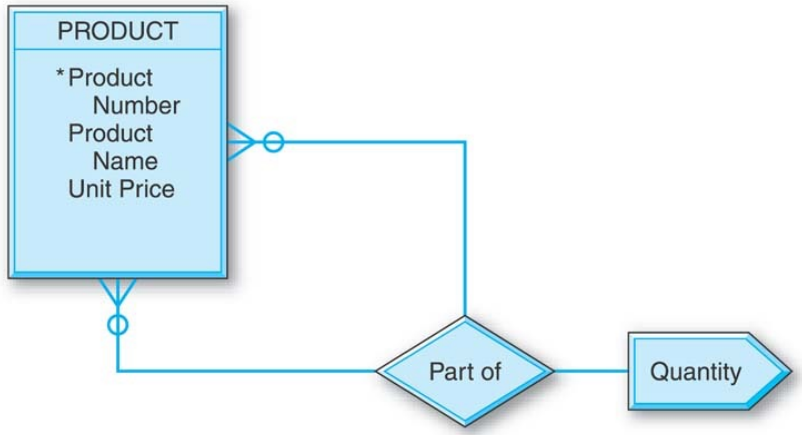
- Very similar to the one-to-one unary case.



Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Manager
SALESPERSON				

Converting Entities in Unary Relationships
Many-to-Many

- This relationship requires two tables in the conversion.
- The PRODUCT table has no foreign keys.
- A second table is created since in the conversion of a many-to-many relationship of any degree — unary, binary, or ternary — the number of tables will be equal to the number of entity types (one, two, or three, respectively) plus one more table for the many-to-many relationship.



Product Number	Product Name	Unit Price

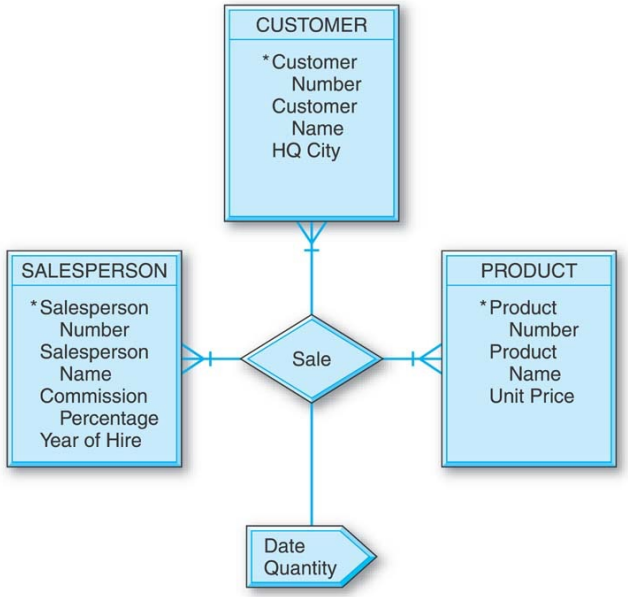
PRODUCT

Product Number	Sub-Assembly Number	Quantity

COMPONENT

Converting Entities in Ternary Relationships

- The primary key of the SALE table is the combination of the unique identifiers of the three entities involved, plus the Date attribute.
  - Adding the Date attribute to the primary key is optional, in case you want the same customer to buy the same product from the same salesperson more than once.



Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire

SALESPERSON

Product Number	Product Name	Unit Price

PRODUCT

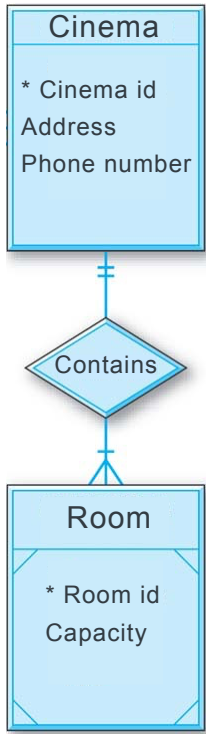
Customer Number	Customer Name	HQ City

CUSTOMER

Salesperson Number	Customer Number	Product Number	Date	Quantity

SALE

Weak Entity



Cinema id	Address	Phone number
CINEMA		

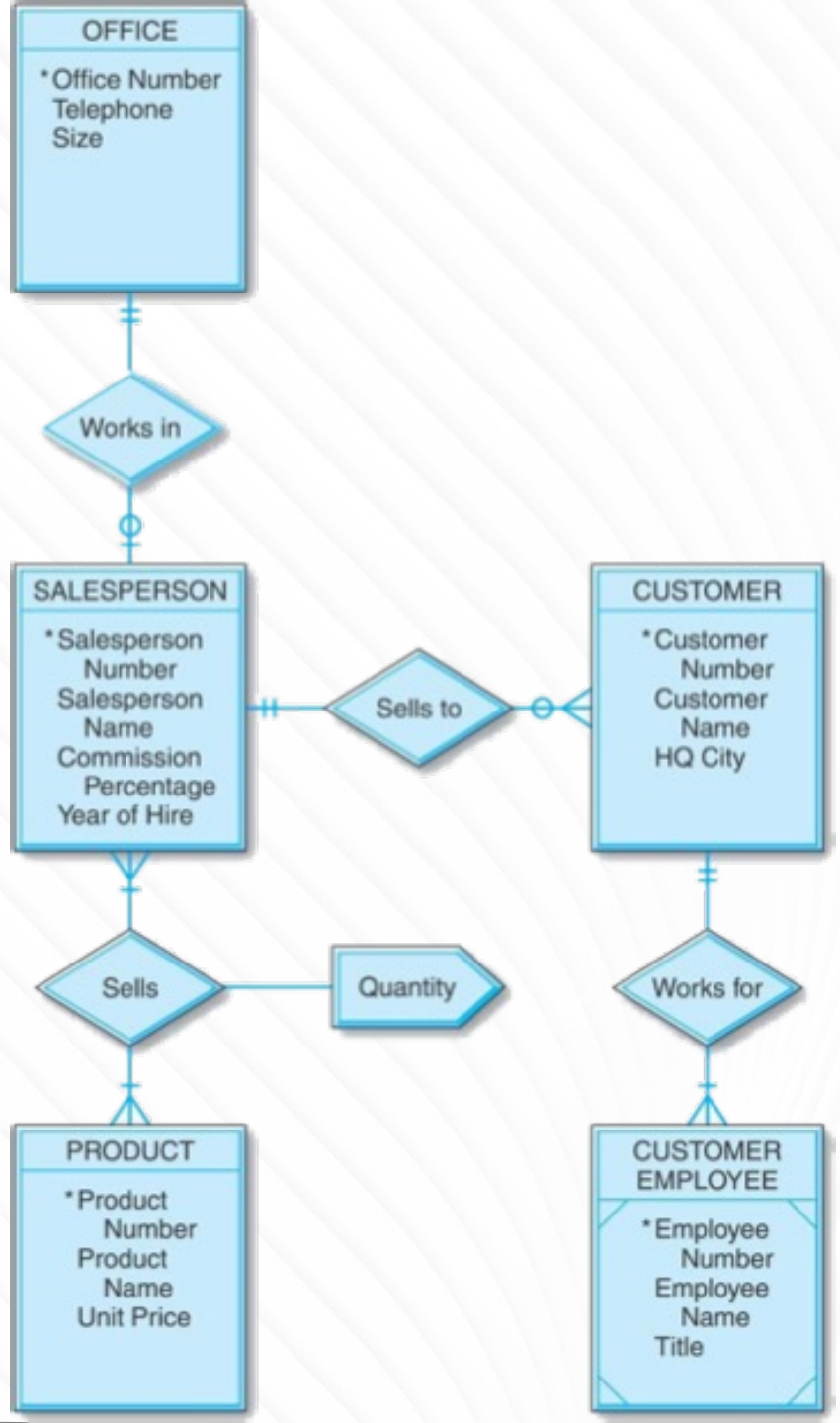
Cinema id	Room id	Capacity
ROOM		

- There are cases where an entity can only exist in close association with another entity, and is identified in relation to that other entity. This is called a **weak entity**.
- It is difficult to imagine representing a room without it being linked to its cinema. Indeed, it is at the cinema level that some general information such as the address or the telephone number will be found.
- It can be considered much more natural to number the rooms by an **internal number for each cinema**.
- The identifier of a room becomes:
  - The **Cinema id**, which indicates in which cinema the room is located;
  - The **room id** within the cinema.
- The room entity does not have an *absolute identification*, but an identification relative to another entity. Of course, this forces the room to always be associated with one and only one cinema.

02

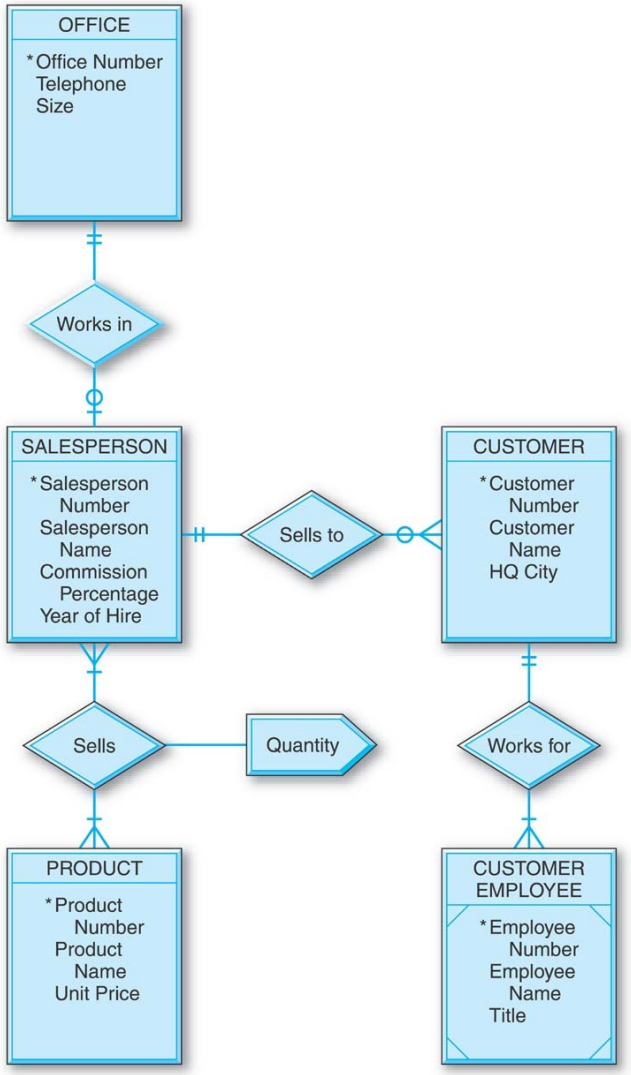
Design Examples

02



The General Hardware Company Conceptual Model

Designing the General Hardware Company Database



Salesperson Number	Salesperson Name	Commission Percentage	Year of Hire	Office Number

SALESPERSON

Customer Number	Customer Name	Salesperson Number	HQ City

CUSTOMER

Customer Number	Employee Number	Employee Name	Title

CUSTOMER EMPLOYEE

Product Number	Product Name	Unit Price

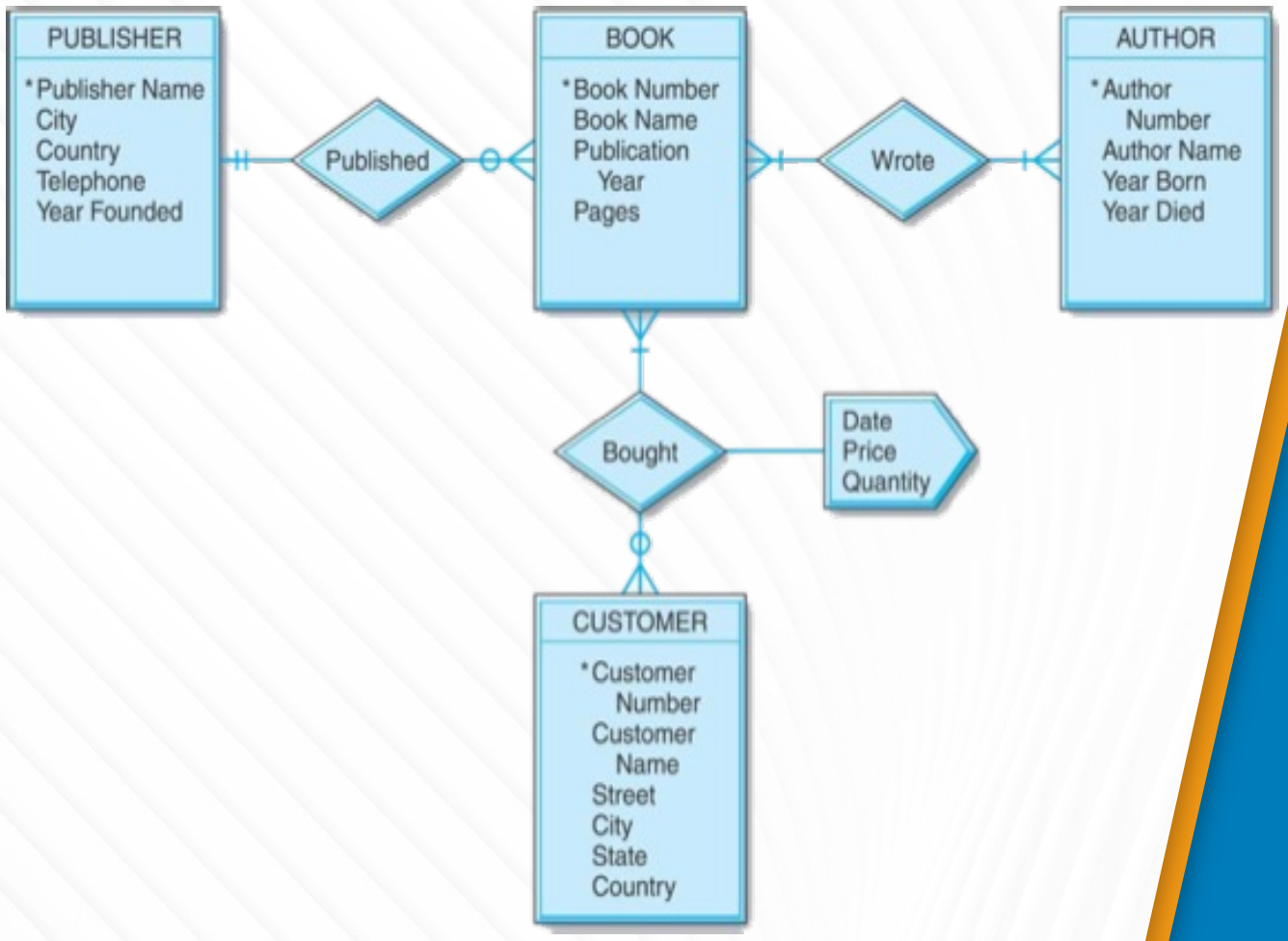
PRODUCT

Salesperson Number	Product Number	Quantity

SALES

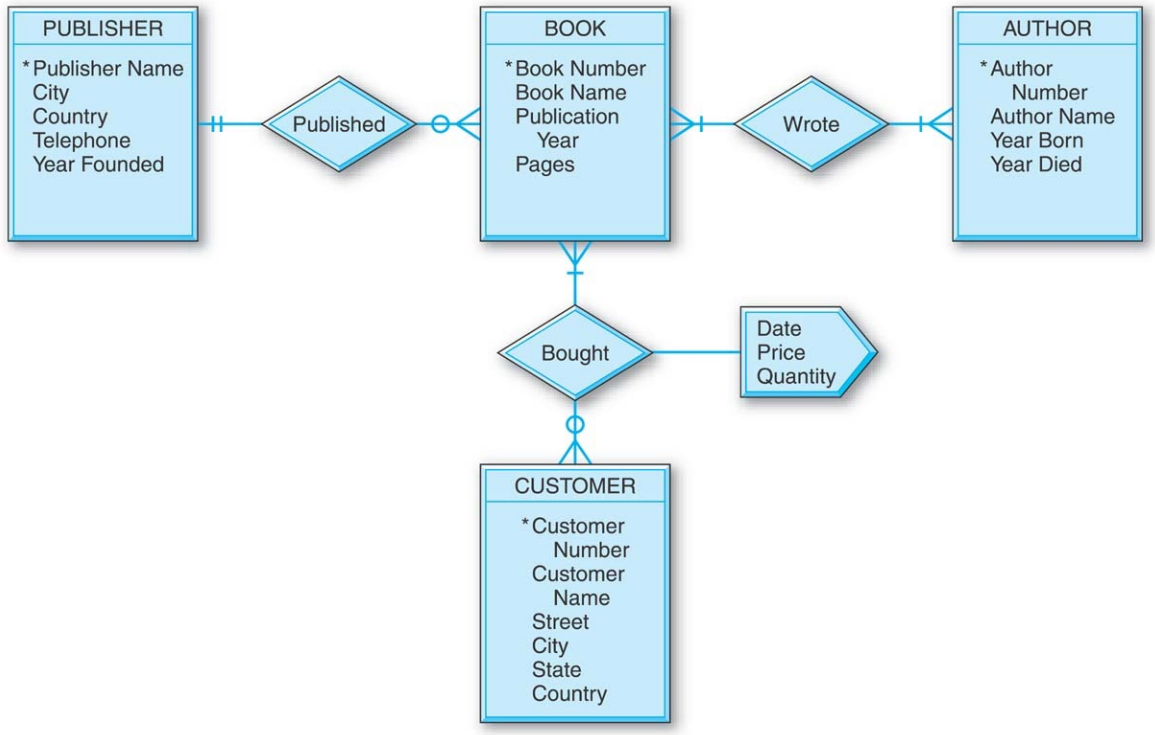
Office Number	Telephone	Size

OFFICE



The Good Reading Bookstores Conceptual Model

Designing the Good Reading Bookstores Database



Publisher Name	City	Country	Telephone	Year Founded

PUBLISHER

Author Number	Author Name	Year Born	Year Died

AUTHOR

Book Number	Author Number

WRITING

Book Number	Book Name	Publication Year	Pages	Publisher Name

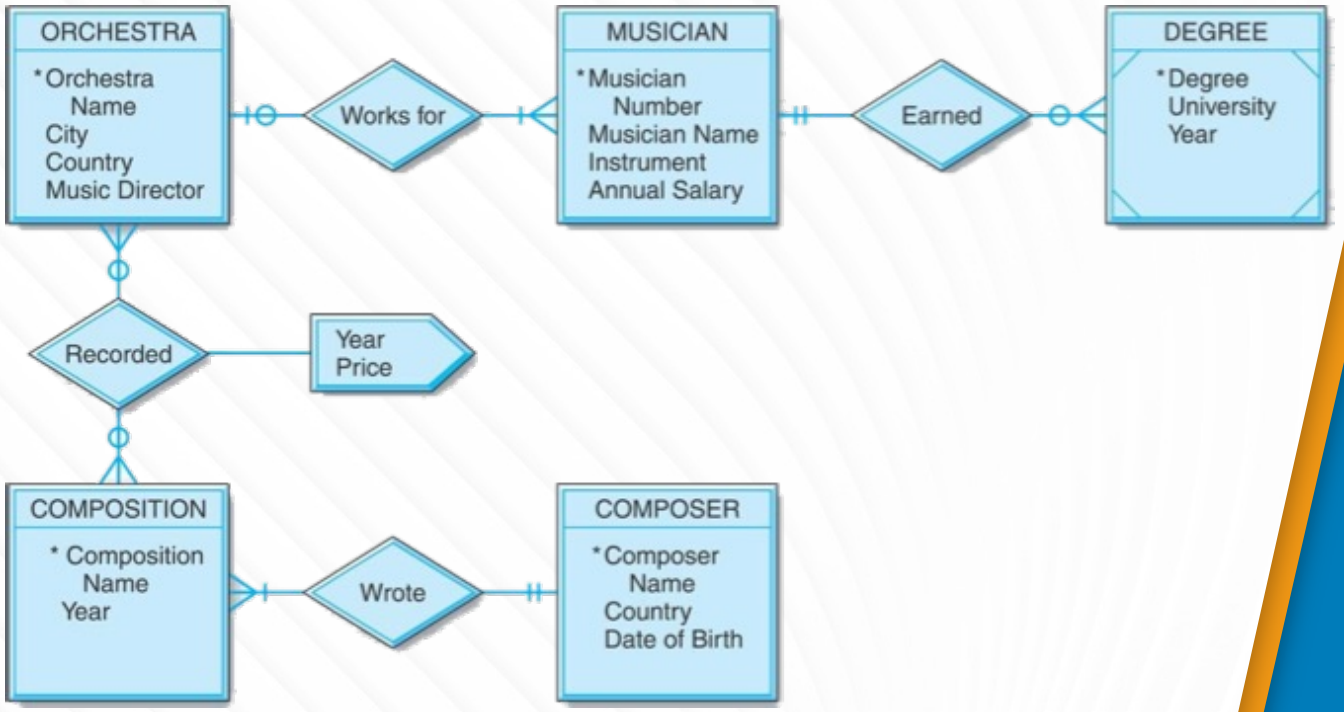
BOOK

Customer Number	Customer Name	Street	City	State	Country

CUSTOMER

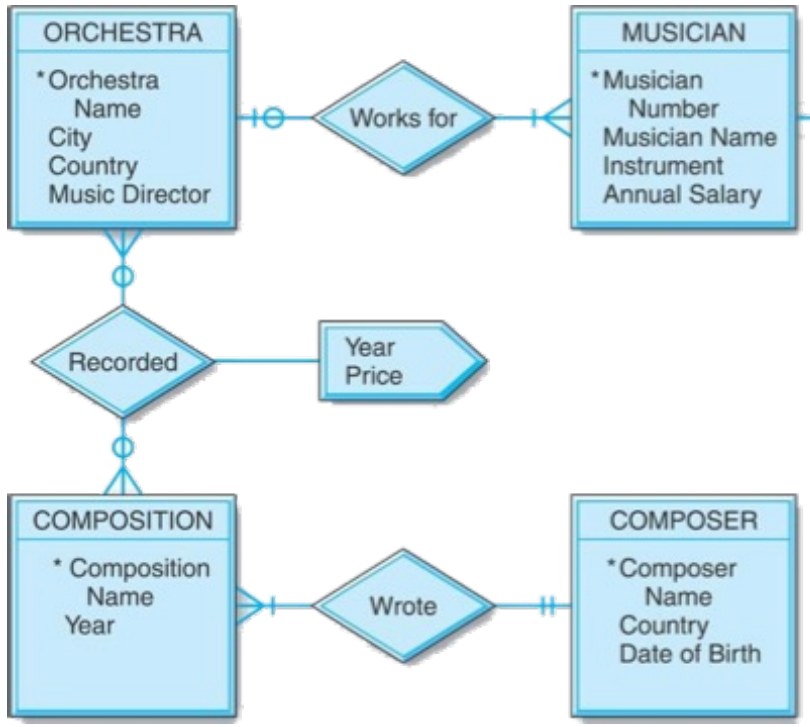
Book Number	Customer Number	Date	Price	Quantity

SALE



The World Music Association Conceptual Model

Designing the World Music Association Database



Orchestra Name	City	Country	Music Director

ORCHESTRA

Musician Number	Degree	University	Year

DEGREE

Musician Number	Musician Name	Instrument	Annual Salary	Orchestra Name

MUSICIAN

Orchestra Name	Composition Name	Year	Price

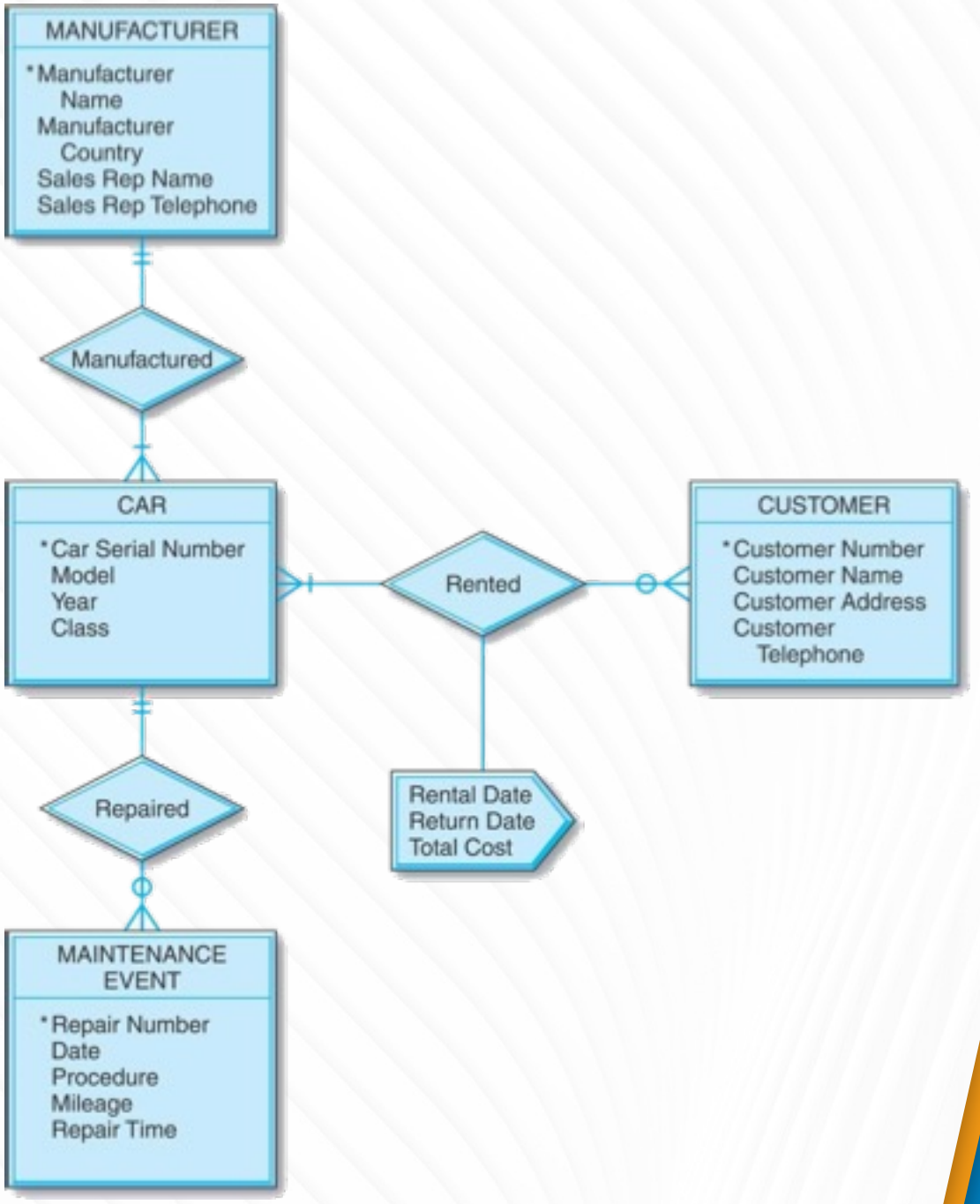
RECORDING

Composition Name	Composer Name	Year

COMPOSITION

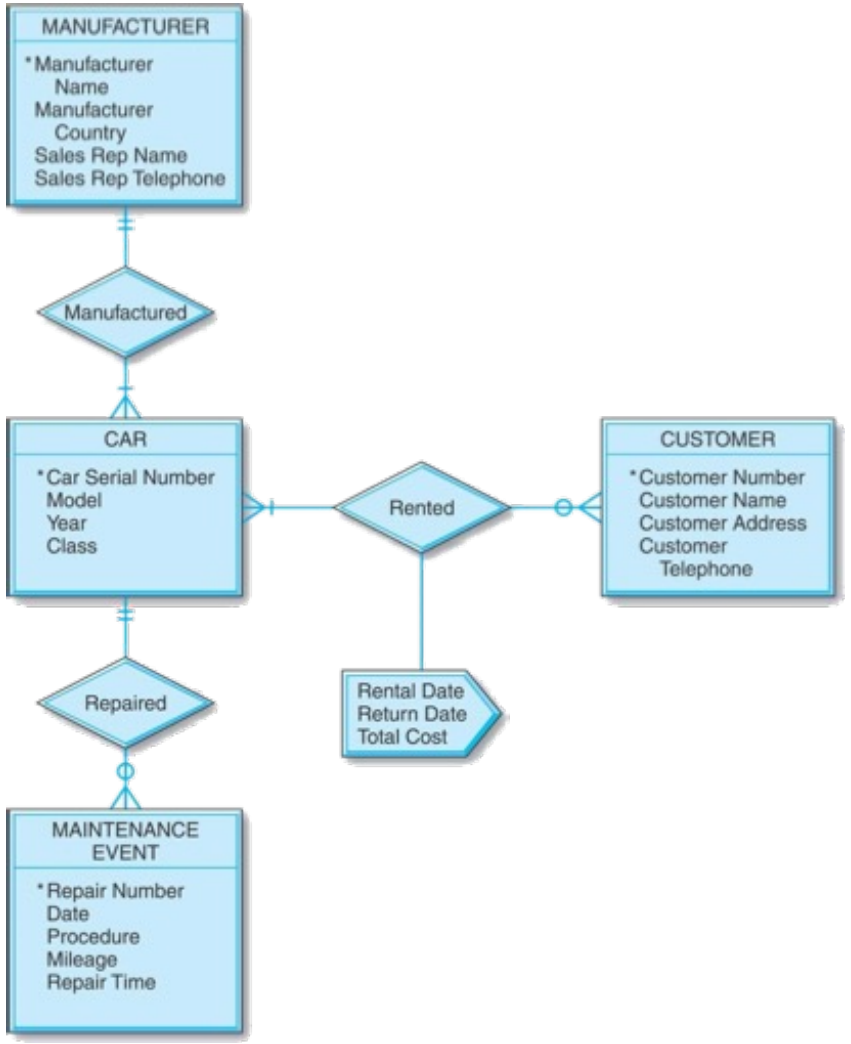
Composer Name	Country	Date Of Birth

COMPOSER



The Lucky Rent-A-Car Conceptual Model

Designing the Lucky Rent-A-Car Database



Manufacturer Name	Manufacturer Country	Sales Rep Name	Sales Rep Telephone

MANUFACTURER

Car Serial Number	Model	Year	Class	Manufacturer Name

CAR

Repair Number	Car Serial Number	Date	Procedure	Mileage	Repair Time

MAINTENANCE EVENT

Customer Number	Customer Name	Customer Address	Customer Telephone

CUSTOMER

Car Serial Number	Customer Number	Rental Date	Return Date	Total Cost

RENTAL

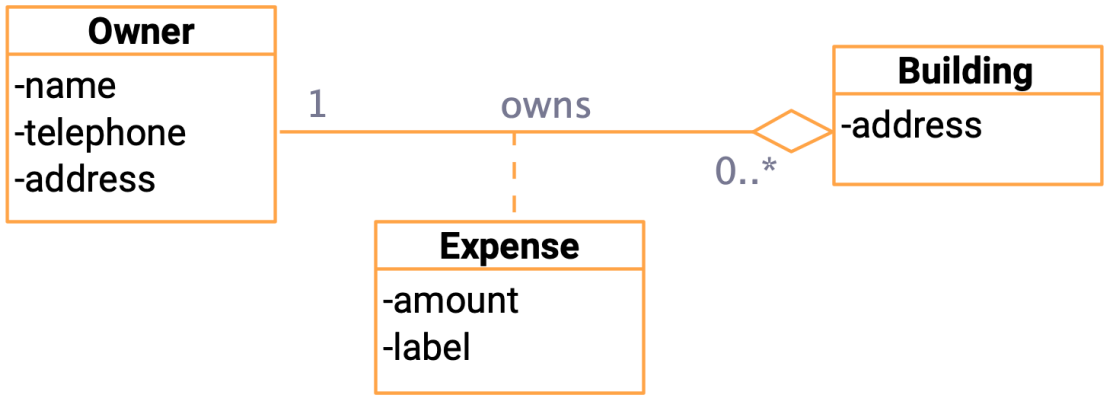
03

Advanced Database Modeling

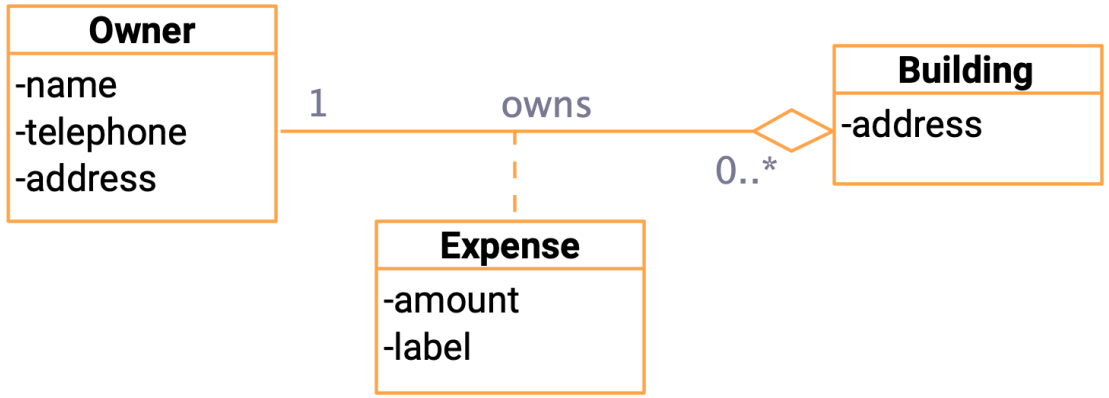
03

Aggregation

- Description of complex entities' types.
- *One type of associations between entities' types is considered as a new type of entities*

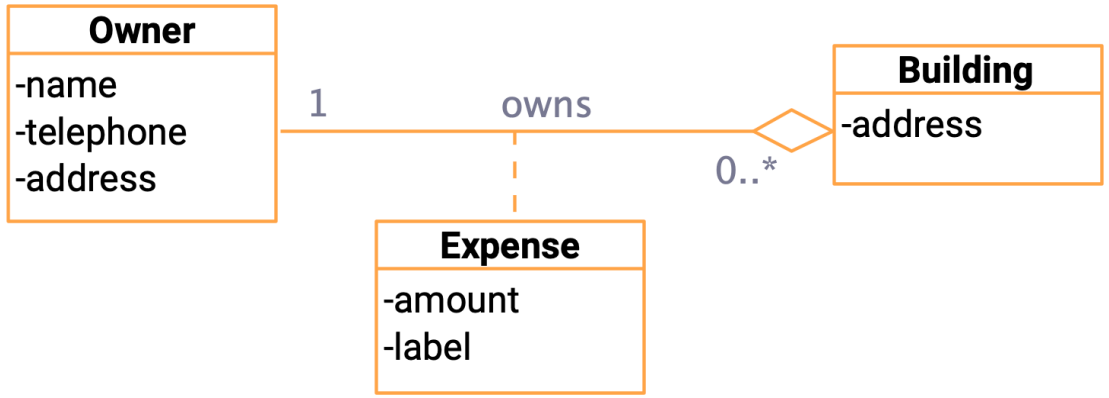


Conversion of Aggregation Relationships



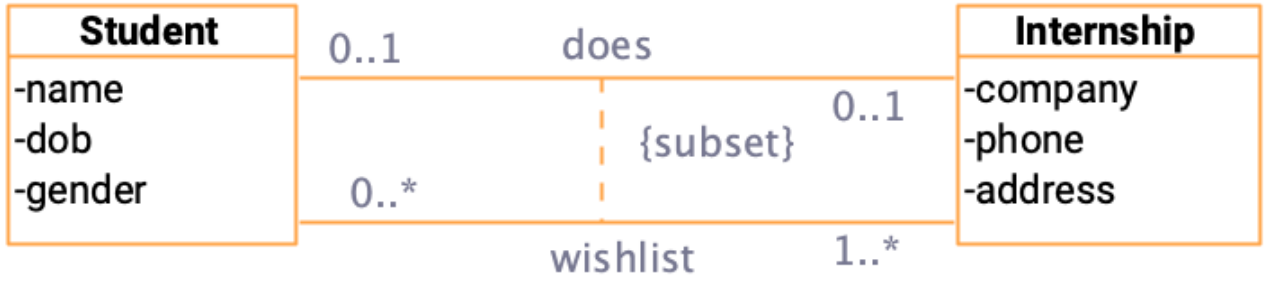

```
CREATE TABLE Owner (
        number VARCHAR(7),
        name VARCHAR(10),
        phone VARCHAR(15),
        address VARCHAR(50),
        constraint PK_OWNER
                Primary Key (number)
);

CREATE TABLE Building (
        number VARCHAR(7),
        address VARCHAR(50),
        constraint PK_BUILDING
                Primary Key (number)
);
```

Conversion of Aggregation Relationships




```
CREATE TABLE Expense (
        numOwner VARCHAR(7),
        numBuilding VARCHAR(7),
        amount NUMBER(10,2),
        label VARCHAR(50),
        constraint PK_EXPENSE
          Primary Key (numOwner,numBuilding)
        constraint FK_EXP_Ow
          Foreign Key (numOwner)
          References Owner(number),
        constraint FK_EXP_Buil
          Foreign Key (numBuilding)
          References Building(number)
);
```

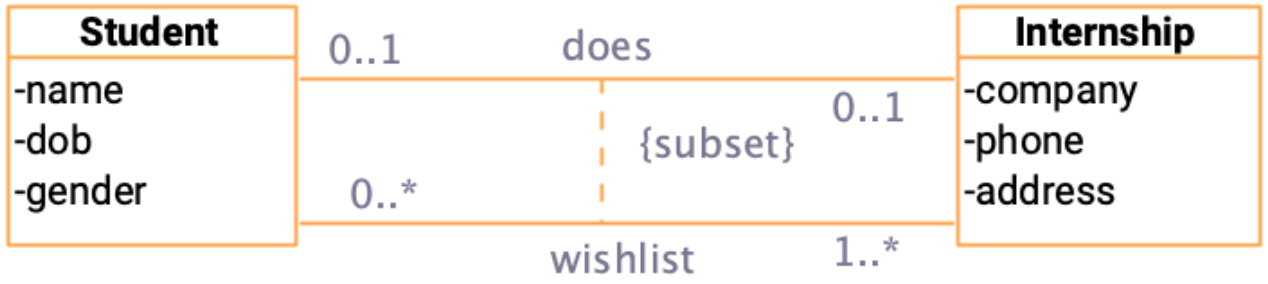
Inclusion constraint




```
CREATE TABLE Internship (
        number VARCHAR(7),
        company VARCHAR(40),
        phone VARCHAR(15),
        address VARCHAR(50),
        constraint PK_INTERNSHIP
                Primary Key (number)
);
```

```
CREATE TABLE Student (
        number VARCHAR(7),
        name VARCHAR(10),
        dob DATE,
        gender CHAR(1),
        numInt VARCHAR(7),
        constraint PK_STUDENT
                Primary Key (number),
        constraint FK_STU_INTER
          Foreign Key (numInt)
          References Internship(number)

);
```

Inclusion constraint

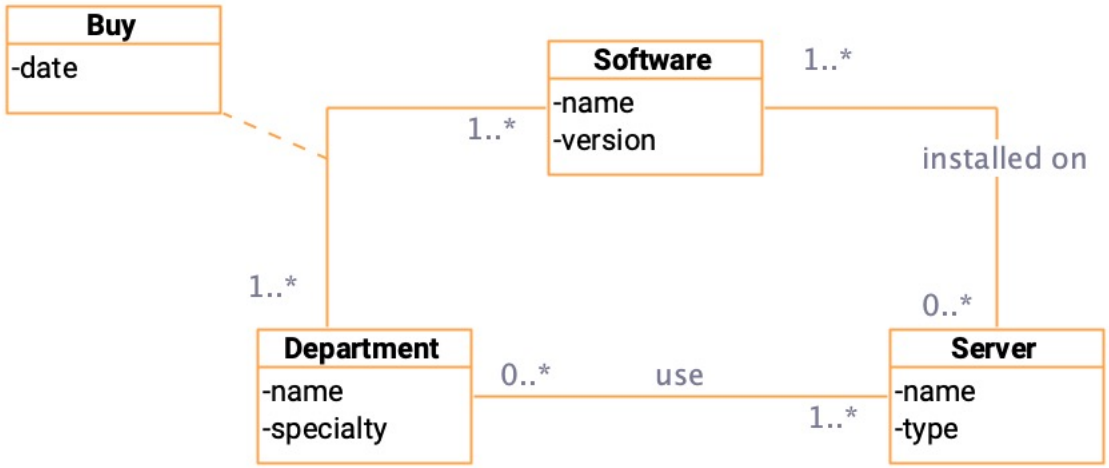



```
CREATE TABLE Wishlist (
        numberStu VARCHAR(7),
        numberInt VARCHAR(7),
        constraint PK_WISHLIST
                Primary Key (numberStu,numberInt),
        constraint FK_STU_WISH
          Foreign Key (numberStu)
          References Student(number),
        constraint FK_INT_WISH
          Foreign Key (numberInt)
          References Internship(number)
);
```

```
ALTER TABLE Student ADD
        Constraint K_DOES_WISH
        Foreign Key (number,numInt)
        References Wishlist(numberStu, numberInt);
```

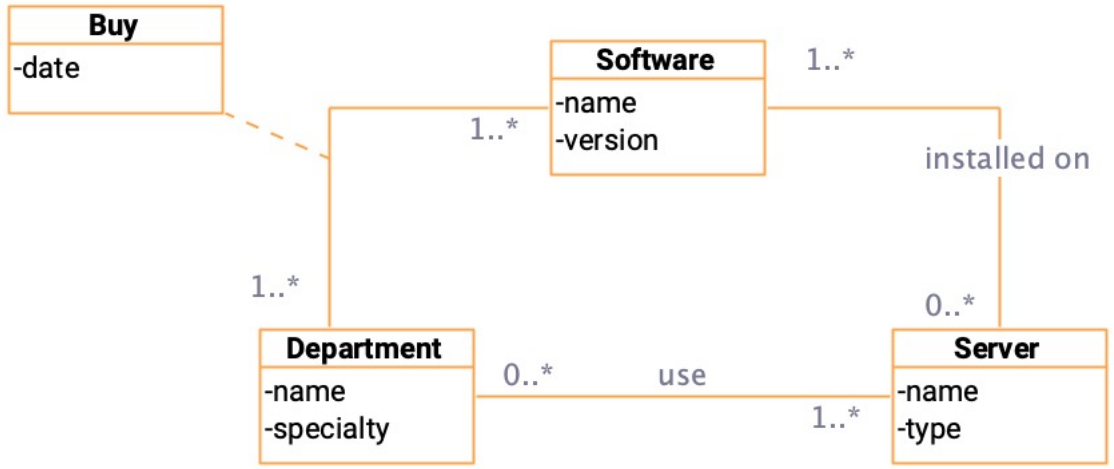
Inclusion constraint (2)

- Inclusion constraint: The software must be installed on a server of the department that purchased the program.



- *The software L purchased by the department D is installed on a server S for, among other things, this department*

Inclusion constraint (2)

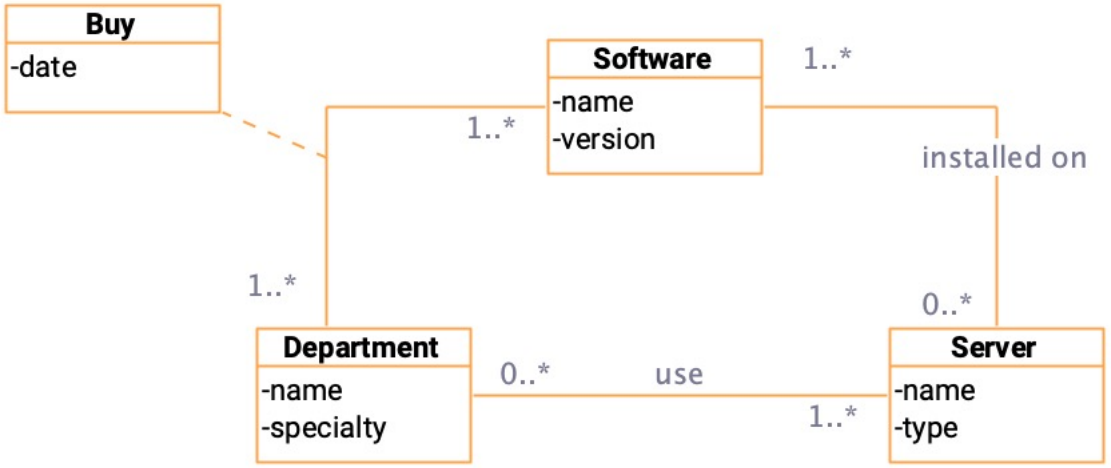



```
CREATE TABLE SERVER (
        number NUMBER(7),
        name VARCHAR(10),
        type VARCHAR(15),
        constraint PK_SERV
                Primary Key (number)
);
```

```
CREATE TABLE Department (
        number NUMBER(7),
        name VARCHAR(10),
        specialty VARCHAR(15),
        constraint PK_DEP
                Primary Key (number)
);
```

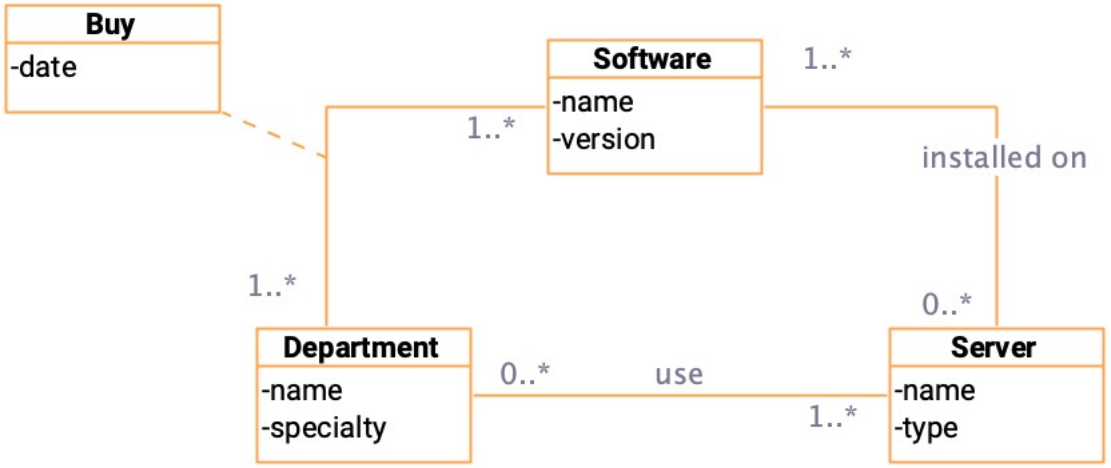
```
CREATE TABLE Software (
        number NUMBER(7),
        name VARCHAR(10),
        version VARCHAR(10),
        constraint PK_SOFTWARE
                Primary Key (number)
);
```

Inclusion constraint (2)



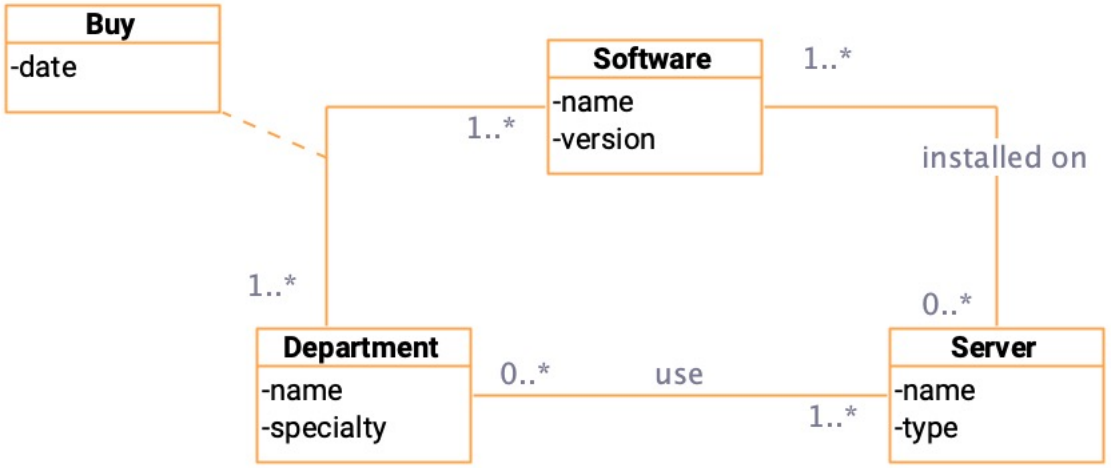

```
CREATE TABLE Buy (
        numDep NUMBER(7),
        numSof NUMBER(7),
        date DATE,
        constraint PK_BUY
                Primary Key (numDep,numSof),
        constraint FK_BUY_DEP
          Foreign Key (numDep)
          References Department(number),
        constraint FK_BUY_SOF
          Foreign Key (numSof)
          References Software(number)
);
```

Inclusion constraint (2)




```
CREATE TABLE Use (
        numDep NUMBER(7),
        numServ NUMBER(7),
        constraint PK_USE
                Primary Key (numDep,numServ),
        constraint FK_USE_DEP
          Foreign Key (numDep)
          References Department(number),
        constraint FK_USE_SERV
          Foreign Key (numServ)
          References Server(number)
);
```

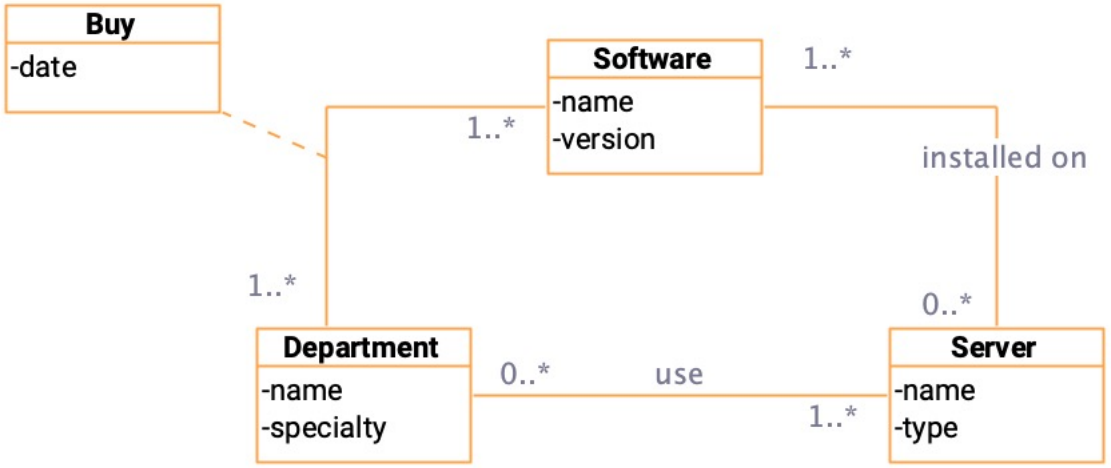
Inclusion constraint (2)




```
CREATE TABLE Install (
        numSof NUMBER(7),
        numServ NUMBER(7),
        constraint PK_INSTALL
                Primary Key (numSof,numServ),
        constraint FK_INST_SOF
          Foreign Key (numSof)
          References Software(number),
        constraint FK_INST_SERV
          Foreign Key (numServ)
          References Server(number)
);
```

Inclusion constraint (2)

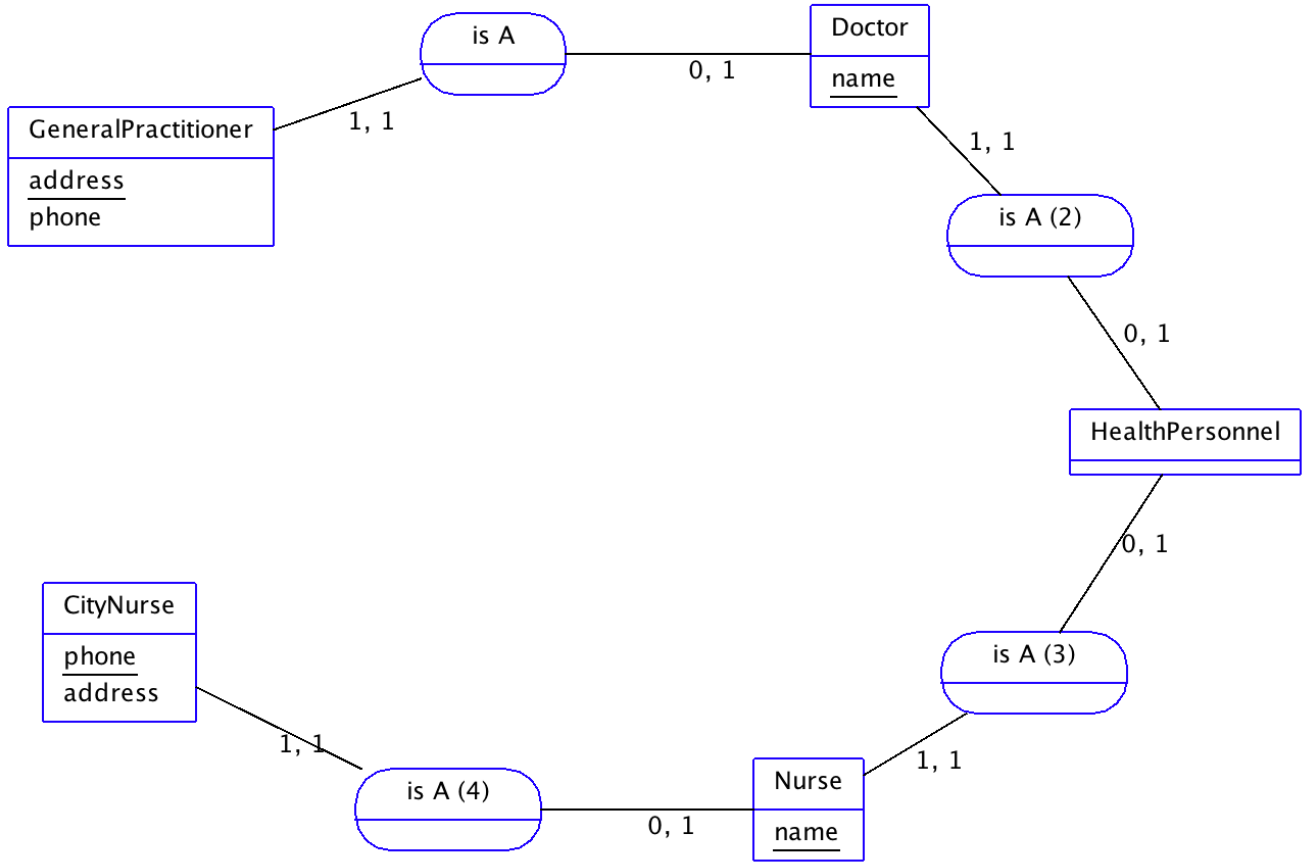
The software purchased by D department is installed on a server S dedicated to this department




```
Create or replace trigger trig_cons_inclusion
before
        insert on INSTALL
    For each row
    declare
        SOFT number (7);
        SERV number (7);
    begin
    Select  BUY.numSof, USE.numServ into SOFT,
    SERV
    From        BUY, USE
    Where       BUY.numDep = USE.numDep and
                BUY.numSof = : new .numSof and
                USE.numServ = : new.numServ;
    exception
        When NO_DATA_FOUND Then
        raise_application_error (-20.100, 'The
    software must be installed on a server in
    the purchasing department ');
     end ;
/
```

Inheritance

- One type of entity A is a specialization of another type of entity B if
  - Each entity A is an entity B
  - One entity (at most) of B is associated with a unit of A

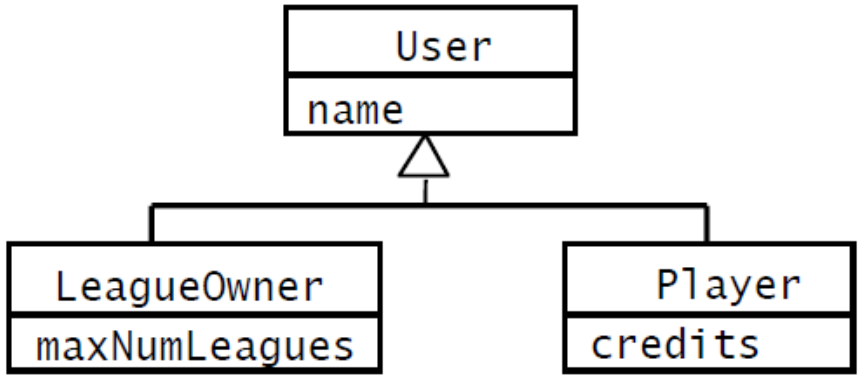


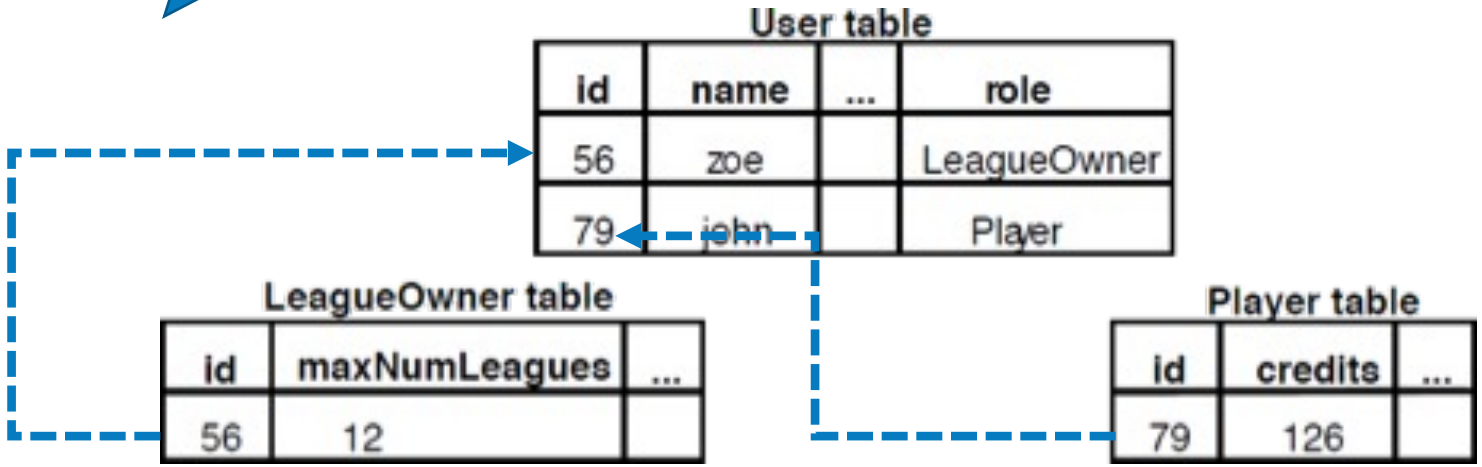
Inheritance

- Relational databases do not support inheritance
- Two possibilities to map an inheritance association to a database schema
- With a separate table (”vertical mapping”)
  - The attributes of the superclass and the subclasses are mapped to different tables
- By duplicating columns (”horizontal mapping”)
  - There is no table for the superclass
  - Each subclass is mapped to a table containing the attributes of the subclass and the attributes of the superclass

Inheritance with a separate table

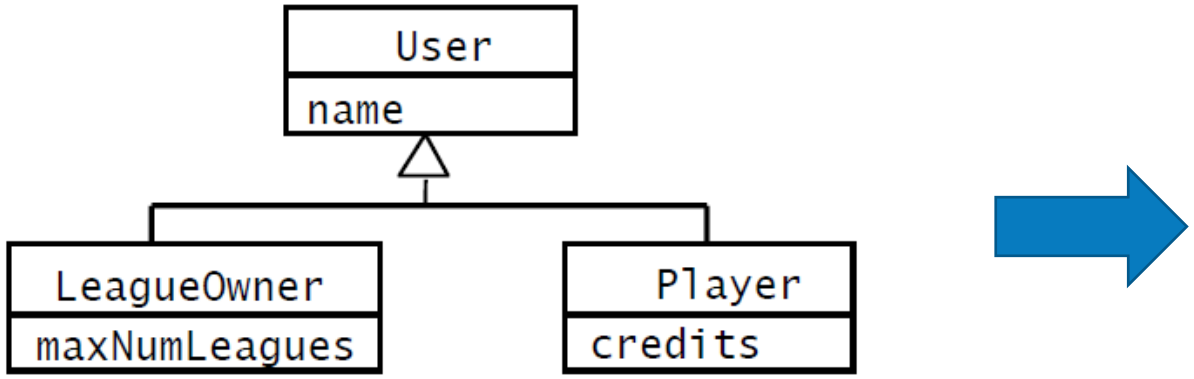
Vertical mapping





Inheritance by duplicating columns

Horizontal mapping



LeagueOwner table

id	name	maxNumLeagues	...
56	zoe	12	

Player table

id	name	credits	...
79	john	126	

Comparison: Separate Tables vs Duplicated Columns

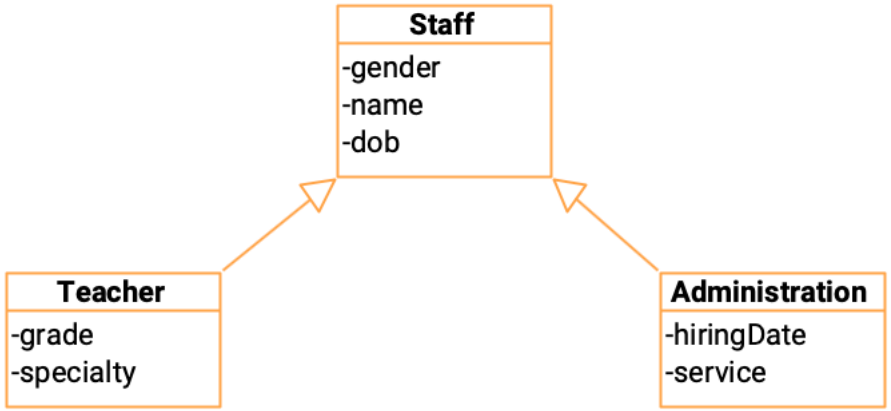
- The tradeoff is between modifiability and response time
  - How likely is a change to occur in the superclass?
  - What are the performance requirements for queries?
- Separate table mapping (vertical mapping)
  - ☺ We can add attributes to the superclass easily by adding a column to the superclass table only
  - ☹ Searching for the attributes of an object requires a join operation
- Duplicated columns (horizontal mapping)
  - ☹ Modifying the database schema is more complex and error-prone
  - ☺ Individual objects are not fragmented across a number of tables, resulting in faster queries

Inheritance using Subtypes in SQL3

- Inheritance is the mechanism that connects subtypes in a hierarchy to their supertypes.
- Subtypes automatically inherit the attributes and methods of their parent type.
- Any attributes or methods updated in a supertype are updated in subtypes as well.
- A subtype can be derived from a supertype either directly or indirectly through intervening levels of other subtypes.
- A supertype can have multiple sibling subtypes, but a subtype can have at most one direct parent supertype (single inheritance).
- Subtypes are created using the keyword UNDER

Inheritance using Subtypes in SQL3

Example: University Staff



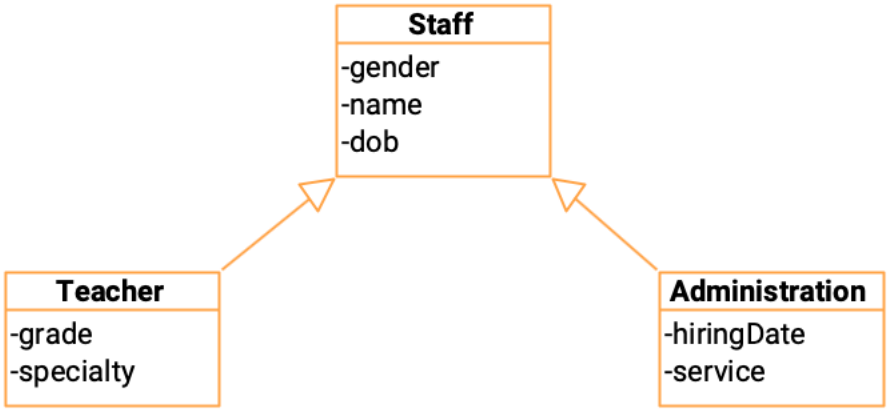

```
CREATE TABLE Staff (
        number NUMBER(7),
        name VARCHAR(20),
        dob DATE,
        gender CHAR(1),
        constraint PK_STAFF Primary Key (number)
);
CREATE TABLE Teacher UNDER Staff (
        grade NUMBER(2),
        specialty VARCHAR(20),
);
CREATE TABLE Administration UNDER Staff (
        hiringDate DATE,
        service VARCHAR(20)
);
```

Inheritance using Object types in SQL3

- Object types are user-defined types that make it possible to model real-world entities, such as customers and purchase orders, as objects in the database.
- New object types can be created from any built-in database types and any previously created object types, object references, and collection types.
- Object types and related object-oriented features, such as varrays and nested tables, provide higher-level ways to organize and access data in the database.
- In Oracle, you create SQL object types with the CREATE TYPE statement.

Inheritance using Object types in SQL3

Example: University Staff

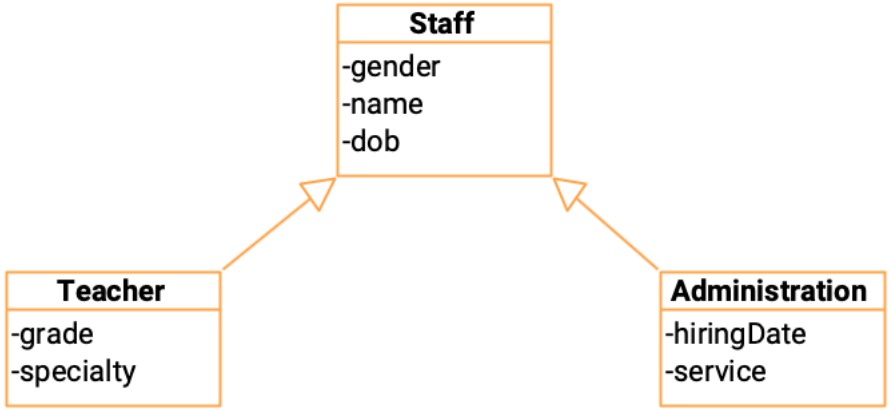



```
CREATE TYPE STAFF_TYPE AS OBJECT (
        number NUMBER(7),
        name VARCHAR(20),
        dob DATE,
        gender CHAR(1)
)
NOT FINAL     /*can include sub classes */
/
CREATE TYPE TEACHER_TYPE UNDER STAFF_TYPE (
        grade NUMBER(2),
        specialty VARCHAR(20)
)
FINAL
/
```

Inheritance using Object types in SQL3

Example: University Staff

- Creating object tables depending on the types previously defined
- No guidelines specifying the inheritance; it is induced by the existing type hierarchy
- **IMPORTANT**: note that the constraints are defined only in the "Staff" table




```
CREATE TABLE STAFF OF STAFF_TYPE(
        constraint PK_STAFF Primary Key (number)
);
CREATE TABLE TEACHER OF TEACHER_TYPE;
```

Inheritance using Object types in SQL3

Example: University Staff

- Inserting data in to “Staff” Table:

```
INSERT INTO STAFF VALUES (1, ‘Bill’,’17-09-2004’,’M’);
INSERT INTO STAFF VALUES (1, ‘Bill’,’17-09-2004’,’M’);

*
ERROR in line 1 :
ORA 00001: UNIQUE constraint violated (FB.PK_STAFF)


SELECT * FROM STAFF;

NUMBER        NAME          DOB              GENDER
--------      --------      ----------       ----------
       1      Bill          17-09-2004       M
```

Inheritance using Object types in SQL3

Example: University Staff

- Inserting data in to “TEACHER” Table:

```
INSERT INTO TEACHER VALUES (1, ‘Bill’,’17-09-2004’,’M’,2,’Info’);
INSERT INTO TEACHER VALUES (1, ‘Bill’,’17-09-2004’,’M’,2,’Info’);
INSERT INTO TEACHER VALUES (3, ‘Mary’,’17-10-2004’,’F’,1,’Maths’);

*
3 lines created!!


SELECT * FROM TEACHER;

NUMBER       NAME          DOB             GENDER       GRADE       SPECIALTY
--------     --------      ----------      ----------   ---------   ----------
       1     Bill          17-09-2004      M            2           Info
       1     Bill          17-09-2004      M            2           Info
       3     Mary          17-10-2004      F            1           Maths
```