**Database Systems: CS-329**

# **Week 07 :** Mapping the ER Model to Relational DBs

Top of Form

Bottom of Form

## Database Design

Goal of design is to generate a formal specification of the database schema

**Methodology**:

1. **Use E-R model to get a high-level graphical view** of essential components of enterprise and how they are related
2. **Then convert E-R diagram to SQL** Data Definition Language (DDL), or whatever database model you are using

E-R Model is not SQL based.

**The E-R Model:**The database represented is viewed as a graphical drawing of

* Entities and attributes
* Relationships among those entities
* --not tables!

**Relational Model**: The database is viewed as a

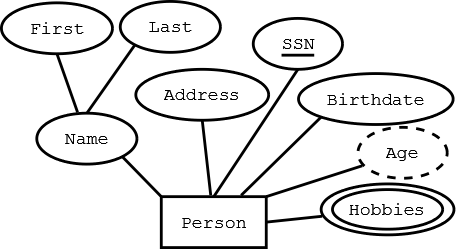
* Tables
* and their attributes (keys)
* --we could include constraints but will not at this stage.

Note on attributes:

* not all attributes in the relational model are attributes on an entity; we may have attributes that are establishing relationships.

## Representation of Entity Type in Relational Model

**Mapping #1 (ENTITIES): Each entity type always corresponds to a relation**

****

**---> Person(....)**

**Mapping #2 (ATTRIBUTES): The attributes of a relation contains at least the simple attributes of an entity type**

* Attributes are single valued
* There may be additional attributes (foreign keys) in the table, not found in the ER diagram

**Persons(SSN, FirstName, LastName, Address, Birthdate)**

**Problem**: Recall that the entity type can have multi-valued attributes.

**Possible solution**: Use several rows to represent a single entity

* (111111, John, 123 Main St, stamps)
* (111111, John, 123 Main St, coins)

Problems with this solution:

* Redundancy of the other attributes (never good)
* Key of entity type no longer can be key of relation

so, the resulting relation must be further transformed--> ***Normalization*** is this decomposition transformation process we will study to help deal with this and would result in:

**Mapping #2-m (MULTIVALUED ATTRIBUTES): The multivalued attributes of a relation and the entity key become their own relation.**

**Persons(SSN, FirstName, LastName, Address, Birthdate)  
Hobbies(SSN, Hobby)**

### Derived attributes

Coded separately in SQL as a **view**. They are not an attribute in a basic relation table.

## Relationship mapping

**Relationship**: connects two or more entities into an association/relationship

* John majors in Computer Science

**Relationship Type**: set of similar relationships

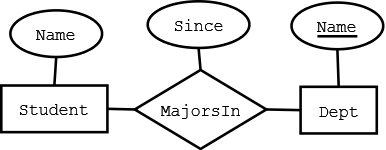
* Student (entity type) related to Department (entity type) by MajorsIn (relationship type).

Distinction -

* relation (relational model) - set of tuples
* relationship (E-R Model) – describes relationship between entities of an enterprise

Entity types and most relationship types in the E-R model are mapped to relations (relational model)

* **Mapping #3 (1-MANY RELATIONSHIP): 1-1 and 1-many relationships between separate entitites need not be mapped to a relation; the primary key attributes of the "1" relation become foreign key attributes of the "many" relation**

****

If no "Since" attribute, the relations could be (with some appropriate attribute renaming and additions)

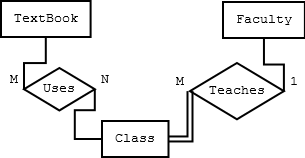
**Students(StudId, Name, Dept)  
Departments(Dept, Chair)**

Relationship Types may also have attributes in the E-R model.

* **Mapping #4 (RELATIONSHIP ATTRIBUTES): Any attributes of the 1-1 or 1-many relationship may be attached to the "many" relation.**

**Students(StudId, Name, Dept, Since)  
Departments(Dept, Chair)**

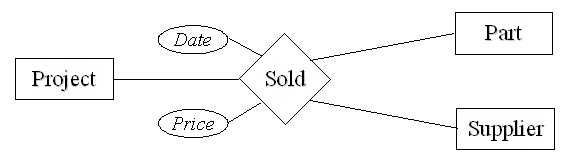
* **Mapping #5 (MANY-MANY RELATIONSHIP): many-many relationships are always mapped to a separate relation.**



**Textbooks(ISBN, Title, Author, Copyright, Edition, Price)  
Class(ClassNo, Name, Room, Days, Time)  
TextUses(ISBN, ClassNo)**

* **Mapping #6: The attributes of many-many relationships become part of the relationship type relation, as well as the primary key attributes of the related entity types**

**TextUses(ISBN, ClassNo, Optional)**



**Projects(ProjId, Name, TotalCost, StartDate)  
Parts(UPC, PartName, Weight, WSPrice)  
Suppliers(SupId, Name, Address)  
Sold(ProjId, UPC, SupId, Date, Price)**

Relationships tend to be **verbs;**attributes of relationships are nouns or adverbs

## Roles

**Problem**: recursive relationships can relate elements of same entity type

e.g., the ***ReportsTo*** relationship type relates two elements of the ***Employee*** entity type:

* Bob reports to Mary since 2000

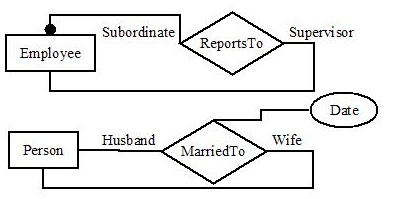
We do not always have distinct names for the roles

It is not clear who reports to whom

**Solution**: the role name of relationship type need not be same as name of entity type from which participants are drawn

* **ReportsTo** has roles ***Subordinate*** and ***Supervisor*** and attribute ***Since***
* Values of ***Subordinate*** and ***Supervisor*** both drawn from entity type **Employee**

**Mapping #7: If the cardinality is 1-many or 1-1 of a recursive relationship, then a second attribute of the same domain as the key may be added to the entity relation to establish the relationship. Attributes of the relationship can also be added to the entity relation, but may be a good reason to create a separate relation with the attributes and keys of the entities.**



**Employees(EmpID, Name, Address, Salary, *SupervisorID*)**

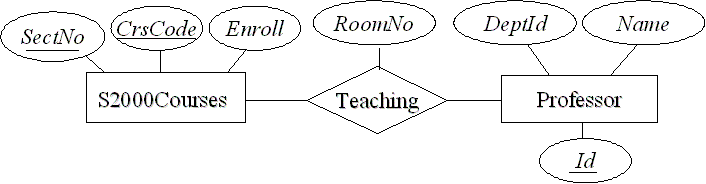
**Persons(PID, Name, Address, *SpouseID*, Mdate)**

**Mapping #8: for many-many recursive relationships, you create a relation including the attributes of the relation but with the primary keys of the entity included twice, one for each role.**

Assume multiple marriages are now recorded, thus many-to-many

**MarriedTo(HusbandID, WifeID, MarDate, DivDate)**

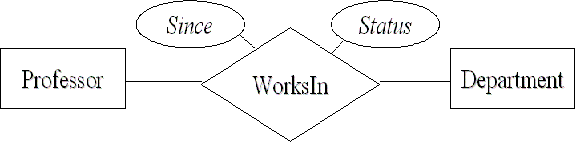
## Examples



**Courses (CrsCode, SectNo, Enroll)**  
**Professor (Id, DeptId, Name)**  
**Teaching (CrsCode, SecNo, Id, RoomNo)**

Assuming a 1-many cardinality for **WorksIn**

**Professor(Id, Name, DeptId,Since,Status)**



Assuming a many to many cardinality for WorksIn

#### Real SQL code

CREATE TABLE WorksIn (

Since DATE, -- attribute

Status CHAR (10), -- attribute

ProfId INTEGER, -- role (key of Professor)

DeptId CHAR (4), -- role (key of Department)

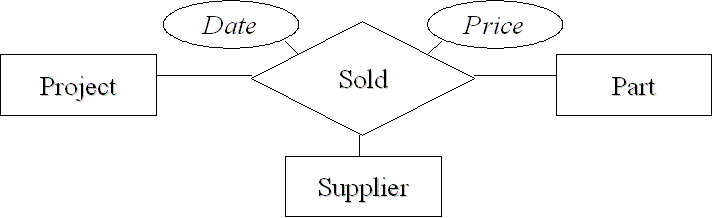
--constraints

PRIMARY KEY (ProfId), -- since a professor works in at most one department

FOREIGN KEY (ProfId) REFERENCES Professor (Id),

FOREIGN KEY (DeptId) REFERENCES Department

)



CREATE TABLE Sold (

Price INTEGER, -- attribute

Date DATE, -- attribute

ProjId INTEGER, -- role

SupplierId INTEGER, -- role

PartNumber INTEGER, -- role

--constraints

PRIMARY KEY (ProjId, SupplierId, PartNumber, Date),

FOREIGN KEY (ProjId) REFERENCES Project (Id),

FOREIGN KEY (SupplierId) REFERENCES Supplier (Id),

FOREIGN KEY (PartNumber) REFERENCES Part (Number)

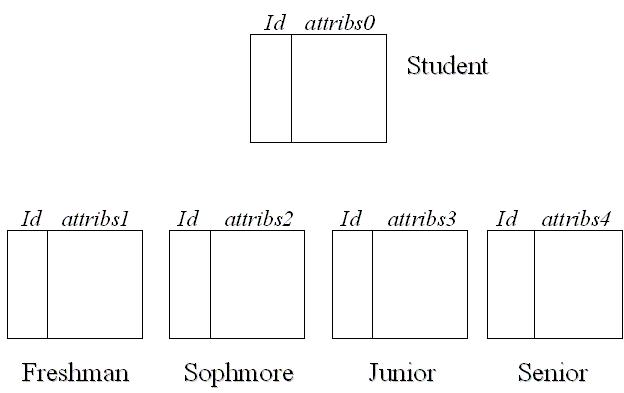
)

## Type Hierarchies and Relational Model

Supertypes and subtypes can be represented as separate relations.

We need a way of identifying each subtype entity with its (unique) related supertype entity

* *Choose a candidate key and make it an attribute of all entity types in the hierarchy  
  e.g.,****Id****below*

**

You have 5 base tables

* **Student (Id, Name, SSn, birthdate,.....)**
* **FirstYear(Id, Advisor,....)**
* **Sophomore(Id, ....)**
* **Junior(Id, Major,.....)**
* **Senior(Id, Major, Capstone,.....)**

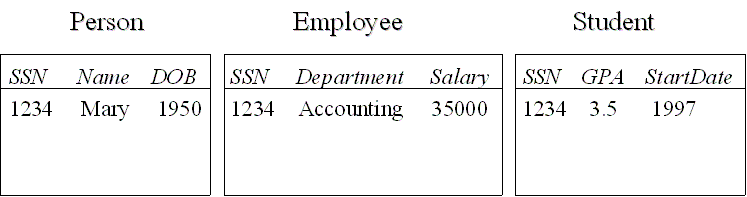
Mostly then, you will join Student with one of the other tables to get the normally expected table.  **Student |X| Sophomore**

This is where a **view** becomes extremely useful to provide a full SophomoreStudent relation

## Type Hierarchies and Relational Model

Redundancy is eliminated if  the IsA hierarchy is not disjoint

* For individuals who are both employees and students, Name and DOB are stored once



Again a join will normally be utilized in order to represent an expected table layout as a view.

## Participation Constraint

NOTE: These constraints must be implemented with more than relations and attributes structures.

If every entity participates in *at least* one relationship, a *participation constraint* holds:

* A participation constraint of entity type E having role r in relationship type R states that for *e* in E there is an *r* in R such that r (*r*) = *e*.
* e.g., every professor works in *at least* one department

### http://jcsites.juniata.edu/faculty/rhodes/dbms/images/ch5fig16.gif

### Representation

*Inclusion dependency*: Every professor works in *at least* one dep’t.

in relational model: (easy)

 Professor (*Id*) references WorksIn (*ProfId*)

in SQL:

 Special case: Every professor works in *exactly one* dep’t. (easy)

FOREIGN KEY *Id* REFERENCES WorksIn (*ProfId*)

General case (not so easy):

CREATE ASSERTION ProfsInDepts

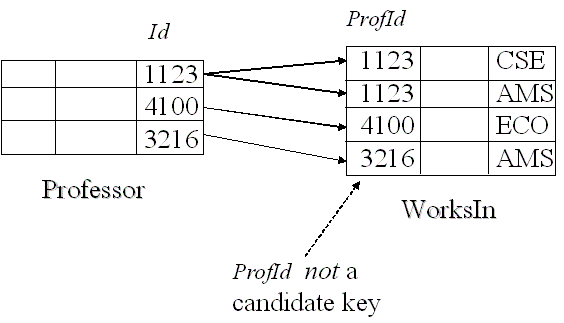
CHECK ( NOT EXISTS (

SELECT \* FROM Professor P

WHERE NOT EXISTS (

SELECT \* FROM WorksIn W

WHERE P.Id = W.ProfId ) ) )



## Participation and Key Constraint

If every entity participates in *exactly* one relationship, both a participation and a key constraint hold:

* e.g., every professor works in *exactly one* department

In SQL:

* If both participation and key constraints apply, use foreign key constraint in entity table (but beware: if candidate key in entity table is not primary, presence of nulls violates participation constraint).

CREATE TABLE Professor (

Id INTEGER,

……

PRIMARY KEY (Id), -- Id can’t be null

FOREIGN KEY (Id) REFERENCES WorksIn (ProfId)

--all professors participate

)

### Alternate solution

if both key and participation constraints apply: merge the tables representing the entity and relationship sets

* Since there is a 1-1 and onto relationship between the rows of the entity set and the relationship sets, might as well put all the attributes in one table

