Systems Analysis and Design II (IS352)

Dr. Ahmed Awad

Spring 2015

Outline

- Model Driven Architecture (MDA)
- Object Relational Mapping (ORM)
- Workflow systems Architectures
- Business Process Execution Language for Web Services (PBEL4WS)
- Process Instantiation
- Process Mining

Transiting from Analysis to Design/Configuration/Enactment

MDA Overview

What is the MDA?

- An approach to IT system specification that separates the specification of system functionality from the specification of the implementation of *that* functionality on a particular technology [platform](#)
- “Design once, build it on any platform”

Basic concepts of MDA

- A *model* is a formal specification of the function, structure and/or behaviour of a system
  - Examples:
    - A BPMN process is a model
    - An UML-based specification is a model

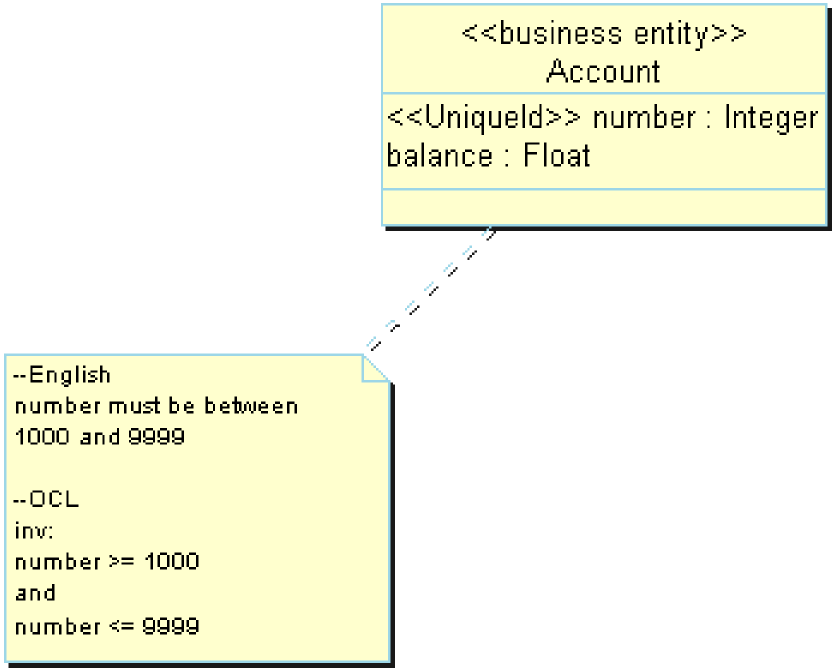
- Models of different systems are structured explicitly into:
  - Platform Independent Models (PIM)
  - Platform Specific Models (PSM)

Platform Independent Model (PIM)

- A “formal” specification of the structure and function of a system that abstracts away technical detail

- Expressed using UML

PIM: an example



<<business entity>>
Account

<<UniqueId>> number : Integer
balance : Float

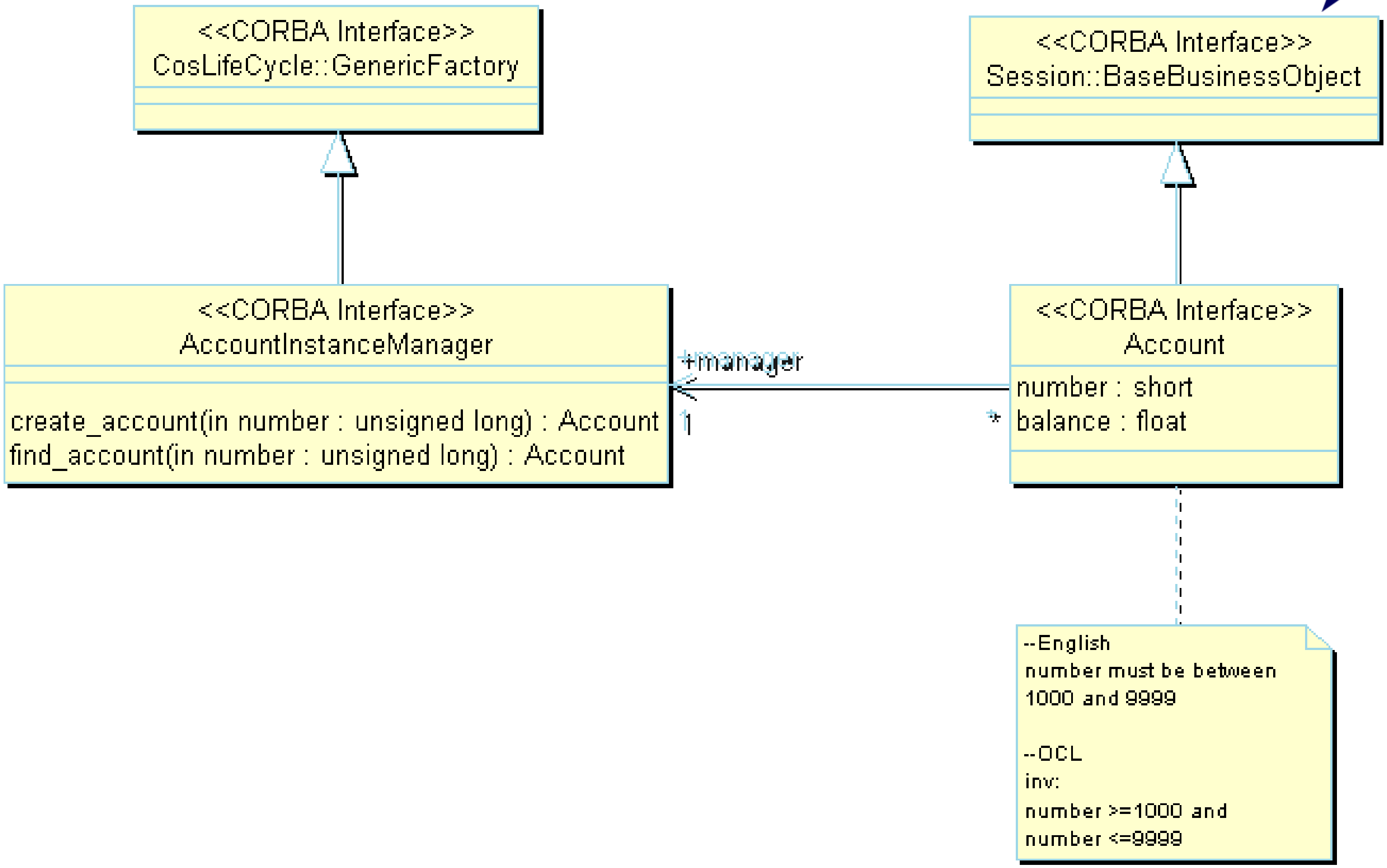
```
--English
number must be between
1000 and 9999

--OCL
inv:
number >= 1000
and
number <= 9999
```

Platform Specific Model (PSM)

- Specifies how the functionality specified in a PIM is realized on a particular platform
- Expressed using UML extended with platform specific UML profiles

PSM: an example



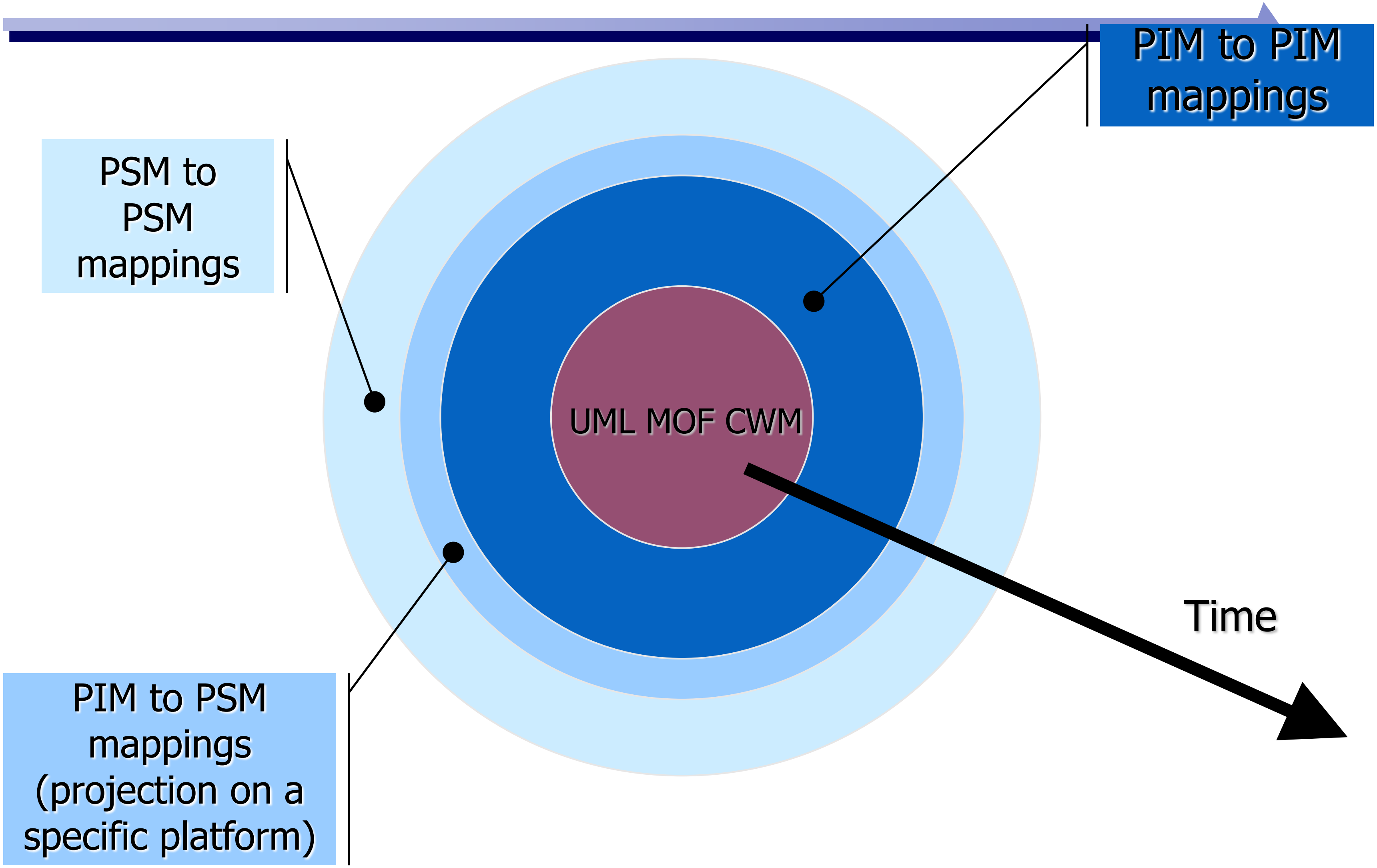
<<CORBA Interface>>
CosLifeCycle::GenericFactory
<<CORBA Interface>>
Session::BaseBusinessObject
<<CORBA Interface>>
AccountInstanceManager
create_account(in number : unsigned long) : Account
find_account(in number : unsigned long) : Account
+manager
1
*
<<CORBA Interface>>
Account
number : short
balance : float

--English
number must be between
1000 and 9999

--OCL
inv:
number >=1000 and
number <=9999

Developing in MDA

System Development Lifecycle and the MDA approach



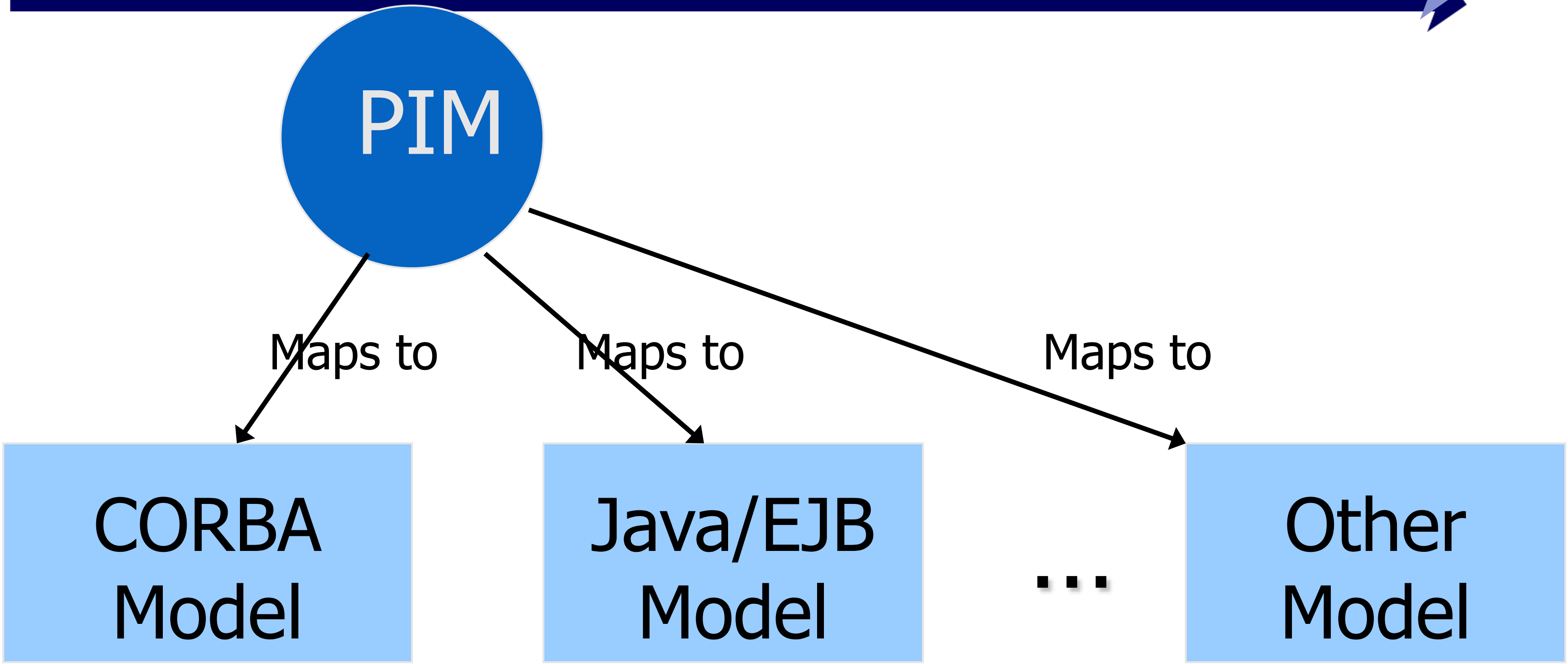
Developing in MDA – Step 1: the PIM

- All MDA development projects start with the creation of a PIM
- PIM at this level represents business functionality and behaviour, undistorted by technology details

Developing in MDA – Step 2: the PSM (1/2)

- Once the first iteration is complete, the PIM is input to the mapping step which will produce a PSM
- Code is partially automatic and partially hand-written
- PIM can be mapped either to a single platform or to multiple platforms

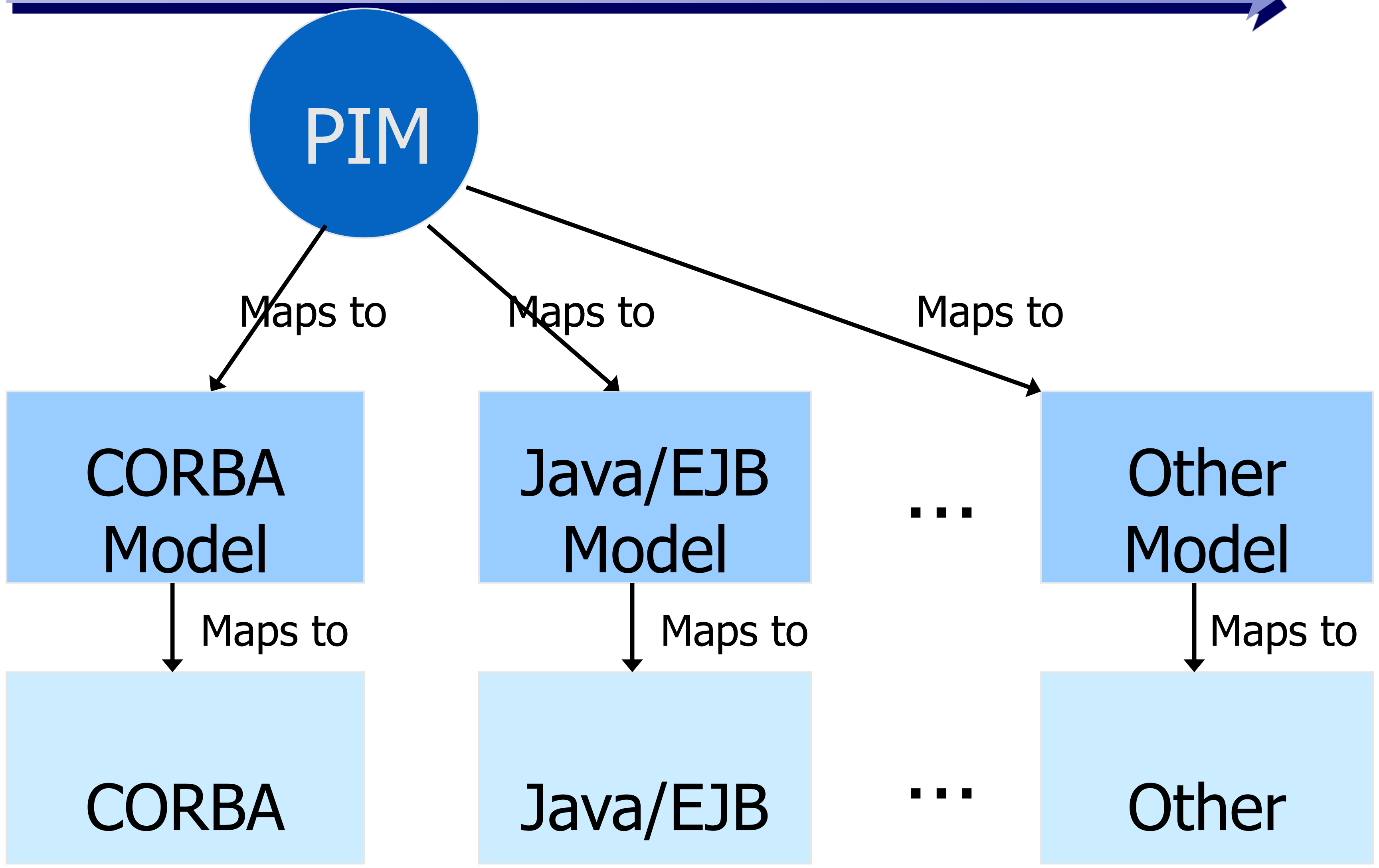
Developing in MDA – Step 2: the PSM (2/2)



Developing in MDA – Step 3: Generating Application (1/2)

- An MDA tool generates all or most of the implementation code for the deployment technology selected by the developer
- Re-integration on new platforms can be done by reverse engineering the existing application into a model and redeploy

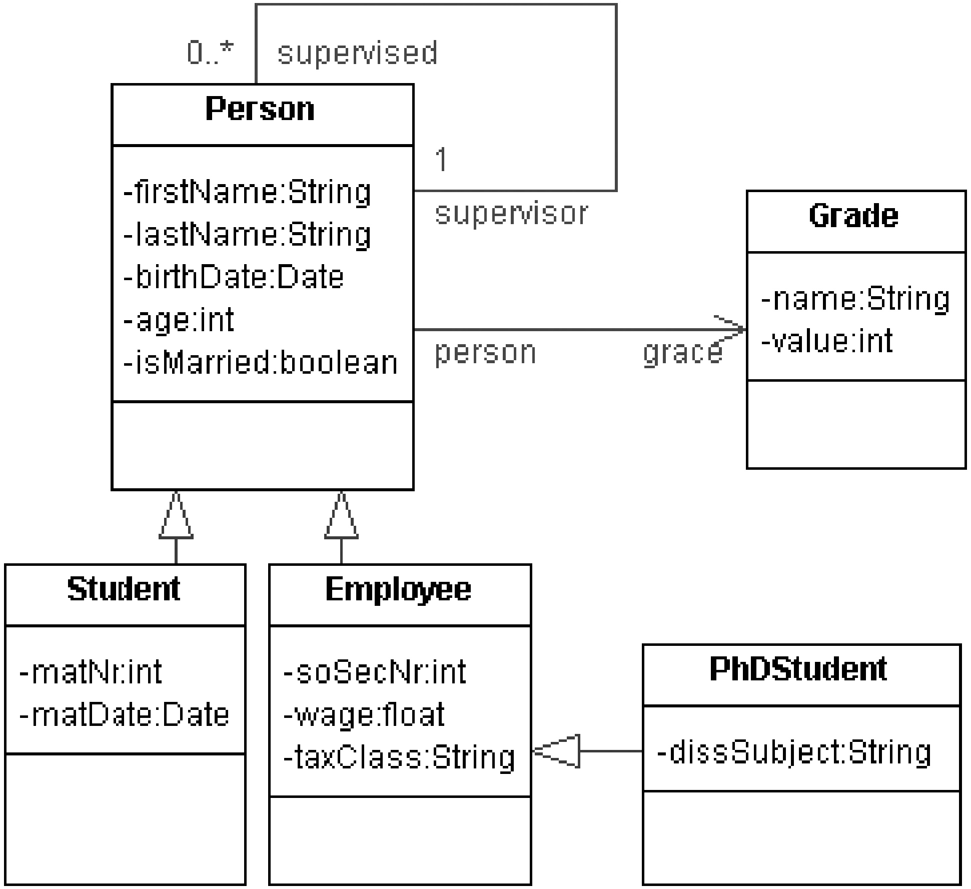
Developing in MDA – Step 3: Generating Application (2/2)



Object-Relational Mapping

What is ORM?

Object-relational mapping, in the purest sense, is a programming technique that supports the conversion of incompatible types in object-oriented programming languages, specifically between a data store and programming objects.


0..*
supervised
Person
-firstName:String
-lastName:String
-birthDate:Date
-age:int
-isMarried:boolean
1
supervisor
Grade
-name:String
-value:int
person
grace
Student
-matNr:int
-matDate:Date
Employee
-soSecNr:int
-wage:float
-taxClass:String
PhDStudent
-dissSubject:String

Why ORM?

- Productivity: The data access code is usually a significant portion of a typical application, and the time needed to write that code can be a significant portion of the overall development schedule. When using an ORM tool, the amount of code is unlikely to be reduced—in fact, it might even go up—but the ORM tool generates 100% of the data access code automatically based on the data model you define, in mere moments.

Why ORM?

- Application design: A good ORM tool designed by very experienced software architects will implement effective design patterns that almost force you to use good programming practices in an application. This can help support a clean separation of concerns and independent development that allows parallel, simultaneous development of application layers.

Why ORM?

- Code Reuse: If you create a class library to generate a separate DLL for the ORM-generated data access code, you can easily reuse the data objects in a variety of applications. This way, each of the applications that use the class library need have no data access code at all.

Why ORM?

- Application Maintainability: All of the code generated by the ORM is presumably well-tested, so you usually don’t need to worry about testing it extensively.

ORM in one sentence...

Never write SQL statements in your source code!

Relational Databases

- Collection of tables
  - Comprised of fields that define entities
  - Primary key has unique values in each row of a table
  - Foreign key is primary key of another table
- Tables related to each other
  - Primary key field of a table is a field of another table and called a foreign key
  - Relationship established by a foreign key of one table connecting to the primary key of another table

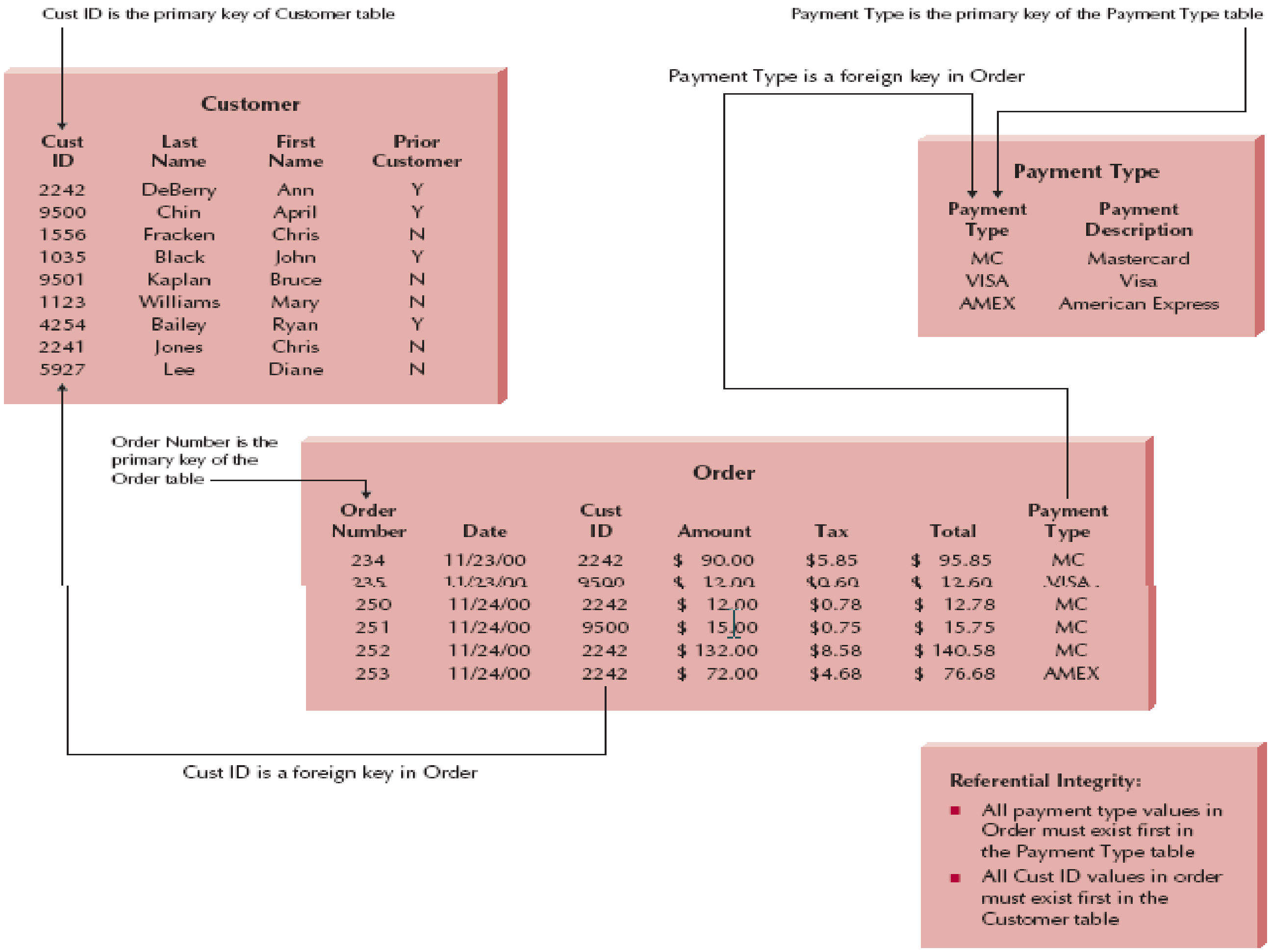
Referential Integrity

- the idea of ensuring that values linking the tables together through the primary and foreign keys are valid and correctly synchronized.

Referential Integrity Example

- Cust. ID is a primary key for the customer table
- Cust. ID is a foreign key for the order table
- A violation of referential integrity would happen if an order was entered in the order table for a Cust. ID that had not been entered into the customer table first
- An RDBMS prevents such a record from being entered

Example of Referential Integrity



Cust ID is the primary key of Customer table

Payment Type is the primary key of the Payment Type table

Payment Type is a foreign key in Order

Customer

Cust ID	Last Name	First Name	Prior Customer
2242	DeBerry	Ann	Y
9500	Chin	April	Y
1556	Fracken	Chris	N
1035	Black	John	Y
9501	Kaplan	Bruce	N
1123	Williams	Mary	N
4254	Bailey	Ryan	Y
2241	Jones	Chris	N
5927	Lee	Diane	N

Payment Type

Payment Type	Payment Description
MC	Mastercard
VISA	Visa
AMEX	American Express

Order Number is the primary key of the Order table

Order

Order Number	Date	Cust ID	Amount	Tax	Total	Payment Type
234	11/23/00	2242	$ 90.00	$5.85	$ 95.85	MC
235	11/23/00	9500	$ 12.00	$0.60	$ 12.60	VISA
250	11/24/00	2242	$ 12.00	$0.78	$ 12.78	MC
251	11/24/00	9500	$ 15.00	$0.75	$ 15.75	MC
252	11/24/00	2242	$ 132.00	$8.58	$ 140.58	MC
253	11/24/00	2242	$ 72.00	$4.68	$ 76.68	AMEX

Cust ID is a foreign key in Order

Referential Integrity:

- All payment type values in Order must exist first in the Payment Type table
- All Cust ID values in order must exist first in the Customer table

Object-Relational Mapping (ORM)

- Mapping attributes
- Mapping classes
- Mapping relationships
  - Inheritance
  - Associations
- Mapping methods?

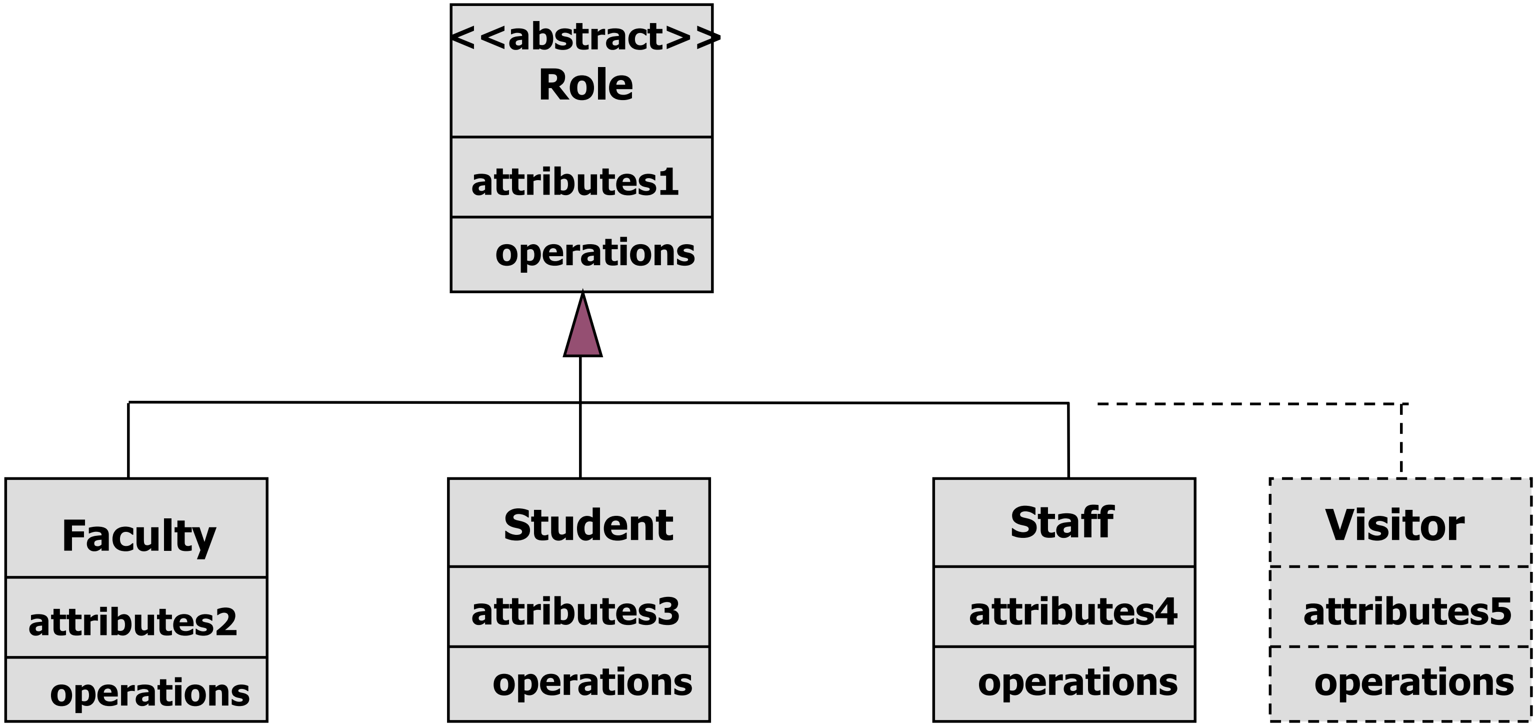
Mapping attributes

- The most straightforward mapping
- Simple types are common between object paradigm and relational models
- Strings, dates, integers are common
  - String in Java is varchar(length) in SQL Server

Mapping classes

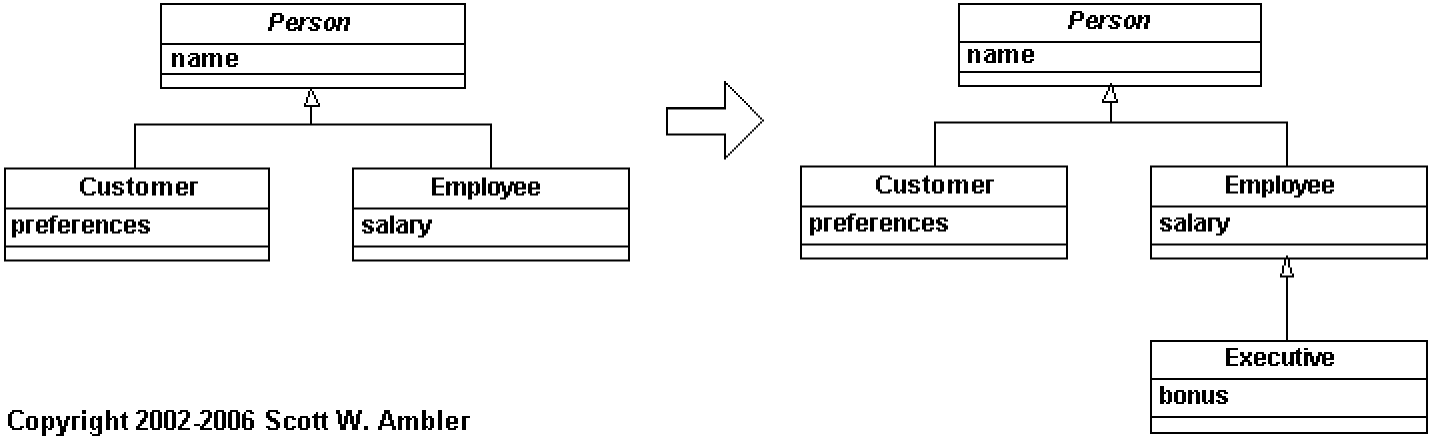
- ☑ In general, classes map to tables in one-to-one mapping
- ☑ In some cases, multiple classes can map to a single table, e.g., inheritance

Mapping Inheritance

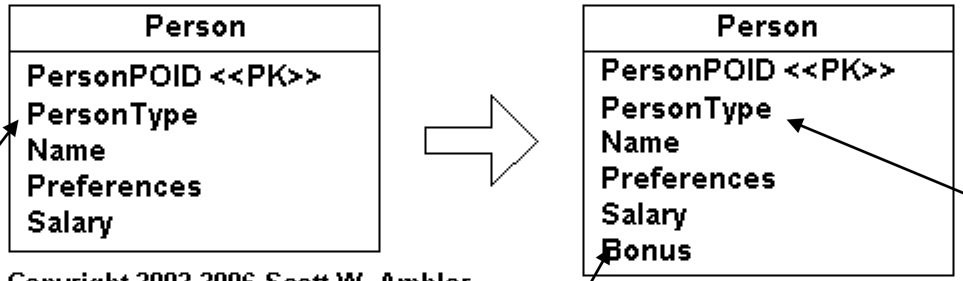


Mapping Inheritance

- ☑ Map the entire class hierarchy to a single table
- ☑ Map each concrete class to its own table
- ☑ Map each class to its own table
- ☑ Map the classes into a generic table structure


Person
name
Customer
preferences
Employee
salary
Person
name
Customer
preferences
Employee
salary
Executive
bonus

Map Hierarchy to a single table



Person

PersonPOID <<PK>>
PersonType
Name
Preferences
Salary

Person

PersonPOID <<PK>>
PersonType
Name
Preferences
Salary
Bonus

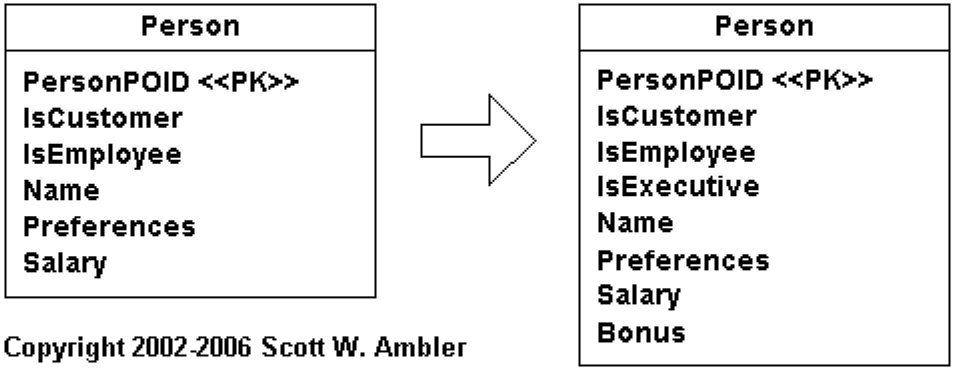
Copyright 2002-2006 Scott W. Ambler

Discriminator:
Customer, Employee

Extra column: for the executive class

Discriminator:
Customer, Employee,
Executive

What if Types overlap



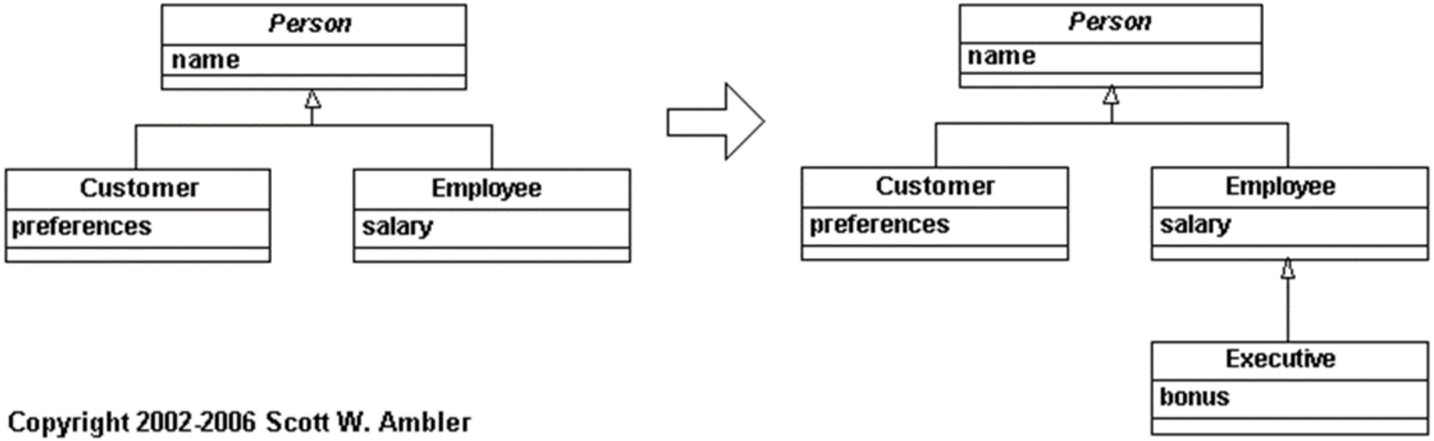
Person

PersonPOID <<PK>>
IsCustomer
IsEmployee
Name
Preferences
Salary

Person

PersonPOID <<PK>>
IsCustomer
IsEmployee
IsExecutive
Name
Preferences
Salary
Bonus

Map Each Concrete Class to its Own Table



Person
name

Customer
preferences

Employee
salary

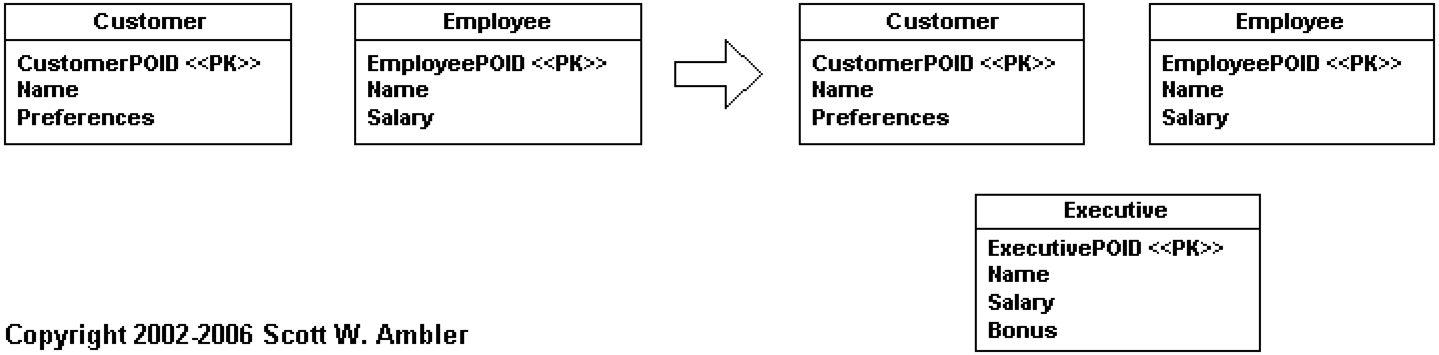
Person
name

Customer
preferences

Employee
salary

Executive
bonus

Copyright 2002-2006 Scott W. Ambler



Customer
CustomerPOID <<PK>>
Name
Preferences

Employee
EmployeePOID <<PK>>
Name
Salary

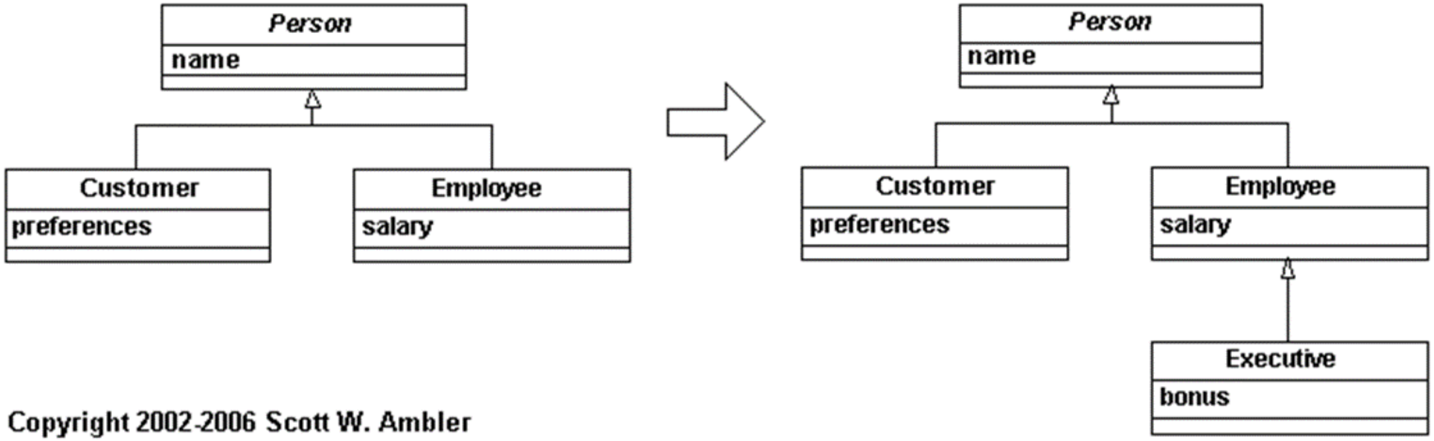
Customer
CustomerPOID <<PK>>
Name
Preferences

Employee
EmployeePOID <<PK>>
Name
Salary

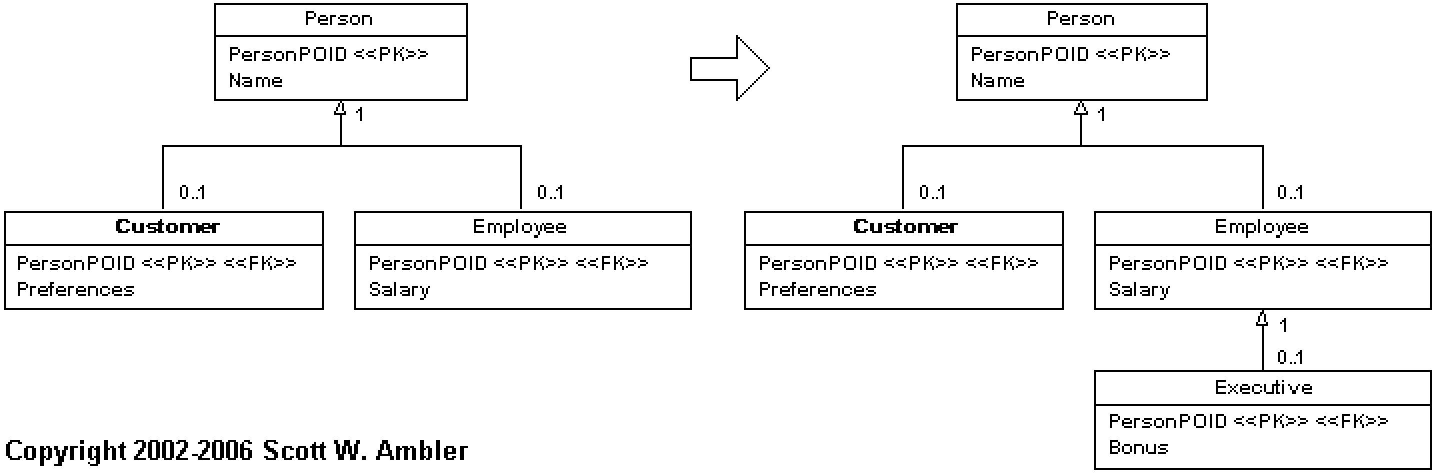
Executive
ExecutivePOID <<PK>>
Name
Salary
Bonus

Copyright 2002-2006 Scott W. Ambler

Map Each class to its Own Table



Copyright 2002-2006 Scott W. Ambler



Copyright 2002-2006 Scott W. Ambler

Comparison

Strategy	Advantages	Disadvantages	When to Use
One table per hierarchy	Simple approach. Easy to add new classes, you just need to add new columns for the additional data. Supports polymorphism by simply changing the type of the row. Data access is fast because the data is in one table. Ad-hoc reporting is very easy because all of the data is found in one table.	Coupling within the class hierarchy is increased because all classes are directly coupled to the same table. A change in one class can affect the table which can then affect the other classes in the hierarchy. Space potentially wasted in the database. Indicating the type becomes complex when significant overlap between types exists. Table can grow quickly for large hierarchies.	This is a good strategy for simple and/or shallow class hierarchies where there is little or no overlap between the types within the hierarchy.

Comparison

Strategy	Advantages	Disadvantages	When to Use
One table per concrete class	easy to do ad-hoc reporting as all the data you need about a single class is stored in only one table. Good performance to access a single object’ s data.	When you modify a class you need to modify its table and the table of any of its subclasses. For example if you were to add height and weight to the Person class you would need to add columns to the Customer, Employee, and Executive tables. Whenever an object changes its role, you need to copy the data into the appropriate table. It is difficult to support multiple roles and still maintain data integrity.	When changing types and/or overlap between types is rare.

Comparison

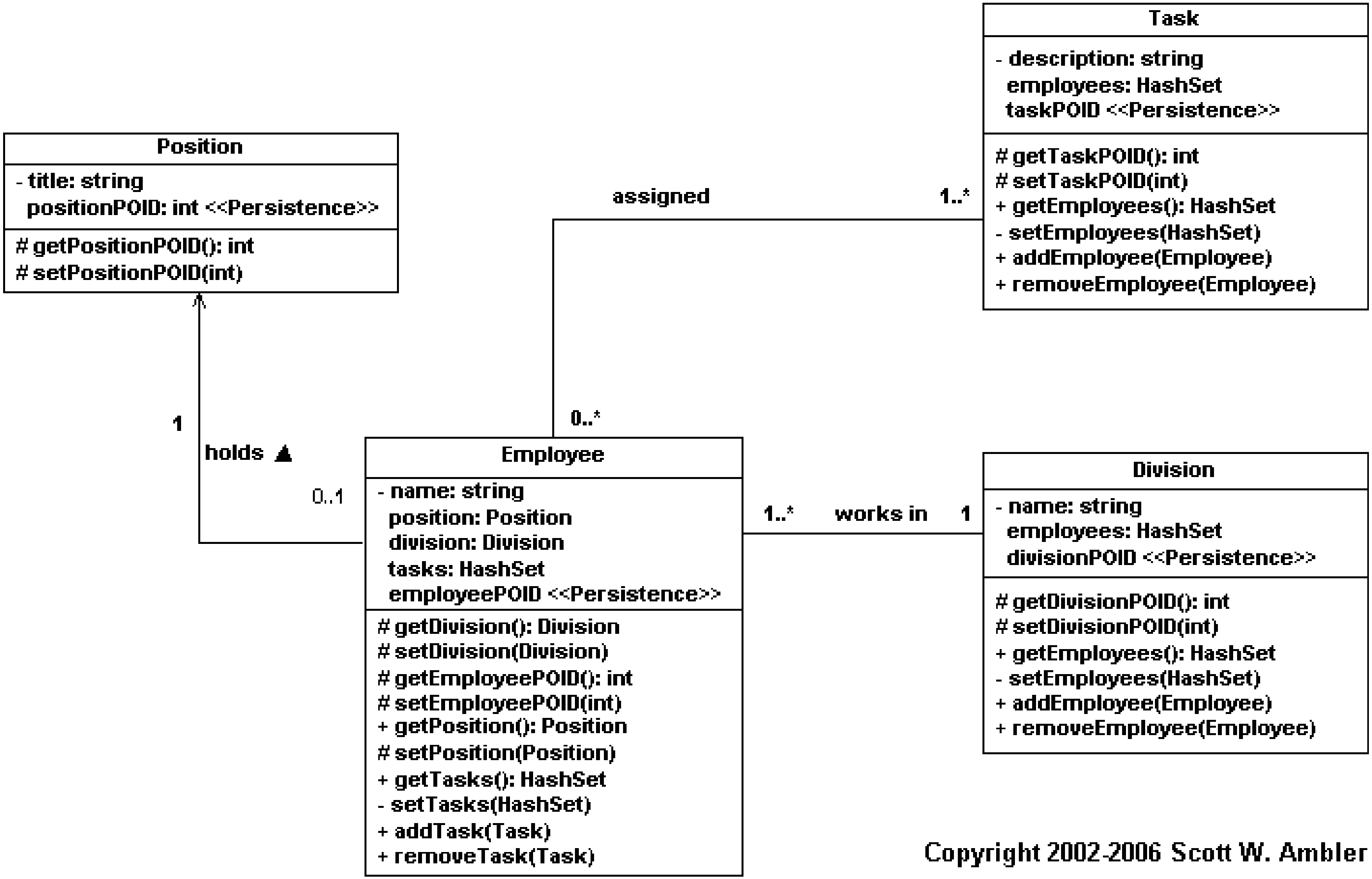
Strategy	Advantages	Disadvantages	When to Use
One table per class	Easy to understand because of the one-to-one mapping. Supports polymorphism very well as you merely have records in the appropriate tables for each type. Very easy to modify superclasses and add new subclasses as you merely need to modify/add one table. Data size grows in direct proportion to growth in the number of objects.	There are many tables in the database, one for every class (plus tables to maintain relationships). Potentially takes longer to read and write data using this technique. Ad-hoc reporting on your database is difficult, unless you add views to simulate the desired tables.	When there is significant overlap between types or when changing types is common.

Mapping Object Relationships

- We have:
  - Association
  - Aggregation
  - Composition
- Recall, Aggregation and composition are special types of association
- All three will be mapped to referential integrity constraints

Mapping Object Relationships

- Cardinality
  - Mapping one-to-one relationships
  - Mapping one-to-many relationships
  - Mapping many-to-many relationships
- Direction
  - Unidirectional
  - Bidirectional



Position

- title: string
positionPOID: int <<Persistence>>

getPositionPOID(): int
setPositionPOID(int)

1

holds ▲

0..1

Employee

- name: string
position: Position
division: Division
tasks: HashSet
employeePOID <<Persistence>>

getDivision(): Division
setDivision(Division)
getEmployeePOID(): int
setEmployeePOID(int)
+ getPosition(): Position
# setPosition(Position)
+ getTasks(): HashSet
- setTasks(HashSet)
+ addTask(Task)
+ removeTask(Task)

0..*

assigned

1..*

Task

- description: string
employees: HashSet
taskPOID <<Persistence>>

getTaskPOID(): int
setTaskPOID(int)
+ getEmployees(): HashSet
- setEmployees(HashSet)
+ addEmployee(Employee)
+ removeEmployee(Employee)

1..*

works in

1

Division

- name: string
employees: HashSet
divisionPOID <<Persistence>>

getDivisionPOID(): int
setDivisionPOID(int)
+ getEmployees(): HashSet
- setEmployees(HashSet)
+ addEmployee(Employee)
+ removeEmployee(Employee)

How Relational Database Relationships Are Implemented

Relationships in relational databases are maintained through the use of foreign keys. A foreign key is a data attribute(s) that appears in one table that may be part of or is coincidental with the key of another table. With a one-to-one relationship the foreign key needs to be implemented by one of the tables.

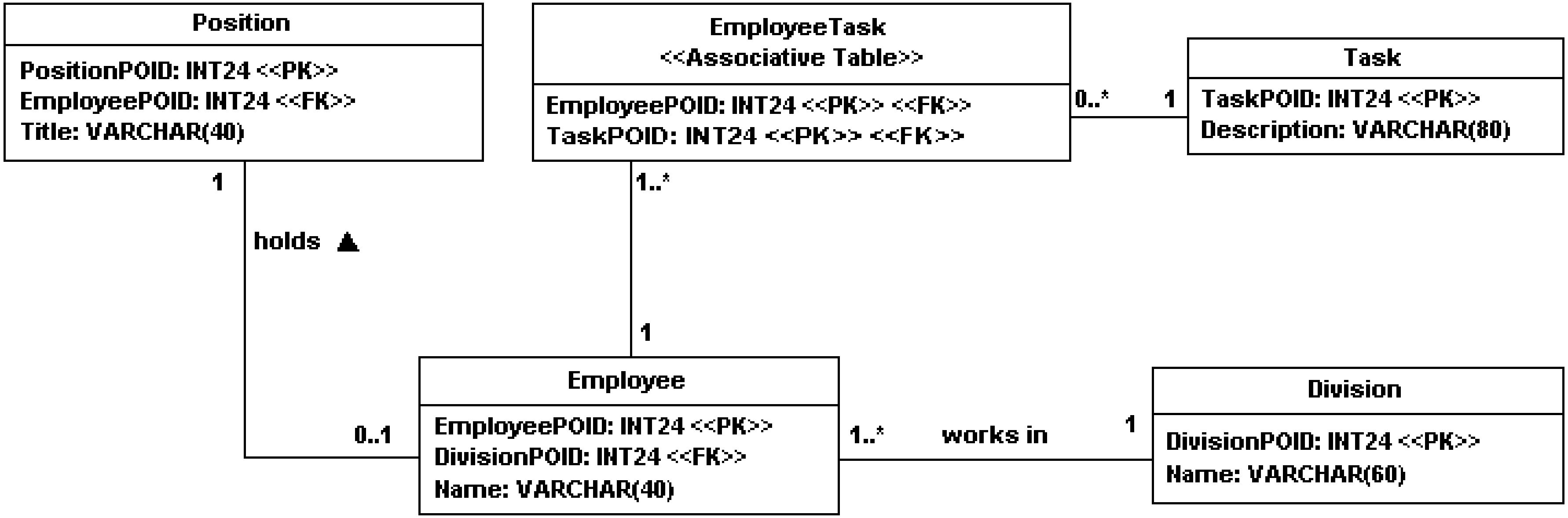
How Relational Database Relationships Are Implemented

To implement a **one-to-many** relationship you implement a foreign key from the “one table” to the “many table”. For example Employee includes a DivisionPOID column to implement the works in relationship to Division. You could also choose to overbuild your database schema and implement a one-to-many relationship via an associative table, effectively making it a many-to-many relationship.

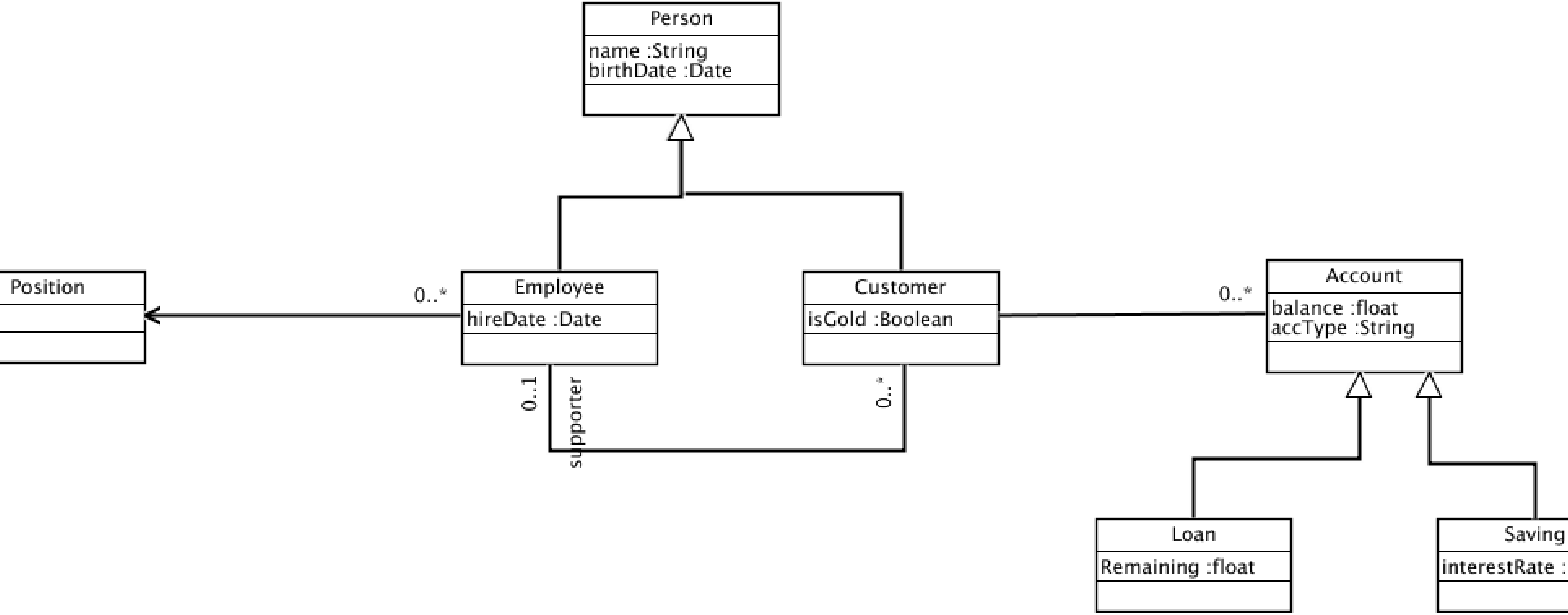
How Relational Database Relationships Are Implemented

To implement **many-to-many** associations in a relational database, is to implement what is called an associative table, an example of which is *EmployeeTask*, which includes the combination of the primary keys of the tables that it associates.

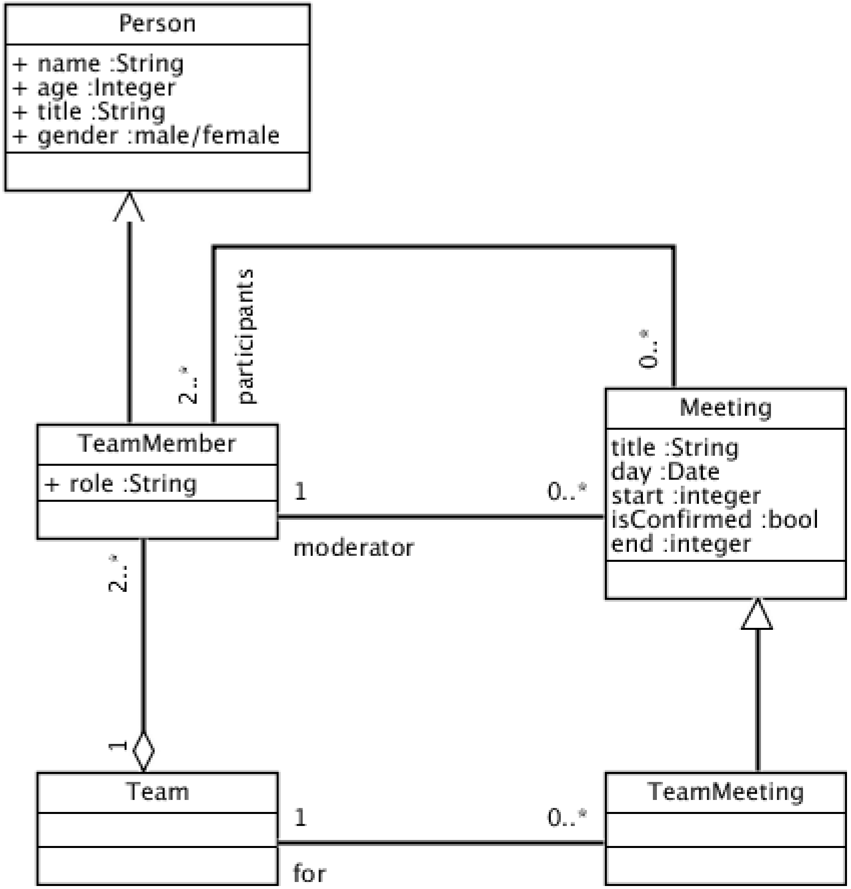
The basic "trick" is that the many-to-many relationship is converted into two one-to-many relationships, both of which involve the associative table.


Position
PositionPOID: INT24 <<PK>>
EmployeePOID: INT24 <<FK>>
Title: VARCHAR(40)
EmployeeTask
<<Associative Table>>
EmployeePOID: INT24 <<PK>> <<FK>>
TaskPOID: INT24 <<PK>> <<FK>>
0..*
1
Task
TaskPOID: INT24 <<PK>>
Description: VARCHAR(80)
1
1..*
holds ▲
1
0..1
Employee
EmployeePOID: INT24 <<PK>>
DivisionPOID: INT24 <<FK>>
Name: VARCHAR(40)
1..*
works in
1
Division
DivisionPOID: INT24 <<PK>>
Name: VARCHAR(60)

More Exercises (1)



More Exercises (2)


Person
+ name :String
+ age :Integer
+ title :String
+ gender :male/female
participants
2..*
0..*
TeamMember
+ role :String
1
0..*
moderator
Meeting
title :String
day :Date
start :integer
isConfirmed :bool
end :integer
2..*
1
Team
1
0..*
for
TeamMeeting