

Systems Design

Object Relational Mapping

Reference:

Scott Ambler article @

<http://www.agiledata.org/essays/mappingObjects.html>

This document may not be used or altered without the express permission of the author.

Dr. Glenn L. Ray

School of Information Sciences

University of Pittsburgh

gray@sis.pitt.edu 412-624-9470

Object Relational Mapping

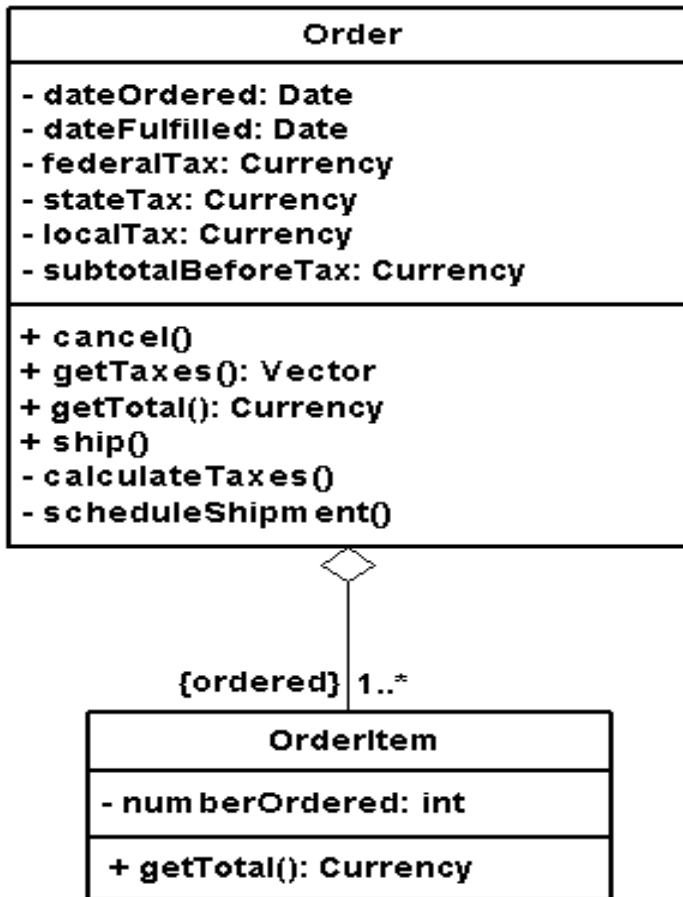
- O/R Impedance mismatch
 - Object modeling is different than RDB modeling!
 - How to map objects to RDBs?
 - Several strategies available
 - All have pros & cons
 - Very few people in IT aware of this issue!

Basic Concepts

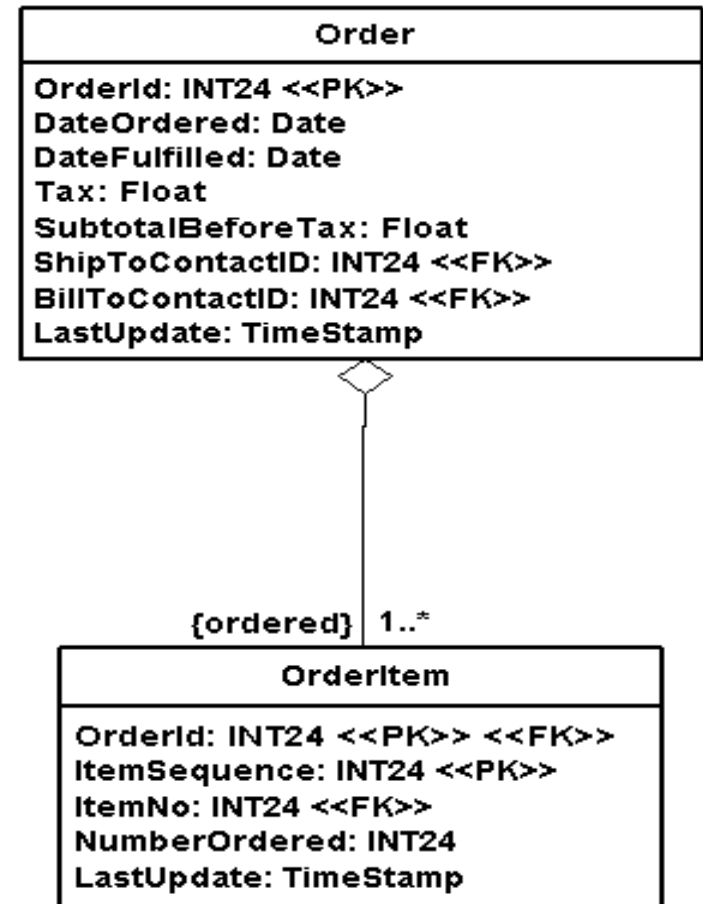
- Object attribute maps to 0..* columns in a DB
- Not all attributes persisted
 - Some are transient, e.g., results of calculations
 - Some refer to other objects
- Mapping: How objects & their relationships persisted in permanent storage
- Property: data attribute
 - physical -> firstName: String
 - virtual -> getTotal(): double

Class Maps To Table

<<Class Model>>



<<Physical Data Model>>



Copyright 2002-2006 Scott W. Ambler

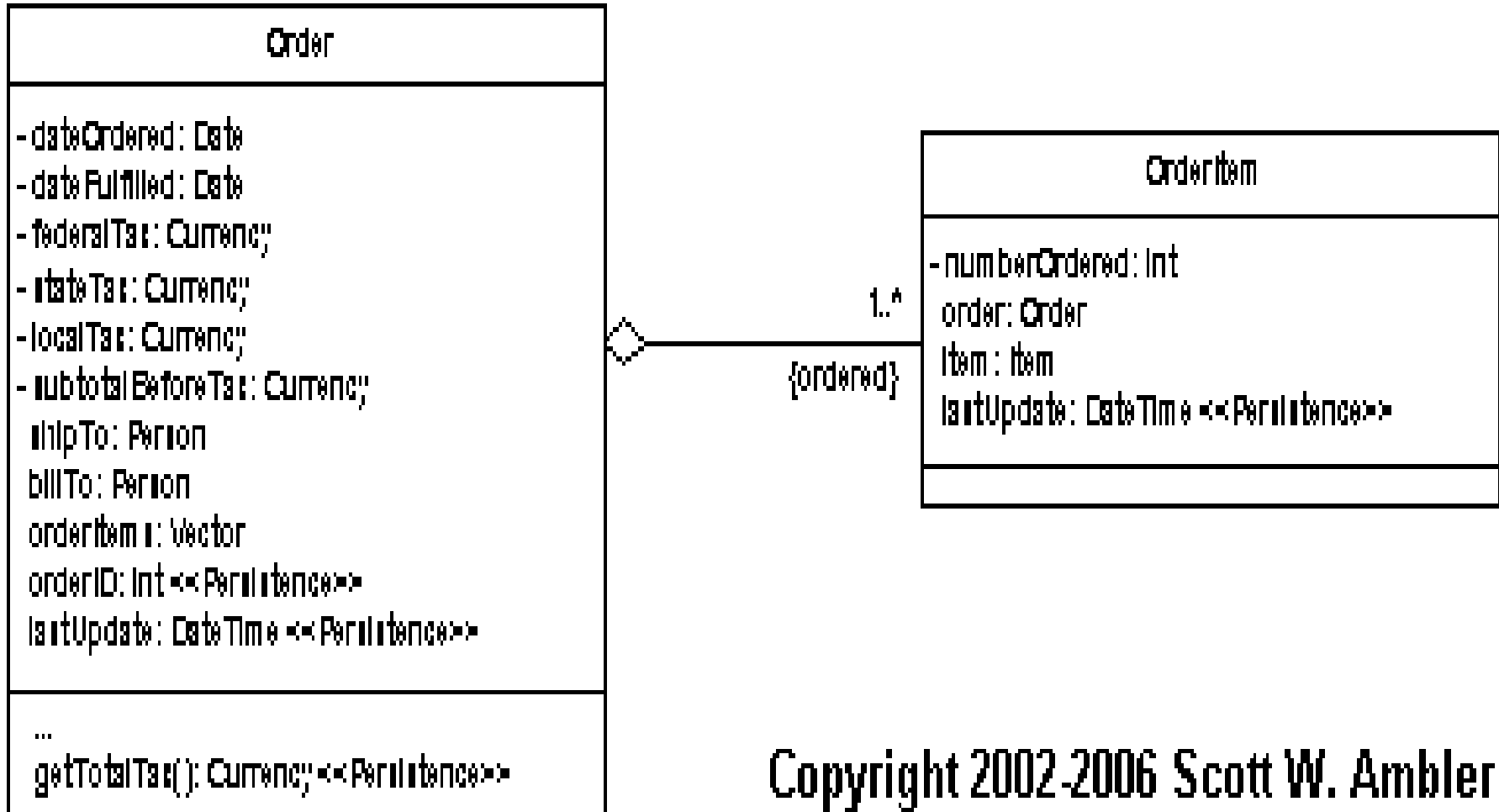
Class Maps To Table

- Use similar names when possible
- For new systems, object schema should drive DB schema
 - Usually, the reverse is true!
- Key schema differences
 - Tax attributes
 - DB schema has PKs & FKs while object schema has attribute references
 - Shadow information
 - Types may be different between schema

Shadow Information

- Any 'extra' data an object needs to persist itself
 - Identification -> PK
 - Concurrency control -> timestamp, version #
- Next slide shows shadow attributes
- Map class properties to DB
 - Can be attribute or method!

Object Relational Mapping



Copyright 2002-2006 Scott W. Ambler

Shadow Info

- `isPersistent` boolean flag
 - false if newly created, true if read from DB
- Shadow info commonly omitted from models because it can be inferred

Mapping Metadata

Fig 3. Mapping object properties to table columns
Note mapping of methods to columns

Property	Column
Order.orderID	Order.OrderID
Order.dateOrdered	Order.DateOrdered
Order.dateFulfilled	Order.DateFulfilled
Order.getTotalTax()	Order.Tax
Order.subtotalBeforeTax	Order.SubtotalBeforeTax
Order.shipTo.personID	Order.ShipToContactID
Order.billTo.personID	Order.BillToContactID
Order.lastUpdate	Order.LastUpdate
OrderItem.ordered	OrderItem.OrderID
Order.orderItems.position(orderItem)	OrderItem.ItemSequence
OrderItem.item.number	OrderItem.ItemNo
OrderItem.numberOrdered	OrderItem.NumberOrdered
OrderItem.lastUpdate	OrderItem.LastUpdate

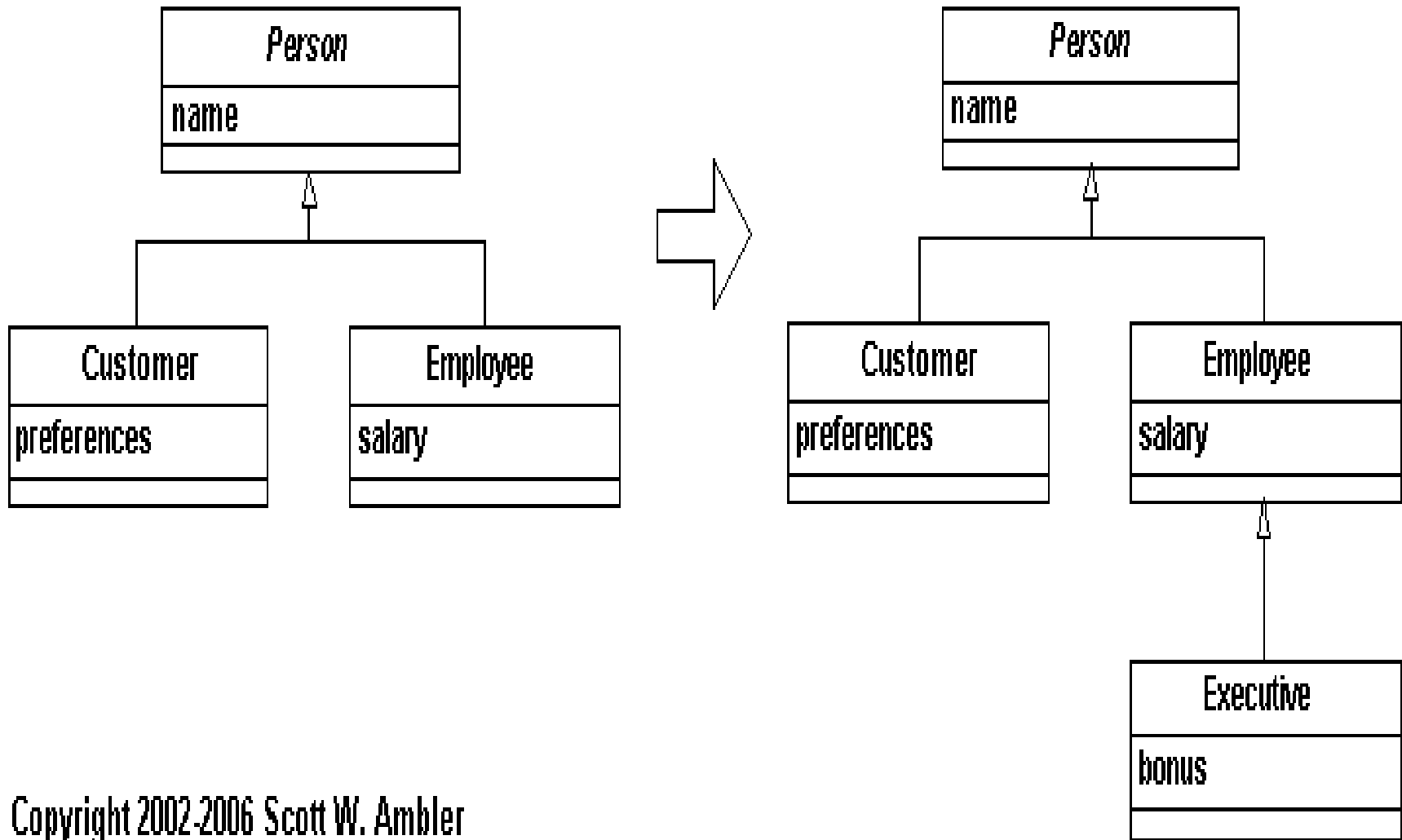
Mapping Metadata

- Impedance mismatch
 - Objects have data and methods
 - Tables only have data
 - `getFirstName()` and `setFirstName()` both map to `FirstName` Column!

Mapping Inheritance

- Impedance mismatch
 - RDBs don't support inheritance
- 4 approaches
 - Map inheritance hierarchy to 1 table
 - Map each concrete class to its own table
 - Map each class to its own table
 - Map classes into generic table structure

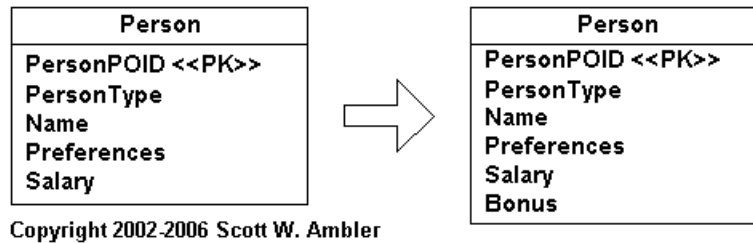
Mapping Inheritance: Case Study



Copyright 2002-2006 Scott W. Ambler

Map Hierarchy To Single Table

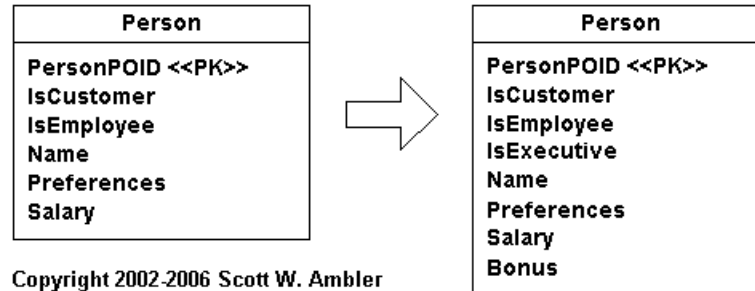
- 1 table contains all attributes for all classes in hierarchy



- Superclass name = table name
- PersonType indicates subclass type

Map Hierarchy To Single Table

- Refactored mapping

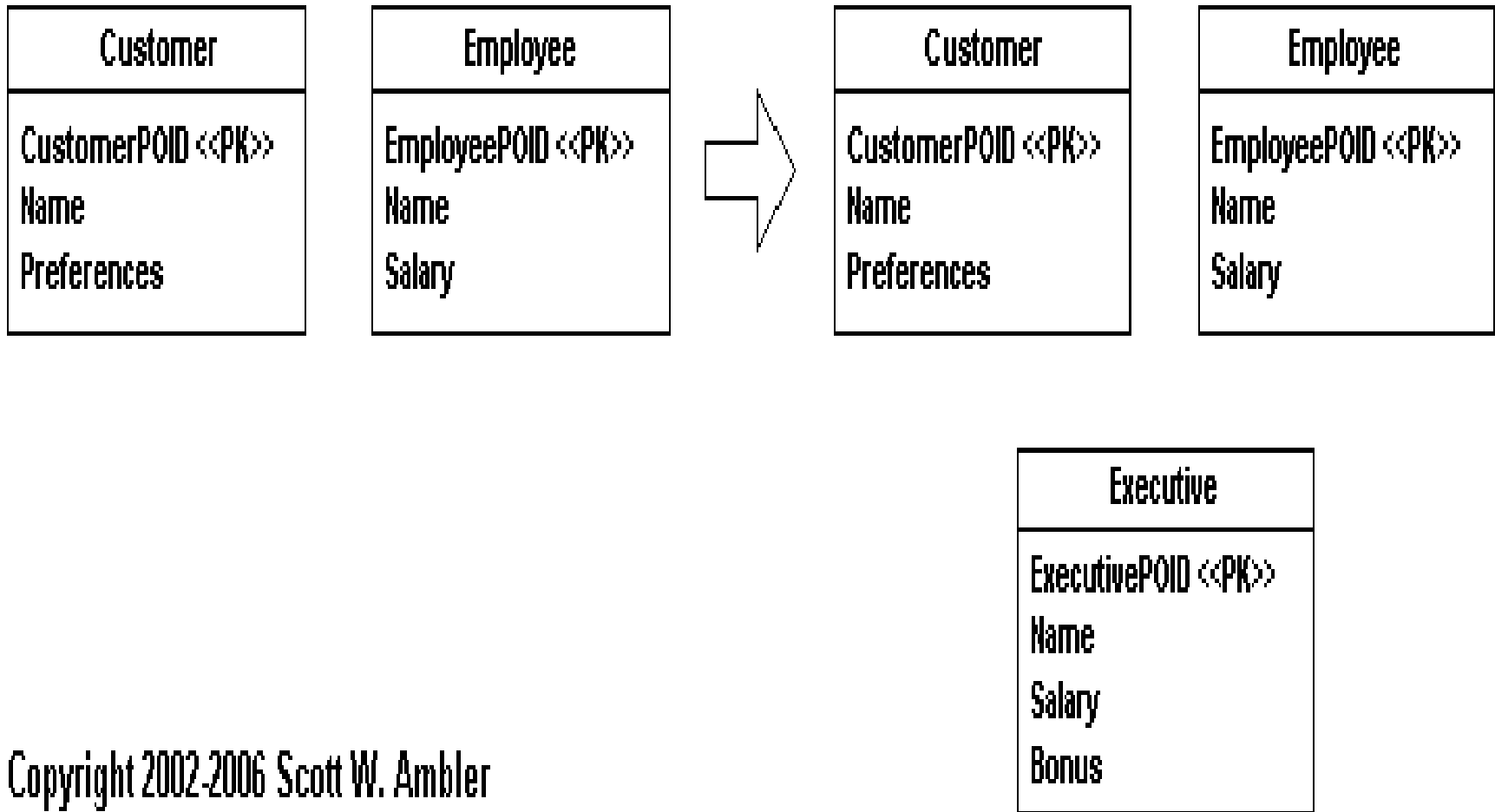


- Boolean attributes indicate subclass type
- Person can be Employee and Executive

Map Concrete Class To Table

- Each concrete class maps to its own table
- Note that inherited attributes are duplicated across tables

Map Concrete Class To Table

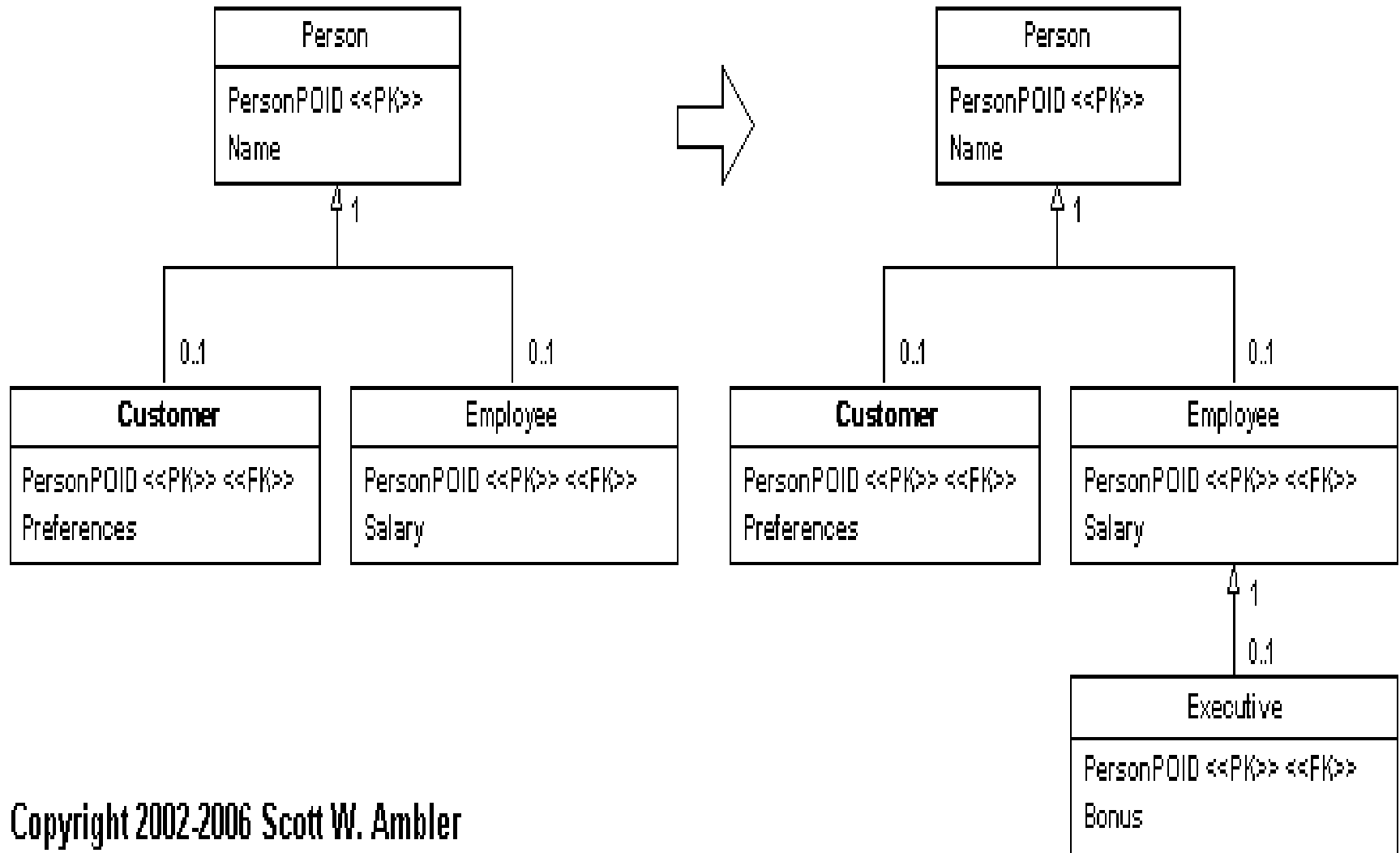


Copyright 2002-2006 Scott W. Ambler

Map Each Class To Own Table

- Avoids duplication of attributes across tables
- To retrieve all Customer data, must join Customer with Person
- PersonOID is both PK and FK

Object Relational Mapping

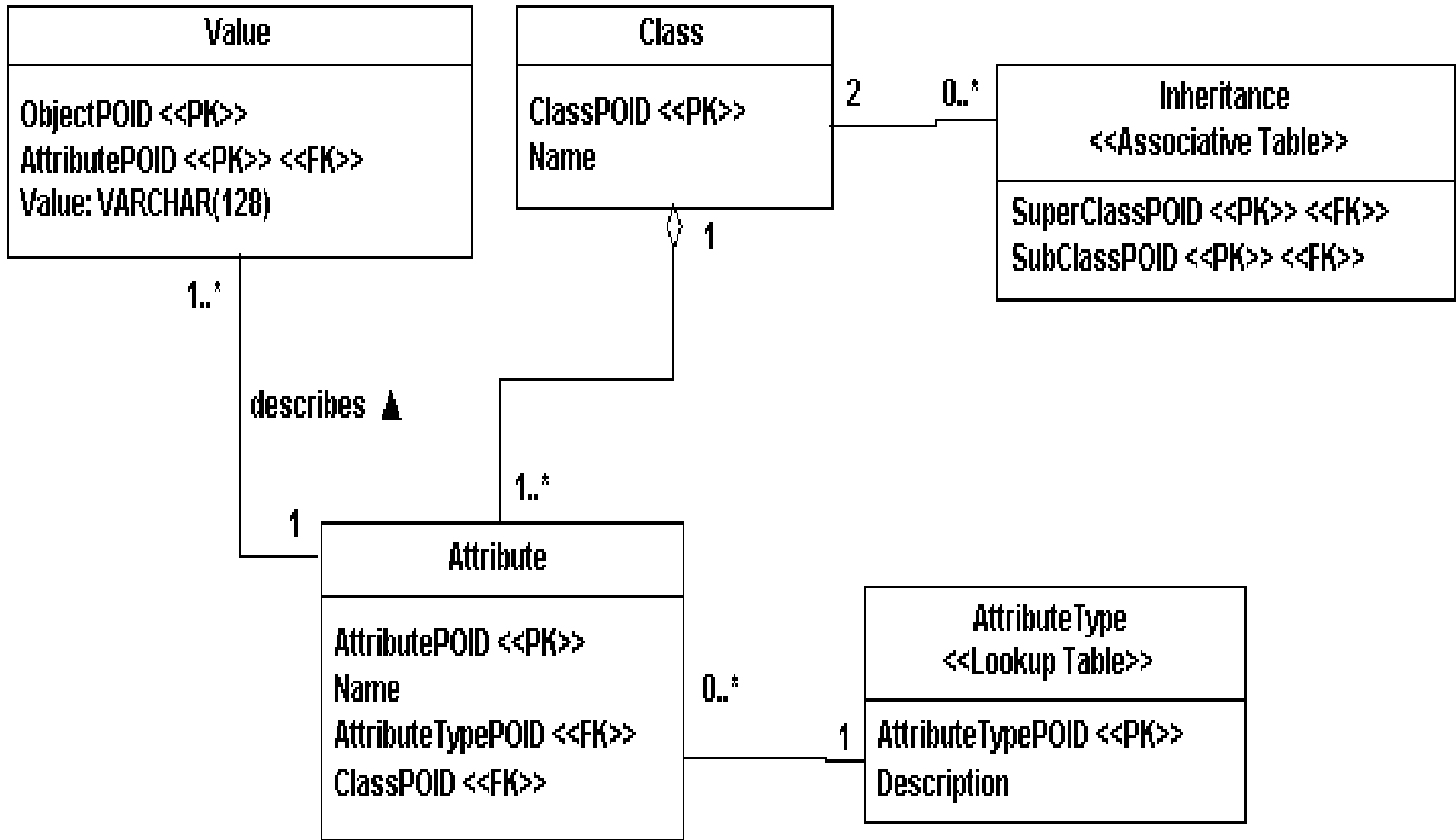


Copyright 2002-2006 Scott W. Ambler

Map Classes To Generic Table

- Metadata driven approach
- Abstract DB representation of a Class
- DB model entirely driven by OO model

Object Relational Mapping



Copyright 2002-2006 Scott W. Ambler

Inheritance Mapping Pros and Cons

- No strategy is always the best
- Easiest is to start with is 1 table/hierarchy
- Can refactor to other mappings as needed
- Generic metadata doesn't scale well
- The top 3 strategies can be used together in a system

One Table Per Hierarchy

- Pros
 - Simplest
 - Handles polymorphism by changing the type of a row
 - 1 table -> fast data access!
 - Easy queries

One Table Per Hierarchy

- Cons
 - Coupling within hierarchy so change to one class affects table -> affecting other subtypes
 - Empty fields -> wasted space in table
 - Type flagging can get messy
 - Table can get very big
- Best practice
 - Use for simple/shallow class hierarchies

One Table Per Concrete Class

- Pros
 - Easy querying since all class data in 1 table
 - Good performance
- Cons
 - Superclass changes require subclass table changes
 - Object properties occur across multiple tables
- Best practice
 - When there is little overlap between types

One Table Per Class

- Pros
 - 1 class -> 1 Table mapping easy to understand
 - Polymorphism indicated by table record occurs in
 - Decouples superclass from subclass -> easier to modify superclass independent of subclass

One Table Per Class

- Cons
 - More tables in DB
 - Slower performance
 - Queries more complex due to table joins
- Best practice
 - Use when significant overlap between types in hierarchy and/or types change often

Mapping Object Relationships

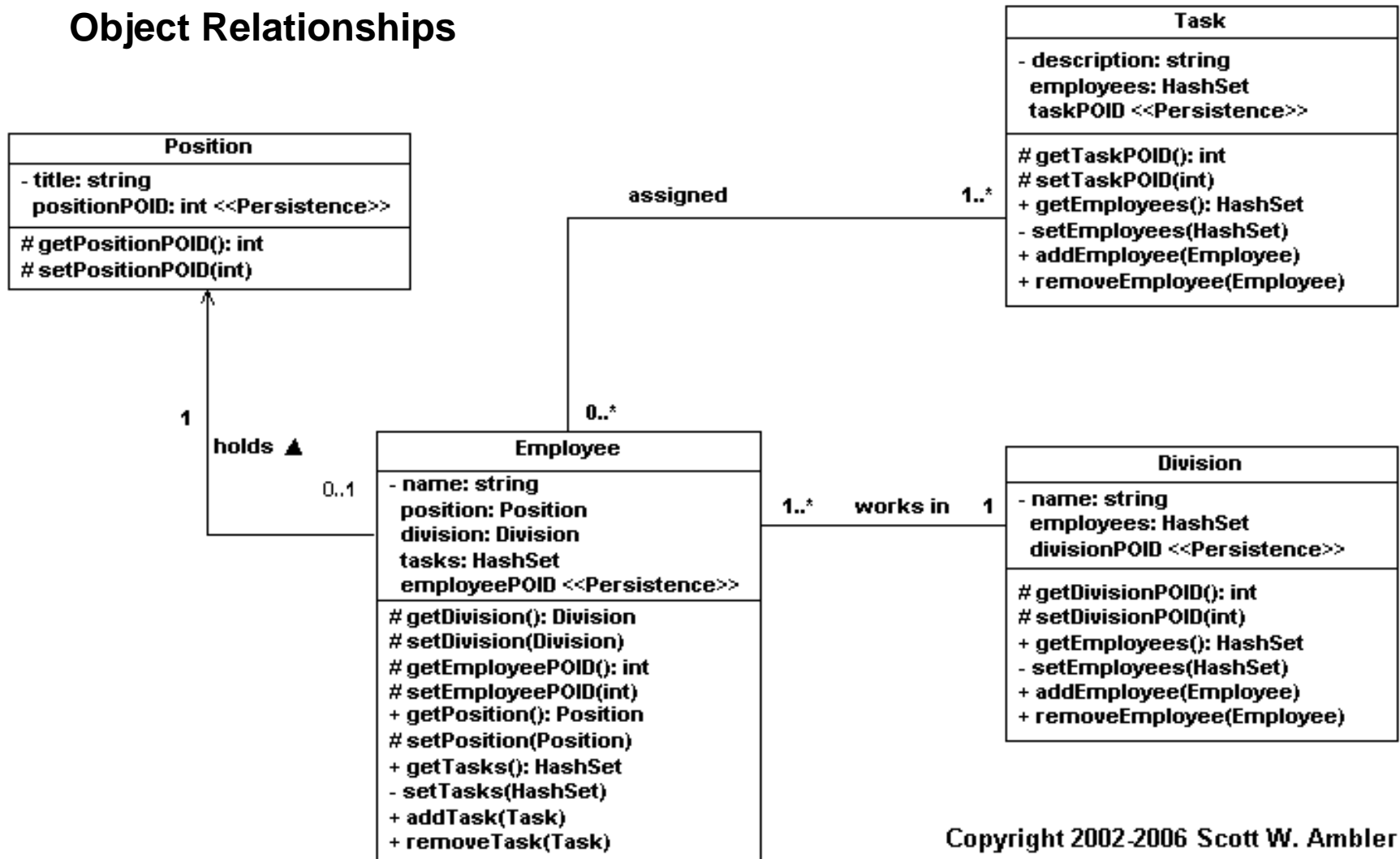
- 2 relationship categories
- Multiplicity-based (3 types)
 - 1:1 1: * * : *
- Directionality-based (2 types)
 - Uni- and bi-directional (object visibilities)
- Yields 6 relationship types

Mapping Object Relationships

- Impedance mismatch
 - In DBs, all relationships are bi-directional
 - FK-based relationships traversable in either direction

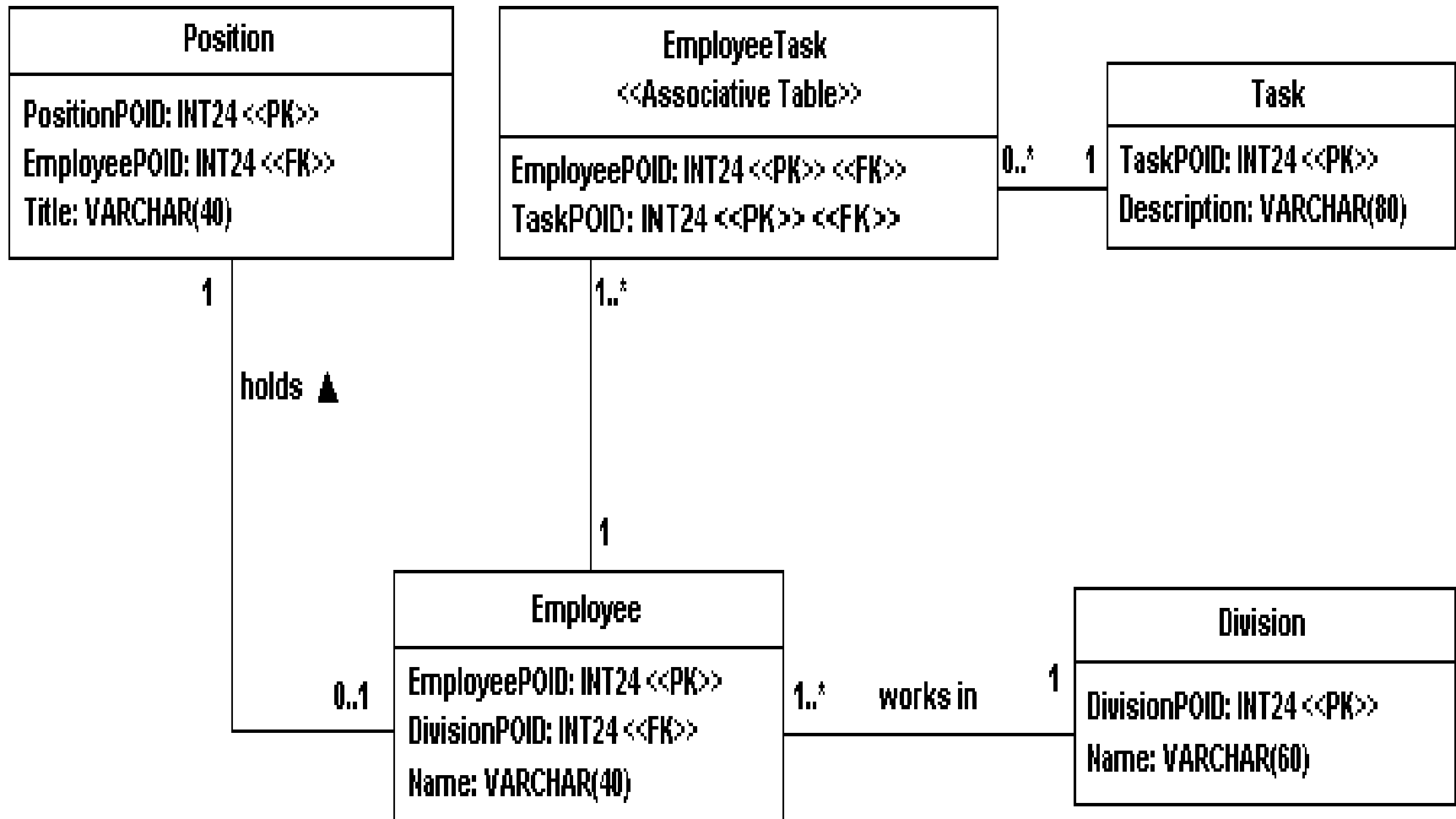
Object Relational Mapping

Object Relationships



Copyright 2002-2006 Scott W. Ambler

Mapping Object Relationships



Data Relationships

Copyright 2002-2006 Scott W. Ambler

Mapping Object Relationships

- Implementing relationships in DB
 - 1:1
 - Make either PK, the FK in the other
 - 1:*
 - Make the PK of the 1, the FK in the many
 - *.*
 - Create associative table, Make each PK a FK in the new table

Mapping Object Relationships

Figure 13. Property mappings.

Property	Column
Position.title	Position.Title
Position.positionPOID	Position.PositionPOID
Employee.name	Employee.Name
Employee.employeePOID	Employee.EmployeePOID
Employee.employeePOID	EmployeeTask.EmployeePOID
Division.name	Division.Name
Division.divisionPOID	Division.DivisionPOID
Task.description	Task.Description
Task.taskPOID	Task.TaskPOID
Task.taskPOID	EmployeeTask.TaskPOID

Mapping Object Relationships

Figure 14. Mapping the relationships.

Object Relationship	From	To	Cardinality	Automatic Read	Column(s)	Scaffolding Property
holds	Employee	Position	One	Yes	Position.EmployeePOID	Employee.position
held by	Position	Employee	One	Yes	Position.EmployeePOID	Employee.position
works in	Employee	Division	One	Yes	Employee.DivisionPOID	Employee.division
has working in it	Division	Employee	Many	No	Employee.DivisionPOID	Division.employees
assigned	Employee	Task	Many	No	Employee.EmployeePOID EmployeeTask.EmployeePOID	Employee.tasks
assigned to	Task	Employee	Many	No	Task.TaskPOID EmployeeTask.TaskPOID	Task.employees

Mapping 1:1 Relationships

- Create objects
 - Employee holds position
 - Read Employee object data from Employee table
 - Employee.POID is used to identify Position data in Position table
 - Employee.position attribute points to Position object
- Persist objects
 - Start a DB transaction
 - Update values for both objects
 - Submit transaction to DB (all or nothing)

Mapping 1: * Relationships

- Employee 'works in' Division
- Read objects
 - Read Employee object, then read Division based on Division FK in Employee
- Persist objects
 - Same as for 1:1

Mapping * : * Relationships

- Need associative table
 - Employee 'assigned to' Task is * : *
 - Create EmployeeTask Table
 - Contains FKs to Employee & Task
- Read all Tasks for an Employee
 - Join EmployeeTask and Task tables
 - SELECT all Tasks, WHERE
EmployeeTask.EmployeePOID =
Employee.EmployeePOID
 - Convert each record into a Task object

Mapping * : * Relationships

- Refer to Ambler article for details on
 - How to persist objects in *: * relationships
 - Mapping ordered collections
 - Mapping recursive relationships
 - Mapping class scope (static) properties
 - Additional advanced topics
- For Java Persistence Frameworks, read about Hibernate (www.hibernate.org)