UNIVERSITY OF Cincinnati

Domain Logic Patterns
(Patterns of Enterprise Application Architecture - Chapter 9)

Mike Helmick
Large Scale Software Engineering - Spring 2014
University of Cincinnati

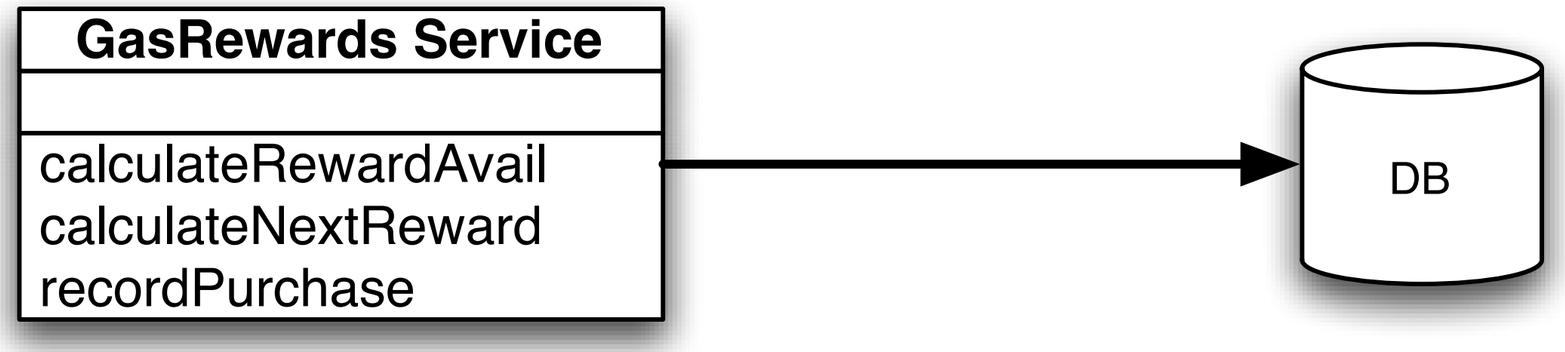
Patterns (Ch 9)

Patterns (Ch 9)

- Transaction Script (Page 110)
- Domain Model (Page 116)
- Table Module (Page 125)
- Service Layer (Page 133)

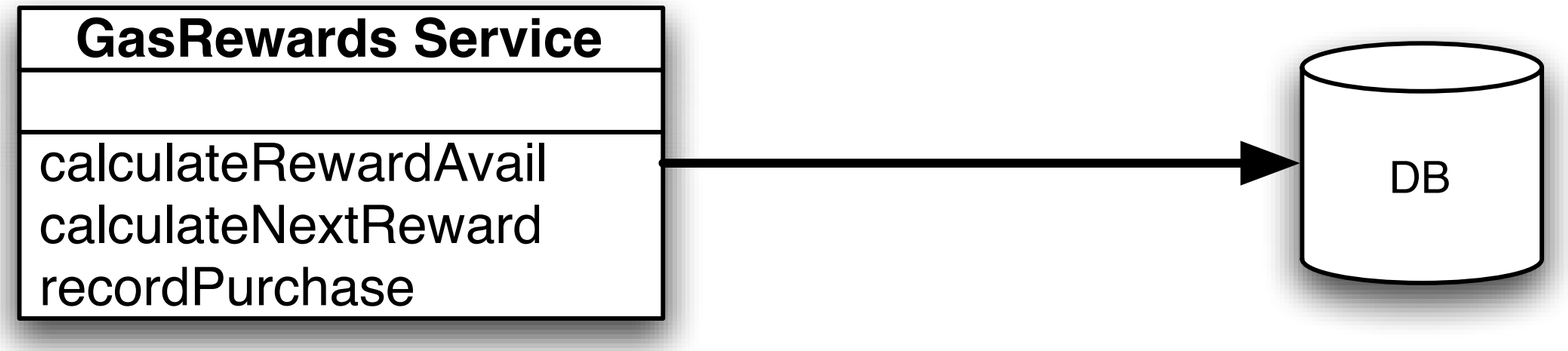
Transaction Script

Transaction Script



Transaction Script

- “Organizes business logic by procedures where each procedure handles a single request from the presentation”



Transaction Script

Transaction Script

- All logic is organized in these large single procedures
- Will usually call database directly or use a thin wrapper

How it works

How it works

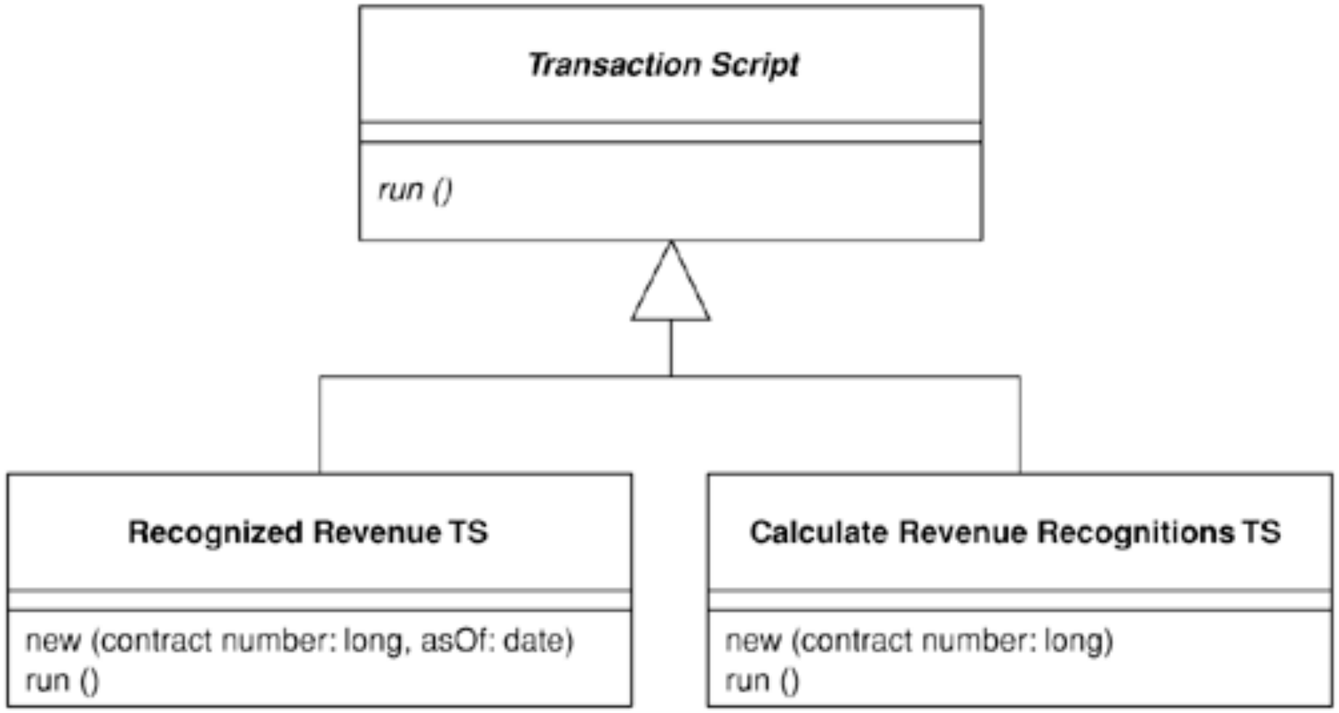
- When discovering your software requirements - think about the coarse steps of the applications
- Your transaction is all in one procedure - so we don’t really have to worry about what other transactions are doing

Where it goes

Where it goes

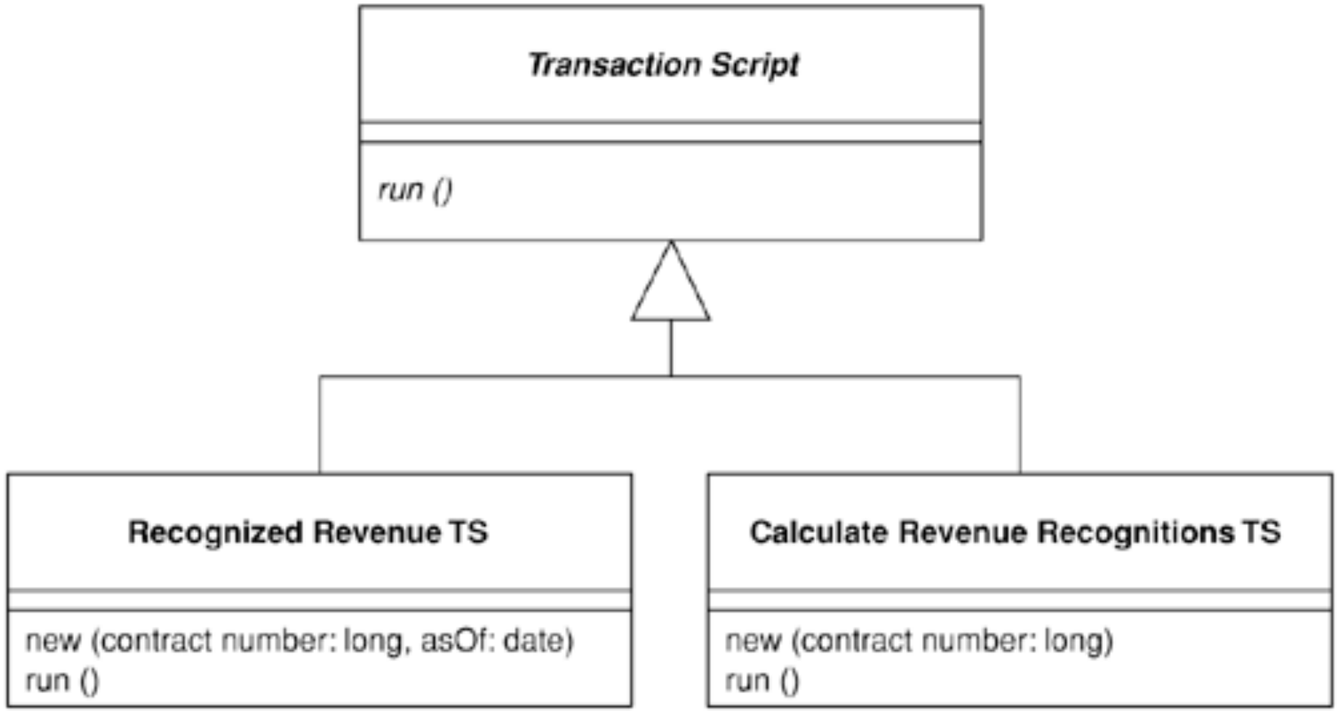
- Depends on your organizations
    - Server Page
    - CGI Script
    - Remote session object

Reuse



Reuse

- We can still organize for reuse
  - putting common functionality in a base class (transaction control / security)



Organization

Organization

- Several transaction script methods in the same class
    - Usually related business logic
- One class for each transaction script
    - Can take advantage of the “command pattern”

When to use

When to use

- When simplicity is important
- Small applications - simple logic
- Have very low overhead - possibly use in extremely performance critical applications

Drawbacks

Drawbacks

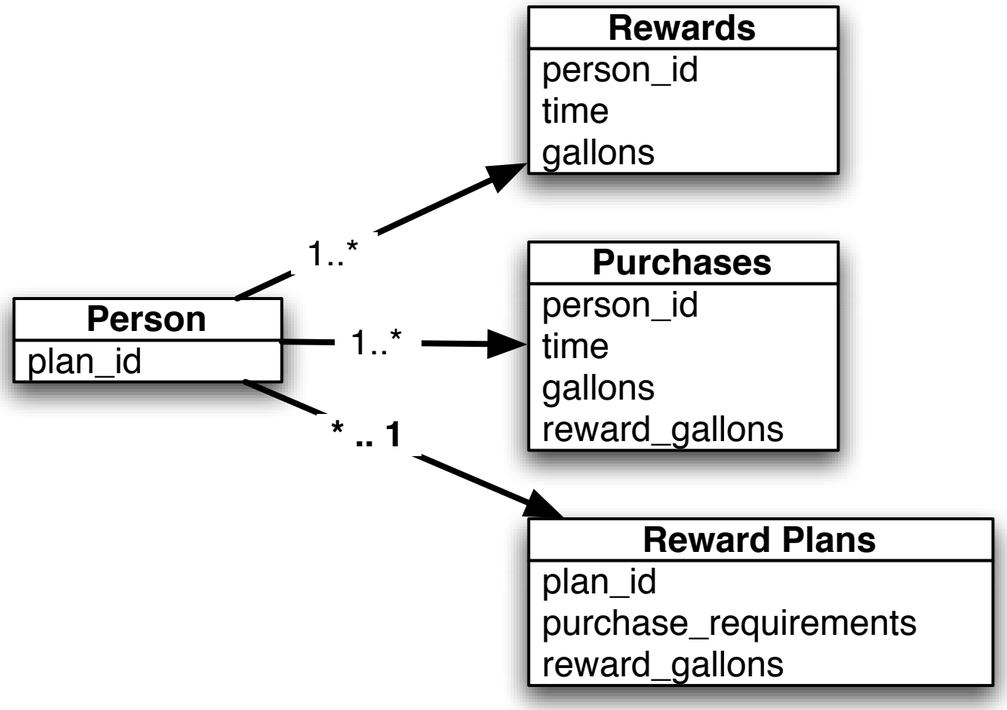
- Sort of becomes unwieldy as your application grows (think of having several hundred transaction script objects)
- A lot of code gets duplicated
- Difficult to refactor

Sample Problem

Sample Problem

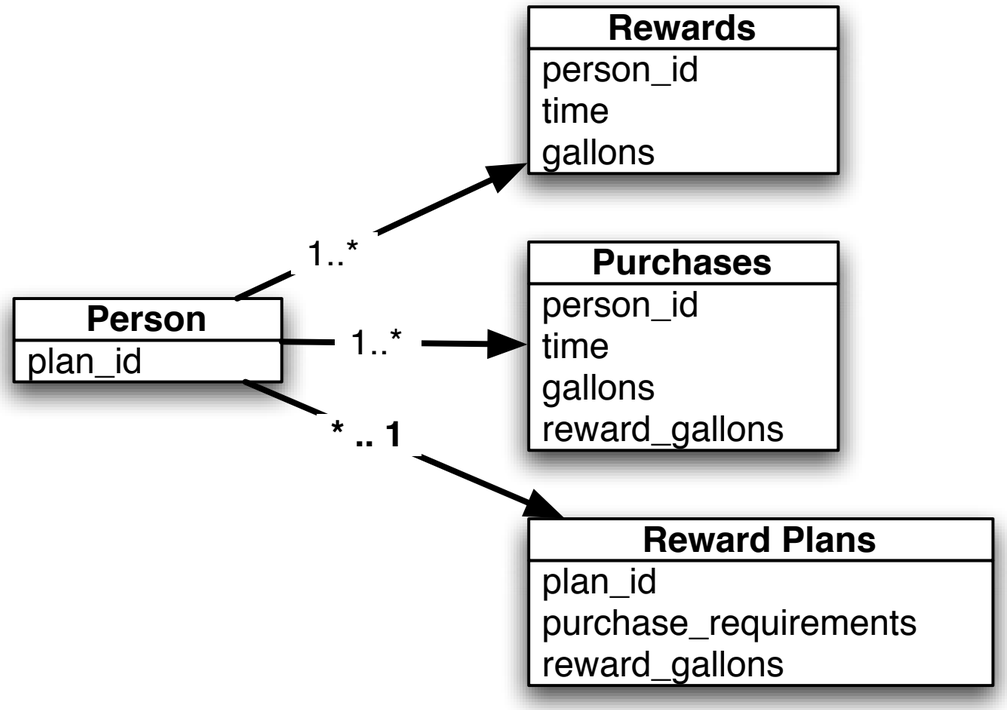
- Gas Rewards
    - Maybe we have different plans
    - Free gallon for every 10 purchased
    - Free gallon for every 15 purchased
- Need to record purchases, calculate reward gallons, calculate what is needed for reward

Rules



Rules

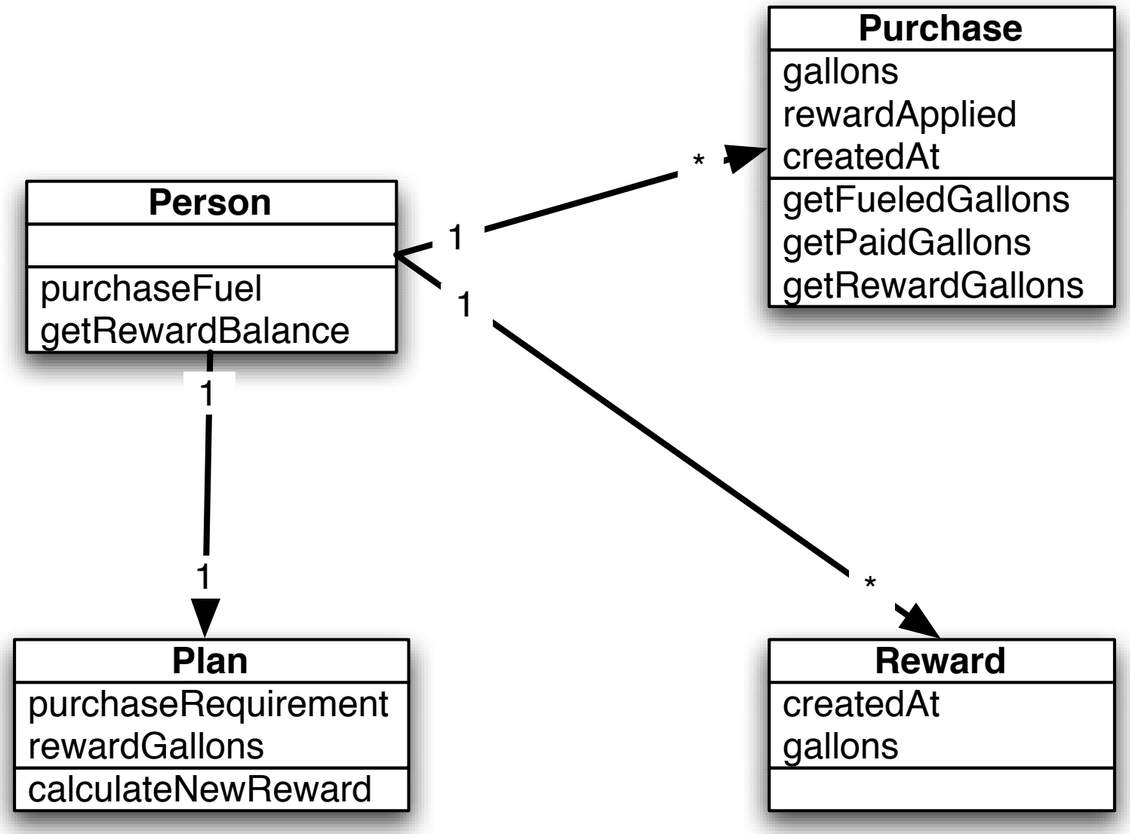
- Depending on the product purchased, revenue is recognized on different settings



Example

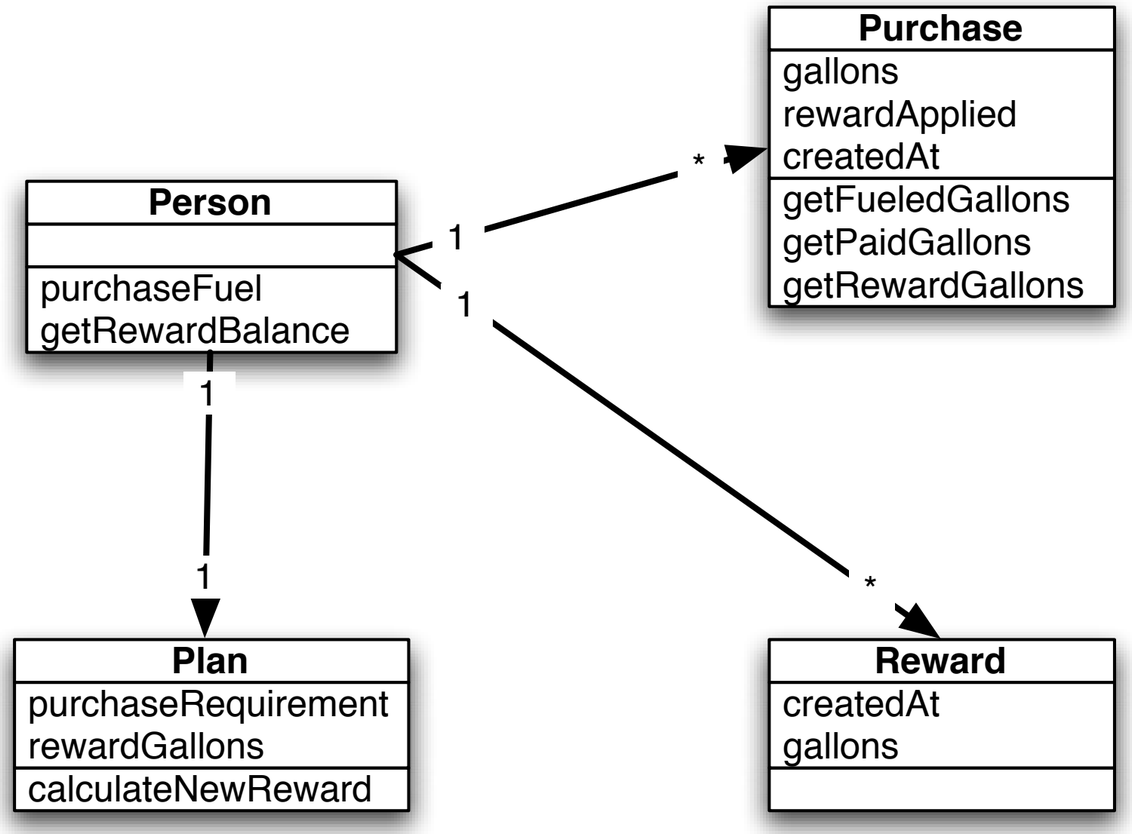
Domain Model

Domain Model



Domain Model

- “An object model of the domain that incorporates both behavior and data.”



Domain Model

Domain Model

- Interconnected object where each object represents an individual entity within the system
- Very object-oriented
    - fits in well with

How it works

How it works

- You have an entire layer that is an object-oriented representation of the entire business logic (entities and actions)
- looks similar to database design
  - but different
  - Database only represents relationships
    - OO model does more

Domain Models

Domain Models

- Simple Domain Model
    - Very closely maps to the database
    - Easy to map to the database
- Rich Domain Model
    - Uses OO style for things - inheritance, etc...
    - Difficult to map to the database

Mapping

Mapping

- Simple domain model
    - can use Active Record (advantages to this)
- Rich domain model
    - needs to use Data Mapper

Testing

Testing

- Minimize coupling between the Domain Model and the other layers
    - Specifically the ideal situation allows the Domain Model to function

Memory

Memory

- We can very rarely hold the entire object model in memory
- So
  - we pull back the parts that we need from the database for the current calculation
  - we will even ignore related data until we need it (Lazy Load)

Bloat

Bloat

- Sometimes with Domain Model developers put too much one time use logic in the objects
- Is something is really specialized - we can maybe move it to a service layer
- Have to balance between bloat and duplication

Java

Java

- As noted on page 118-119
- Using POJOs (Plain Old Java Objects) has many advantages
  - Easy to develop
  - No dependencies

When to Use

When to Use

- Depends on your complexity
- Domain Model scales very well (in terms of both complexity and number of entities)
- Learning how to design domain models takes lots of practice / experience

Considerations

Considerations

- Author prefers Data Mapper when using the Domain Model pattern
    - I prefer the Active Record pattern
- A Service Layer is an excellent addition since it gives your program an API
    - This is preferable to exposing your domain model & possibly allowing for violations of business logic

Example

Table Module

Table Module

Table Module

- “A single instance that handles the business logic for all rows in a database table or view.”

Table Module

Table Module

- Organizes domain logic with 1 class per table
- Contains methods to act on the data for that table

How It Works

How It Works

- Works by packaging data and behavior (OO)
    - but more closely models the relational database
- What's missing
    - Identity - the Table Module object has no identity

How It Works

How It Works

- Deals with tabular data (Record Set) which looks like the database table
- Could be an object that you instantiate - or just a collection of static methods

Queries

Queries

- Can be built into the Table Module
    - or in a separate Table Data Gateway
- When you use a Table Data Gateway
    - adds complications
    - allows you to create a record set from scratch and test the Table Module

When to use

When to use

- When your data is tabular - when it makes sense to operate on a whole table at the same time
- Doesn’t represent relationships well
- Can’t represent complex logic well
- Primarily used in .NET/COM

Example

Example

- See the C# example on pages 129 - 132

Service Layer

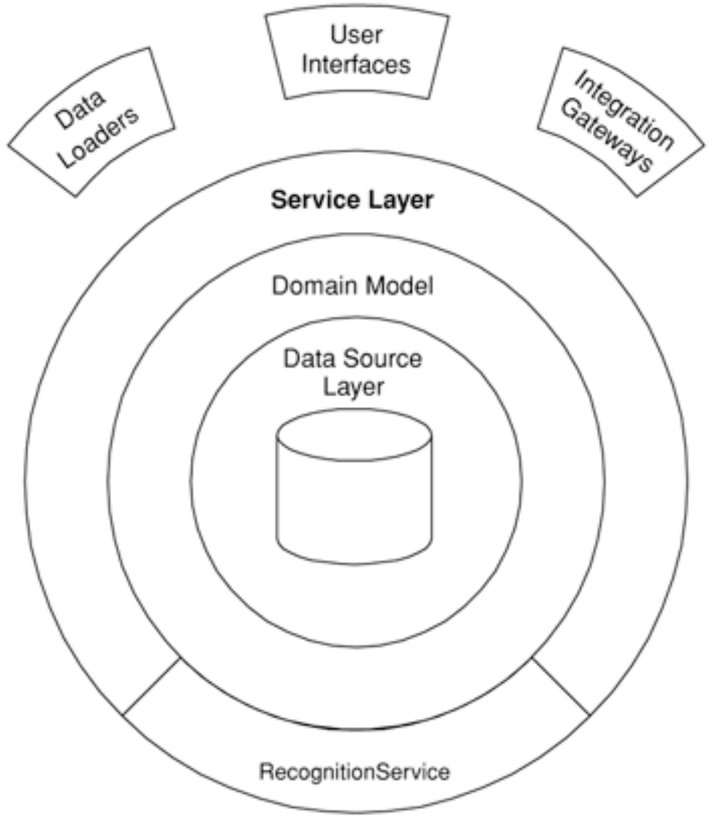
Service Layer

Service Layer

- “Defines an application’s boundary with a layer of services that establishes a set of available operations and coordinates the application’s response in each operation.”

Service Layer

- “Defines an application’s boundary with a layer of services that establishes a set of available operations and coordinates the application’s response in each operation.”



Service Layer

Service Layer

- Enterprise applications typically have multiple interfaces
    - GUI, Web Service, Batch Jobs, etc...

How It Works

How It Works

- There are many various ways to build a service interface

Kinds of Business Logic

Kinds of Business Logic

- There are two central kinds of logic
  - domain logic - having to do with the specific problem at hand
  - application logic - application responsibilities - integration, notifications

Implementation Variations

Implementation Variations

- Domain Facade
  - The Service Layer manipulates the object-oriented Domain Model
  - Presents a well defined boundary between the layers of your application

Implementation Variations

Implementation Variations

- Operation script
  - Set of “thicker” classes that implement application logic
  - Still delegates domain logic to another layer

Remoting

Remoting

- Your service layer should be coarse grained
    - this makes it suitable for acting as a remote service layer
- Remote invocation is comparatively expensive

Remote

Remote

- If you provide a good separation of layers...
  - you can start coding locally and then distribute your service layer later

Identifying Services

Identifying Services

- The services you provide are typically defined by the requirements of the service consumers
    - i.e. if there are two comparable services - we want the one thats easier to work with

Identifying

Identifying

- Most operations are CRUD type operations

- They still have lots of work that goes around them (validations, etc...)

Java

Java

- Can be implemented using POJOs or Stateless Session Beans
    - Ease of testing vs. transaction control
- Author recommends a hybrid approach
    - Stateless Session Beans - delegating to POJOs
    - We can now control transactions without EJBs

UNIVERSITY OF Cincinnati

Data Source Patterns
(Patterns of Enterprise Application Architecture - Chapter 10)

Mike Helmick
Large Scale Software Engineering - Spring 2014
University of Cincinnati

Patterns (Ch 10)

- Table Data Gateway
- Row Data Gateway
- Active Record
- Data Mapper

Table Data Gateway

Table Data Gateway

- “An object that acts as a Gateway to a database table. One instance handles all the rows in the table.”

Person Gateway
find (id) : RecordSet findWithLastName(String) : RecordSet update (id, lastname, firstname, numberOfDependents) insert (lastname, firstname, numberOfDependents) delete (id)

Table Data Gateway

Table Data Gateway

- Holds all of the SQL for the table
    - isolated from the application

How It Works

How It Works

- A very simple interface
    - Finder methods
    - Insert new record
    - Update existing
    - Delete record
- Usually stateless (could be static methods)

How It Works

How It Works

- All data is returned as multiple values (using a Record Set)
    - This requires you to know the underlying makeup of the data (raw database layout)
- You could modify this to return a Map based structure
    - Requires a pass through the mapping layer

How It Works

How It Works

- The Record Set still binds you to the specific data interface
- You can modify the Table Data Gateway to work with your Domain Model
    - This can cause problems with relationship mapping though
- Usually one class per database table

When to Use It

When to Use It

- First - decide if a gateway should be used at all
- If there is a very simple database interface
  - easy to understand
  - quick to implement

When to Use It

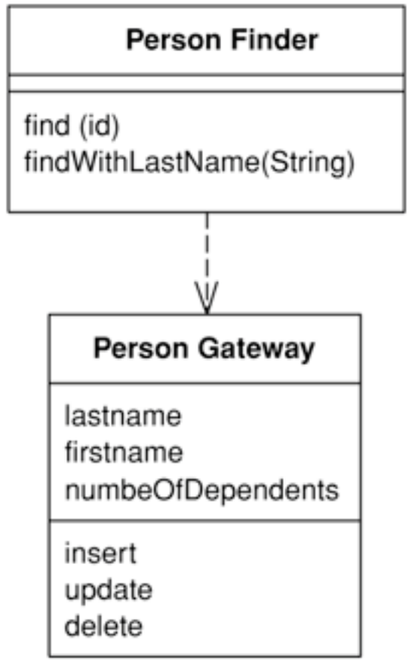
When to Use It

- Works well with the Table Module data model
- Works well with Transaction Script
- Can also encapsulate stored procedures

Example

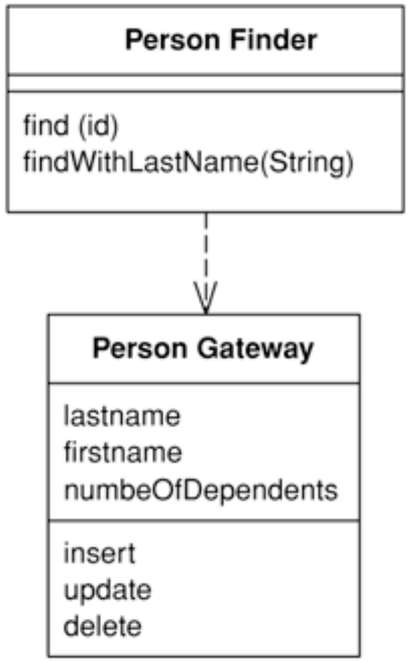
Row Data Gateway

Row Data Gateway



Row Data Gateway

- “An object that acts as a Gateway to a single record in a data source. There is one instance per row.”



How It Works

How It Works

- An object that looks exactly like one row in the database (appropriate fields, etc)
- Does type conversion to/from database
- Only contains database access logic
    - & does not contain domain logic

Finders

Finders

- Can go in 2 locations
    - static method on the row gateway object
    - static method on “finder” class
- Putting them on a different class allows you to manage inheritance / polymorphism better

vs Active Record

vs Active Record

- The difference is
  - Active Record includes domain logic
  - Row Data Gateway does not include domain logic

Caution

Caution

- Row Data Gateway records are not created uniformly
    - There is no way to enforce that row ‘2’ isn’t loaded (and possibly manipulated) more than once
- They can interfere with each other

Writing

Writing

- Can take a long time to write
- Can be uniformly generated pretty easily
    - Code generation is something that you should always be thinking of as a possibility

When to Use It

When to Use It

- Again - should you use a gateway at all?
- Works well with transaction script
- Doesn’t work very well with Domain Model
  - Active Record does more with less code
  - Data Mapper works better if more complex

When to Use It

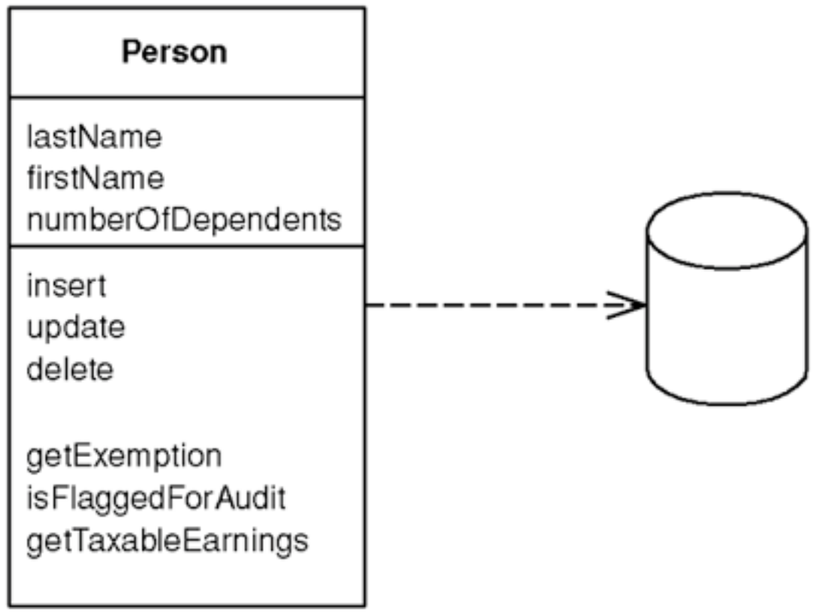
When to Use It

- Row Data Gateway provide good insulation from the database
    - You can even present different field names
    - You can aggregate some fields, etc...

Example

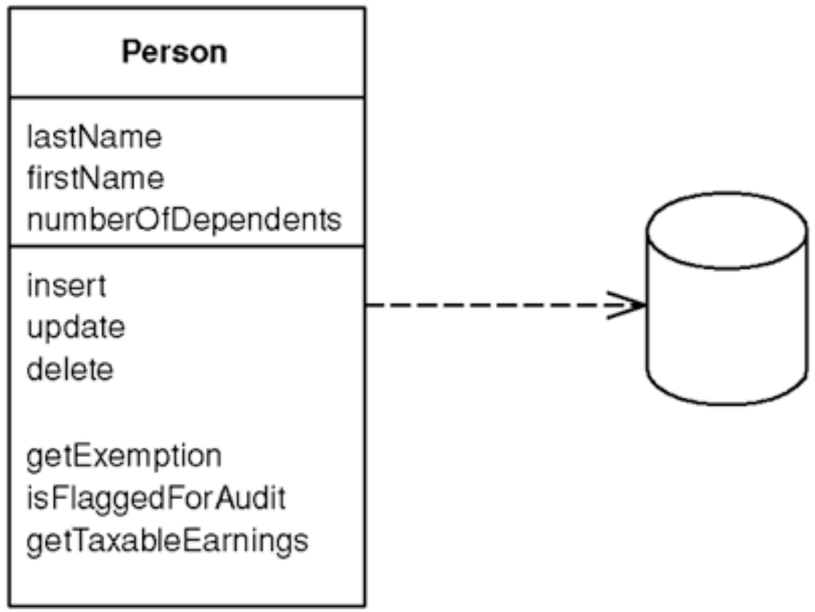
Active Record

Active Record



Active Record

- “An object that wraps a row in a database table or view; encapsulates the database access, and adds domain logic on that data.”



Active Record

Active Record

- A very obvious approach

- Every object is “activated” and knows how to create, find, update, and delete themselves

How It Works

How It Works

- Active Record is a Domain Model with classes that look very much like the database tables
    - Including field names, types, and relationships

How It Works

How It Works

- Each object
  - Has all of it's appropriate domain logic
  - Logic for loading/saving to the database

Fields

Fields

- Best if fields in the class match exactly to the fields in the database
    - This allows for automation

Methods

Methods

- “Construct an instance of the Active Record from a SQL result set row”
- “Construct a new instance for later insertion into the table”
- “Static finder methods to wrap commonly used SQL queries and return Active Record objects”
- “Update the database and insert into it the data in the Active Record”
- Get and set the fields
- Implement some pieces of business logic

Methods

Methods

- ‘Get’ methods
  - Can do SQL type on the fly
  - Can load dependent data (related Active Records - either on creation or using Lazy Load)

When to Use It

When to Use It

- When domain logic isn’t overly complex
    - most of the actions are CRUD type
    - this is usually the case

Active Record

Active Record

- Active Record is a simple
    - it is very easy to use
- But the objects must correspond directly to the database
    - Data Mapper works better when the logic / relationships are overly complex

Real Implementations

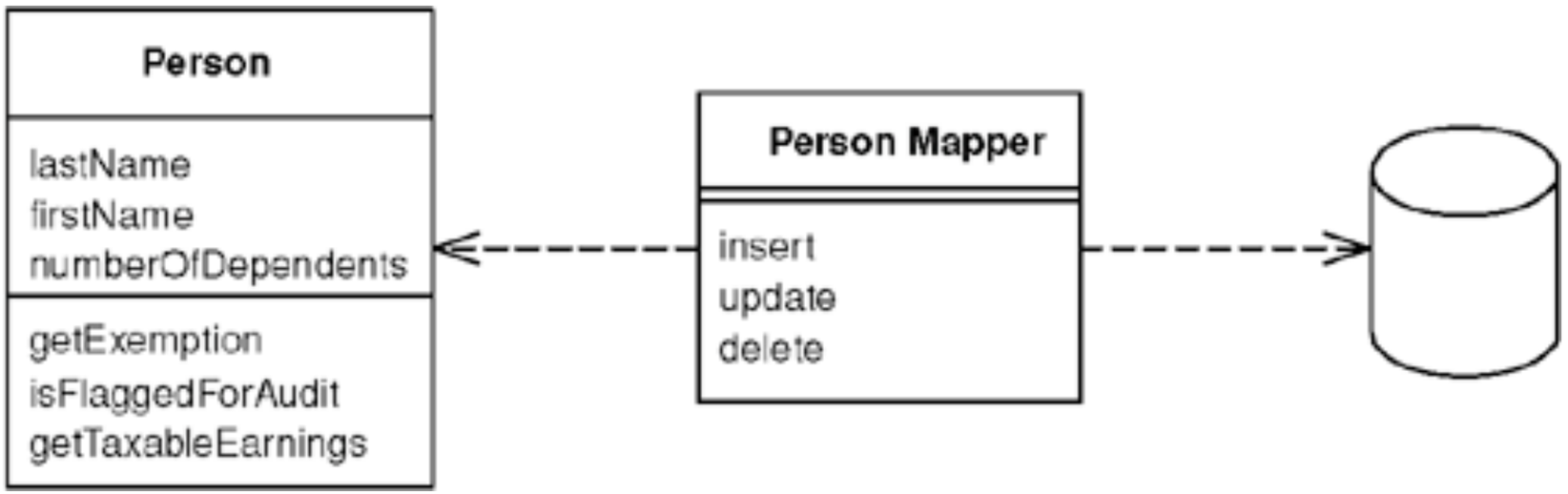
Real Implementations

- This is what Ruby on Rails uses
    - And one of the primary reasons that development with Rails is so quick / easy
- Other languages now use this pattern in frameworks
    - Java - ActiveJDBC - https://github.com/javalite/activejdbc
    - Objective-C - https://github.com/supermarin/ObjectiveRecord

Example

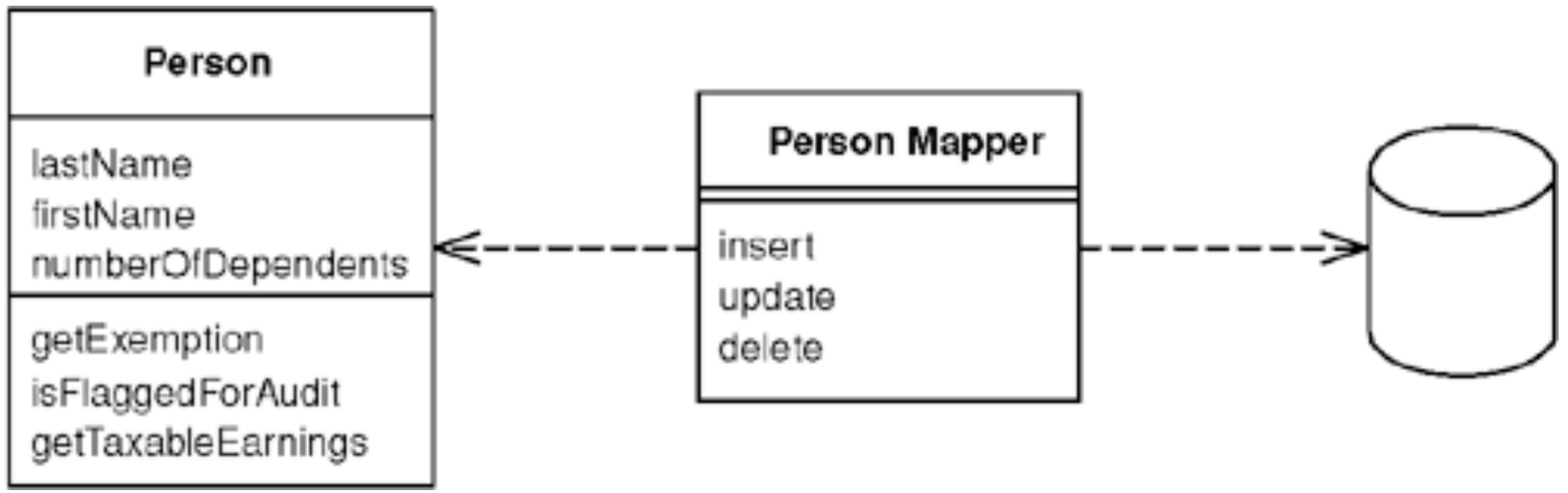
Data Mapper

Data Mapper



Data Mapper

- “A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.”



Data Mapper

Data Mapper

- We've talked about things in OO technology that don't translate well to relational databases
- Coupled with complex business logic
    - We can use a Data Mapper to better organize things

Isolation

Isolation

- The in memory objects don’t necessarily know about the database
- But -
    - changes in one will cause changes in the other

How It Works

How It Works

- The key is that the layers are separated
- Works well with caching mechanisms such as the Identity Map
- The Data Mapper layer can be swapped out for testing
    - or re-configured to point to a test database

How It Works

How It Works

- The Data Mapper needs to be intelligent enough to determine which fields of changed
- And to organize items into a transactional framework
    - Unit of Work is a candidate for this

How It Works

How It Works

- Lazy Load can be utilized
    - at a cost of making your Domain Model somewhat aware of the mapping layer

How It Works

How It Works

- Can be coded 2 ways
  - One mapper for each class - when you hardcode them
  - One data mapper for everything - when using Metadata Mapping

Finders

Finders

- Typical scenario: Service Layer finds a few objects and then kicks off domain logic
- The Domain Objects themselves usually need to find other objects
  - Can be handled with Lazy Load
  - Or an intermediary between the Domain Model and the Data Mapper

Data to Domain Fields

Data to Domain Fields

- The programming language created a problem
    - For some fields, we want them to be read-only to the user
    - But they must be writable by the Data Mapper

Data to Domain Fields

Data to Domain Fields

- A rich constructor can be used
  - A constructor which initializes the minimal set of fields required for a valid object
  - Created no invalid objects
  - Prevents immutable fields from being changed

Metadata Mapping

Metadata Mapping

- The simplest way to do mapping is to hard code everything
- Metadata mapping will store the database to domain model mapping in
  - a class, a file, the database itself
  - Can then either generate the mapper
  - or be used by the mapper

When to Use It

When to Use It

- When you want your object model and your database model to be isolated from each other
- Can potentially evolve them separately from each other
    - but that is not always the case

When to Use It

When to Use It

- An extra layer of software - means that you have an extra layer to test
- If your business logic is simple, you can probably use active record
    - Then use Data Mapper for more complex business logic

Example

UNIVERSITY OF Cincinnati

Object-Relational Behavioral Patterns
(Patterns of Enterprise Application Architecture - Chapter 11)

Mike Helmick
Large Scale Software Engineering
University of Cincinnati
Spring 2014

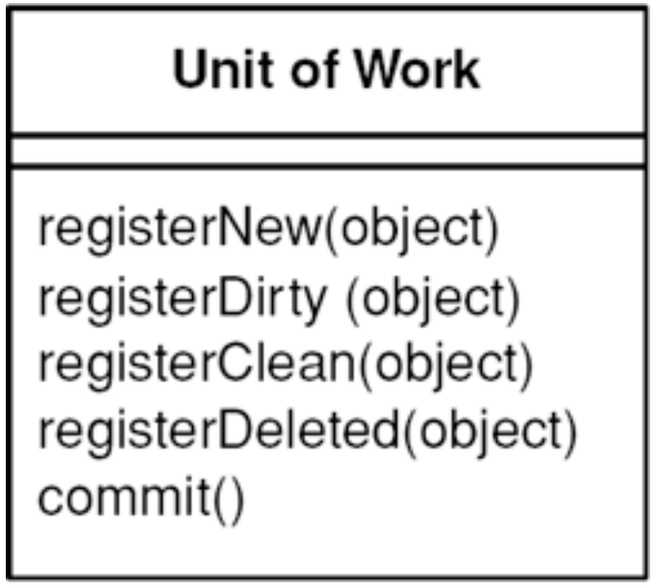
Patterns (Ch 11)

Patterns (Ch 11)

- Unit of Work
- Identity Map
- Lazy Load

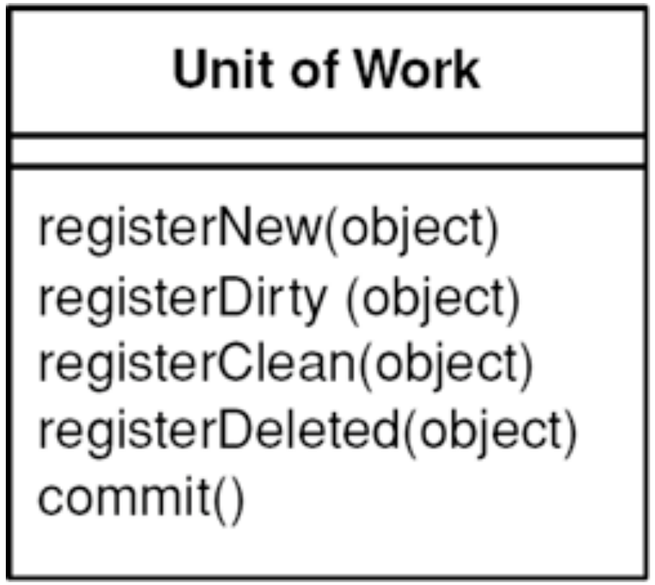
Unit of Work

Unit of Work



Unit of Work

- “Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems”



Unit of Work

Unit of Work

- Changes made to domain objects are useless if they can’t be written back to the database
- We can make multiple SQL calls (each time something changes)
- or - We can keep track of the changes

How it Works

How it Works

- Unit of Work is an object that keep track of
    - new, updated, and deleted objects
- The object handles inconsistent reads
- Tracks changed objects

How it Works

How it Works

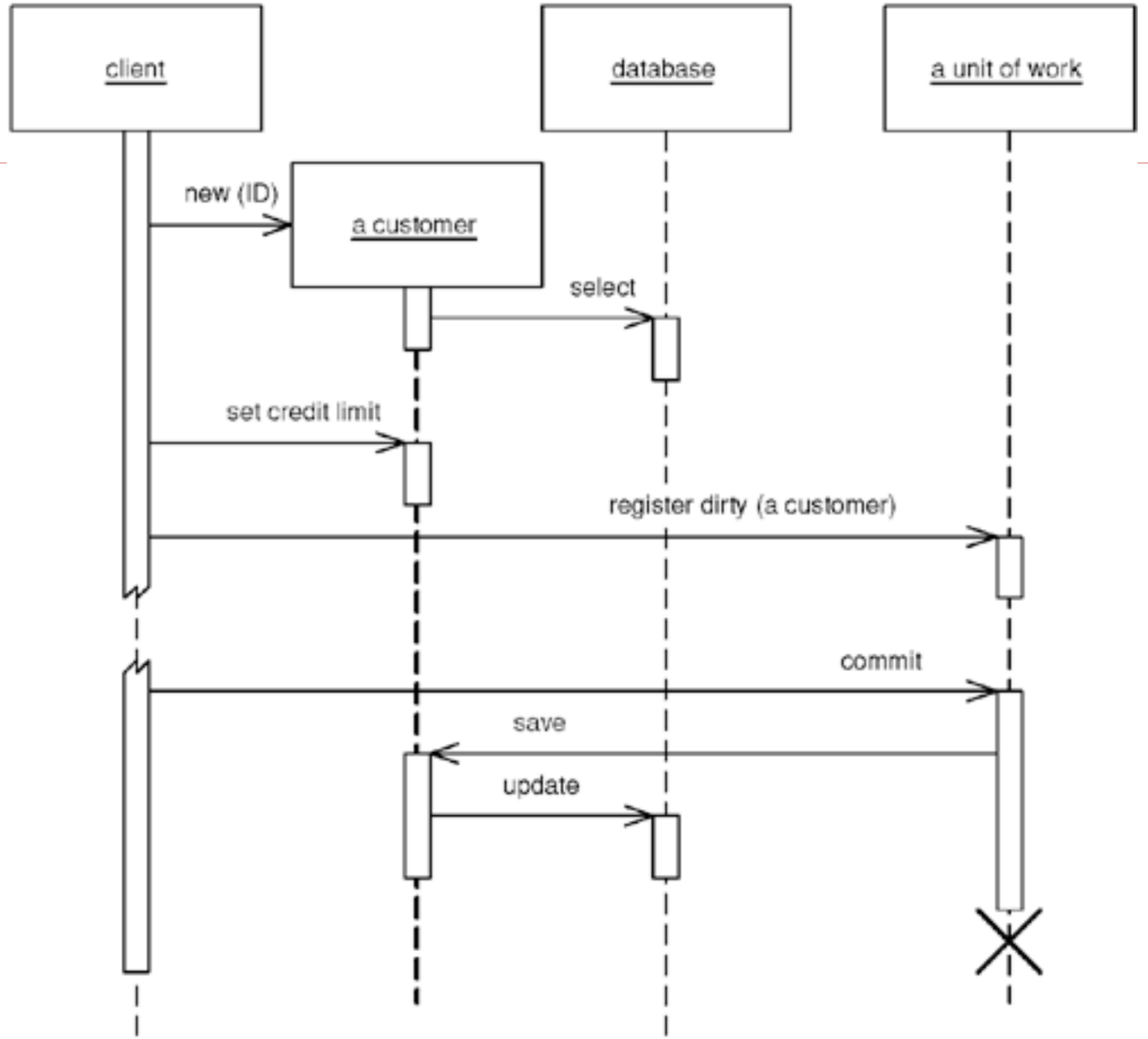
- At commit time
    - The Unit of Work object decides what to do
    - Checks for concurrency violations
        - Pessimistic or Optimistic Offline Lock
- Application programmers never call methods that change the database

Registering

Registering

- An object can be registered with the Unit of Work
    - Manually by the developer
    - Automatically when the change
        - This obviously requires extra wiring
        - You could possibly do some interesting things with the Observer pattern

Caller Registration

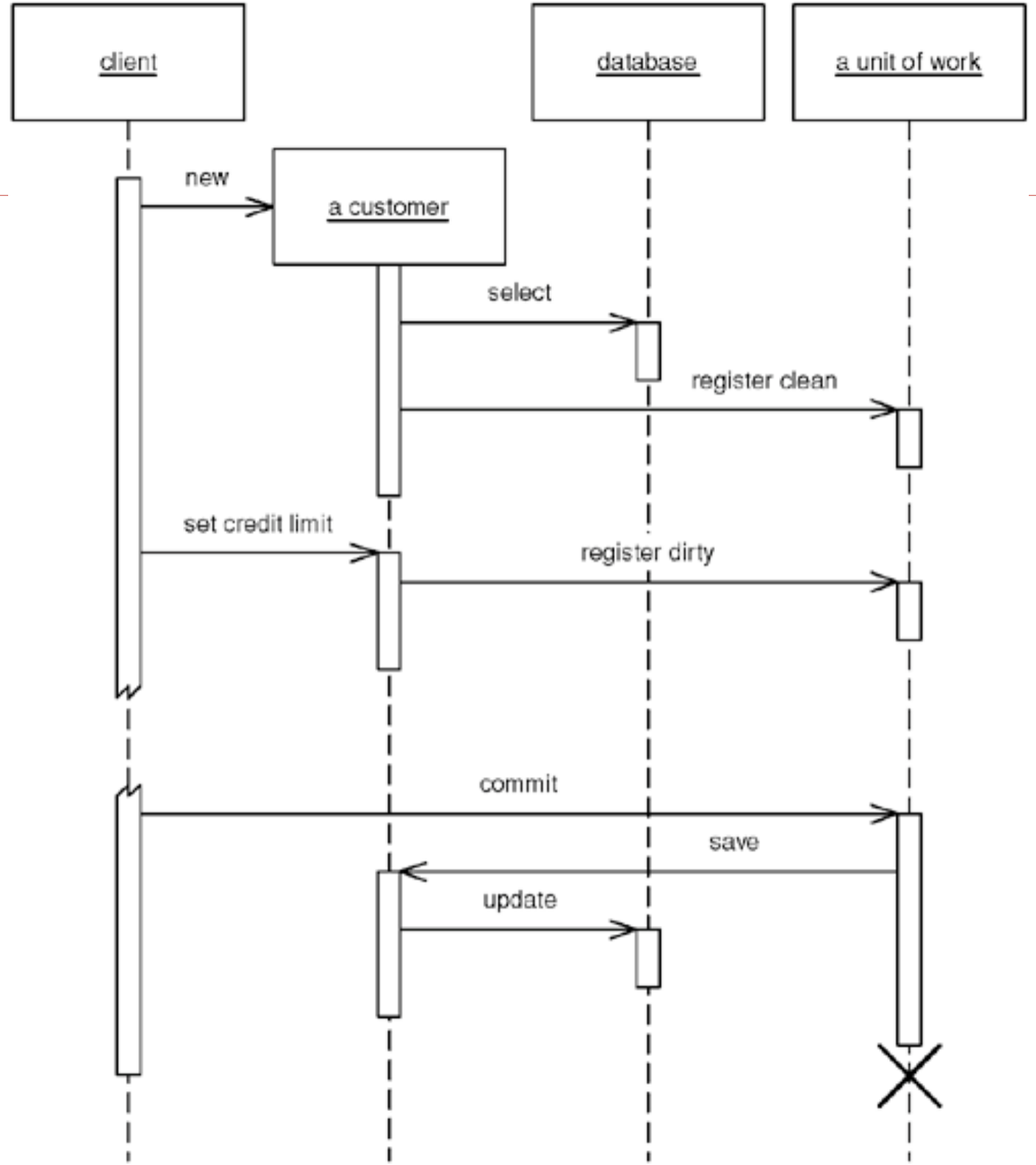


Caller Registration

Caller Registration

- You have to remember to notify the Unit of Work
- Can lead to confusion
- Framework is easier to develop

Object Registration



Object Registration

Object Registration

- Place a registration call in the mutator methods of each domain object
- The Unit of Work needs to be passed to the object or in an easy to find place
  - Thread local storage is a candidate solution
  - Session object access can work as well

Aspects

Aspects

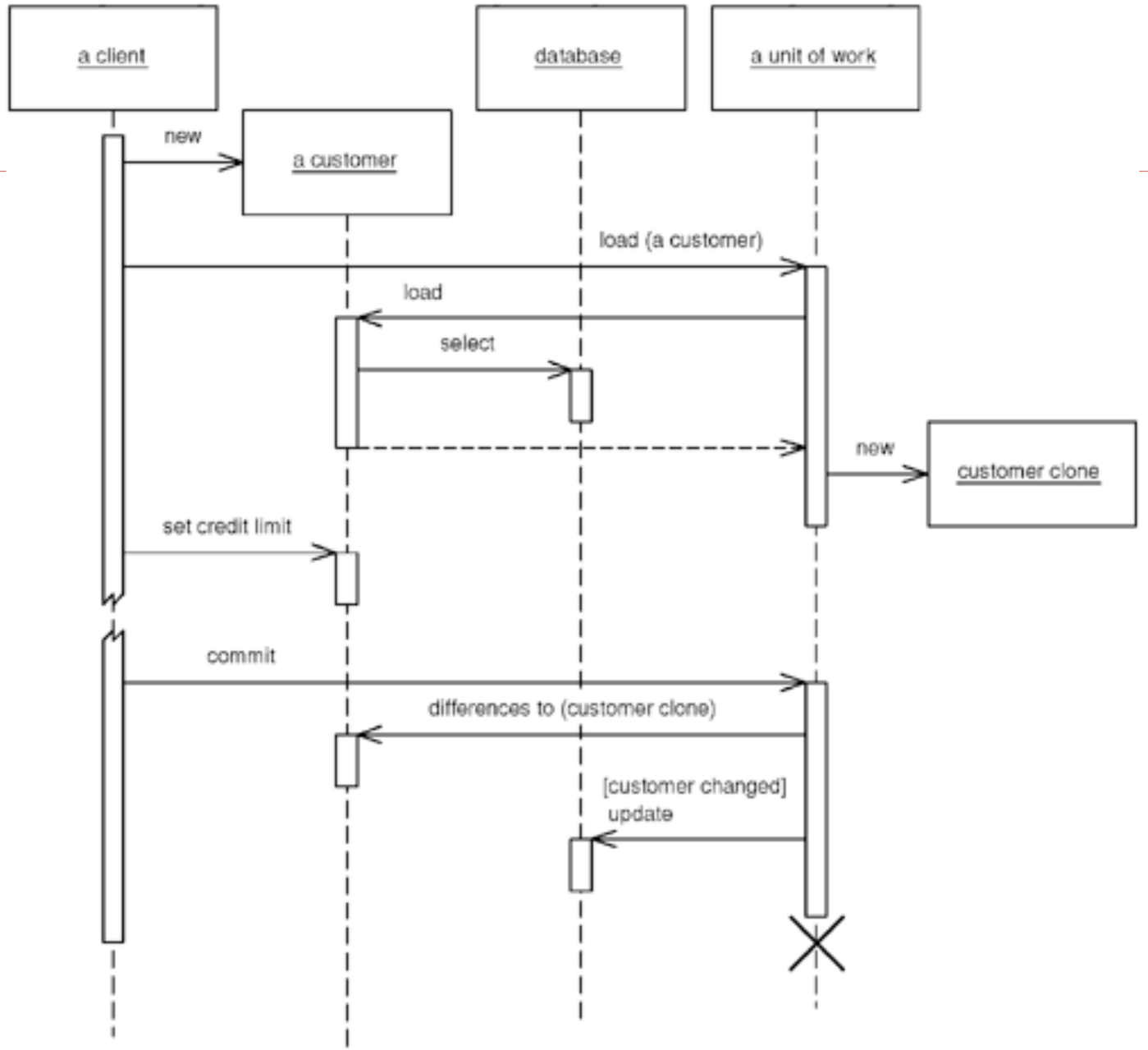
- The problem of inserting the calls to ‘setDirty’ is a candidate for Aspect Oriented Programming
- This can enforce that the method is called after every call to a ‘set’ method

Controller

Controller

- Another option is the unit of work controller
- The Unit of Work handles all reads
  - The object is created twice
    - 1 copy goes to the application
    - 1 copy stays with the unit of work
  - Objects are then compared at commit time

Controller



Referential Integrity

Referential Integrity

- Unit of Work can solve this problem
    - Although your DB software might be able to solve it too
- You can define an absolute ordering to the tables & the objects can be saved in that order
    - Use metadata
    - A defined order can help eliminate deadlock

Batch Update

Batch Update

- A Unit of Work can also perform batch updates
- This is where SQL commands are grouped into a single database call
    - can be more efficient than separate calls for each statement

When to Use It

When to Use It

- When you want to combine database calls
- Usually the best way handle your updates from an efficiency

A Practical Matter

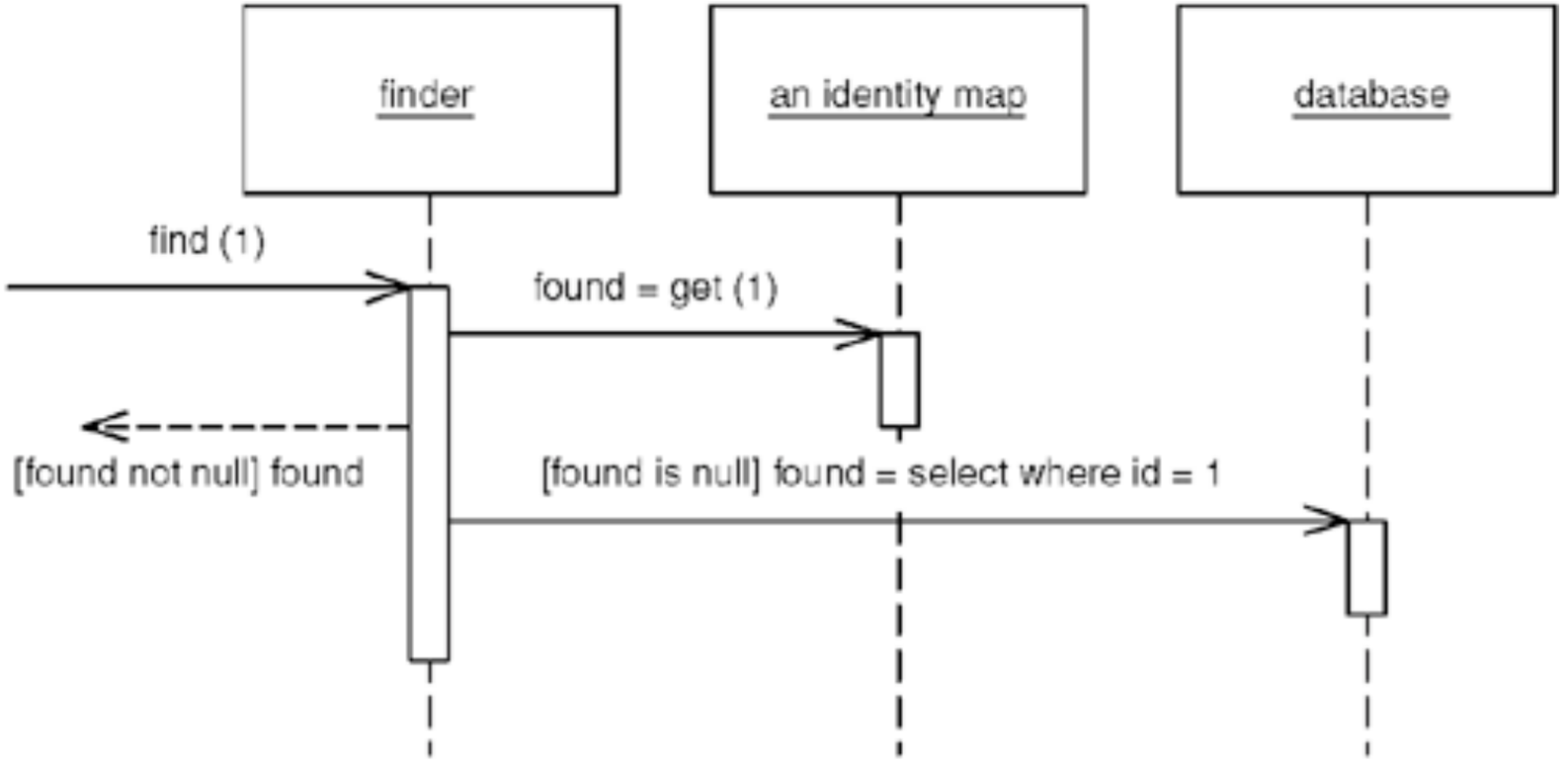
A Practical Matter

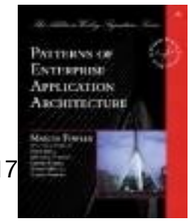
- This always sounds nice in theory to me
- I’ve never seen it used in a practical system
- Doesn’t mesh well with certain data source layers

Example

Identity Map

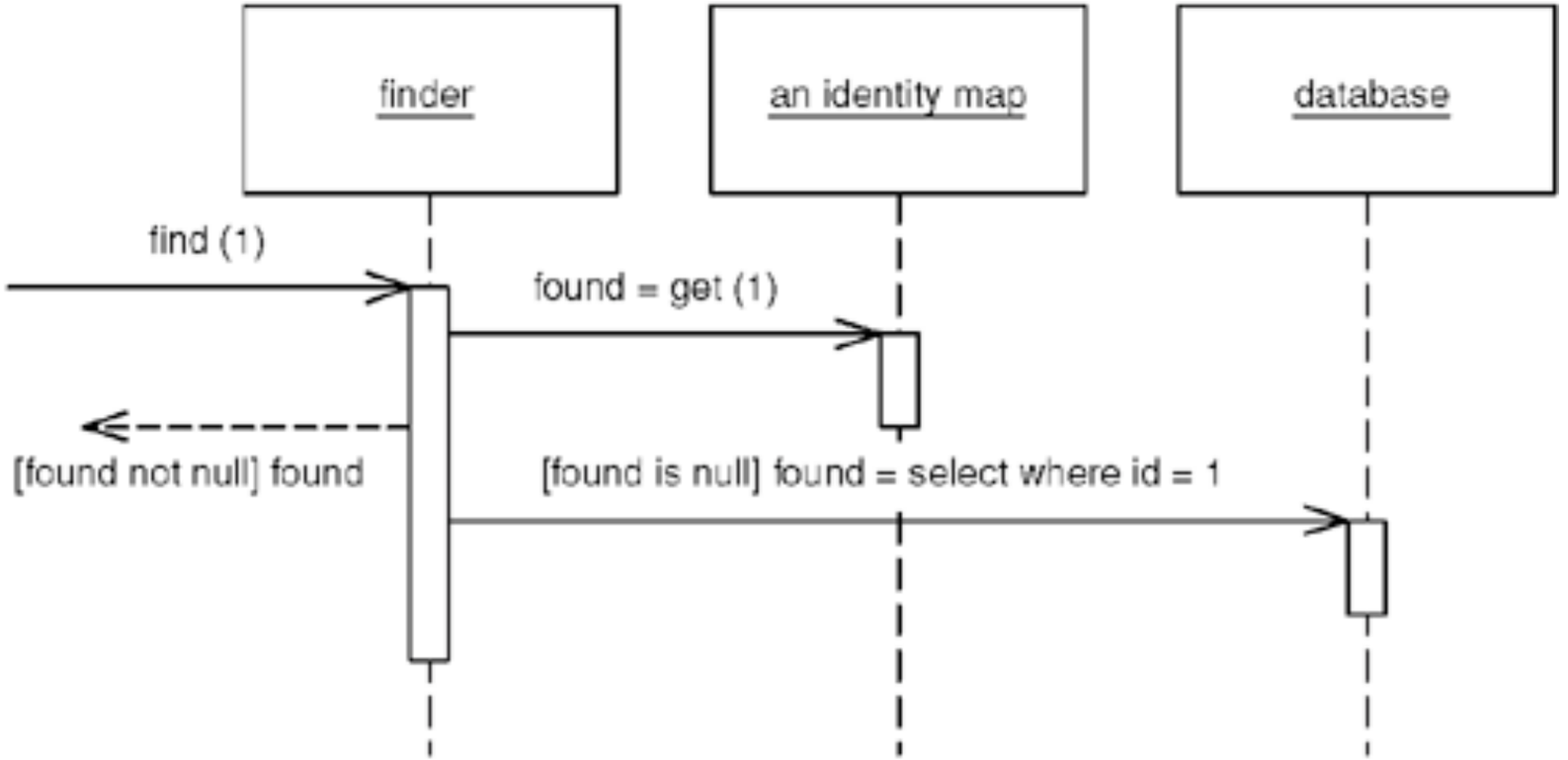
Identity Map

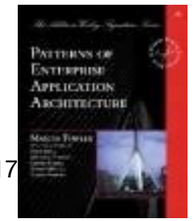




Identity Map

- “Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them.”





Identity Map

Identity Map

- Will make sure that you never load the same object from the database into memory twice
    - Can help performance
    - Cuts down on confusion (more important)

How It Works

How It Works

- Some kind of map structure to be the authoritative in-memory data source
- Can have one map per table or centralized maps, or whatever works for your schema

Keys

Keys

- You must choose a uniform key so that the identity map will function correctly
- Often a uniform ID column will work well

Explicit or Generic

Explicit or Generic

- You have to decide whether to use an explicit data structure or a generic data structure
    - Generic - need to cast
    - Explicit - no casing
        - This is much easier to do through Java Generics - i.e. specialize your data structure

How Many

How Many

- One map per class
- One map for everything
- My thought
  - use a map of maps
  - or - if you’re using data mapper
    - Maybe you can put one in each mapper
    - or one in each active record

Where

Where

- The Identity Map has to be easy to access from multiple locations
- Identity Maps should be isolated on a per-session (or even per-request) basis
    - We can get creative though - may some data is read-only and can be shared across all sessions

Where

Where

- From a practical standpoint
  - One Identity Map per transaction /request is probably the easiest to deal with
  - You can be sure to flush the identity map when appropriate

When to Use

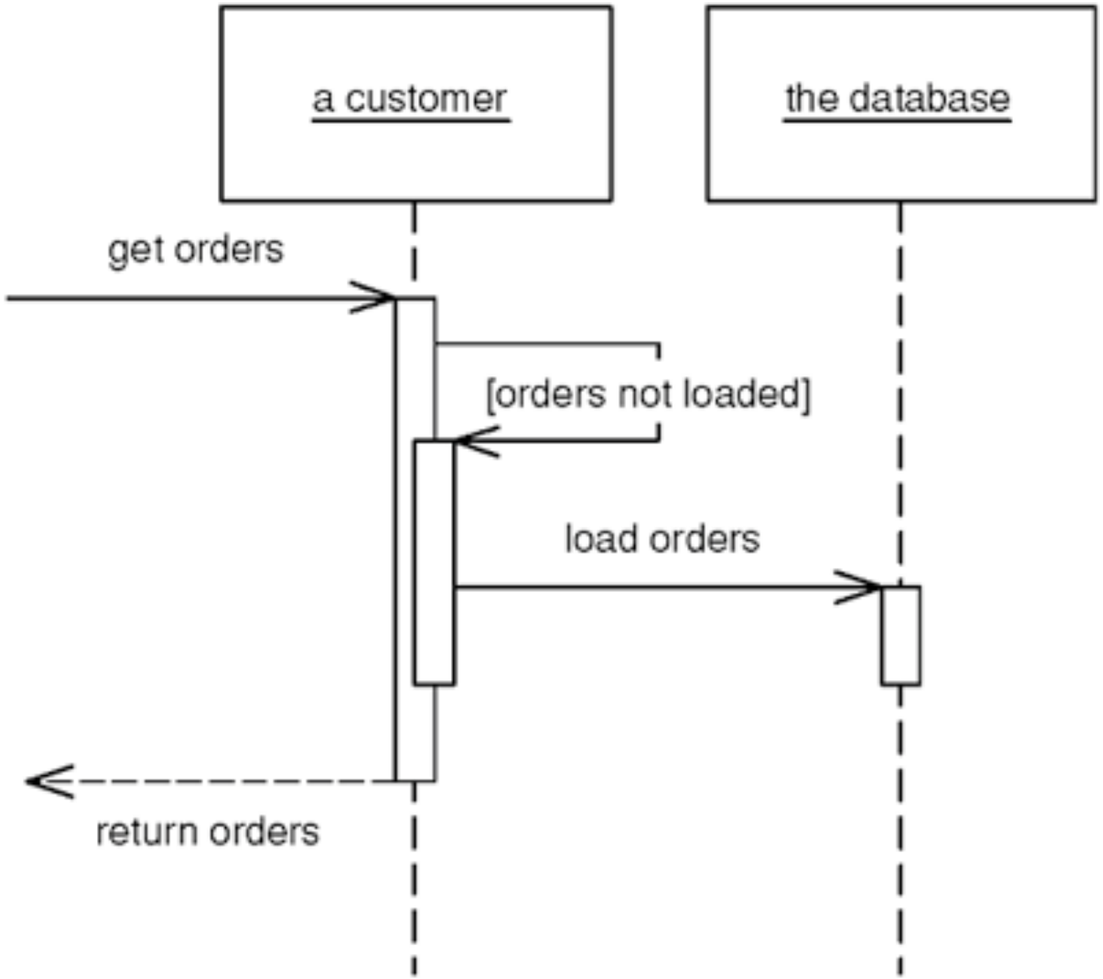
When to Use

- When you want to be certain that you don’t double load information from the database
- Might occur when the same record is read in multiple places during a transaction

Example

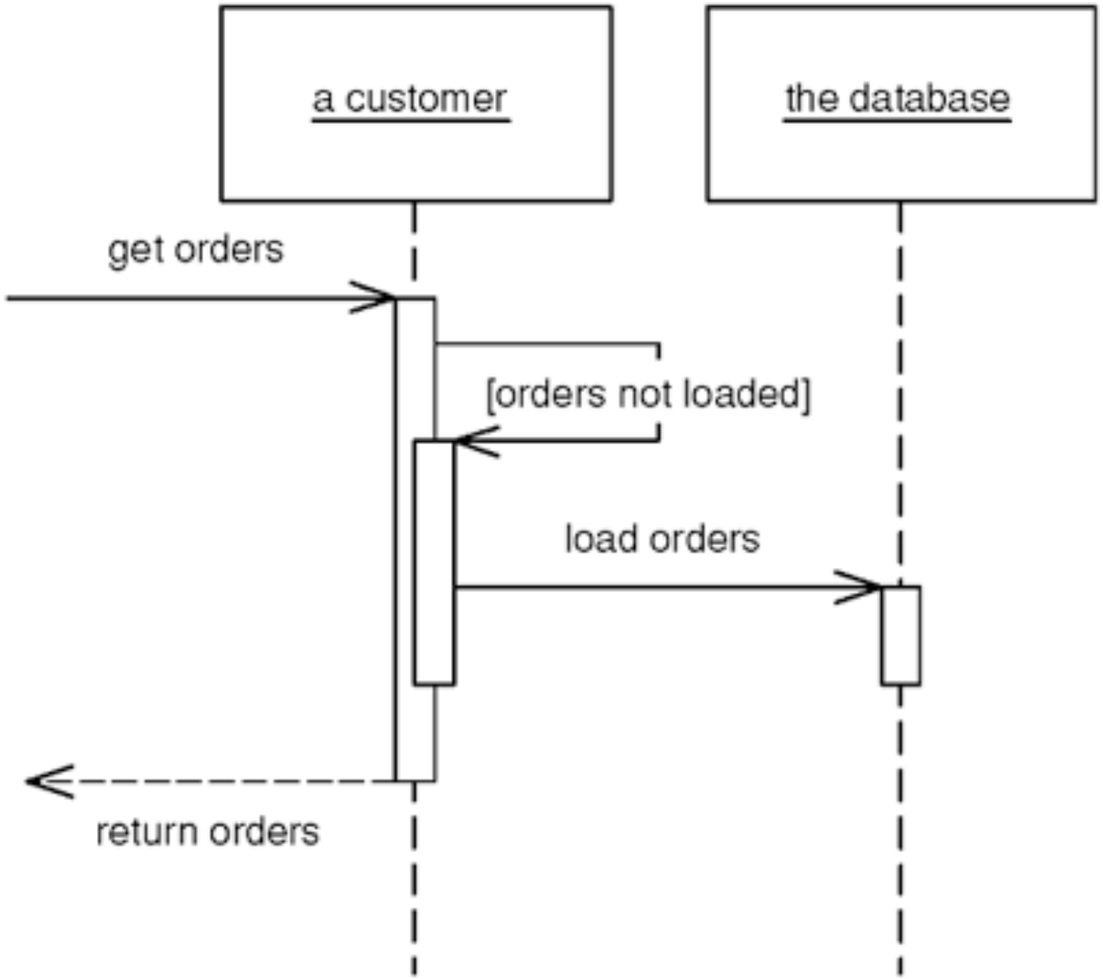
Lazy Load

Lazy Load



Lazy Load

- “An object that doesn’t contain all of the dat you need but knows how to get it.”



Lazy Load

Lazy Load

- The most obvious thing to do is to load and object and all related data into memory at the same time
    - A customer and all of their transactions
    - Over time - this can lead to problems as the customer builds up more and more transactions
    - or sometimes we just want the address of the customer and don’t need the transactions

How it Works

How it Works

- 4 was to implement (from the book)
    - lazy initialization
    - virtual proxy
    - value holder
    - ghost

Lazy Initialization

Lazy Initialization

- Easiest to implement
- Each getter method that requires this must
  - first see if the field / collection / etc has been initialized
  - if not - initialize it
  - return the value

Lazy Initialization

Lazy Initialization

- Usually null will work for detailing fields that haven't been loaded
    - This doesn't work if null is a valid value

Lazy Initialization

Lazy Initialization

- Can cause a dependency between the objects and the database
- Works best with Active Record, Table Data Gateway, and Row Data Gateway
- Can be made to work with Data Mapper
    - but works better with virtual proxy

Virtual Proxy

Virtual Proxy

- GOF Book - Page 207
- An object that has the same interface as the object, but doesn’t actually hold the data
- Both will implement the same interface

Virtual Proxy

Virtual Proxy

- Good
  - Looks identical to the real object
  - Easy to program
- Bad
  - There is an identity problem - 2 in memory objects representing the same object

Value Holder

Value Holder

- An object that actually hold the real value
- But it only loads the value on the first access

Ghost

Ghost

- The object is created containing just the ID field
- When any other field is loaded
    - The rest of the object is loaded

Collections

Collections

- For performance reasons
  - You will usually want to load an entire dependent collection of objects at the same time
  - Rather than loading the individual members as they are accessed

When to Use It

When to Use It

- Consider using lazy load if data would already require a second call to read the data
- Might as well delay the call in the event that the data ends up not being needed

UNIVERSITY OF Cincinnati

Object-Relational Structural Patterns
(Patterns of Enterprise Application Architecture - Chapter 12)

Mike Helmick
Large Scale Software Engineering - Spring 2014
University of Cincinnati

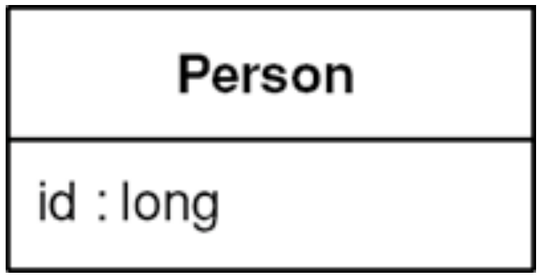
Patterns (Ch 12)

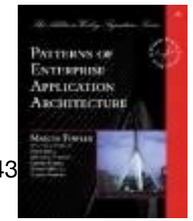
Patterns (Ch 12)

- Identity Field
- Foreign Key Mapping
- Association Table Mapping
- Dependent Mapping
- Embedded Value
- Serialized LOB
- Single Table Inheritance
- Class Table Inheritance
- Concrete Table Inheritance
- Inheritance Mappers

Identity Field

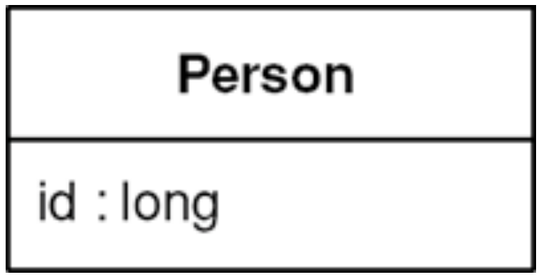
Identity Field

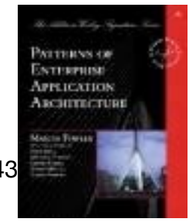




Identity Field

- “Saves a database ID field in an object to maintain identity between an in-memory object and a database row.”





Identity Field

Identity Field

- Database table require a primary key on every record to uniquely identify the records
- In memory objects do not have this requirement
- Identity Field bridges this gap

How It Works

How It Works

- The overall idea is very simple
    - We designate a key and use it to find rows by identity
    - Several complications come up

Key

Key

- Choosing the appropriate key is a difficult task
- Should we use naturally occurring keys or generated keys
  - Natural: A Student's SSN
  - Generated: A numeric ordering or hashed value

Natural Keys

Natural Keys

- Usually sound like a good idea
- Just doesn't work out that way all of the time
  - Mistyped SSN
  - Address (gets reused after a period of time)
  - Phone Number (same)

Key Width

Key Width

- A simple key is a single field that uniquely identified a record
- A compound key is when 2 or more fields are required to uniquely identify each record
  - Compound keys are more difficult to deal with
  - Can be avoided with generated keys
  - But they can be necessary with many-to-many relationships

Inheritance

Inheritance

- When using Concrete Table Inheritance or Class Table Inheritance
    - You must make sure the key is unique across several tables (the object hierarchy)

Identity Field

Identity Field

- In its simplest form it is a field that matches the type and name of the key on the database
    - private long id;

New Keys

New Keys

- Needed on all inserts / object creations
    - Auto-generate (From database)
    - Use GUID
    - Generate your own (possibly from database)

Auto-Generate

Auto-Generate

- Sounds easy
    - but it is database dependent
    - it is hard to get the generated value back right away

GUID

GUID

- Globally Unique IDentifier
  - A unique identifier that is guaranteed to be unique across all machines and times
  - Generates big keys (large numerical values) - which can lead to performance and storage problems

Roll your own

Roll your own

- Use the SQL max function to find the highest
    - can severely impact performance
    - can lead to contention & race conditions
- Better approach
    - Use a key table

Key Table

Key Table

- A table with 2 fields, table name, last ID
    - Before each insert, read the last ID, add 1, and update
    - Best to have this in a separate transaction so as not to lock the key table
- Best choice for portability and efficiency

When to Use It

When to Use It

- When there is a direct mapping between domain model objects and database rows
    - Domain Model or Row Data Gateway
- Can assists with use of an Identity Map

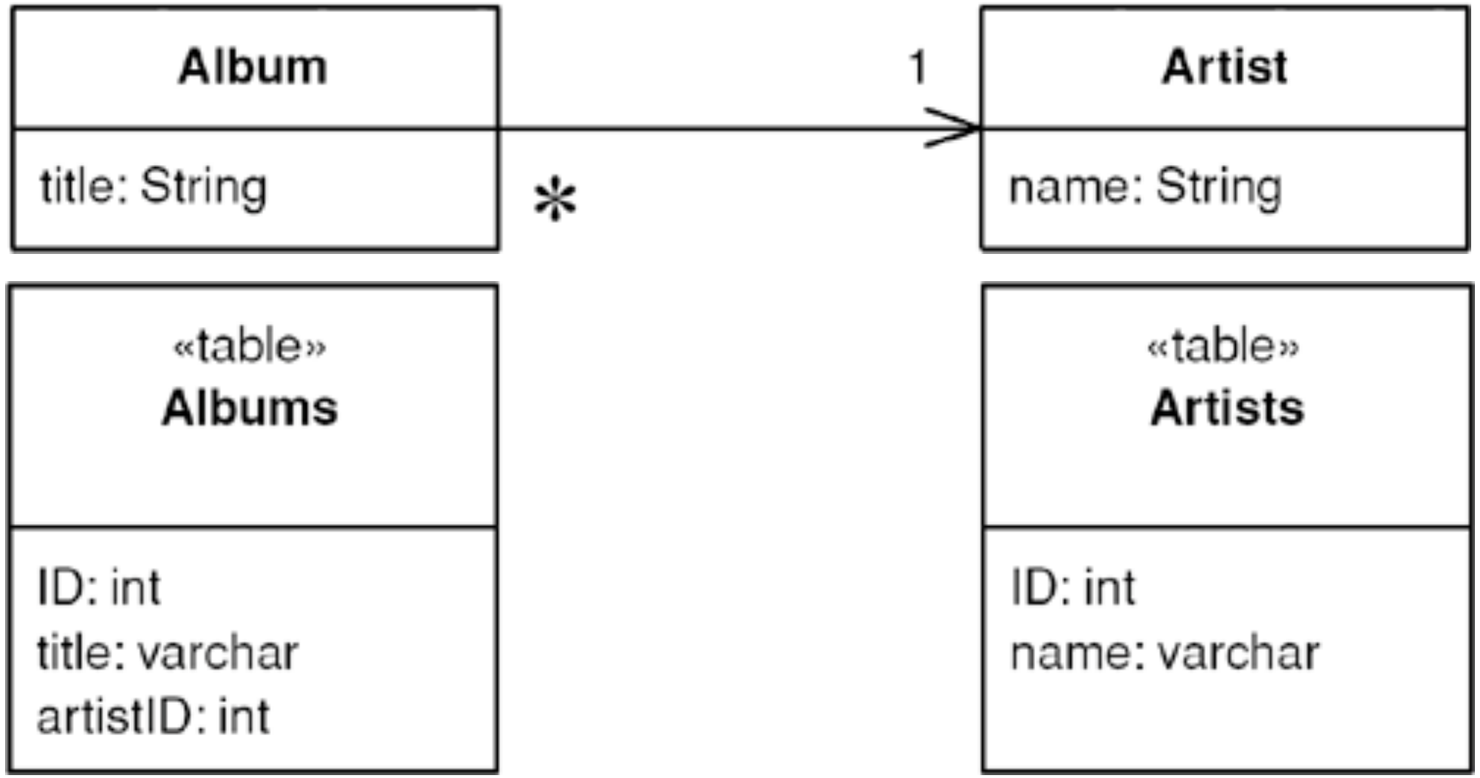
Example

Example

See pages 224 - 234 for an in-depth example using compound keys.

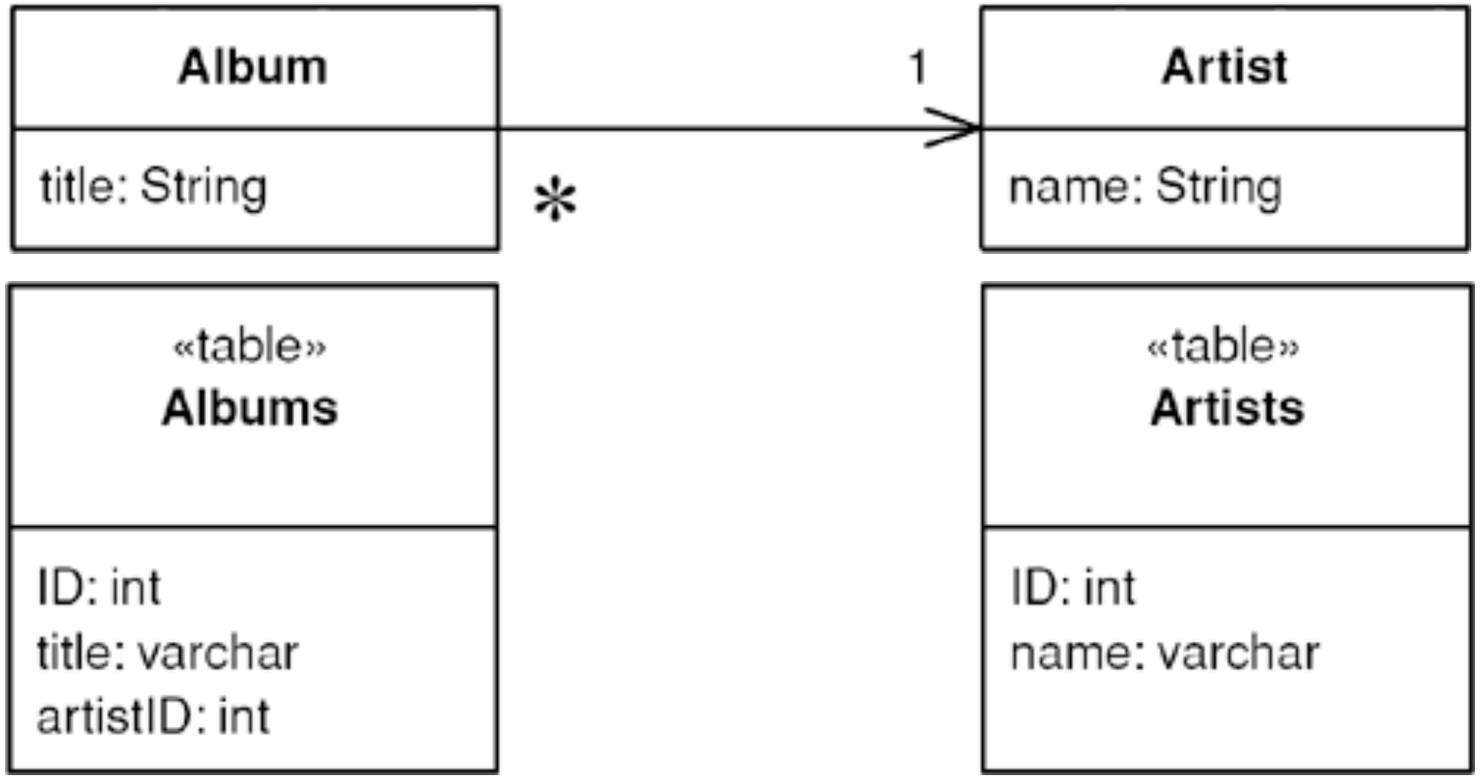
Foreign Key Mapping

Foreign Key Mapping



Foreign Key Mapping

- “Maps an association between objects to a foreign key reference between tables.”



Foreign Key Mapping

Foreign Key Mapping

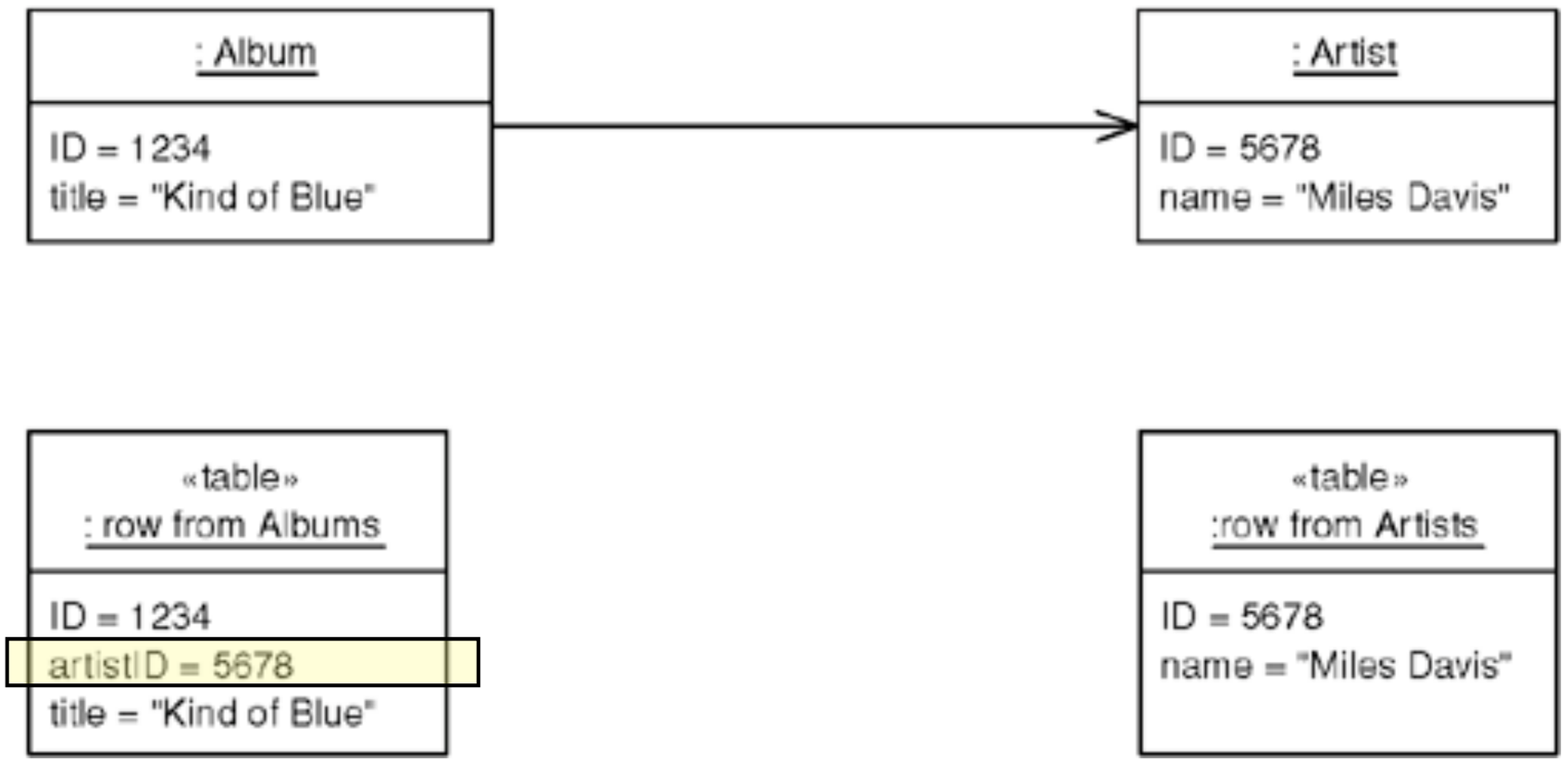
- In memory - objects refer to each other seamlessly
    - Object references
- When we save/load to/from a database
    - Object references must be mapped to foreign key identities

How It Works

How It Works

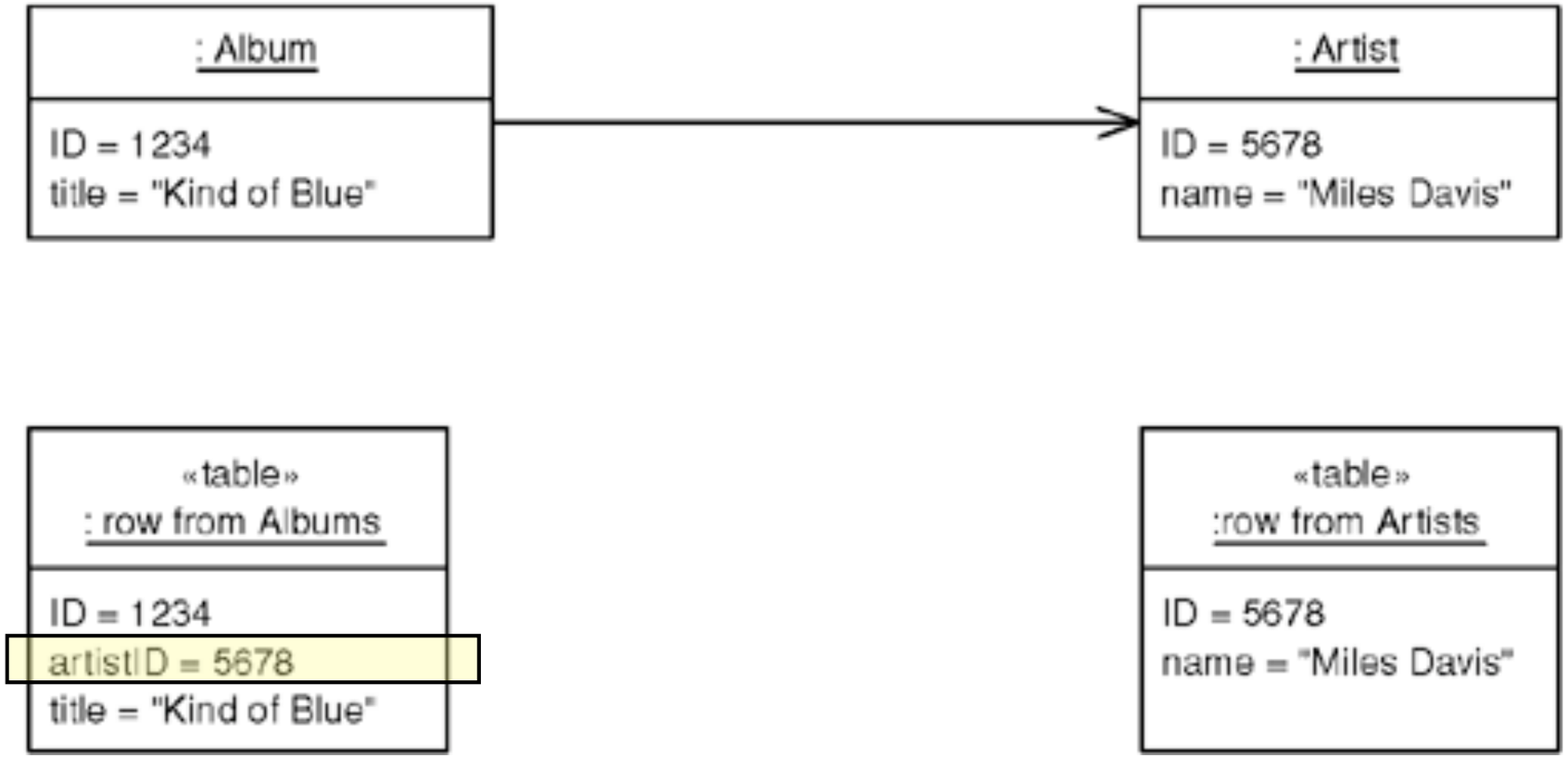
- Each record has an identity field of its own
- We must use the identity fields to link to the owning record / owned records
    - Usually the owning record

1 to 1

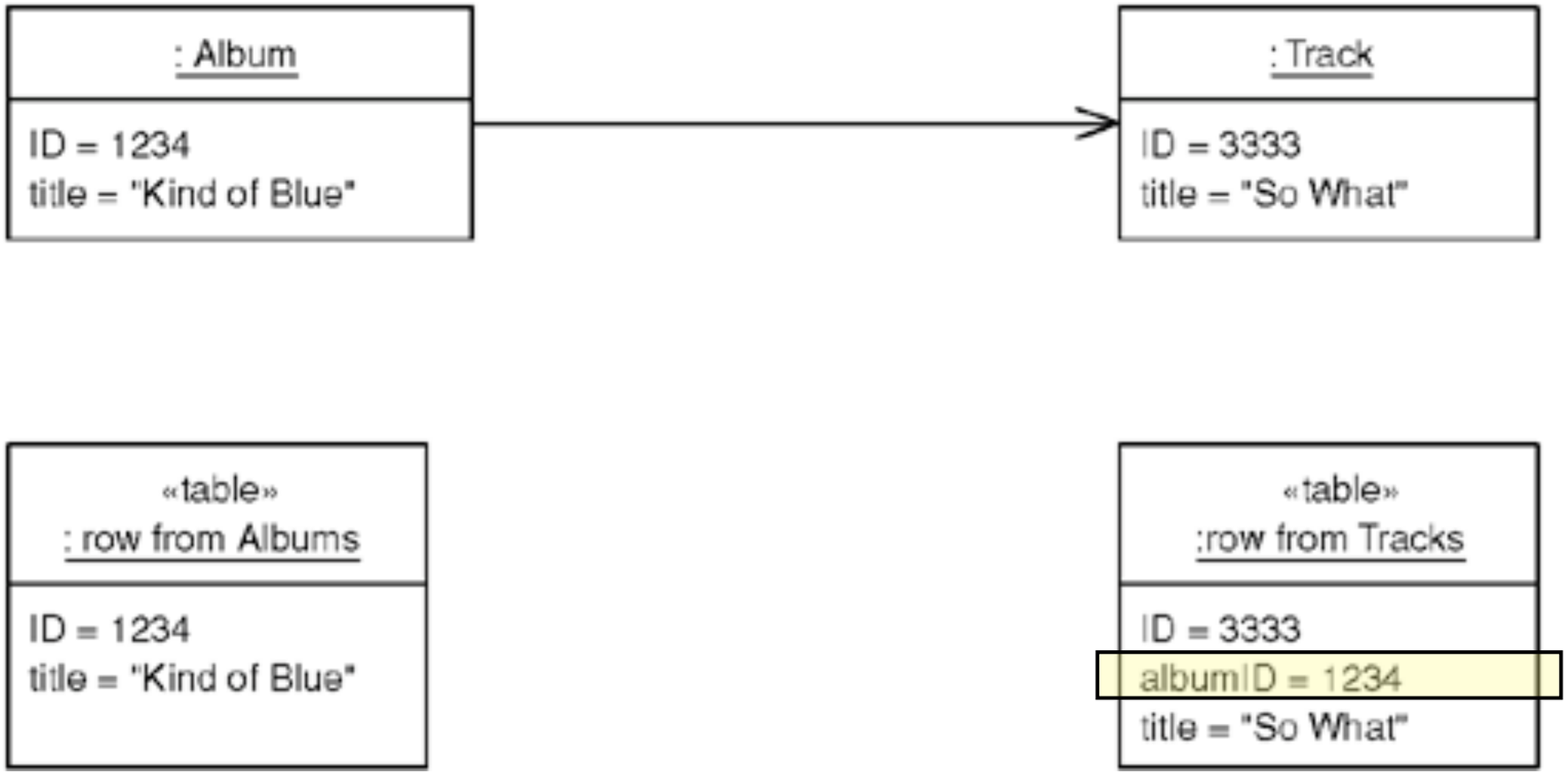


1 to 1

- With the 1 to 1 case the parent record can have the ID field of the child record

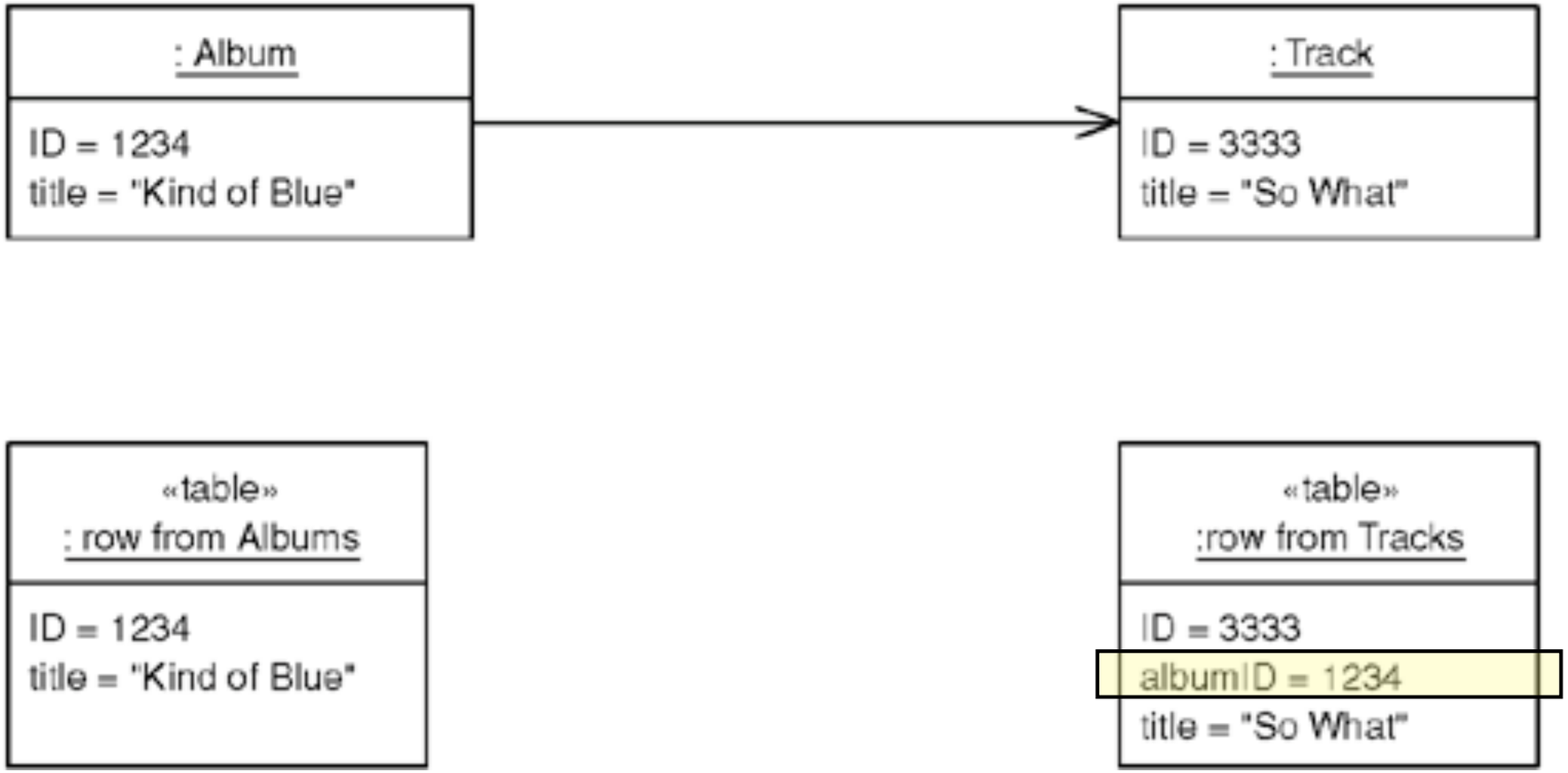


Collection



Collection

- With a collection (1 to many) we must put the ownership on the owned record



Updating

Updating

- Dependent items can be inserted / updated / removed
    - commits can get complicated

Updating

Updating

- “delete & insert”
    - delete all dependent record on the database
    - insert all the dependent records in memory
- Easy to code
- Not so great for performance reasons

Updating

Updating

- back pointers
  - give dependent records a reference back to the owning object
  - this makes it easier to set the ID field on the owned objects

Updating

Updating

- “Diff”
  - It is possible to compare the current state of the database to the list in memory
    - or copy when it is first read

Updating

Updating

- My recommendation
  - Use back-pointers
  - Save all the objects as if they exist on their own

Ownership

Ownership

- If we allow dependent records to be “orphaned”
    - Then code needs to be built to to handle knowing if a record is deleted (Because its back-pointer is nulled out) or just doesn’t have an owner

Cycles

Cycles

- Links between objects are great
- When loading objects
  - be careful not to follow cycles in order to load the objects
  - Lazy load usually takes care of this since fields aren't loaded until accessed

When to Use It

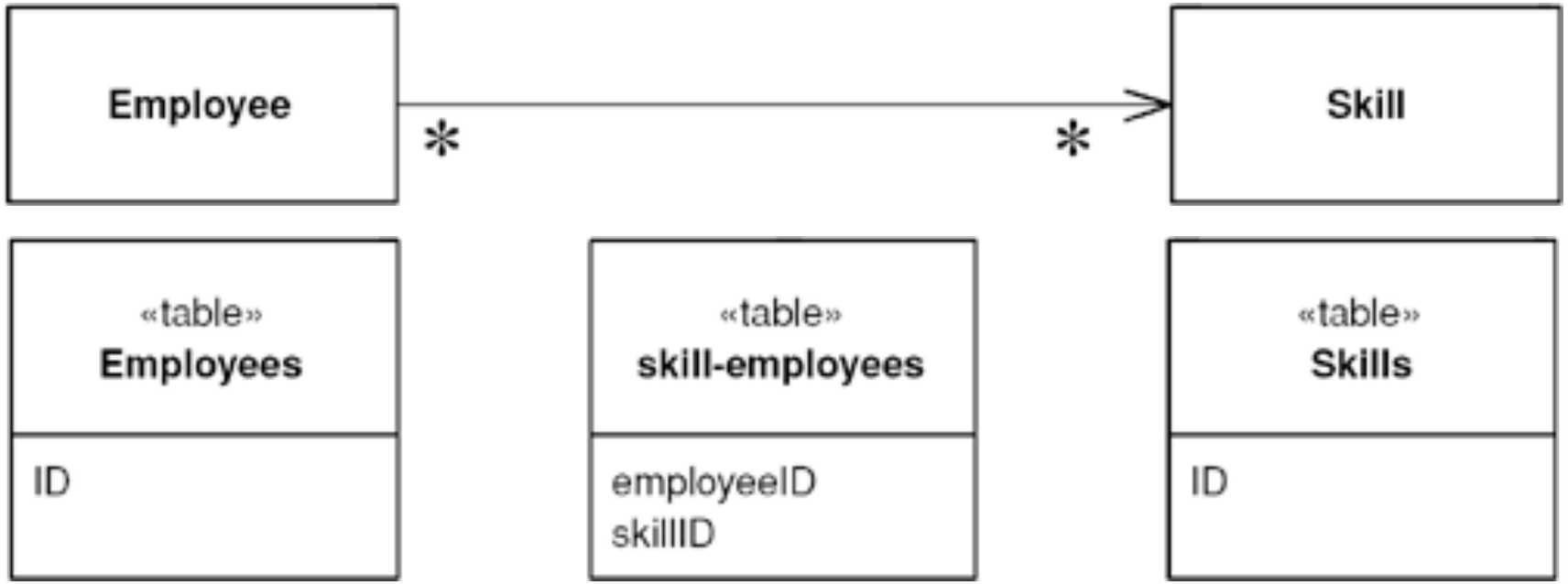
When to Use It

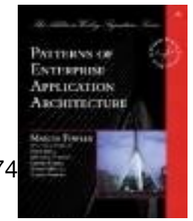
- Can be used for almost all associations
    - many-to-many is the one exception

Example

Association Table Mapping

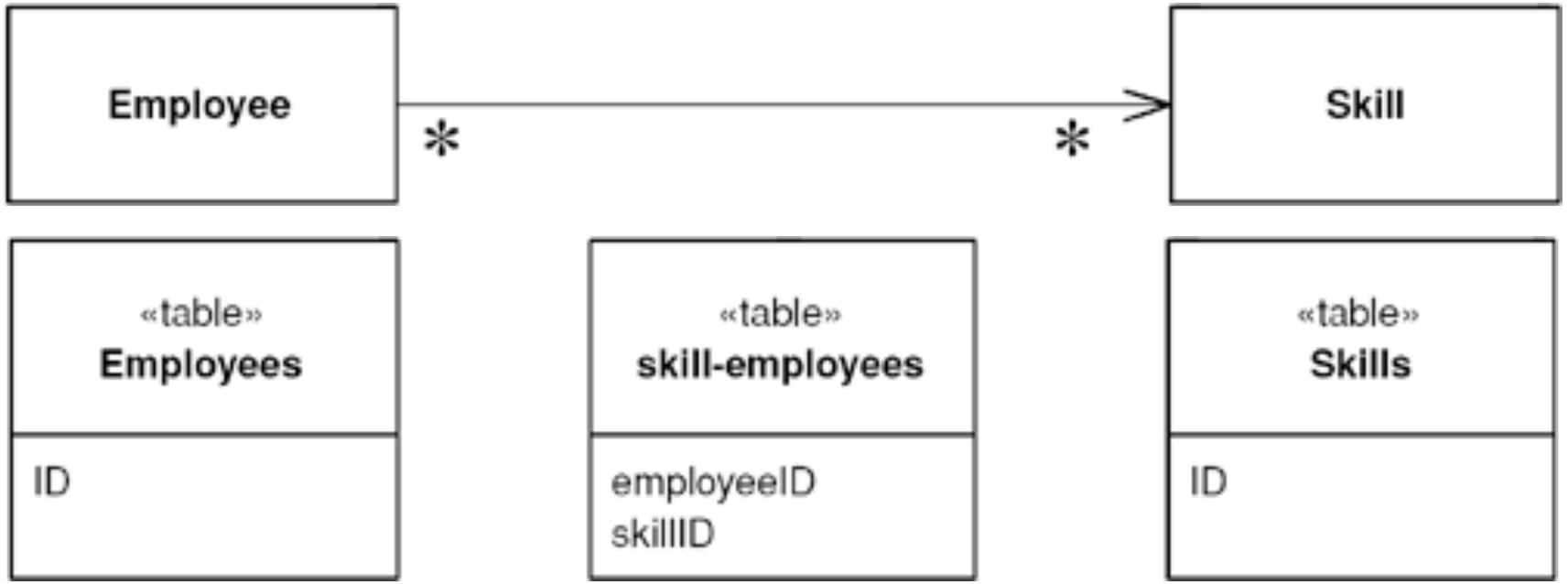
Association Table Mapping

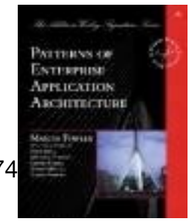




Association Table Mapping

- “Saves an association as a table with foreign keys to the tables that are linked by the association.”





Many to Many

Many to Many

- This is the many to many association
- Allows for reuse of common data
  - category, skills, zip codes
  - Anything that multiple objects might be associated to and associated with
- Solved with an extra table

How It Works

How It Works

- Use a third ‘link table’ to store the association
- The table will typically only have the two foreign keys
    - You can store data specific to the association

How It Works

How It Works

- The link table will not normally have a corresponding data model object
- Can be loaded with
  - two queries (can be hard to coordinate)
  - using a join

When to Use It

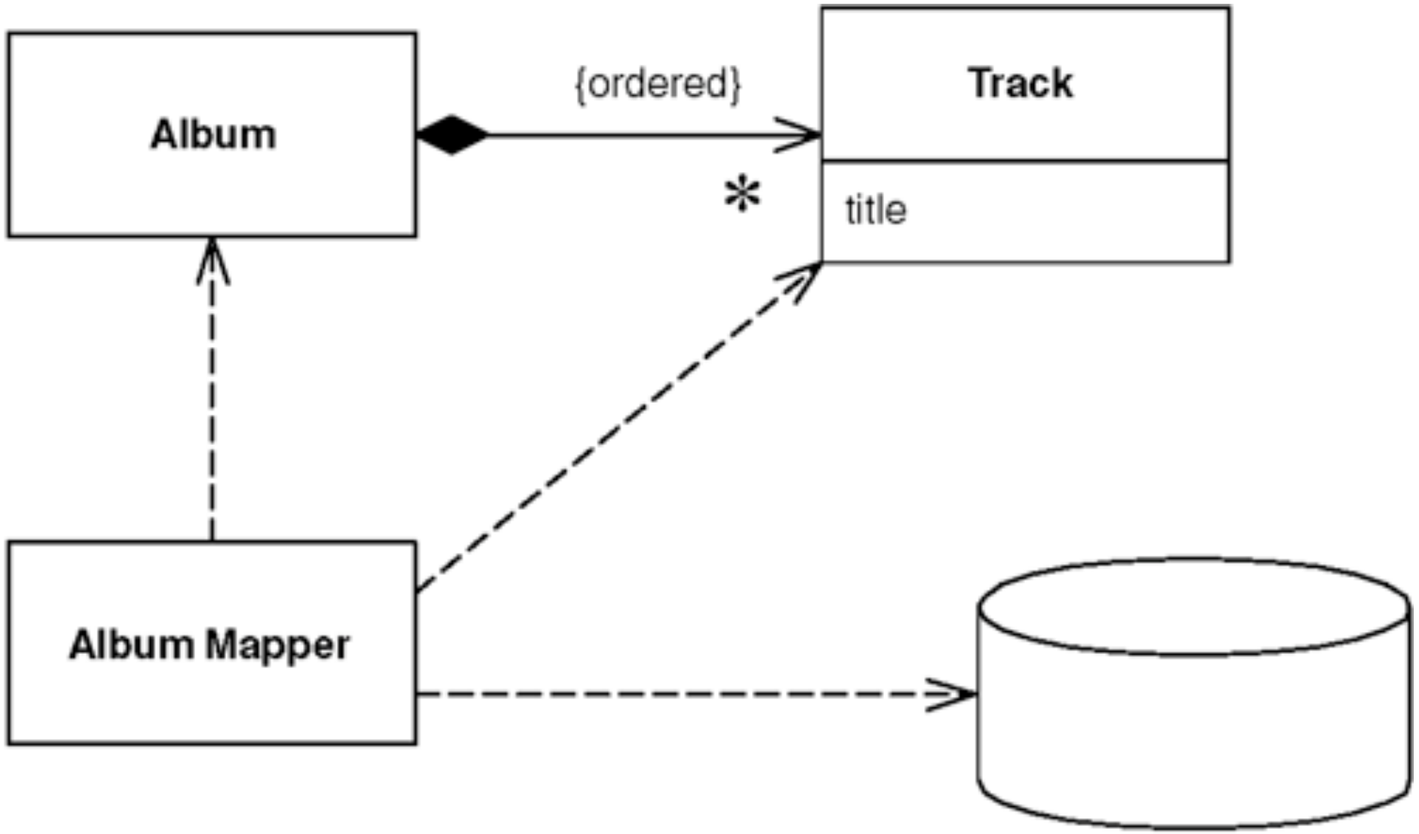
When to Use It

- When you have a many-to-many relationship to map
- When you have extra information to store
    - works fine, but you need the corresponding domain object

Example
(from book)

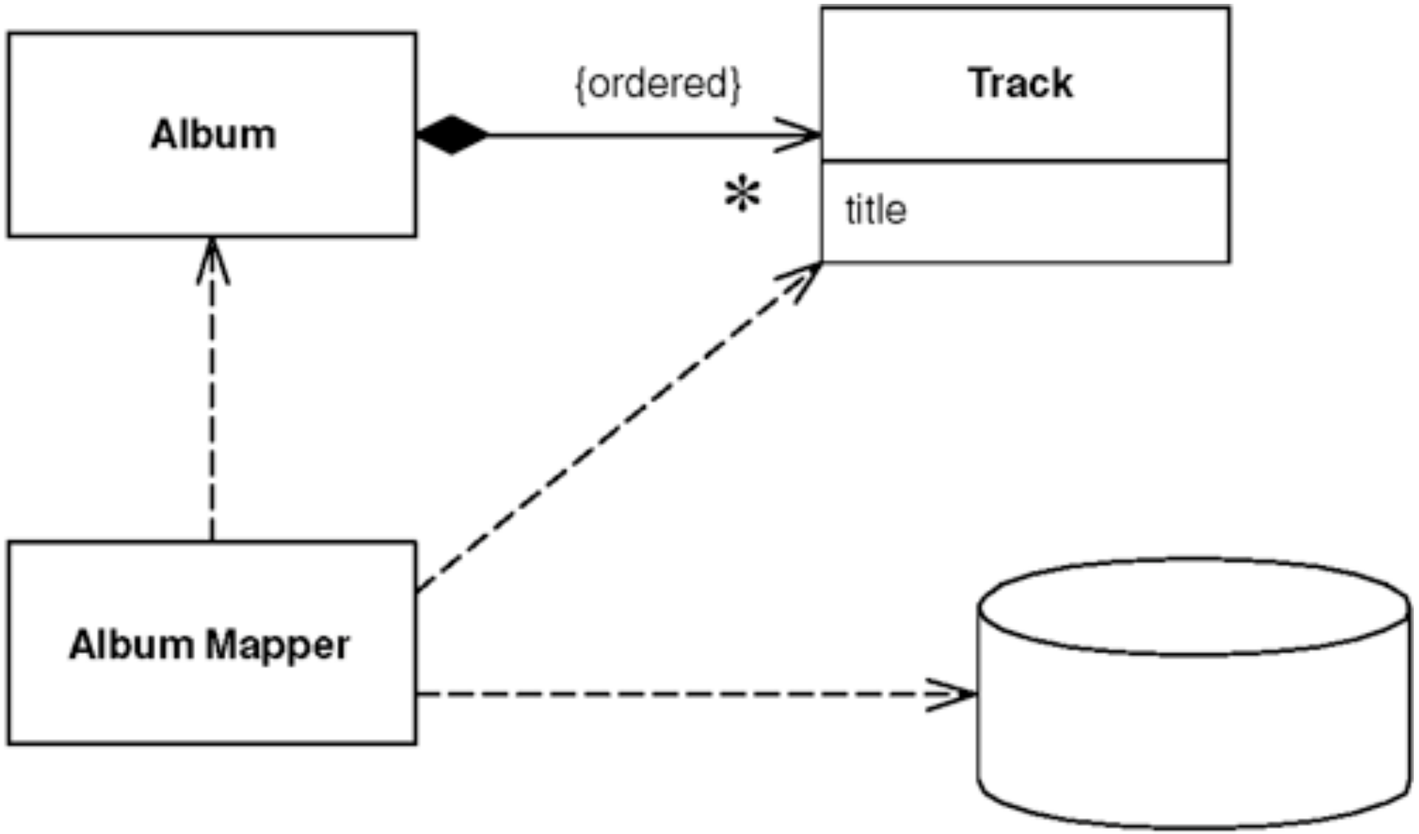
Dependent Mapping

Dependent Mapping



Dependent Mapping

- “Has one class perform the database mapping for a child class.”



Dependent Mapping

Dependent Mapping

- For objects that are naturally part of their parent object
    - but they are a collection, so they have to be stored on a separate table

How It Works

How It Works

- A dependent class relies on its owner
    - Every dependent must have an owner
    - And can not be loaded unless it is in the context of its owner

How It Works

How It Works

- Active Record, Row Data Gateway
    - There is no DB logic in the domain object
- Data Mapper
    - There is no mapper for the dependent object
- The dependent object can be lazy loaded

Identity

Identity

- The dependent object does not have an identity field
- It will be identified by the same ID as its owning object

Changes

Changes

- Changes to the dependent object
    - require you to save the owning object even if it hasn’t changed on its own

When to Use It

When to Use It

- When you have an object (or collection) that only exists in the context of another object
- I like to use this when a record is large (say 30 fields) but not all of them are needed in all situations
    - Push to separate table & lazy load

Example (Book)

Embedded Value

Embedded Value

Embedded Value

- “Maps an object into several fields of another object’s table.”

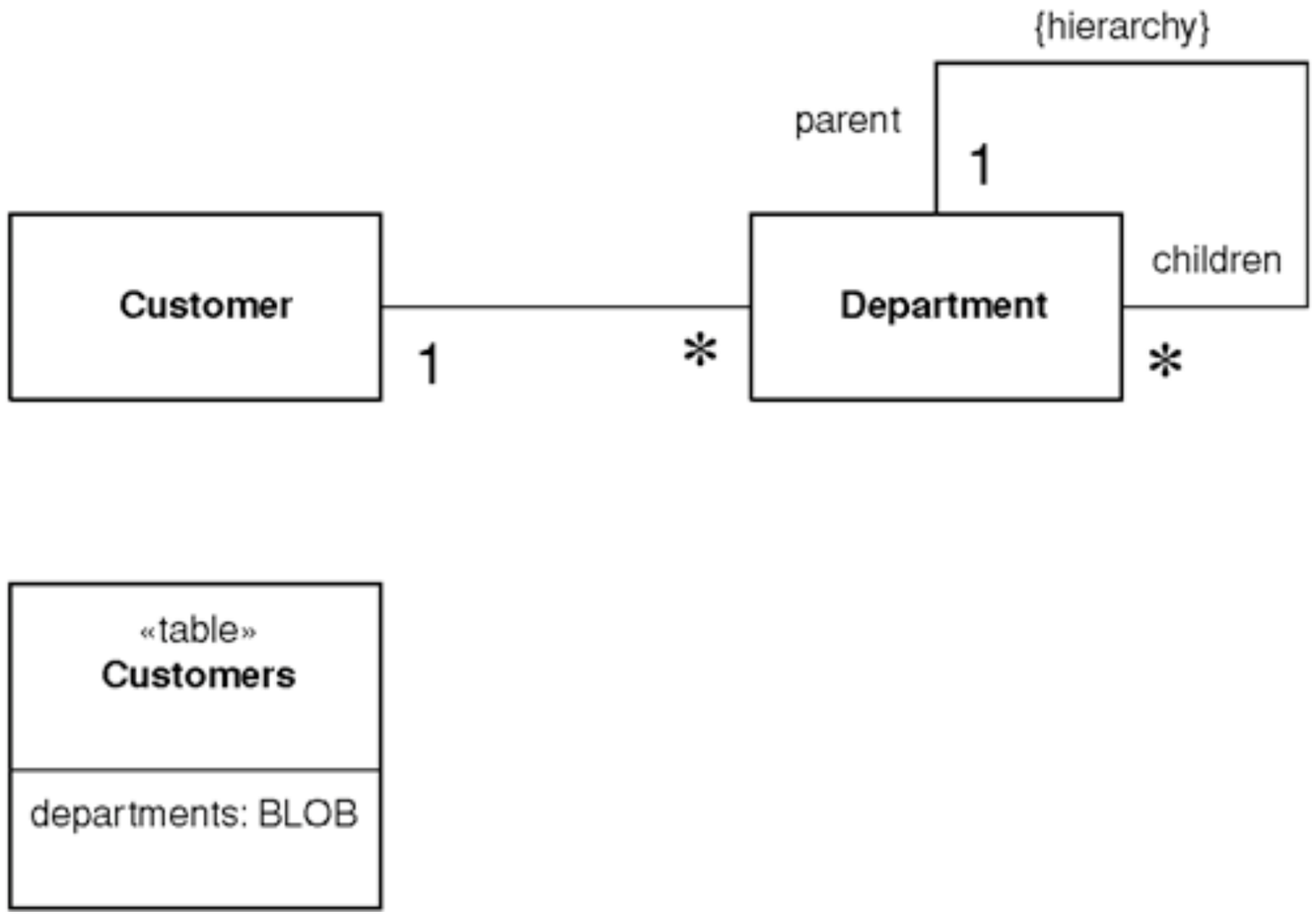
Embedded Value

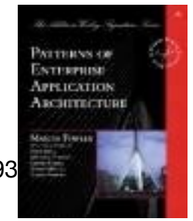
Embedded Value

- Really no different than Dependent Mapping except it is explicitly a single dependent object, rather than a collection

Serialized LOB

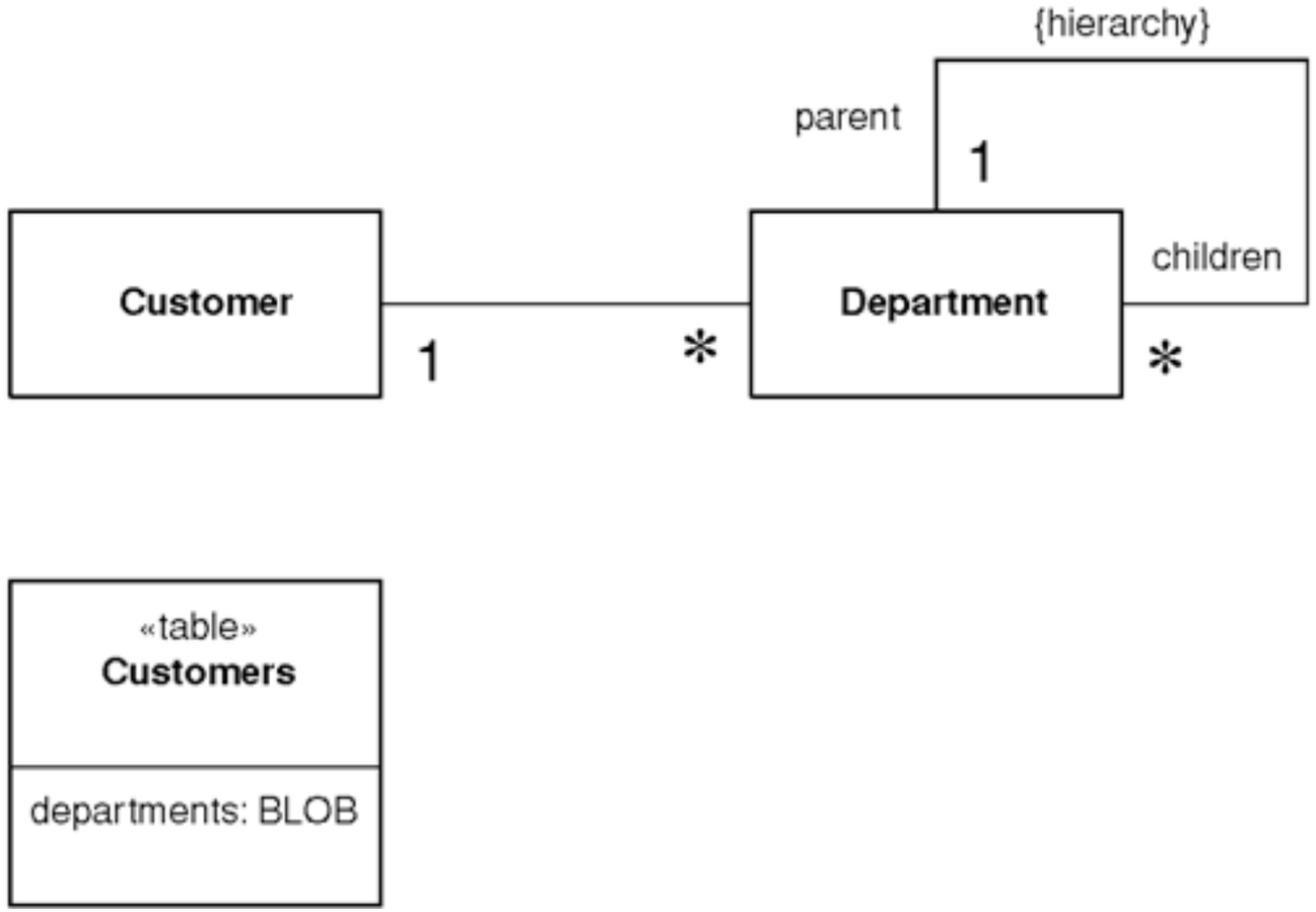
Serialized LOB

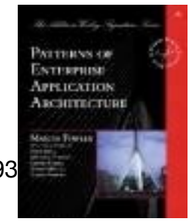




Serialized LOB

- “Saves a graph of objects by serializing them into a single large object (LOB), which it stores in a database field.





LOB

LOB

- Databases now have to “large object columns”
    - BLOB - Binary Large OBject
    - CLOB - Character Large OBject
- In HSQLDB
    - LONGVARBINARY, LONGVARCHAR

Serialized LOB

Serialized LOB

- Allows serialized objects to be persisted
    - i.e. the object doesn't have to be broken down into table / fields
    - Good for objects that you don't have to search
        - or they have a dynamic structure over time

How it Works

How it Works

- Two main types of serialization
  - Binary - BLOB
  - Text/XML - CLOB

Blob

Blob

- The Good:
  - Easy to program (Java binary serialization isn’t so bad)
- The Bad:
  - DB must support BLOBs
  - Can force lengthy and complicated conversions when the software is upgraded

CLOB

CLOB

- Can be platform independent
    - XML that can be written / read by multiple systems
    - Can be read by humans (DBAs, Developers)

CLOB

CLOB

- Downsides to XML
    - Parsing XML can be time consuming
    - XML also takes up much more space than the binary version of an object

Identity Issues

Identity Issues

- Since data in the LOB column has a singular ID (for the whole serialized list)
  - What happens if the same customer is saved in two different rows?
  - Is this a problem?
  - How do we identity the correct one?

Duplication

Duplication

- We can avoid this by limiting (or eliminating) duplication
    - Try to store data in the LOB column that pertains only to that particular record and doesn't overlap with another

When to Use It

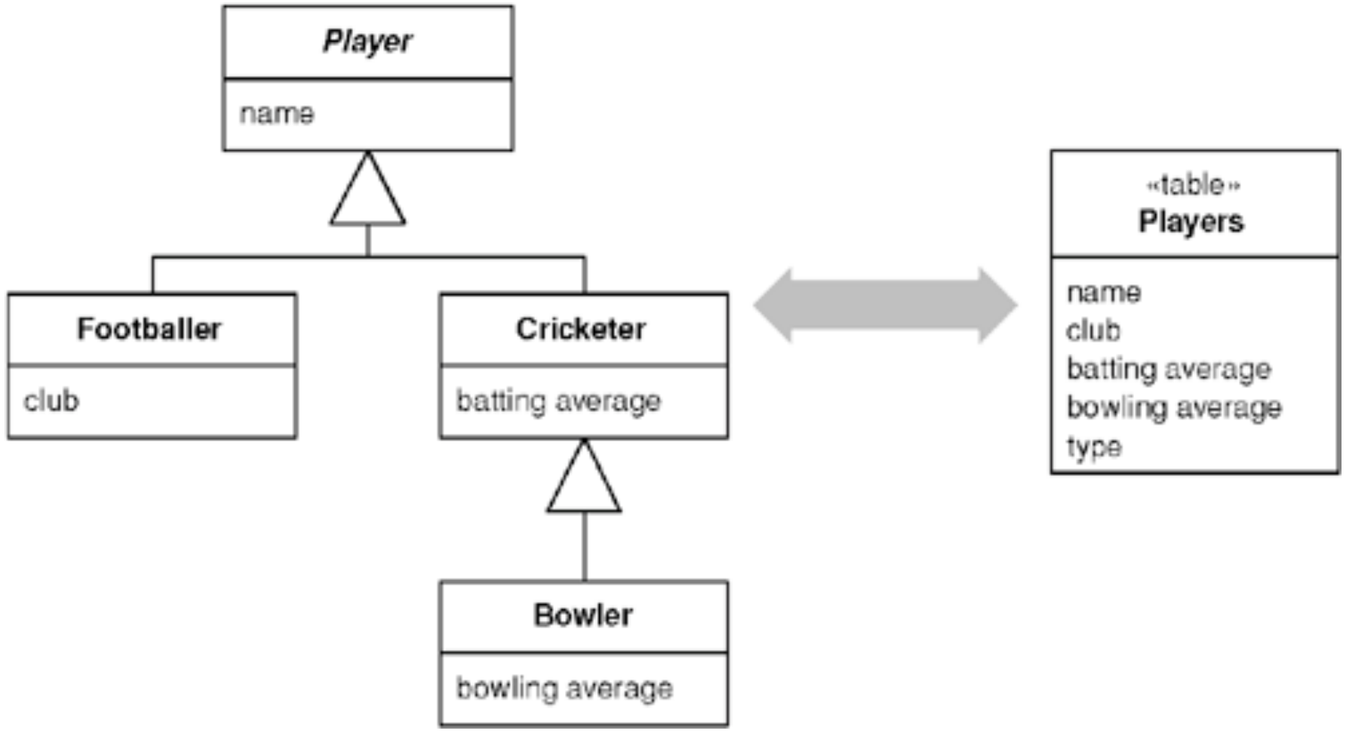
When to Use It

- A good way to store objects that will never be queried on
- When you can't alter the database structure
    - I've actually seen this

Example

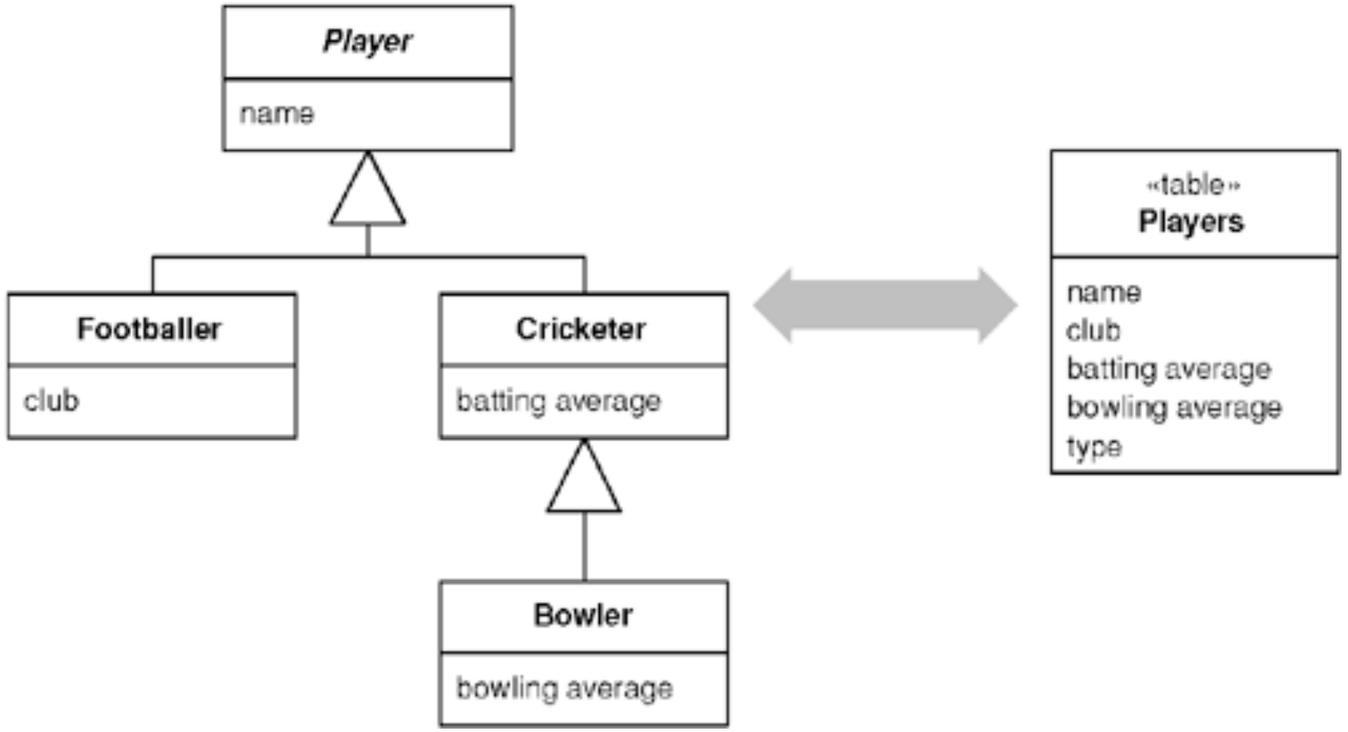
Single Table Inheritance

Single Table Inheritance



Single Table Inheritance

- “Represents an inheritance hierarchy f classes as a single table that has columns for all the fields of the various classes.”



Single Table Inheritance

Single Table Inheritance

- As we noted earlier - RDBMs don’t support inheritance
- In many instances, it is nice to have inheritance in our object model
    - The resolution of this is in the mapping (one of the next 4 patterns)

How It Works

How It Works

- One table contains all data
    - for all classes in the hierarchy
- Each object is stored in a single row
    - This leaves the possibility of empty columns in any given row

How It Works

How It Works

- As each row is read
    - Determine the correct concrete class to instantiate
    - Typically there will be a field indicating which type it is (type code or class name)

Naming

Naming

- Using the class name
  - creates a coupling between the DB & object model
  - can be used to instantiate the object (using reflection)

When to Use It

When to Use It

- The Good:
  - only a single database table
    - no joins
  - field level refactoring doesn’t force a database change

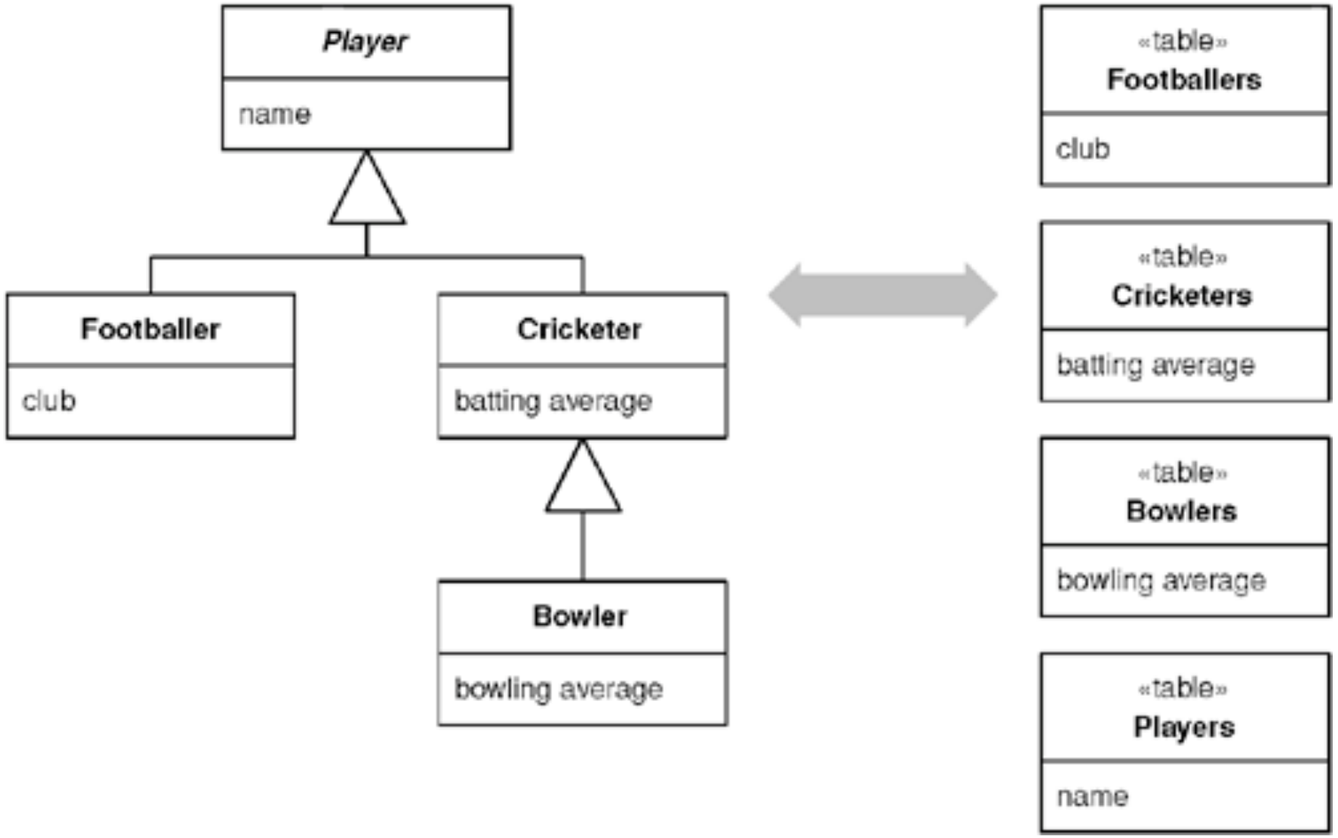
When to Use It

When to Use It

- The Bad:
  - seemingly random empty fields can be confusing
  - wasted space in database (most databases efficiently represent null values though)
  - the single table could be very large - leads to contention
  - single namespace
  - can’t use database to enforce required fields

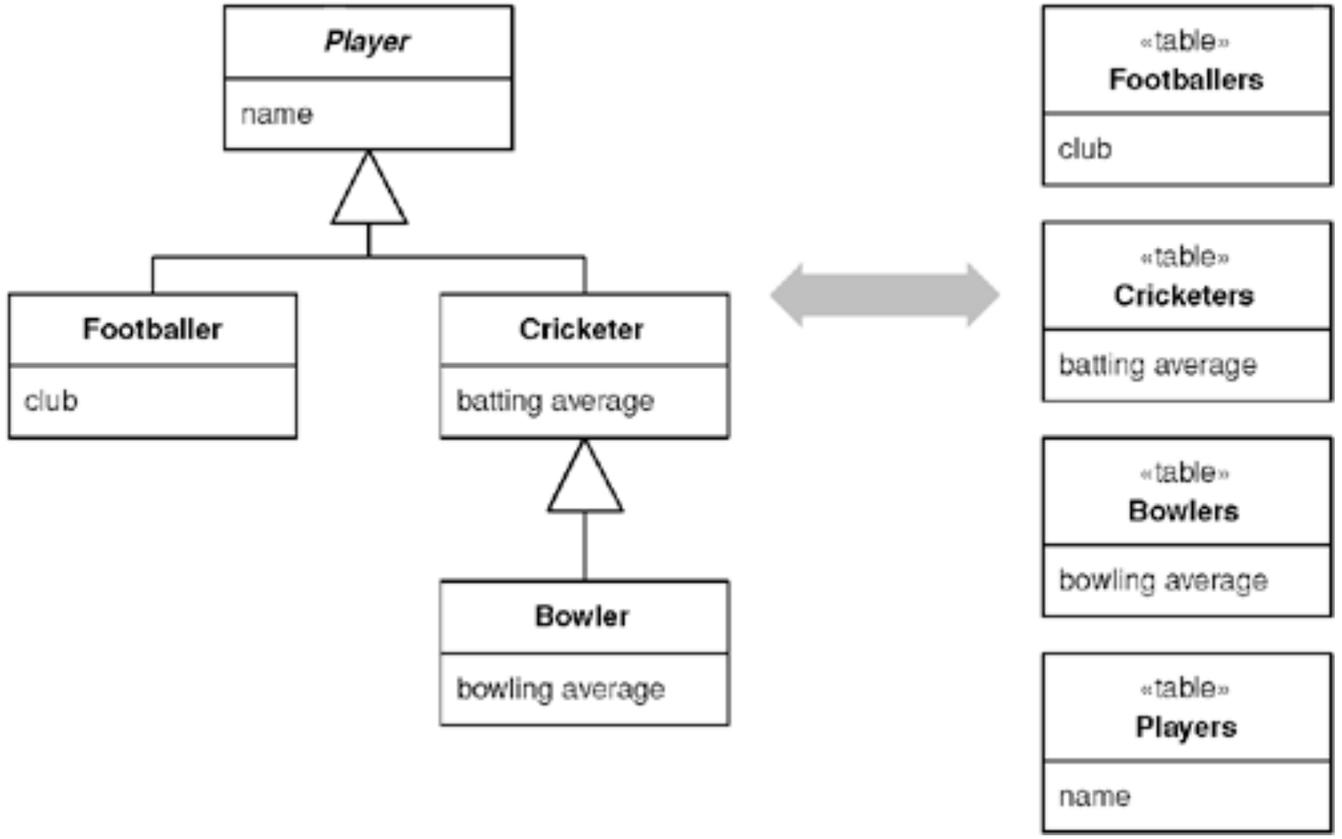
Class Table Inheritance

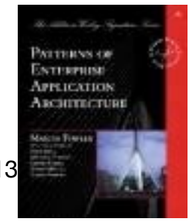
Class Table Inheritance



Class Table Inheritance

- “Represents an inheritance hierarchy of classes with one table for each class.”





Class Table Inheritance

Class Table Inheritance

- Supports one table for every class in the hierarchy
    - Including the abstract classes

How It Works

How It Works

- Easy to understand with the 1 to 1 correspondence of table/class column/field
- You have to know how to join the correct values in
  - Use a shared primary key
  - Each key must be unique across ALL tables (football / baseball player can’t both have an ID of 1)

How It Works

How It Works

- Need to do a complex join to get all of the data back
    - but - if the hierarchy is complex enough, you might not be able to join everything together in one call
- Plus - you don’t know where to start reading

When to Use It

When to Use It

- The Good:
    - All columns are meaningful on all tables
    - Easy to see the relationships between the object model and database tables

When to Use It

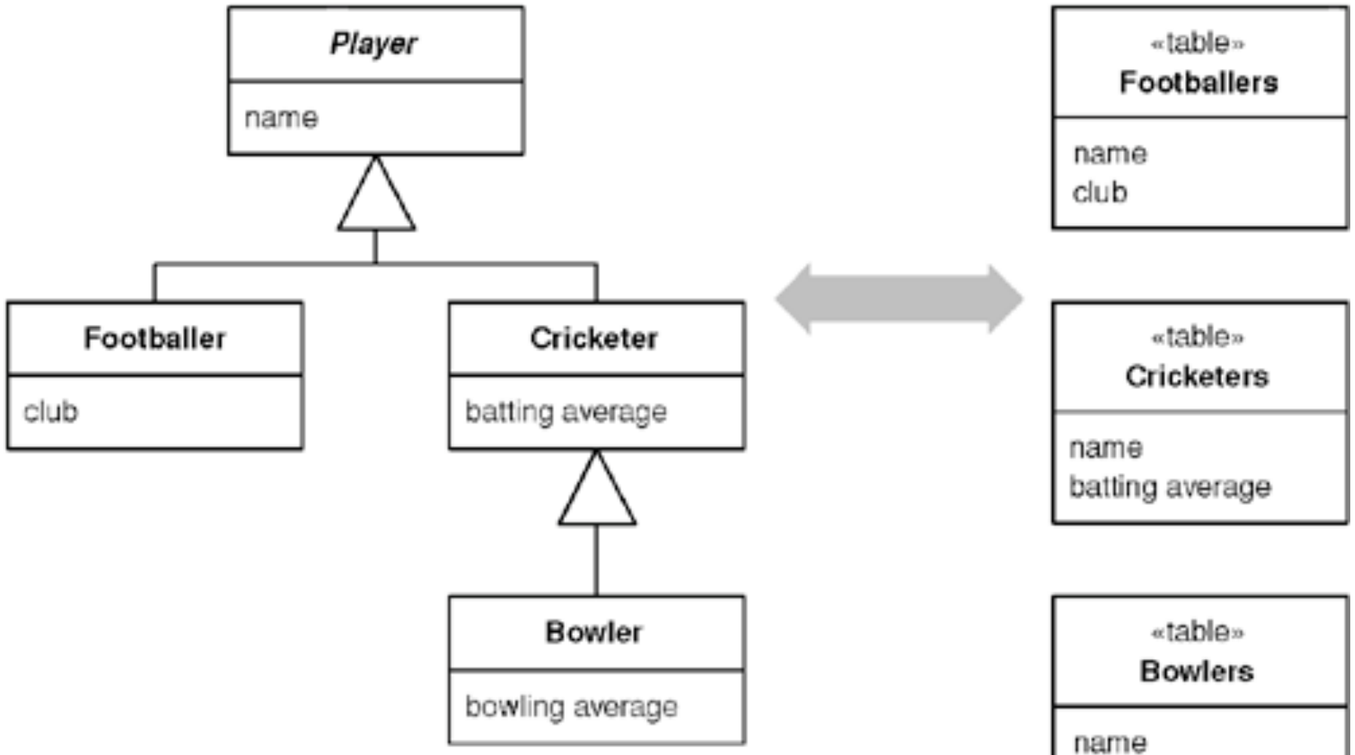
When to Use It

- The bad:
  - need to read from multiple tables (joins)
  - Refactoring of fields causes changes to multiple tables
  - Ad hoc queries become difficult to write

Example

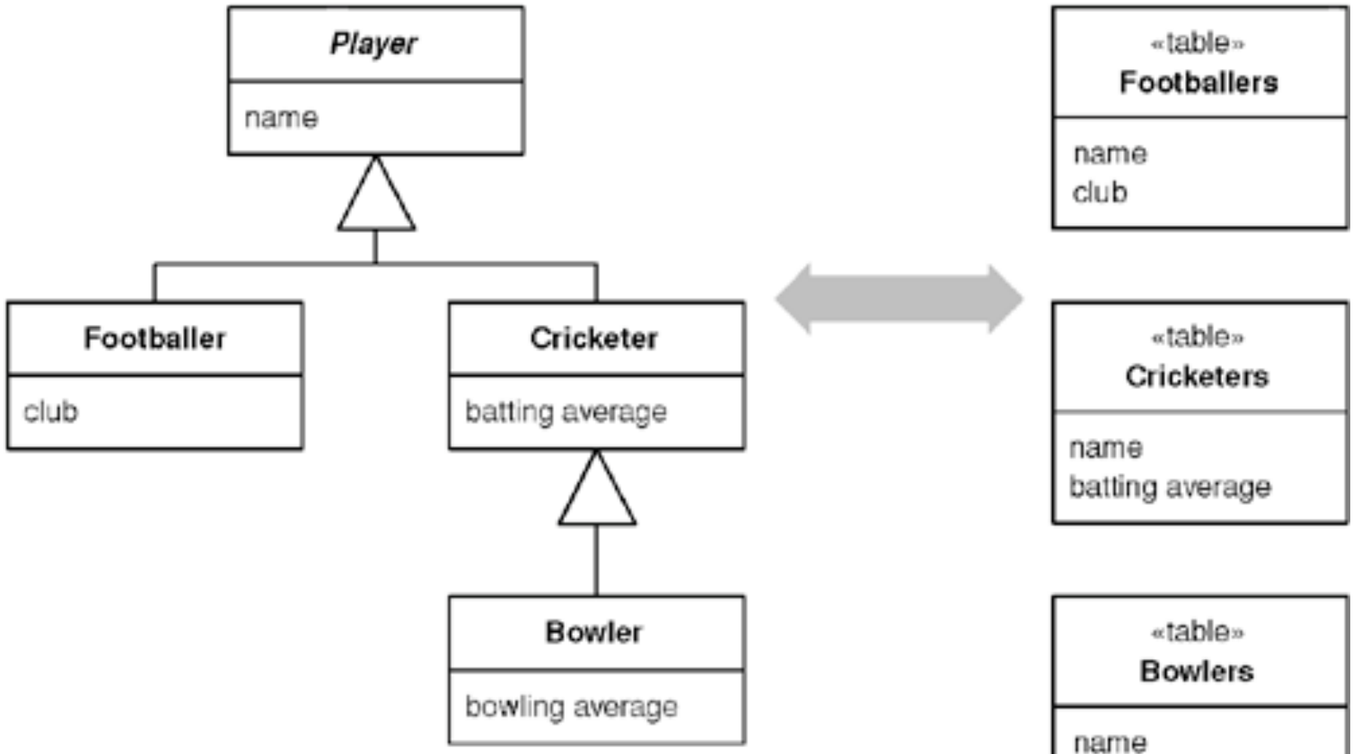
Concrete Table Inheritance

Concrete Table Inheritance



Concrete Table Inheritance

- “Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.”



How It Works

How It Works

- One table for each concrete class in the object hierarchy
- Each table contains the necessary columns for all fields in that silo of the hierarchy
    - some fields appear on multiple tables

How It Works

How It Works

- Keys are important
    - They must still be unique across all tables in the hierarchy

How It Works

How It Works

- Finding the correct object by ID is difficult.
- You need to search all of the appropriate tables

When to Use It

When to Use It

- The Good:
    - each table contains all of the necessary data to stand on its own
    - no joins to load
    - tables can balance the load
        - a somewhat empty argument

When to Use It

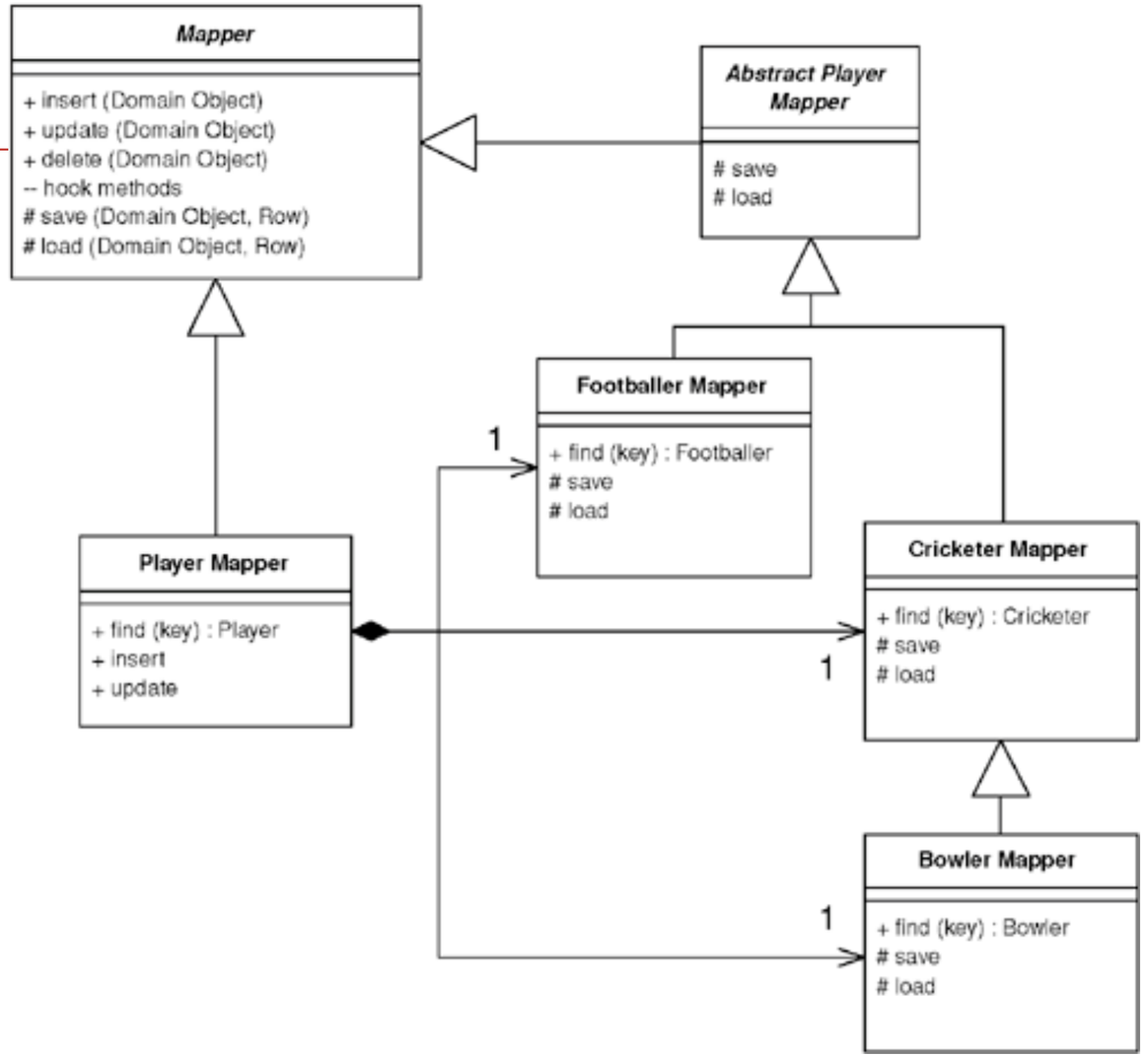
When to Use It

- The bad
  - must have unique IDs across tables
  - hard to enforce relationships with abstract classes
  - refactoring may require database changes
  - a change at the superclass requires changes to all tables
  - find requires multiple queries

Example

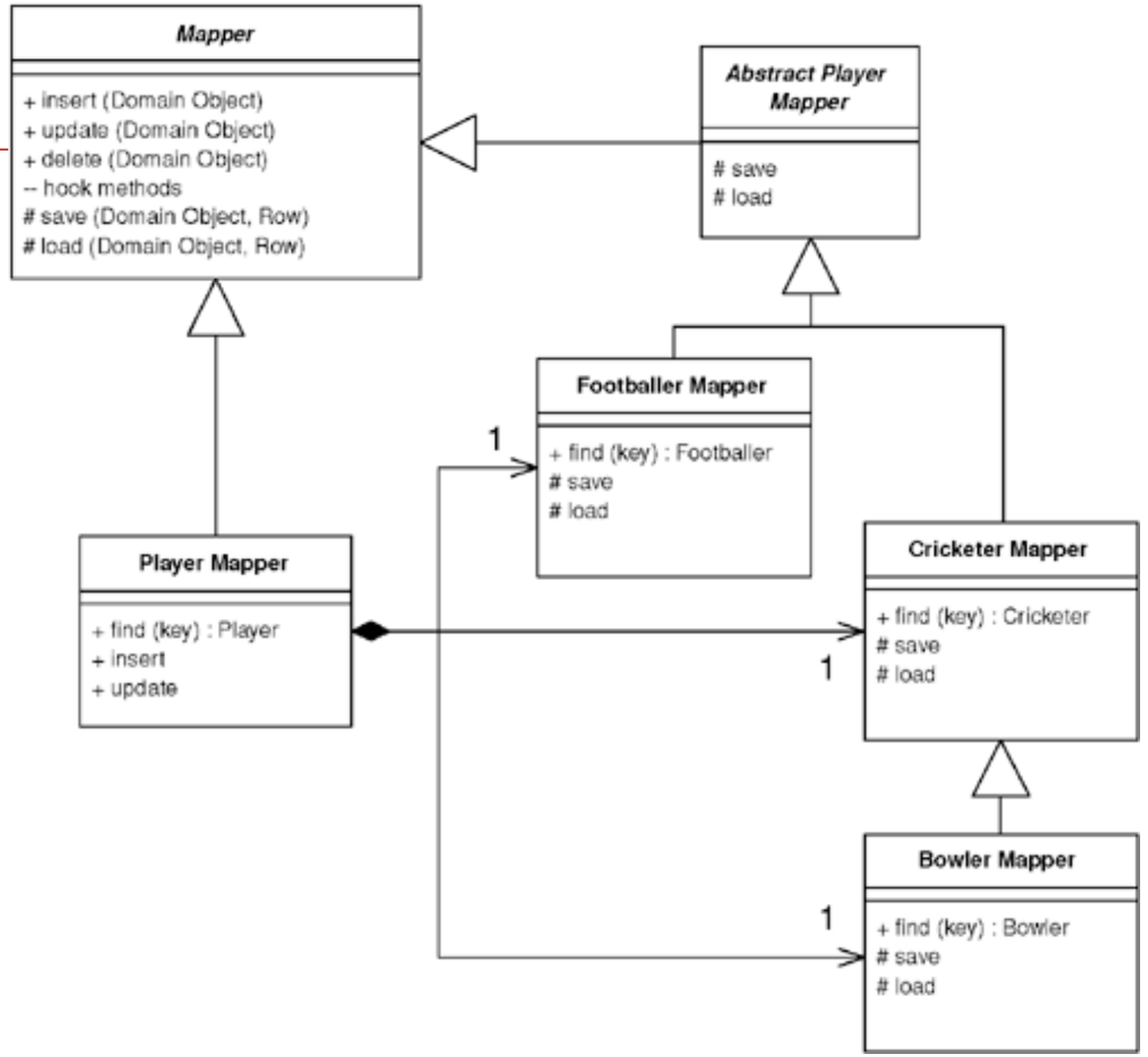
Inheritance Mappers

Inheritance Mappers



Inheritance Mappers

- “A structure to organize database mappers that handle inheritance hierarchies.”



Inheritance Mappers

Inheritance Mappers

- We want to minimize the amount of code used to load/save hierarchies
- We also want to separate
  - abstract - reusable portion of the mapper
  - concrete - class specific portion of the mapper

How It Works

How It Works

- A hierarchy of finders - corresponding to the domain object hierarchy
- Find methods
  - locate the appropriate database row and instantiate the correct objects
  - once you have the correct object - it should be easier to save

When to Use It

When to Use It

- Any time you have inheritance involved in the object hierarchy

UNIVERSITY OF Cincinnati

Object-Relational Metadata Mapping Patterns (Patterns of Enterprise Application Architecture - Chapter 13)

Mike Helmick
Large Scale Software Engineering - Spring 2014
University of Cincinnati

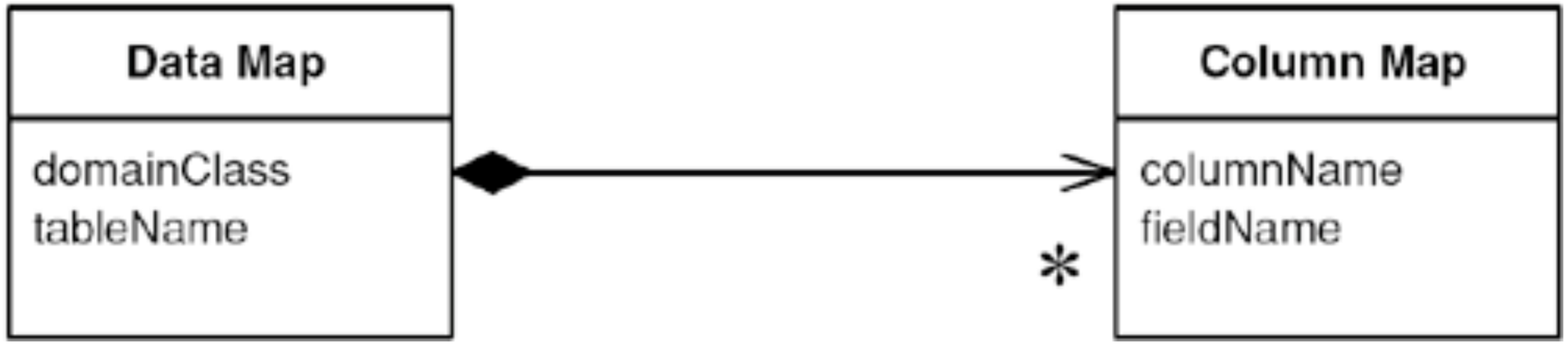
Patterns (Ch 13)

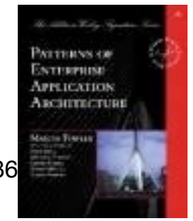
Patterns (Ch 13)

- Metadata Mapping
- Query Object
- Repository

Metadata Mapping

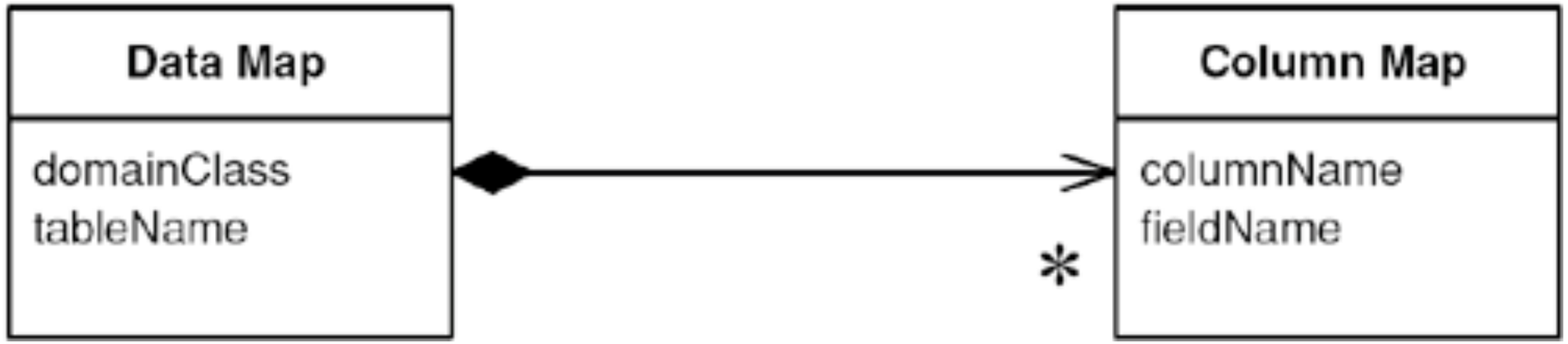
Metadata Mapping

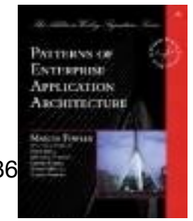




Metadata Mapping

- “Holds details of object-relational mapping in metadata.”





Metadata Mapping

Metadata Mapping

- So far we've seen a lot of hard coding of mapping into individual classes
- Not such a great approach is it?

How It Works

How It Works

- On a basic level
    - metadata is turned into code at runtime
- Two basic architecture approaches - this is a big decision
    - code generation
    - reflective programming

Code Generation

Code Generation

- Write a program that reads the metadata and outputs source code
- The classes look like someone coded them
- Usually done as part of the systems build
    - No reason to check these in to a source code control system
- Code should never be edited

JIT Code Generation

JIT Code Generation

- This technique is not in the book
- It is possible to generate code at runtime
  - In Java this is throw dynamic subclassing and byte code generation
  - I think this is a very valuable approach, but requires coding to interfaces
  - Better suited to dynamic languages (Ruby)

Reflective Programming

Reflective Programming

- Treat both the methods and fields as data
- Find a field called name
  - ask the object if it has a method called setName
  - Invoke the method dynamically

Reflective Programming

Reflective Programming

- Author says he usually recommends against it
    - hard to debug
    - slow (when comparing to a standard function call)
- But acknowledges that it can be appropriate

Reflective Programming

Reflective Programming

- My opinion:
  - I am actually a big fan of the reflective properties of modern OO languages
  - They can seriously simplify generic code making it easier to maintain
  - I will almost always sacrifice a little runtime speed for maintenance speedup

Code Generation

Code Generation

- Changes to the mapping layer
  - require regeneration and full build
  - or just restart server if it is dynamic generation

Debugging

Debugging

- Both approaches are somewhat difficult to debug
- With generated source code - you can step into the methods (if you keep the source code around that long)
- Less experienced developers will have trouble debugging a reflection based mapping layer

Metadata

Metadata

- Needs to be stored somewhere
- XML
  - Hierarchical format
    - can have a table tag, field tags inside
  - Parsers / editors / languages (XQuery) already exist

Metadata

Metadata

- Some other format - you have to write your own parsers
- Can actually be stored in code
  - As compiled source
  - As annotations (Java 5 has these now)

Metadata

Metadata

- Store it in the database
  - As a separate mapping table
  - Or just read the table structure
    - JDBC metadata API

Metadata

Metadata

- Speed of accessing / loading the metadata can be ignored
  - Even if it takes a while
  - Since it should only be loaded once and kept around
    - As generated code or as generated reflective mappings

Complexity

Complexity

- You probably won't be able to guess at everything you need when constructing this layer
- Take an evolutionary approach to development

Special Cases

Special Cases

- For most of what we do (CRUD type operations)
  - Some complex mappings can’t be expressed with the same metadata
  - Or the effort to code the mapping generator is too great
  - You can always sub-class generated code to modify behavior
  - Handle these situations as they arise

When to Use It

When to Use It

- Metadata mapping can greatly speed up development
- Developing the mapping framework is quite complex
    - Takes a good amount of time - requires (generally) a senior developer / architect - needs to be right

When to Use It

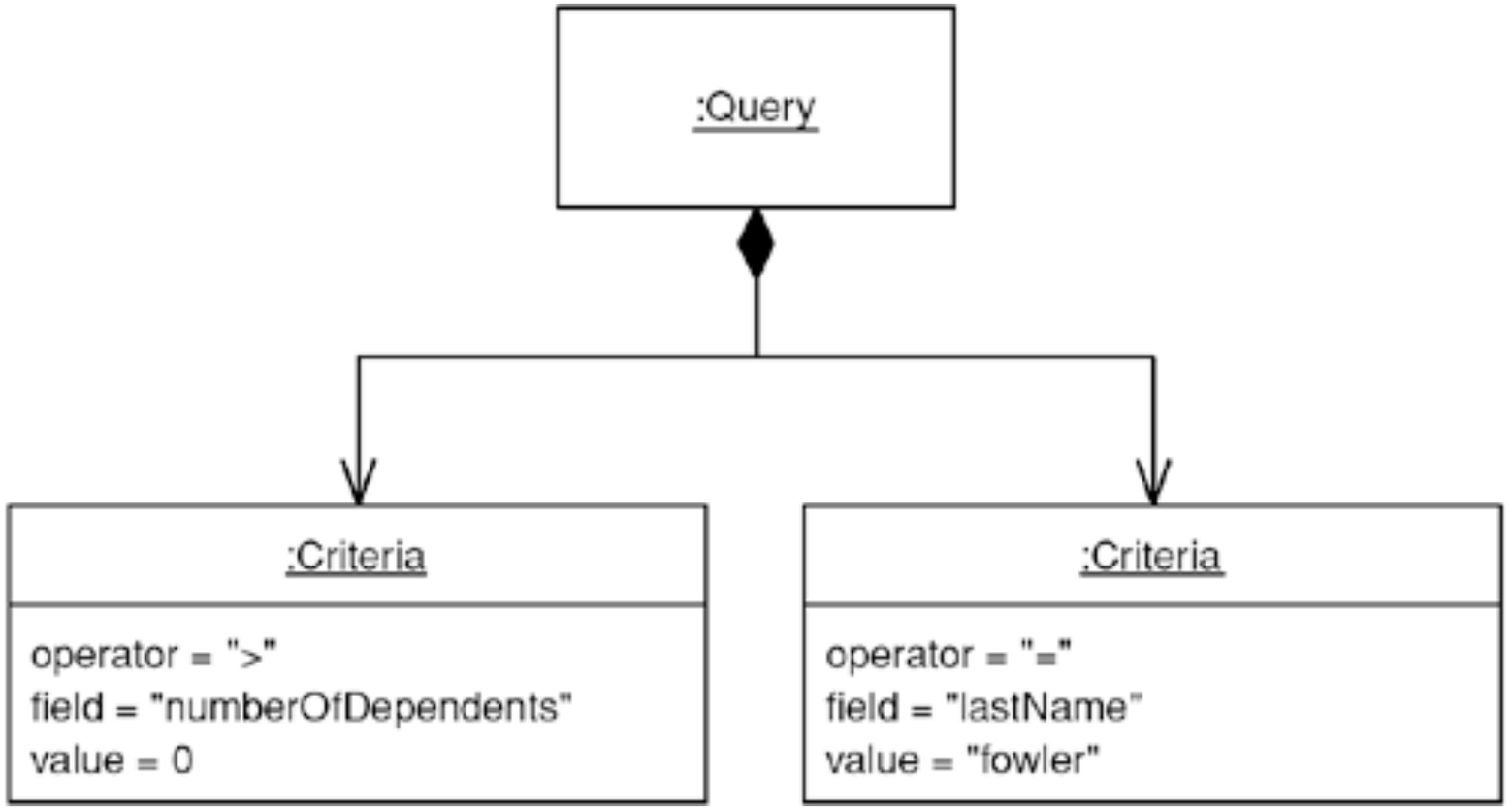
When to Use It

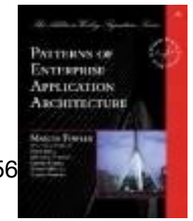
- In general:
  - I think this is the best approach
  - Many commercial / OSS tools exist for a variety of programming languages

Example

Query Object

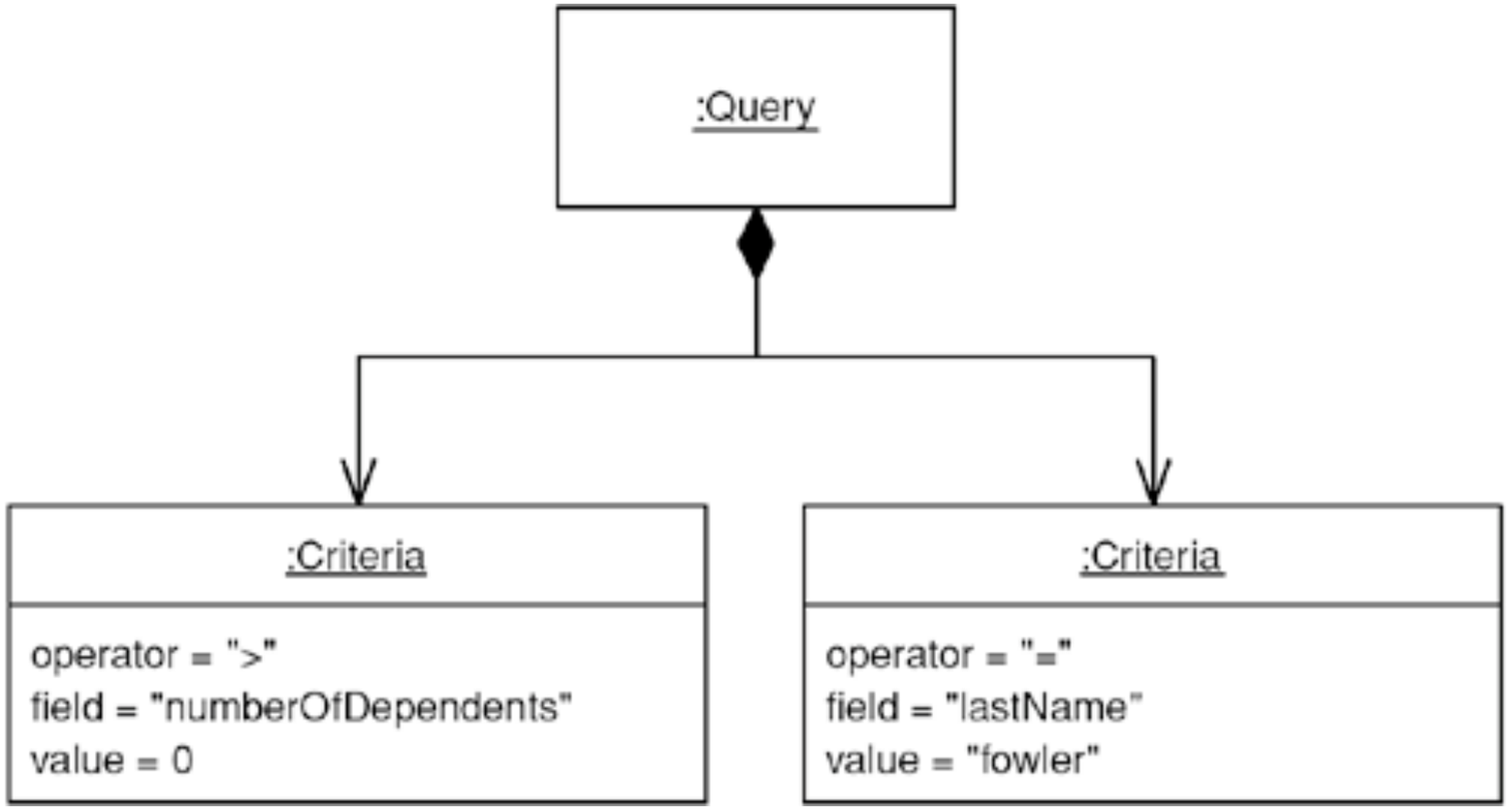
Query Object

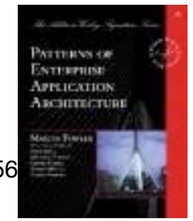




Query Object

- “An object that represents a database query.”





Query Object

Query Object

- On a given project - there is a good chance that not all of the developers will know (or know well) SQL
- Queries are created by referencing fields and objects

How It Works

How It Works

- Turns in-memory objects into an SQL query
- A simple implementation will usually work to start with
    - and can be expanded upon as new features are required

How It Works

How It Works

- Queries are formulated in the programming language that you are using
  - Often times the use of the field name is exactly the same as using the database column name
  - This can help to limit the impact of changes in one or the other
    - Assuming a good mapping layer

How It Works

How It Works

- Can help isolate the developers from the differences in SQL from one database to another
- Hibernate / HQL does this

Variation

Variation

- Use a partially populated domain object to infer a query

When to Use It

When to Use It

- Doesn’t work with a hand coded data source layer
- works well with the combination of
  - Domain Model, Data Mapper, Metadata Mapping
  - Domain Model, Active Record, Metadata Mapping

When to Use It

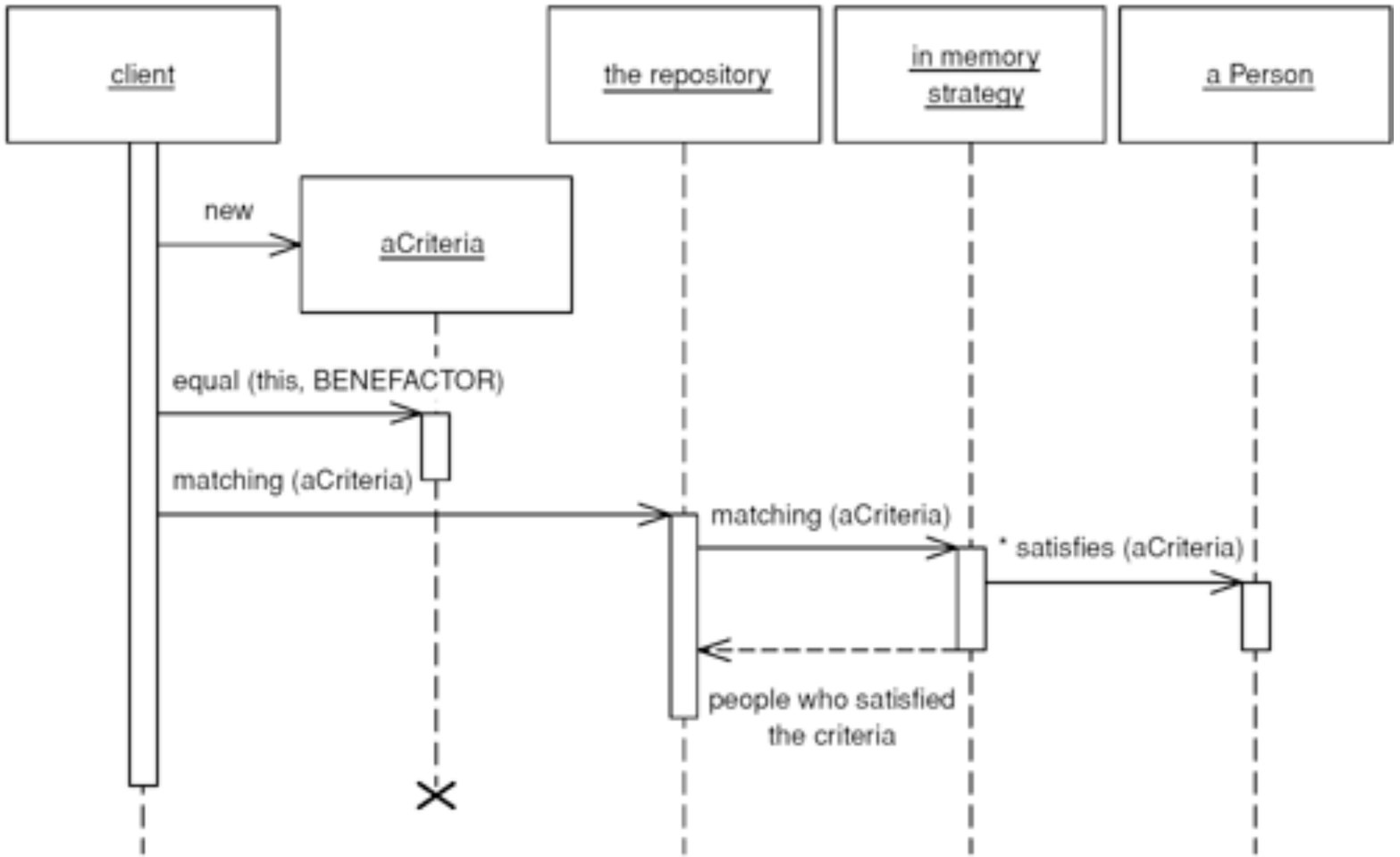
When to Use It

- Developers aren't too familiar with SQL
- You have a tool that can be purchased to accomplish this goal
    - These are very difficult to construct

Example

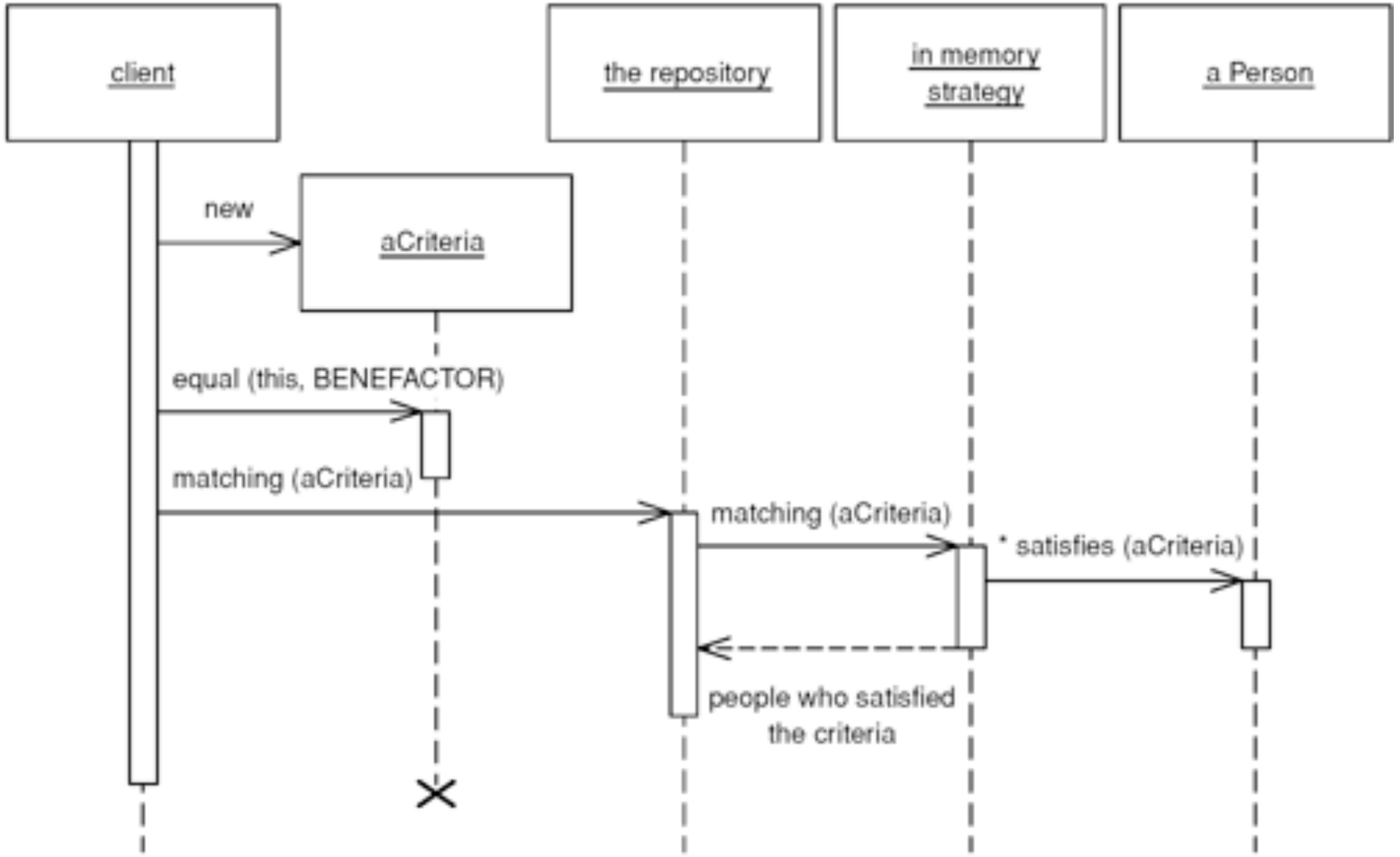
Repository

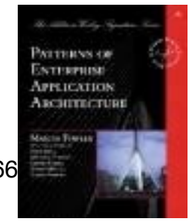
Repository



Repository

- “Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.”





Repository

Repository

- Another layer that abstracts out querying functionality
- Eliminates coupling between query building and the data access layer

How It Works

How It Works

- Works well with the query object
    - And while difficult to build
    - Not a far leap if you are already using Query Object

How It Works

How It Works

- Create a criteria based query
- The repository then presents the database as if all objects are in memory and you are just asking for a subset
- Eliminates the need to specialized finders (Java)

How It Works

How It Works

- Uses Metadata Mapping and Query Object to power its functionality

When to Use It

When to Use It

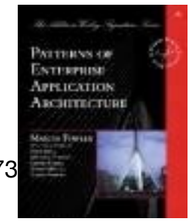
- SQL is totally removed
- All queries are done based on the in-memory objects

When to Use It

When to Use It

- Multiple data sources
    - When your application is pulling data form more than one database
    - This can encapsulate this detail
- Can also encapsulate a simulated database (testing)

Example



Example

- From the book
    - Page 325