Object/Relational Mapping 2008: Hibernate and the Entity Data Model (EDM)

Elizabeth (Betty) O'Neil
Dept. of Computer Science
University of Massachusetts Boston

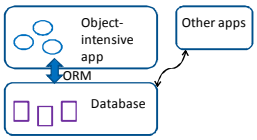
Love of Objects

- Programmers love objects
- Objects are normally ephemeral
- Programming languages provide object persistence, but it is fragile
- Databases provide robust data persistence.
- So… need way to persist object data in the database
- Or think bigger: use data model across object-DB boundaries.
- But a central authority on an object model worries some people: not clearly agile.

2

Object-Relational Mapping (ORM)

- A software system that shuttles data back and forth between database rows and objects
- Appears as a normal database user to the database
- Can share the database and tables with other apps



3

Object-Relational Mapping

- Has a history, but widely adopted only since open-source Hibernate project started, 2001-2002.
- The Hibernate project [7] was founded and led by Gavin King, a Java/J2EE software developer, now part of JBoss. King wrote an excellent book [3].
- Microsoft has adopted a similar approach with EDM, Entity Data Model and its Entity Framework [1, 10]. V1 was released with Visual Studio SP1 in August.
- Both Hibernate and EDM support (or will support) multiple databases: Oracle, DB2, SQL Server, …

4

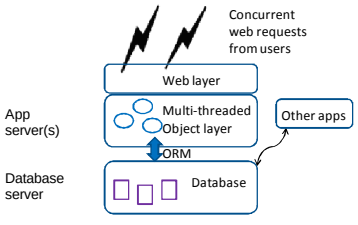
Java Persistence Architecture (JPA)

- JPA is part of current JEE (previously J2EE), Sun's Java Enterprise Edition
- JPA is a standardized version of the Hibernate architecture
- EJB 3 (current Entity Java Beans) uses JPA for EJB persistence, i.e., persistence of "managed" objects
- JPA and EJB 3 are now available in major application servers: Oracle TopLink 11g, OpenJPA for WebSphere and WebLogic, Hibernate JPA for JBoss
- JPA can be used outside EJB 3 for persistence of ordinary objects, as Hibernate is used in this talk

5

Current ORM Impetus: the web app

A web app, with its multi-threaded object layer, particularly needs help with the correct handling of persistent data



6

Our Universe of Discourse

- Object-oriented web apps with database backed data
- Let's consider sites with
  - Possibly many application servers, where the objects live
  - A single database server with plenty of CPUs and disks
- Given today's fast machines and databases, this configuration scales up to many 100s of transactions/second (over 1 M Tx/hour)
- We will concentrate on common characteristics of Hibernate and EDM

7

Outline of Presentation

-------Ask questions any time

- Schema mapping
- Entities and their identity
- Relationships
- Inheritance
- The Pizza Shop Example
- Sample code using entity objects
- Development tools, artifacts
- The ORM runtime system
- Transactions, performance
- Summary

8

Data Modeling

- Three modeling methodologies:
  - We all know the venerable Chen E-R models for database schemas; the extended E-R models (EER) incorporate inheritance
  - The object modeling approach uses UML class diagrams, somewhat similar to EER
  - The tables of the database define the "physical" schema, itself a model of underlying resources
- The relationship between these models involves schema mapping, covered in last SIGMOD's keynote talk by Phil Bernstein[9]

9

Even simple cases need help

- In the simplest case, a program object of class A has fields x, y, z and a table B has columns x, y, z
  - Each instance of A has a row in B and vice versa, via ORM
  - Are we done?
  - If x is a unique id, and x, y, and z are simple types, yes.
  - --Or some unique id in (x, y, z), possibly composite
- If no unique id in (x, y, z), the object still has its innate identity, but corresponding rows involve duplicate rows, against relational model rules
- So in practice, we add a unique id in this case:
- Class A1 has id, x, y, z and table B1 has id, x, y, z

10

Persistent Objects & Identity

- A "persistent object" is an ordinary program object tied via ORM to database data for its long-term state
- The program objects come and go as needed
- Don't confuse this with language-provided persistence (Java/C#), a less robust mechanism
- Persistent objects have field-materialized identity
- It makes sense—Innate object identity depends on memory addresses, a short-lived phenomenon
- So long-lived objects (could be years…) have to be identified this way, it's not the database's fault

11

Persistent Objects need tracking

- We want only one copy of each unique object in use by an app, a basic idea of OO programming
- Each persistent object has a unique id
- We can no longer can depend on object location in memory to ensure non-duplication
- So we have a harder problem than before—need an active agent tracking objects
- This agent is part of ORM's runtime system
- The ORM uses hashing to keep track of ids, detect duplicates

12

ORM Entities

- Like E/R entities, ORM entities model collections of real-world objects of interest to the app
- Entities have properties/attributes of database datatypes
- Entities participate in relationships—see soon (but relationships are not "first class" objects, have no attributes)
- Entities have unique ids consisting of one or more properties
- Entity instances (AKA entities) are persistent objects of persistent classes
- Entity instances correspond to database rows of matching unique id

13

Value Objects

- In fact, persistent objects can be entities or value objects
- Value objects can represent E/R composite attributes and multi-valued attributes
- Example: one address consisting of several address attributes for a customer.
- Programmers want an object for the whole address, hanging off the customer object
- Value objects provide details about some entity, have lifetime tied to their entity, don't need own unique id
- Value objects are called Hibernate "components", EDM "complex types"
- We'll only discuss entities for persistent objects
- For this presentation, persistent object = entity object

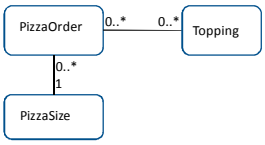
14

Creating Unique IDs

- A new entity object needs a new id, and the database is holding all the old rows, so it is the proper agent to assign it
- Note this can't be done with standard SQL insert, which needs predetermined values for all columns
- Every production database has a SQL extension to do this
  - Oracle's sequences
  - SQL Server's auto-increment datatype
  - …
- The ORM system coordinates with the database to assign the id, in effect standardizing an extension of SQL
- Keys obtained this way have no meaning, are called "surrogate keys"
- Natural keys can be used instead if they are available.

15

Entity Model

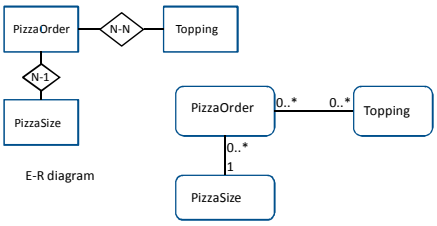


- Uses UML-like diagrams to express object models that can be handled by this ORM methodology
- Currently handles only binary relationships between entities, expects foreign keys for them in database schema
- Has a SQL-like query language that can deliver entity objects and entity object graphs
- Supports updates and transactions

16

Classic Relationships

A PizzaOrder has a PizzaSize and a set of Toppings



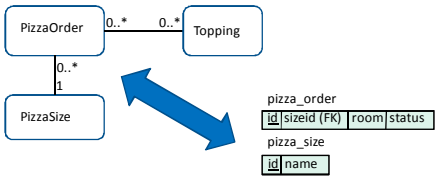
E-R diagram

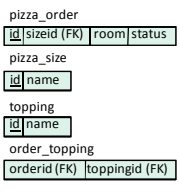
UML class diagram or entity model: no big diamonds, type of relationship is inferred from cardinality markings

17

Classic Relationships

Schema mapping, entities to tables and vice versa





Needed database schema: has one table for each entity, plus a link table for N-N relationship

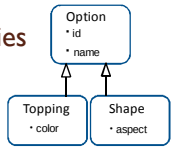
18

Inheritance

- Example: generalize Topping to PizzaOption, to allow other options in the future:
  - Topping ISA PizzaOption
  - Shape ISA PizzaOption, …
- Then a PizzaOrder can have a collection of PizzaOptions
  - We can process the PizzaOptions generically, but when necessary, be sensitive to their subtype: Topping or Shape
  - It is important to have "polymorphic associations", such as PizzaOrder to PizzaOption, that deliver the right subtype object when followed.
- Inheritance is supported directly in Java, C#, etc., ISA "relationship"
- Inheritance is not native to RDBs, but part of EER, extended entity-relationship modeling, long-known schema-mapping problem

19

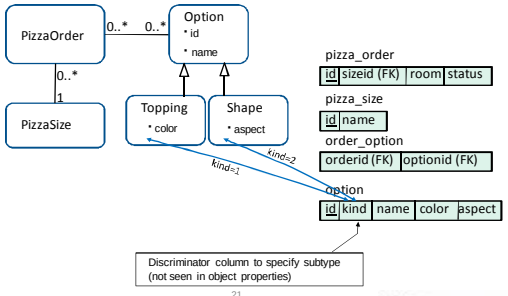
Inheritance Hierarchies



- Both Hibernate and EDM can handle inheritance hierarchies and polymorphic associations to them
- Both Hibernate and EDM provide single-table and multiple-tables per hierarchy solutions
  - Single-table: columns for all subtypes, null values if not appropriate to row's subtype
  - Multiple-table: table for common (superclass) properties, table for each subclass for its specific properties, foreign key to top table
  - Also hybrid: common table plus separate tables for some subclasses

20

Inheritance Mapping (single table)



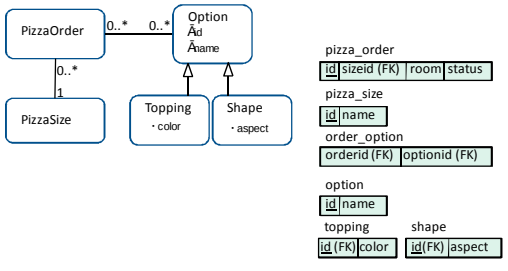
21

Inheritance using a single table

- The discriminator column (here "kind") is handled by the O/R layer and does not show in the object properties
- The hierarchy can have multiple levels
- Single-table approach is usually the best performing way
- But we have to give up non-null DB constraints for subtype-specific properties
- Alternatively, use multiple tables…

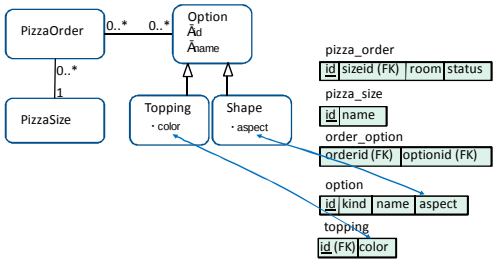
22

Inheritance Mapping (3 tables)



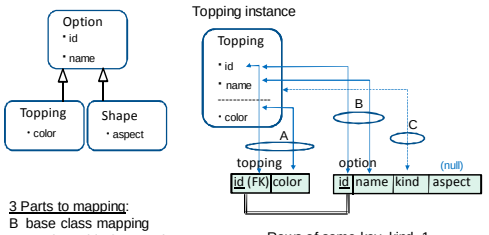
23

Inheritance Mapping (hybrid)



24

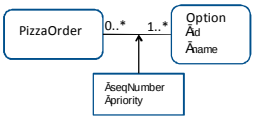
A Mapping dissected



3 Parts to mapping:
B base class mapping
A topping table for Topping
C option table for Topping
Together, 2-way mapping

25

Example of an object model that doesn't fit current ORM



- Case of attributes of the N-N relationship
- We know how to do this in the database…
- We can introduce a new entity in the middle but lose crispness
- Related to problem of ternary relations, also missing

26

The Pizza Shop Example

- Free pizza for students in a dorm
- Student can:
  - Choose toppings, size, order by room number
  - Check on orders by room number
- Admin can:
  - Add topping choices, sizes
  - Mark the oldest pizza order "done"
- Available at www.cs.umb.edu/~eoneil/orm

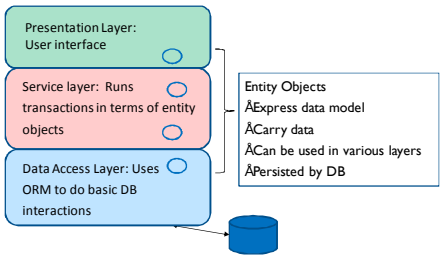
27

The Pizza Shop Example

- Implemented using Hibernate and Microsoft EDM: same architecture, similar code, same database schema
- Implemented as client-server app and web app: only the top-level code changes
  - Client-server means all the app code runs on the client, with network connection to DB
  - Web app means all the app code runs on the web server/app server, clients use browser, DB can be on another server.
- Transactions are explicitly coded, not using container-managed transactions (EJB/COM+ Serviced Components)

28

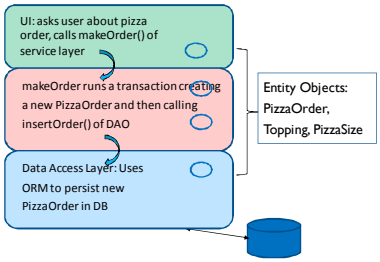
The Pizza Shop: layers



Note: layers are not required by ORM, they are just a good idea for such apps

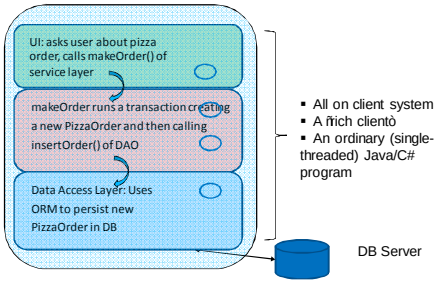
29

The Pizza Shop: objects, calls



30

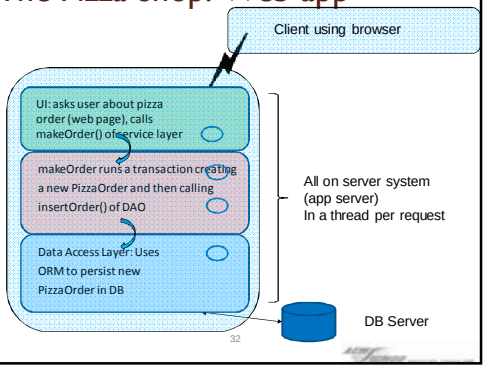
The Pizza Shop: Client-server



- All on client system
- A ñrich clientò
- An ordinary (single-threaded) Java/C# program

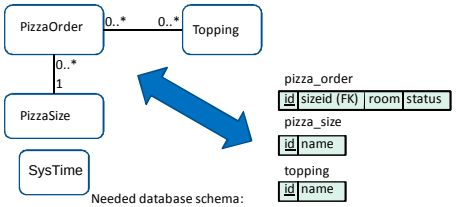
31

The Pizza Shop: Web app



32

Pizza Shop Entities, Mapping



Needed database schema: has one table for each entity, plus a link table for N-N relationship

pizza_order: id | sizeid (FK) | room | status

pizza_size: id | name

topping: id | name

order_topping: orderid (FK) | toppingid (FK)

sys_time: id | current_day | report_day

33

Entity Objects: ideally POJOs/POCOs

- POJO=plain old Java object, POCO=plain old CLR object (C#, etc., CLR= Common Language Runtime of .NET)
  - No special methods need to be implemented
  - Objects are created with normal "new", not some required factory
- Compare to EJB2 Entity Java Bean, COM "managed" objects: these are hard to unit-test, tend to be "heavy-weight"
- EDM: entity objects are POCOs in some ways, but need to extend system class EntityObject, and implement certain methods. Version 2 of EDM is expected to improve this.
- Hibernate: entity objects are POJOs, with no required superclass. All they need is a no-args constructor.

34

Example Simple POJO with properties id and sizeName

```
public class PizzaSize {
    private int id;
    private String sizeName;
    public PizzaSize () {}
    public int getId() { return this.id;}
    public String getSizeName() { return
                        this.sizeName;}
    public void setSizeName(String sizeName) {
        this.sizeName = sizeName;
    }
…   // equals, hashCode, other methods
}
```

Private fields for properties

No-args constructor
Getter for id: ñidòis a read-only property ñ

Getter and setter for sizeName: ñsizeNameò is a read-write property

35

Example Simple POCO

Note: C# has handy property-specific syntax

```
public class PizzaSize
{
   private int _ID;
   private string _SizeName;
   public int ID {
        get  { return this._ID;}
   }
   public string SizeName {
        get  { return this._SizeName;}
        set  { this._SizeName = value;}
   }
  … // other methods
}
```

Private fields for properties

Getter for id: ñidòis a read-only property

Getter and setter for sizeName: ñsizeNameò is a read-write property

36

Hibernate entity POJO with Relationships (can be generated)

```
public class PizzaOrder {
    // Fields, constructors, property getters, setters
    // as in simple POJO
    public PizzaSize getPizzaSize() { return this.pizzaSize;}
    public void setPizzaSize(PizzaSize pizzaSize) {
        this.pizzaSize = pizzaSize;
    }
    public Set<Topping> getToppings() {
        return this.toppings;
    }
…
}
```

N-1 relationship to PizzaSize

N-N relationship to Topping

Standard collection type

37

EDM entity with Relationships (generated code)

Note: italics indicate pseudocode

```
public partial class PizzaOrder:
   global::System.Data.Objects.DataClasses.EntityObject
{  // Fields, constructors, property getters, setters
   // setters have code to report property change
   public PizzaSize PizzaSize
     {
       get  { return PizzaSize object from superclass RelationshipManager; }
       set  { set value in superclass RelationshipManager; }
     }
   public global::System.Data.Objects.DataClasses.EntityCollection<Topping>
        Topping
     {
       get  {  return EntityCollection<Topping> from superclass's
                RelationshipManager; }
     }
…
}
```

←superclass

N-1 relationship to PizzaSize

N-N relationship to Topping

Special collection type, with familiar API

38

Sample code using entity objects

Just "dot through" the N-1 relationship: an order has a unique PizzaSize object bound to it

```
order.getSize().getSizeName() //Hibernate/Java
order.Size.SizeName           // EDM/C#
```

The N-N relationship to Toppings: an order has a collection of Toppings:

```
for (Topping t: order.getToppings())   //Hibernate/Java
   // do something with t
foreach (Topping t in order.Topping)   // EDM/C#
  // do something with t
```

39

Sample Code: persist new object

Hibernate:

```
session.persist(order);
```

Later, after commit, or optional earlier

`session.flush()`, order id is valid

EDM:

```
context.AddObject("PizzaOrder", order);
```

Later, after `context.SaveChanges()`, order id is valid

40

Sample code for a "finder"

To get PizzaOrder objects for a certain room and day, including available Toppings (and PizzaSize) for each

Hibernate HQL: Toppings available in a lazy way

```
List<PizzaOrder> orders =
   session.createQuery("from PizzaOrder o
    where o.roomNumber = "+ roomNumber +
    " and o.day = " + day ).list();
```

EDM Entity SQL: Toppings available by explicit request here:

```
List<PizzaOrder> orders = new ObjectQuery<PizzaOrder>("select
   value o from PizzaEntities.PizzaOrder as o
    where o.RoomNumber = " + roomNumber +
    " and o.Day = " + day, context).
   Include("Topping").Include("PizzaSize").ToList();
```

41

Hibernate lazy fetch

- In the finder query, Hibernate returns PizzaOrder objects with a "proxy" for the associated PizzaSize and a "collection wrapper" for the Toppings collection
- As long as the runtime system is still alive, first access to such an association results in a DB hit for the actual data
- First access after the runtime is shut down results in an exception: it's too late to be lazy
- This default strategy can be overridden in the mapping file: lazy="false" for PizzaOrder's Toppings, for example.
- EDM: no implicit database access, so need to code what you need

42

More queries: some joins

- Example: Find all pizza orders for today (order's day matches sys_time's current_day). No relationship for this, so no connections in the object graph.
- Note no mapped association on day, so no handy collection of object references to use. Use a join…
  - EDM Entity SQL:
    "select value o from PizzaEntities.PizzaOrder as o join PizzaEntities.SysTime as t on o.Day = t.CurrentDay"
  - EDM LINQ: language-integrated query
    List<PizzaOrder> l = (from o in context.PizzaOrder join s in context.SysTime on o.Day equals s.CurrentDay select o).ToList(); // no quotes! C# knows query syntax and does type checking
  - HQL:
    "select o from PizzaOrder o, SysTime t where o.day = t.currentDay"

43

More query features

- Group by, having, order by
- Parameterized queries
- Pairs of objects returned, etc.
- Scalars and aggregates
- Build up queries using methods
- Hibernate: direct SQL queries
- stored procedures
- Control of fetch strategies

44

Entity objects in two layers

Service layer

- Create context and transaction
- Call DAO to get entity objects to work on
- Or Call DAO to add objects
- Or Call objects' own methods
- EDM: context.SaveChanges()/ Hibernate: session.flush() (can be done automatically)
- Commit transaction, drop context

DAO

- Run query for objects or scalar results
- Add new objects

45

Entity Objects can do more

- So far, entity objects carry data to/from database
- i.e, represent persistent *data*
- But objects should have related behavior too
- No problem: add methods to entity classes
- Suppose app needs to compute optimal ordering of toppings for building pizza
- List<Topping> x.getToppingsInBuildOrder()
- This should be method of PizzaOrder, an entity

46

Adding business methods to the entity classes

- Hibernate: relatively simple entity classes, can expand as needed
- EDM: generated code for entity classes: how can we add to it?
- Partial classes of C# come to the rescue:
  - Generated code for PizzaOrder provides data methods in one partial class for PizzaOrder
  - We code business methods in another partial class for PizzaOrder, in another source file.
  - Compiler puts them together

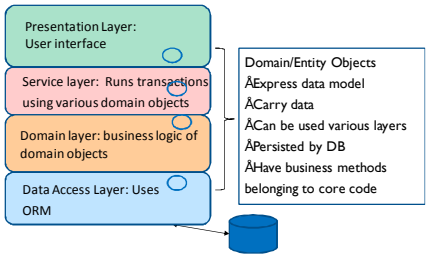
47

"Rich" Domain model

- Domain classes (entities) manage their persistent data, and "rich" ones also provide app-related actions on their data
- Idea of DDD, domain-driven design (Fowler[13], Evans[14], 2004)
- Service layer coordinates actions *between* entities as needed for transactional actions
- Service layer should be thin, delimiting transactions and calling on domain classes for most of the work
- Data-only entities dubbed "anemic domain model"

48

The Pizza Shop: layers refined



49

Development Tools

Goal of using GUI to incrementally build a data model is doable, coming, will be great
Example: EDM data model display:

50

Development Tools

- The tools
  - Can turn the entity model into program classes
  - Can turn the entity model into database schema
  - Or turn database schema into an entity model
  - Or turn a set of classes into an entity model
- Pizza Shop is simple enough to be specified by database schema + one execution of tool
- For complex systems, you need to work with the XML mapping files to get the full use of these systems today. (Or use Hibernate/JPA source annotations instead of XML.)
- Luckily, only elementary XML is needed, let's look at some files…

51

XML: Hibernate PizzaSize

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//ENö
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="pizza.domain.PizzaSize" table="PIZZA_SIZE">
    <id name="id" type="int">
      <column name="ID" />
      <generator class="native" />
    </id>
    <property name="sizeName" type="string">
      <column name="SIZE_NAME" length="30" not-null="true" unique="true" />
    </property>
  </class>
</hibernate-mapping>
```

52

XML for EDM PizzaSize: 3 parts

```
<EntityType Name="PizzaSize">
    <Key><PropertyRef Name="ID" /></Key>
    <Property Name="ID" Type="Int32" Nullable="false" />
    <Property Name="SizeName" Type="String" Nullable="false" MaxLength="50" />
    <NavigationProperty Name="PizzaOrder" Relationship = "PizzaModel.FK_PizzaOrder_PizzaSize"
                FromRole="PizzaSize" ToRole="PizzaOrder" />
</EntityType> …
<EntityType Name="PizzaSize">
    <Key><PropertyRef Name="ID" /></Key>
    <Property Name="ID" Type="int" Nullable="false" StoreGeneratedPattern="Identity" />
    <Property Name="SizeName" Type="nvarchar" Nullable="false" MaxLength="50" />
</EntityType> …
<EntityTypeMapping TypeName="IsTypeOf(PizzaModel.PizzaSize)">
    <MappingFragment StoreEntitySet="PizzaSize">
        <ScalarProperty Name="ID" ColumnName="ID" />
        <ScalarProperty Name="SizeName" ColumnName="SizeName" />
    </MappingFragment>
</EntityTypeMapping>
```

Conceptual schema

Storage schema

Mapping

53

EDM XML for Association

```
<EntityType Name="PizzaSize">
    <Property …>
    <NavigationProperty Name="PizzaOrder"
        Relationship = "PizzaModel.FK_PizzaOrder_PizzaSize"
            FromRole="PizzaSize" ToRole="PizzaOrder" />
</EntityType>
<Association Name="FK_PizzaOrder_PizzaSize">
    <End Role="PizzaSize" Type="PizzaModel.PizzaSize" Multiplicity="1" />
    <End Role="PizzaOrder" Type="PizzaModel.PizzaOrder" Multiplicity="*" />
</Association> …
<Association Name="FK_PizzaOrder_PizzaSize">
    <End Role="PizzaSize" Type="PizzaModel.Store.PizzaSize" Multiplicity="1" />
    <End Role="PizzaOrder" Type="PizzaModel.Store.PizzaOrder" Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="PizzaSize"><PropertyRef Name="ID" /></Principal>
        <Dependent Role="PizzaOrder"><PropertyRef Name="SizeID" /></Dependent>
    </ReferentialConstraint>
</Association>
```

Conceptual schema

Storage schema

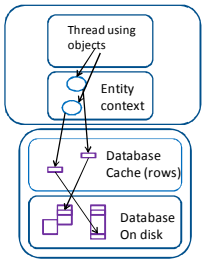
54

The Entity Context

- Hibernate Session, EDM ObjectContext, seen as "session" and "context" variables in previous code.
- Belongs to one thread as its object cache, usually for one transaction lifetime, one request-response cycle.
- Usually defers updates to the database to end of transaction
- Ensures only one entity object for each id
- As a cache, avoids some rereads (loads by id) of database, preventing some repeated-read anomalies if running at lower isolation level.
- Caching: the database itself has the definitive cache, global to all apps using it…

55

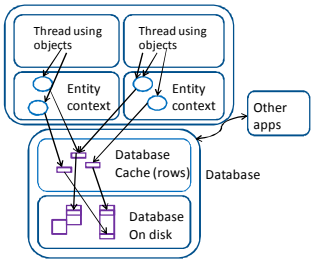
The Entity Context at work



1. Thread creates entity context, starts transaction
2. Accesses entity objects
3. Commits
4. Drops entity context
5. Uses (reads) entity objects outside context

56

The Entity Contexts & the DB cache Under Concurrent Access



57

Entity Object Life Cycle

- Birth: as a POJO or (semi) POCO, unconnected to the context
- Or read in by query, so already connected to context
- Possibly modified by app
- If new, needs addition to context: save()/AddObject()
- By commit, its updates are saved to DB
- Lives on after context dropped, useful for results display
- Can be reattached to new context, but not covered here
- Normally, abandoned soon, garbage-collected

58

Queries and the object cache

- When a query delivers entity objects, the id's may already be in the cache
- Need to avoid duplicates, preserve app's changes
- Hibernate flushes changes to DB before query by default
- EDM, by default, preserves the older object of a certain id, avoiding some DB writes at this point
- But tricky: need to flush changes affecting search conditions to maintain query correctness.

59

Transactions

- Hibernate and EDM are designed for transactional apps
- Both support transactions involving single or multiple DBs/resource managers, via JTA or DTC for distributed case
  (JTA=Java Transaction API, DTC= Microsoft's Distributed Transaction Coordinator)
- Both support both explicit transactions and container-managed transactions
- We're considering simple case of single DB, explicit transactions
- Still have choice in isolation level, mainly:
  - Read-committed (RC)
  - Read-committed with ORM-coordinated versioning
  - Serializable (SR)

60

Transactions

- RC, SR have usual meaning, handled in DB
- Also Snapshot Isolation for some DBs
- Pizza project uses SR, but easy to change
- RC often not enough, but RC + versioning is attractive
- Versioning by ORM provides "optimistic CC":
  - Context remembers original object state, or row version if supported by DB
  - For changed objects, compares saved vs. current DB state at commit-time, throws an exception if changed
  - Avoids lost updates otherwise possible with RC
  - Refresh action available to help with retries

61

Conversations

- So far, each request has had one transaction, good enough for Pizza Shop
- Some actions perceived as a unit to the user are made up of several requests
- Example: Read a current bid amount, let user decide on new bid, then make the new bid
- Two DB transactions here, since no DB transaction should span the think time
- The two transactions are related: a "conversation" or "session" or "business transaction" with one user

62

Conversations

- Example Conversation: Look at bid, think, update bid
- Someone else can slip in a bid update between my look and update
- One solution: make my bid update contingent on the bid amount still being what I saw before, abort second transaction if not
- This is versioning again, now used across multiple system transactions in the same context

63

Conversations and Sessions

- We're thinking about a context spanning several requests of a conversation so it can do version checking

  ----|---look----|----------think----------|--update--|----

  Tx1:10 ms 2 min Tx2:10 ms
- Expensive in memory, however, since the context must be kept alive between requests, while the user thinks: above 20 ms vs 2020 ms, factor of 100
- Rather than holding a whole context for a conversation, we can condense it down to a usually-small dataset as part of "session data", save nearly a factor of 100
- This can be held in the common database, but as unshared data, has other possibilities too

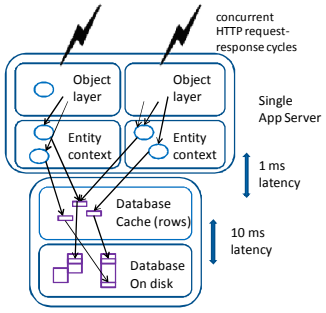
64

Performance & Scalability: Assumptions

- One database server with plenty of memory
- Warm data is in DB server's cache
- One or multiple application servers
- No shared mutable data on app servers
- All shared mutable data is in the database
- DB server's cache data is fast to access over a local network
- Standard use of pooled DB connections to avoid connection-setup delays

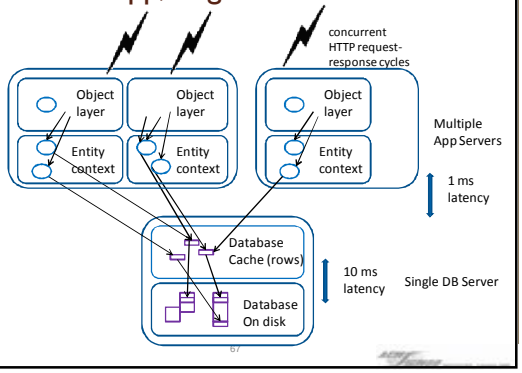
65

Web app, small site



66

Web app, larger site



67

Second-level caching

- If this simple design is maxing out, say with 1000s of Tx/sec
  - And is not sharing DB with other apps
  - And specifically is overwhelming the DB
- Can try second-level caching to offload the DB
- Example: JBoss Cache, a transactional replicated distributed cache
- i.e, can handle case of multiple app servers
- Involves more configuration, tuning, not easy
- Hopefully plenty of money for consultants at this point

68

Summary

- Relational technology continues to prove its worth, and ORM is using it in full
- Only one deficiency of SQL 92 uncovered: standard way to generate new unique id
- The object-relational impedance mismatch has been largely overcome
- No textbooks yet, little academic work: Hibernate semantics are "example-driven"…
- See sources, run web app (Hibernate version) at www.cs.umb.edu/~eoneil/orm

69

Bibliography

In proceedings (subset):

[1] Adya, Atul, Blakely, Jose, Melnik, Sergey, Meralidhar, S., and the ADO.NET Team, 2007, Anatomy of the ADO.NET Entity Framework, In Proceedings of SIGMOD 2007, ACM Press, New York, NY.

[3] Bauer, Christian, King, Gavin 2006 Java Persistence with Hibernate, Manning

[5] Bernstein, Phil, Melnik, Sergey. 2007 Model Management 2.0—Manipulating Richer Mappings. In Proceedings of SIGMOD 2007, ACM Press, New York, NY, 1-12.

[7] Hibernate, http://www.hibernate.org

[10] MSDN Library, 2006 The ADO.NET Entity Framework Overview, http://msdn2.microsoft.com/en-us/library/aa697427(VS.80).aspx

Added:

[13] Fowler, Martin 2003 Patterns of Enterprise Application Architecture, Addison-Wesley

[14] Evans, Eric 2004 Domain-Driven Design, Addison-Wesley

70