Polyglot Persistence

EclipseLink JPA for NoSQL, Relational, and Beyond

ORACLE

Shaun Smith

shaun.smith@oracle.com

AGETO

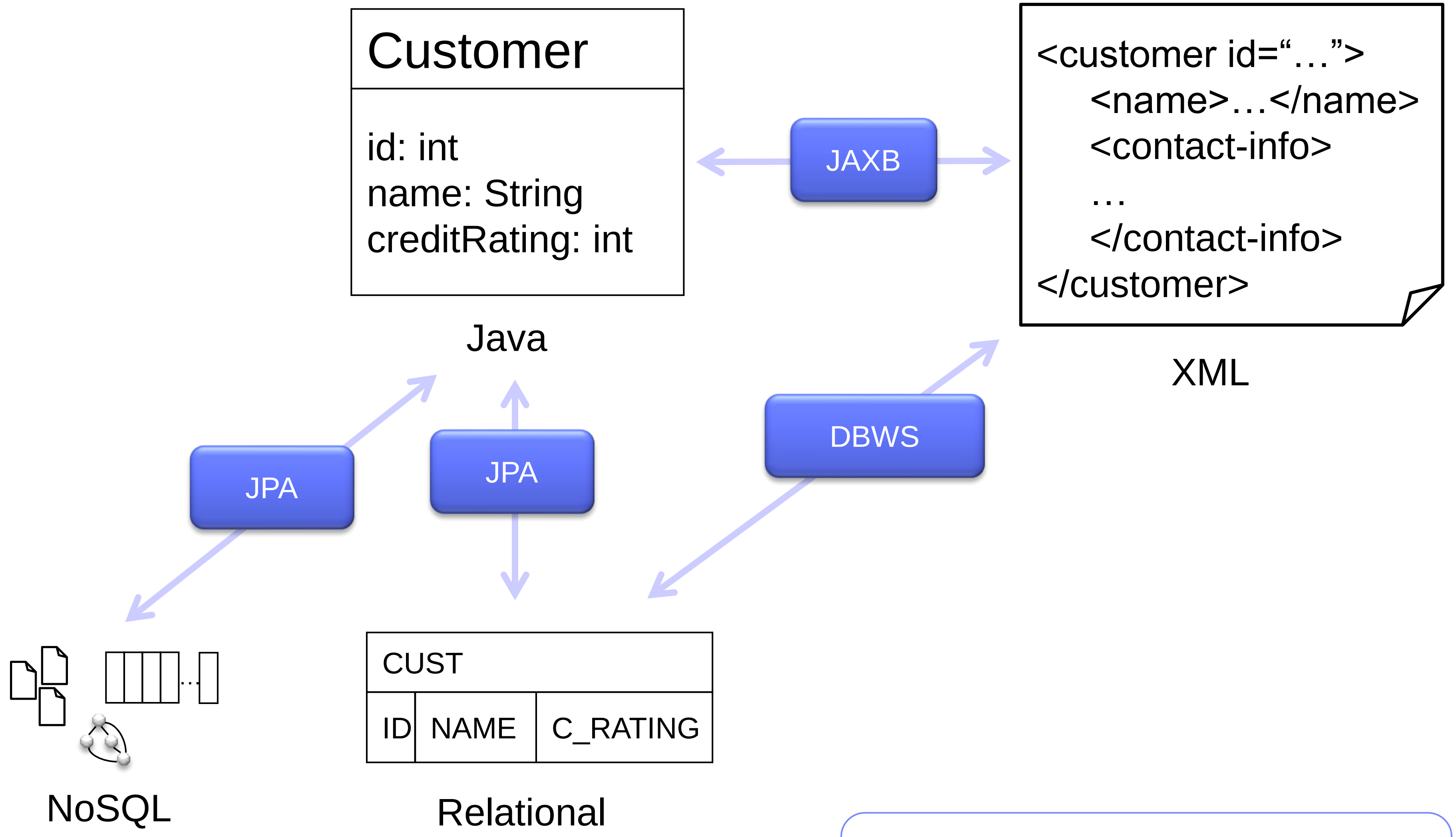
Gunnar Wagenknecht

gunnar@wagenknecht.org

About Us

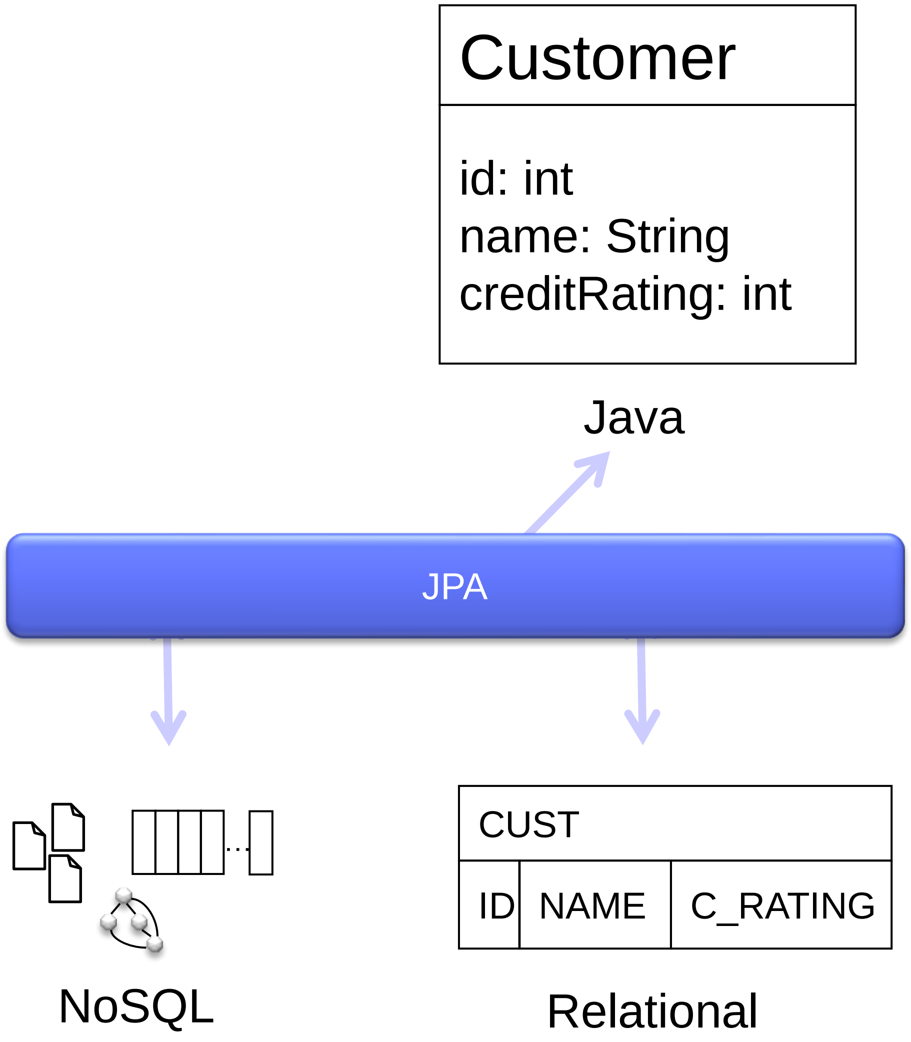
- Shaun Smith
  - Oracle TopLink Product Manager
    - TopLink Grid, Coherence GoldenGate Adapter, and NoSQL persistence
  - Eclipse committer on EclipseLink and related projects
- Gunnar Wagenknecht
  - CTO @ AGETO
  - Committer and contributor @ Eclipse
  - Java since 1999; Eclipse since 2001

Java Persistence: The Problem Space



JPA: Java Persistence API
JAXB: Java Architecture for XML Binding
DBWS: EclipseLink Database WebServices

Java Persistence: The Problem Space

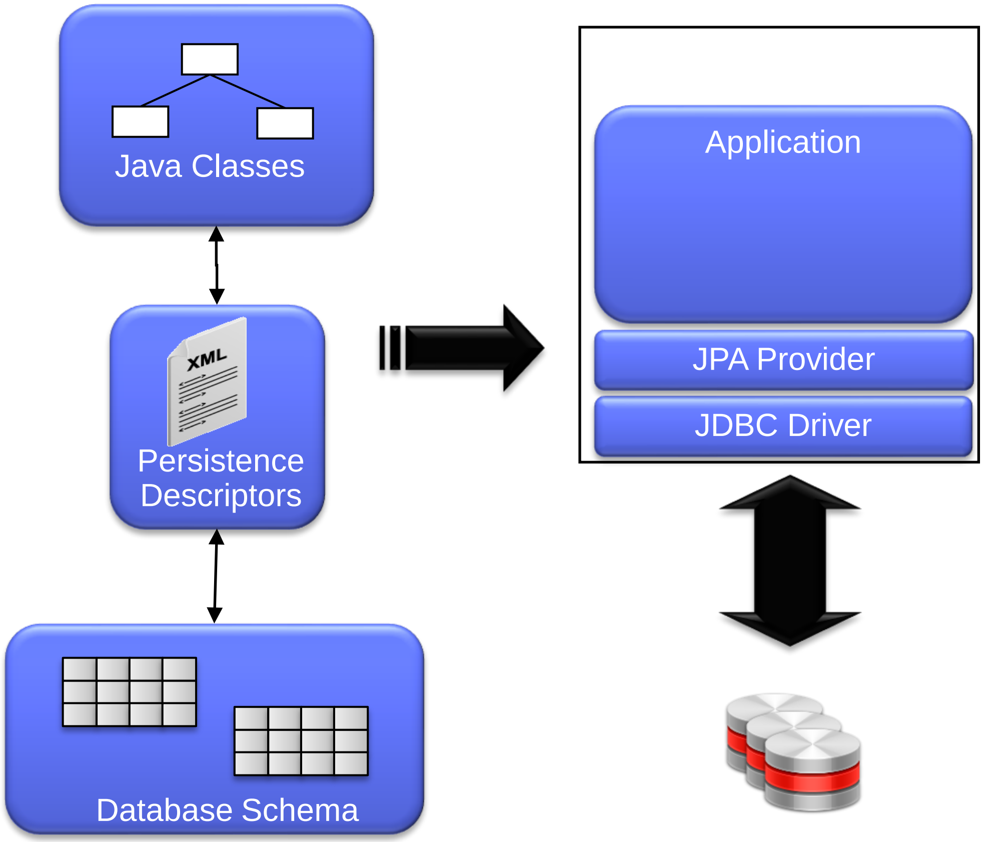


JPA - Background

- A standardization of industry practices for Java POJO Object Relational Persistence
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals

Java Persistence API (JPA) - in a Nutshell

- Defines:
  - How Java objects are stored in relational db
  - A programmer API for reading, writing, and querying persistent Java objects (“Entities”)
  - A full featured query language in JP QL
  - a container contract that supports plugging any JPA runtime in to any compliant container



NoSQL Databases

- NoSQL database are increasingly popular
- No common definition (document, graph, columnar)
  - Differing feature sets
  - Some offer query language/API—some not
- No standards
- Every database offers a unique API
  - Cost in terms of learning
  - Zero portability across databases

EclipseLink NoSQL

- Support JPA access to NoSQL databases
  - Leverage non-relational database support for JCA (and JDBC when available)
- Define annotations and XML to identify NoSQL stored entities (e.g., @NoSQL)
- Support JPQL subset for each
  - Key principal: leverage what's available
- Initial support for MongoDB and Oracle NoSQL.
- Support mixing relational and non-relational data in single composite persistence unit ("polyglot persistence")

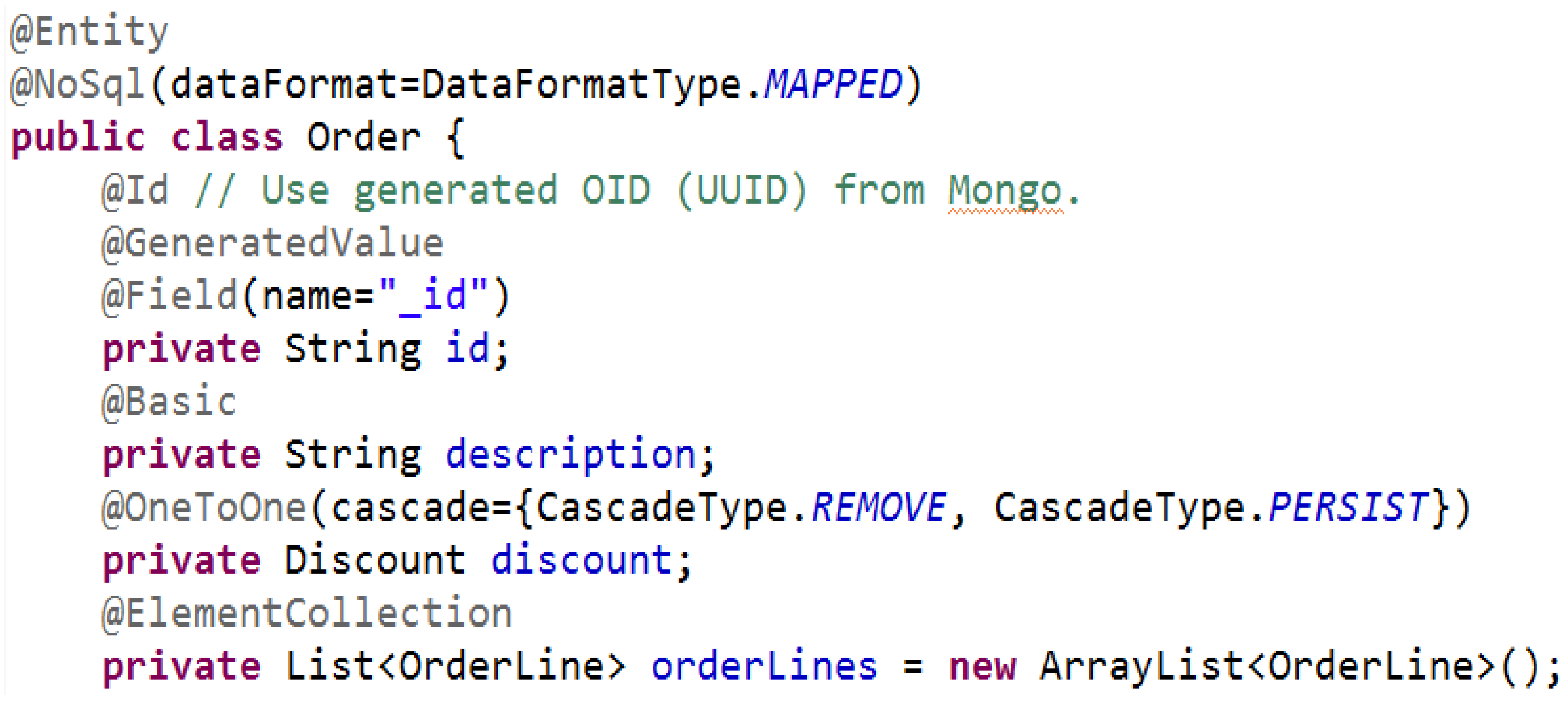
Applicability of JPA to NoSQL

- Core JPA concepts apply to NoSQL:
  - Persistent Entities, Embeddables, ElementCollection, OneToOne, OneToMany, ManyToOne, Version, etc.
- Some concepts apply with some databases:
  - JPQL, NamedNativeQuery
- Pure relational concepts don’t apply:
  - CollectionTable, Column, SecondaryTable, SequenceGenerator, TableGenerator, etc.

Querying NoSQL with JPA

- Two kinds of queries
  - JQPL—portable query language defined by the spec
  - Native query—lets you leverage database specific features
  - Dynamic or static @NamedQuery
- JPQL translated to underlying database query framework.

Example MongoDB Mapped Entity



```java
@Entity
@NoSql(dataFormat=DataFormatType.MAPPED)
public class Order {
    @Id // Use generated OID (UUID) from Mongo.
    @GeneratedValue
    @Field(name="_id")
    private String id;
    @Basic
    private String description;
    @OneToOne(cascade={CascadeType.REMOVE, CascadeType.PERSIST})
    private Discount discount;
    @ElementCollection
    private List<OrderLine> orderLines = new ArrayList<OrderLine>();
```

MongoDB Query Examples

- JPQL

```
Select o from Order o
  where o.totalCost > 1000

Select o from Order o
  where o.description like 'Pinball%‘

Select o from Order o
  join o.orderLines l where l.cost > :cost
```

- Native Queries

```
query = em.createNativeQuery(
  "db.ORDER.findOne({\"_id\":\"" +
  oid + "\"})", Order.class);

Order order =
  (Order) query.getSingleResult();
```

NOSQL PERSISTENCE

DEMO

Polyglot Persistence

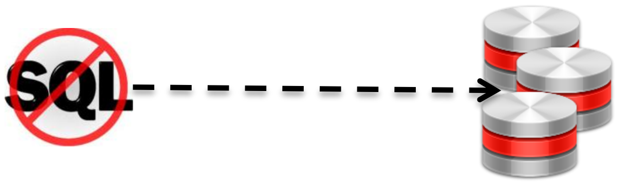
- **Different** data storage **technologies** for **different kinds of data**
- First ask, **how we** want to **manipulate** the **data**, **then figure out** what **technology** is the best bet for it
- Apparent even within a **single application**
- Cost of **complexity**

http://martinfowler.com/bliki/PolyglotPersistence.html

NoSQL Distilled (Sadalage and Fowler, 2012)

Polyglot Persistence

- Relational and NoSQL databases each have their strength - choose the right one for the job
- A single application may have need for both relational and NoSQL data
- EclipseLink JPA supports use of multiple database technologies in the same application
  - Relationships can span databases and database technologies



POLYGLOT PERSISTENCE

DEMO

Experience / Best Practices

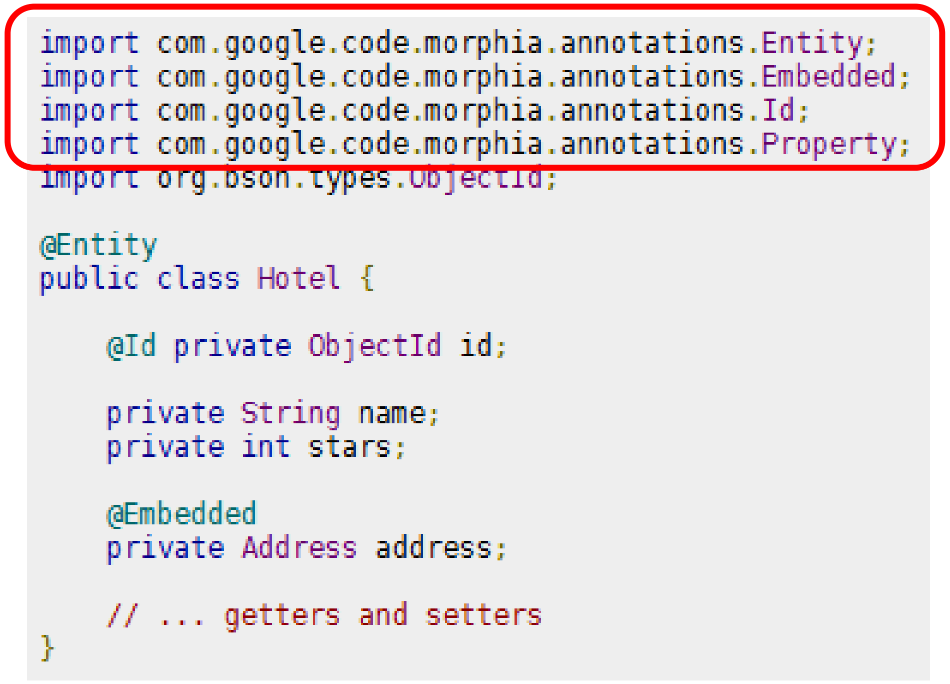
- MongoDB great for Sandbox development
  - Start with NoSQL
  - Go relational when model complete
- Remember: Two Worlds
  - Think Auto-Commit
  - No complex queries
- Don’t be cool just because you can!

Tools

- Dali JPA Development Tools
- NoSQL specific
  - Zero schema development in MongoDB
  - Many MongoDB Admin GUIs
    (eg. MonjaDB in Eclipse)

EclipseLink vs. Morphia

- Morphia
  - is MongoDB only
  - uses JPA-like mappings—clearly the JPA approach is amenable to NoSQL persistence



```
import com.google.code.morphia.annotations.Entity;
import com.google.code.morphia.annotations.Embedded;
import com.google.code.morphia.annotations.Id;
import com.google.code.morphia.annotations.Property;
import org.bson.types.ObjectId;

@Entity
public class Hotel {

    @Id private ObjectId id;

    private String name;
    private int stars;

    @Embedded
    private Address address;

    // ... getters and setters
}
```

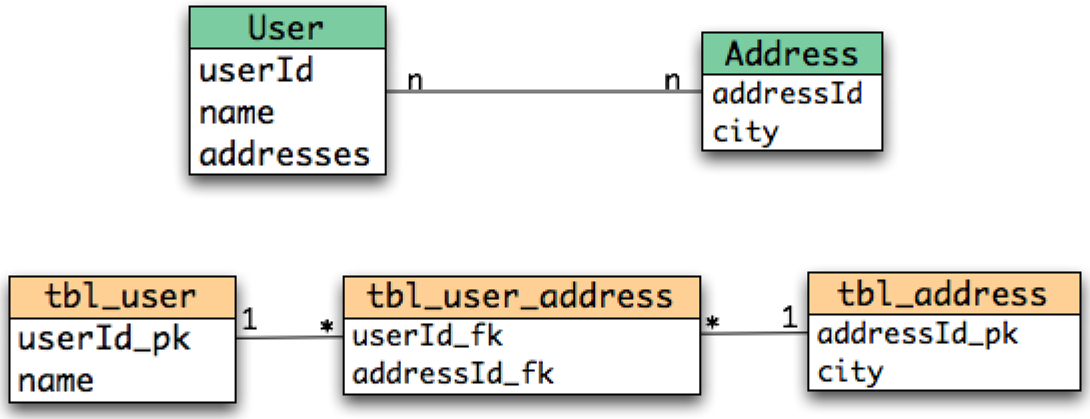
Morphia Observations

- Cannot combine with other DB technologies
  - Non-Polyglot
- DB operation order must be managed by developer
  - Extremely painful in practice
- EclipseLink offers comparable features with distinct advantages
  - Polyglot, technology independent, ease of use

EclipseLink vs. Hibernate OGM

- OGM—"Object Grid Mapper"
  - initially focused on Infinispan
- Focused on storing entities in the Grid/NoSQL
  - **Imposes specific data storage format**
  - EclipseLink is focused on JPA access to NoSQL not simply NoSQL as dedicated entity store
- Relies on Lucene for search
  - No native or JPQL translation to native query

Hibernate OGM Storage Format



key	value
tbl_user,userId_pk,1	{userId_pk=1,name="Emmanuel"}
tbl_user,userId_pk,2	{userId_pk=2,name="Caroline"}
tbl_address,addressId_pk,3	{addressId_pk=3,city="Paris"}
tbl_address,addressId_pk,5	{addressId_pk=5,city="Atlanta"}
tbl_user_address,userId_fk,1	{ {userId_fk=1, addressId_fk=3}, {userId_fk=1, addressId_fk=5} }
tbl_user_address,userId_fk,2	{ {userId_fk=2, addressId_fk=3} }
tbl_user_address,addressId_fk,5	{ {userId_fk=1, addressId_fk=5} }
tbl_user_address,addressId_fk,3	{ {userId_fk=1, addressId_fk=3}, {userId_fk=2, addressId_fk=3} }

EclipseLink NoSQL Next Step

- Gather community feedback
  - **Please contribute to the conversation!**
- Support additional databases:
  - Cassandra
  - HBase
  - CouchDB
  - …
- Long term—standardization?

Useful Links

- EclipseLink
  - http://www.eclipse.org/eclipslink
- EclipseLink NoSQL Examples
  - http://wiki.eclipse.org/EclipseLink/Examples/JPA/NoSQL
- James Sutherland’s Blog
  - http://java-persistence-performance.blogspot.com/

Q&A