Persistence in Java: technical perspective

jfernan@ua.pt

Disclaimer

- This is no a persistence course
- Assuming (some) knowledge
  - Relational DB
- Some details will be left out
  - Very specific ORM annotations
  - DB engine

ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, Pages 9–36.

The Entity-Relationship Model—Toward a Unified View of Data

PETER PIN-SHAN CHEN

Massachusetts Institute of Technology

A data model, called the entity-relationship model, is proposed. This model incorporates some of the important semantic information about the real world. A special diagrammatic technique is introduced as a tool for database design. An example of database design and description using the model and the diagrammatic technique is given. Some implications for data integrity, information retrieval, and data manipulation are discussed.

The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model. Semantic ambiguities in these models are analyzed. Possible ways to derive their views of data from the entity-relationship model are presented.

Key Words and Phrases: database design, logical view of data, semantics of data, data models, entity-relationship model, relational model, Data Base Task Group, network model, entity set model, data definition and manipulation, data integrity and consistency
CR Categories: 3.50, 3.70, 4.33, 4.34

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance

A Relational Large Shared

E. F. CODD
IBM Research Labora

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

described in
spects to the
ogue for non-
scribing data
out superim-
epresentation
r a high level
pendence be-
e representa-

ew is that it
redundancy,
sed in Section
as spawned a
is mistaking
ation of rela-
ection trap").
er evaluation
nt formatted
from a logical
data within a
rspective are
mentations of
not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance

Home > Data Management

IDG CONTRIBUTOR NETWORK Want to Join?

BUSINESS OF DATA

By Barry Morris, CIO | AUG 3, 2017 6:11 AM PT

Opinions expressed by ICN authors are their own.

OPINION

The promise and payoff of NoSQL, Part 1

As we move toward a new industry view of data management, we are also moving towards a world in which smarter data
at where we

OPINION

Simpler applications and smarter databases, Part 2

Here's a more detailed look into the trade-offs represented by NoSQL.

OPINION

Back to the (SQL) Future, Part 3

In the first post in this series, I discussed the emergence of NoSQL to address the need to make databases compatible with cloud needs. And in my last blog post, I talked about the dichotomy between smart databases and smart applications. I asked whether we can have smart databases that can scale on the cloud. So now let's talk about why I think the answer is yes.

MORE LIKE THIS

Simpler applications and smarter databases, Part 2

ise and payoff of
rt 1

T and the myth of
ntime

VIDEO

Does the Cloud Free up the

Worth reading, out of scope of course

The promise and payoff of NoSQL, Part 1
https://www.cio.com/article/3213165/data-management/the-promise-and-payoff-of-nosql-part-1.html#tk.drr_mlt
Simpler applications and smarter databases, Part 2
https://www.cio.com/article/3215084/software/simpler-applications-and-smarter-databases-part-2.html
Back to the (SQL) Future, Part 3
https://www.cio.com/article/3218105/sql/back-to-the-sql-future-part-3.html

Ajay Kulkarni Follow
Co-Founder/CEO @timescaledb. Ex @GroupMe, @Sensobi, @Skype, @Microsoft x2, @MIT x3, @MITSloan, others. Perpetual optimist.
Sep 26, 2017 · 12 min read

Why SQL is beating NoSQL, and what this means for the future of data

After years of being left for dead, SQL today is making a comeback. How come? And what effect will this have on the data community?

(Update: #1 on Hacker News! Read the discussion here.)

(Update 2: TimescaleDB is hiring! Open positions in Engineering, Marketing/Evangelism, and Office Management. Interested?)

Worth reading, out of scope of course

Why SQL is beating NoSQL, and what this means for the future of data
https://blog.timescale.com/why-sql-beating-nosql-what-this-means-for-future-of-data-time-series-database-348b777b847a

SQL awakens to fight the dark forces of NoSQL

Worth reading, out of scope of course

Don't Believe the Hype, It's a SQL: A Brief Guide to the NoSQL Landscape

Let's explore the evolution of NoSQL development from the early days of SQL and what the database ecosystem looks like today.

by Cliff Hall MVB · Apr. 04, 18 · Database Zone · Opinion

SQL: It's so last century, amirite?

Although it was created around ten years earlier, Structured Query Language (SQL) really began to catch hold in the commercial world during the mid-80s — and brought with it some great improvements over what had come before.

Probably the most important feature was transactionality, which enabled concurrency and rollback capabilities. SQL also lent itself to ad-hoc business reporting since queries could be easily written by sales and other departments to gain fresh insights into their company's data without application programming.

Don't Believe the Hype, It's a SQL: A Brief Guide to the NoSQL Landscape
https://dzone.com/articles/dont-believe-the-hype-its-a-sql-a-brief-guide-to-t

Worth reading, out of scope of course

Don't Believe the Hype, It's a SQL: A Brief Guide to the NoSQL Landscape

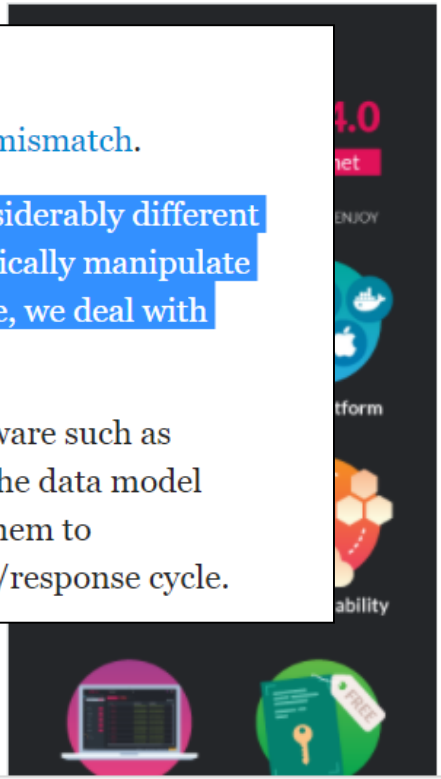
Let's explore the evolution of NoSQL development from the early days of SQL and what the database ecosystem looks like today.

1. Data Isn't Stored in the Same Shape as It Is Used

One of the biggest hassles with SQL is what is called object-relational impedance mismatch.

In short, the way we represent and reason about data in our user interfaces is considerably different from the way we store that data in a relational database. With modern UIs, we typically manipulate hierarchies of objects, often without a defined schema. But in a relational database, we deal with tables of data related to each other by keys. Schemas are not optional.

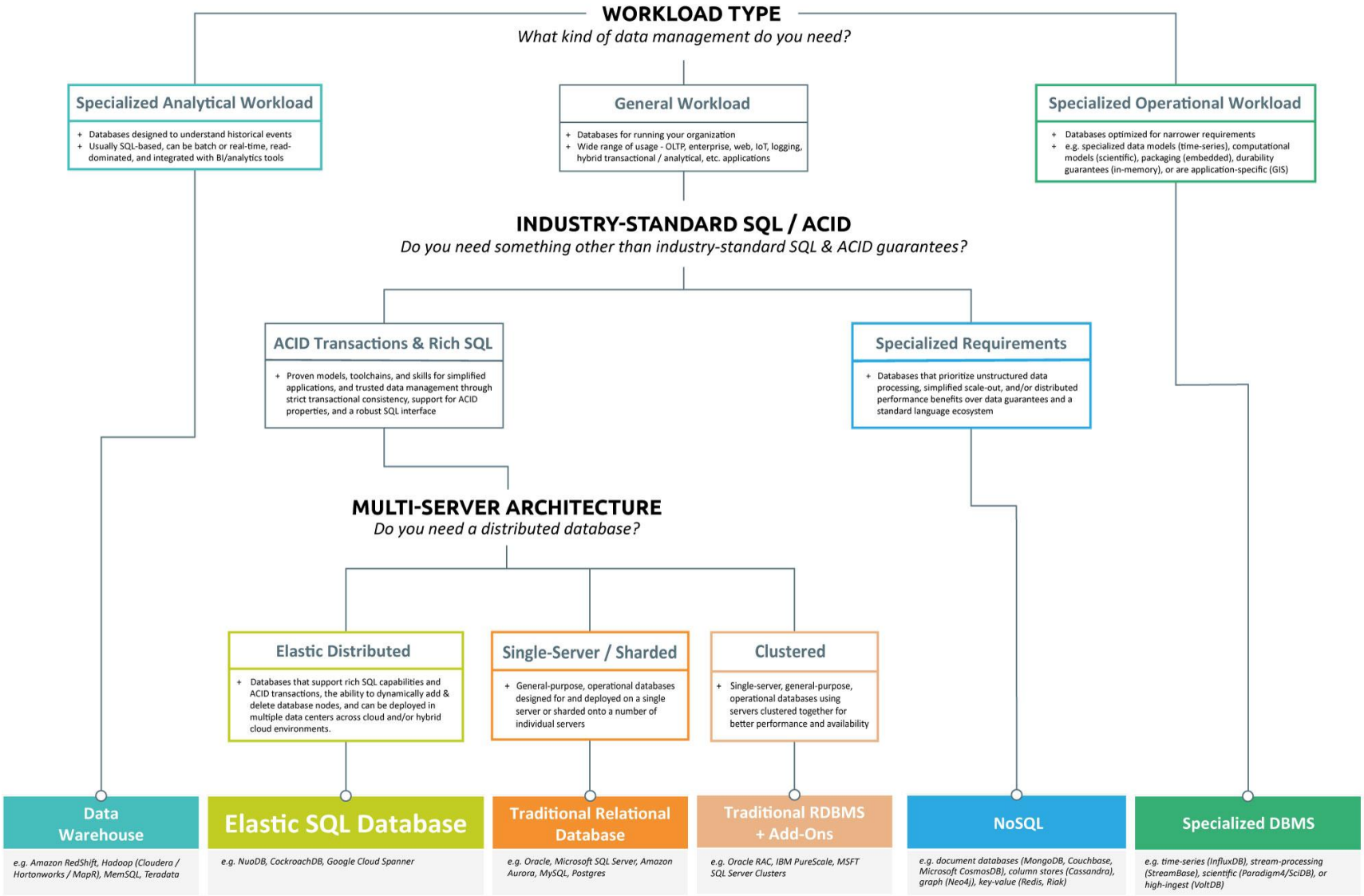
This disparity in data organization gives rise to object-relational mapping middleware such as Hibernate. Along with that comes extra management overhead, since changes to the data model require changes not only to the UI and DB but also to the ORM layer that allows them to communicate. It can also add a certain amount of processing time to each request/response cycle.



sales and other departments to gain fresh insights into their company's data without application programming.

Don't Believe the Hype, It's a SQL: A Brief Guide to the NoSQL Landscape
https://dzone.com/articles/dont-believe-the-hype-its-a-sql-a-brief-guide-to-t

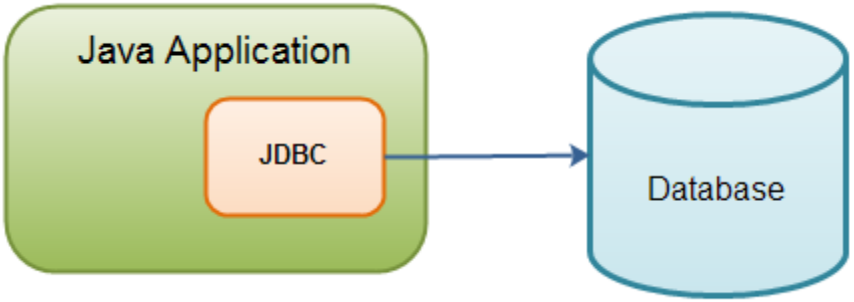
CHOOSING A DATABASE



https://www.nuodb.com/sites/default/files/graphics/conceptual-diagrams/database-decision-tree_text_FINAL.jpg

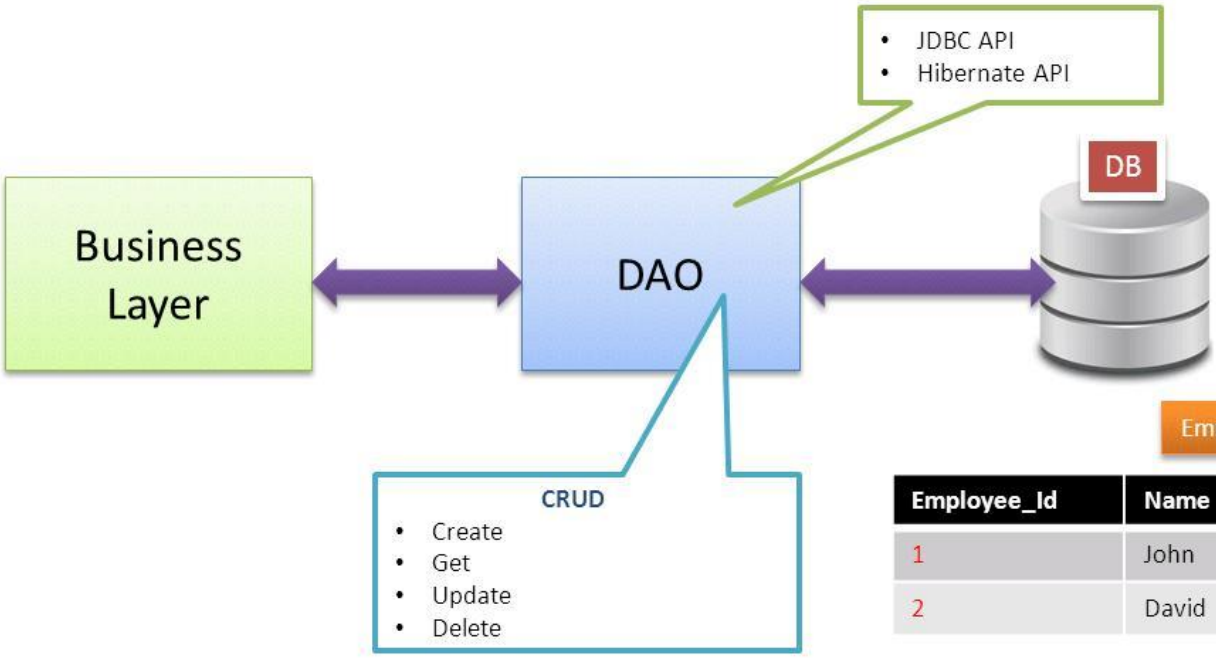
The relational model

- Typically supported on relational models
- Using SQL
- Traditional CRUD
- In java supported via JDBC



Data Access Object pattern

- ✓ Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services.
- ✓ DAO layer is responsible for Data access from the persistence storage[DB/LDAP/File system] and manipulation of Data in in the persistence storage
- ✓ Decouple the persistent storage implementation from the rest of your application.



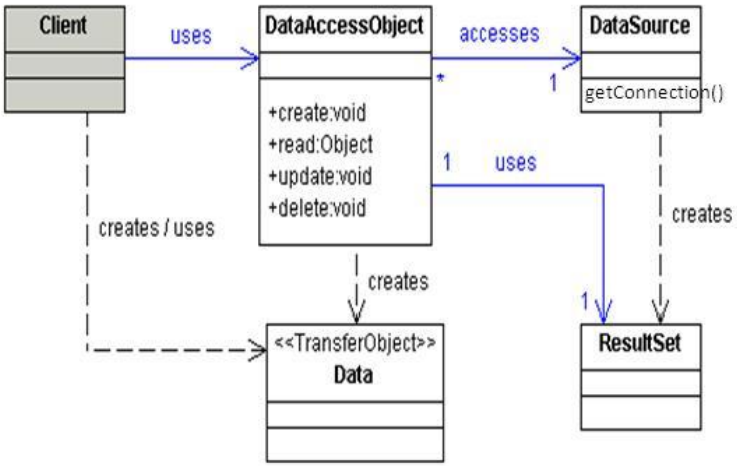
Employee

Employee_Id	Name	Age
1	John	28
2	David	39

http://ramj2ee.blogspot.pt/2013/08/data-access-object-design-pattern-or.html#.WLmZVzvyjb0

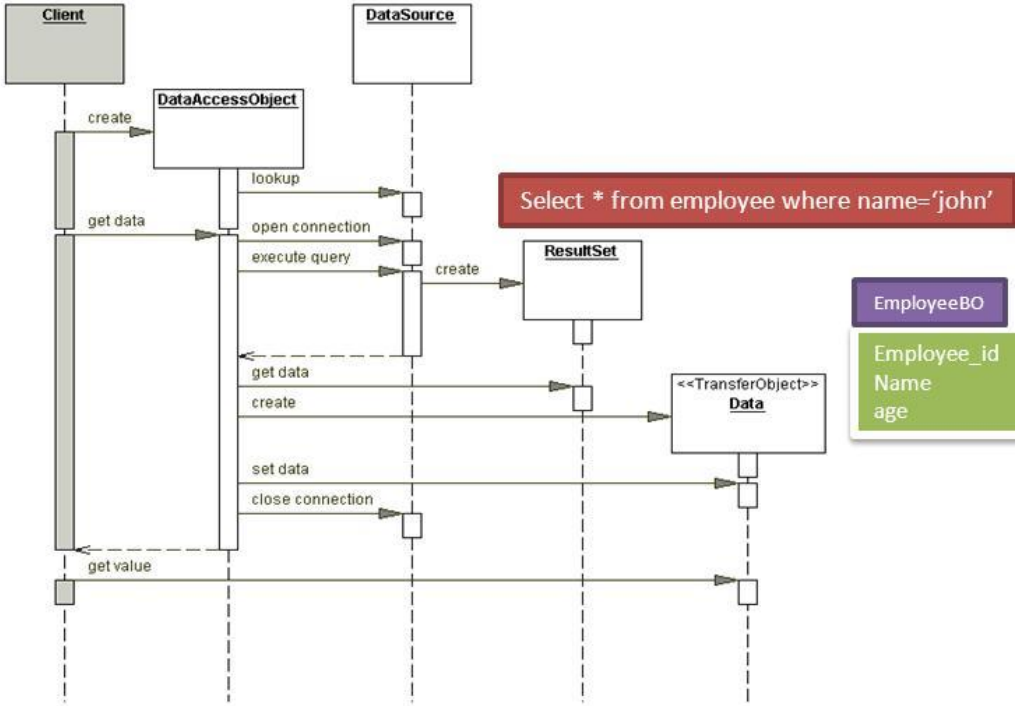
Data Access Object pattern

- ✓ Data sources are the resources that provide connections to your relational database.
- ✓ An SQL result set is a set of rows from a database.

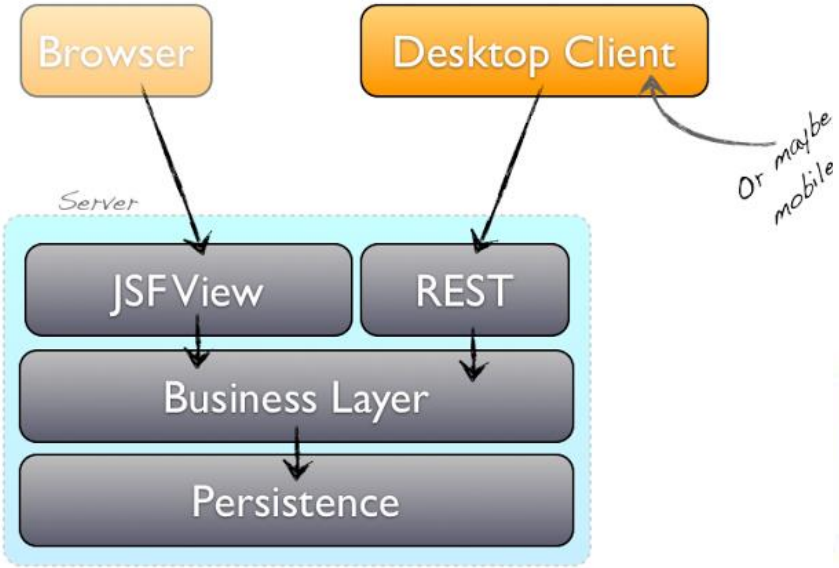


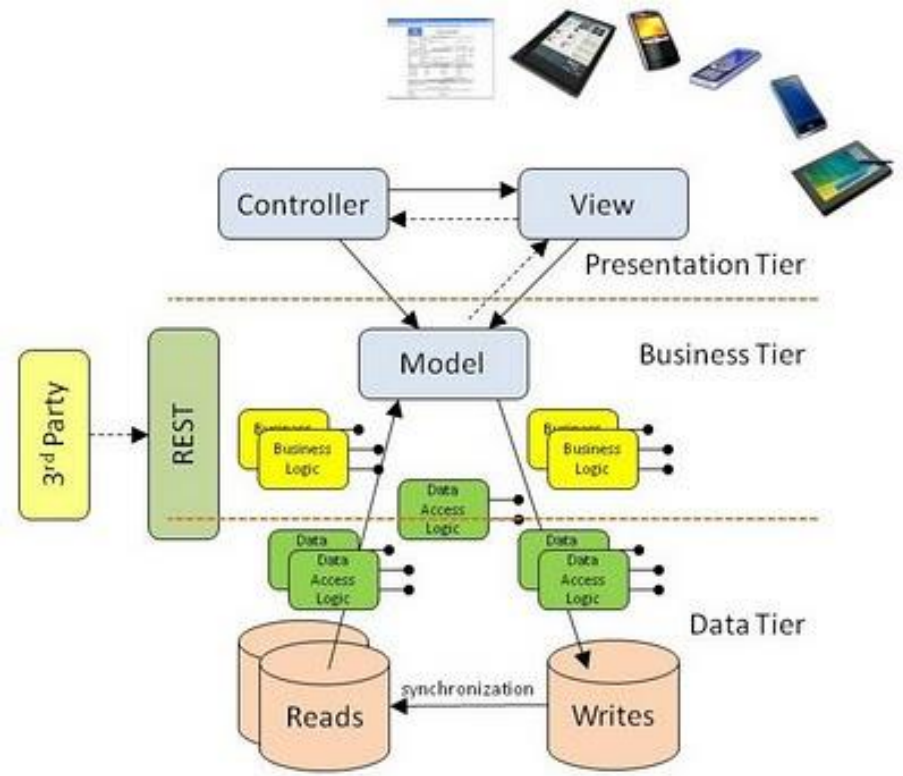
Employee

Employee_Id	Name	Age
1	John	28
2	David	39



Enough to simple architectures





Presentation tier

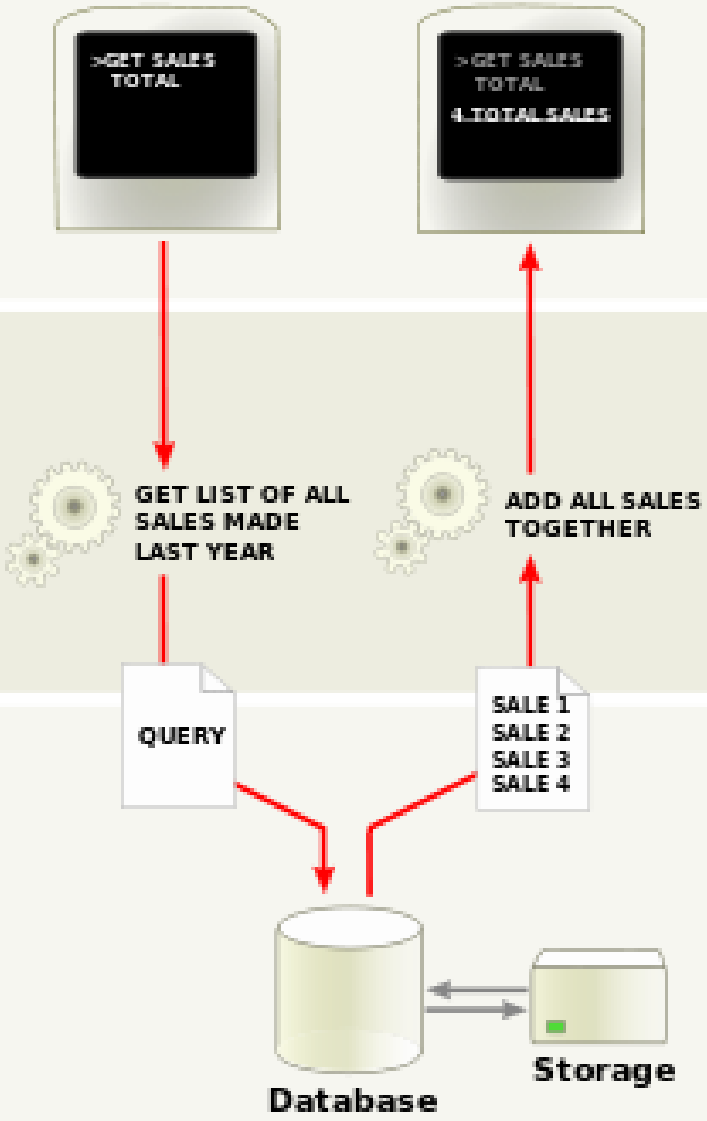
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

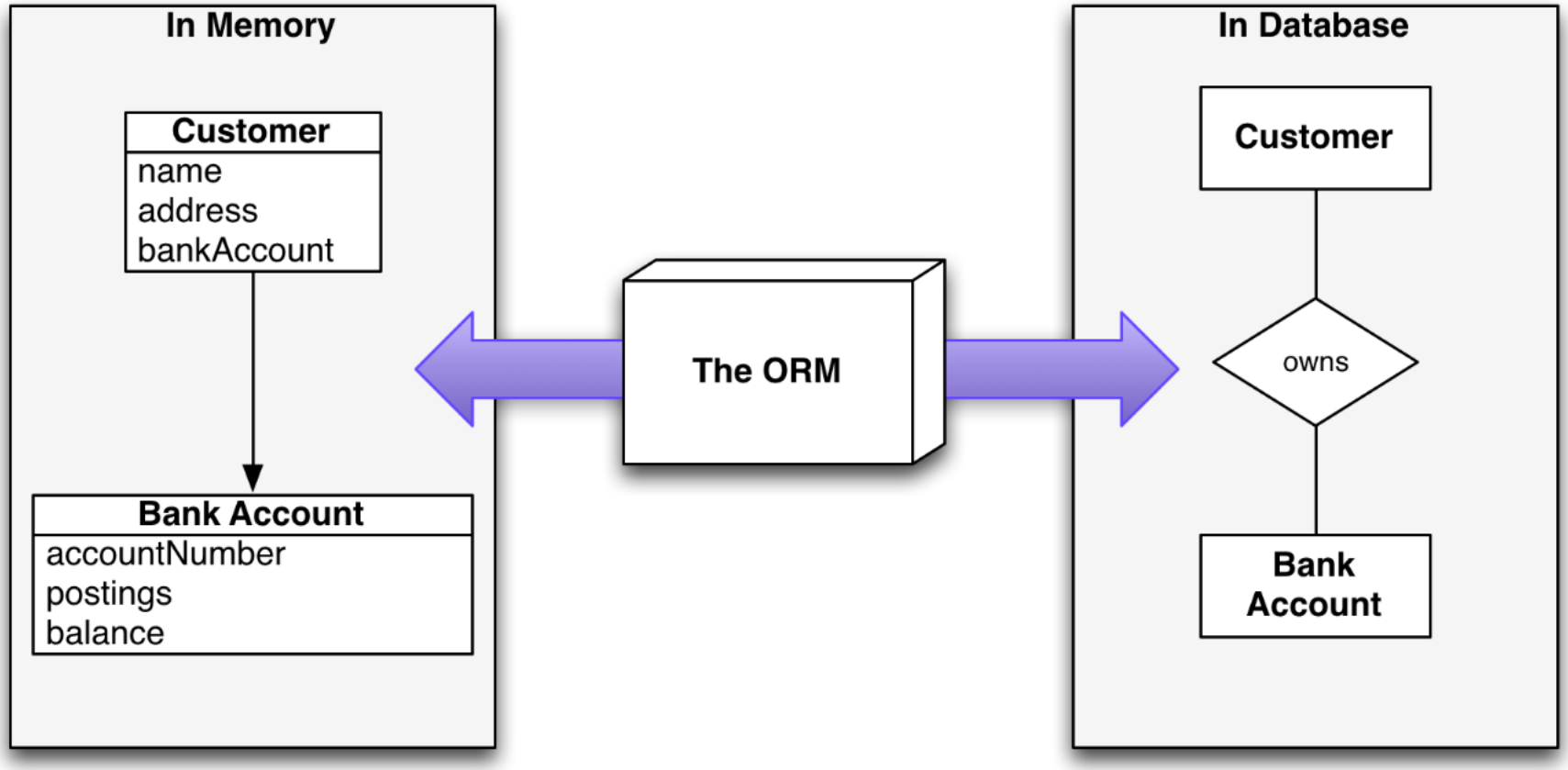
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



Relational is not OO

- OO languages are pervasive
- Need domain model to decouple from relational model
  - Concepts should no depend on representation model
- Object to Relational Mapping (ORM)
  - A solution
- Java Persistence API (API) is java ORM
  - An interface + annotations
  - Semantics & patterns
    - Factory, façade, …
  - Implementations should comply
  - A Factor

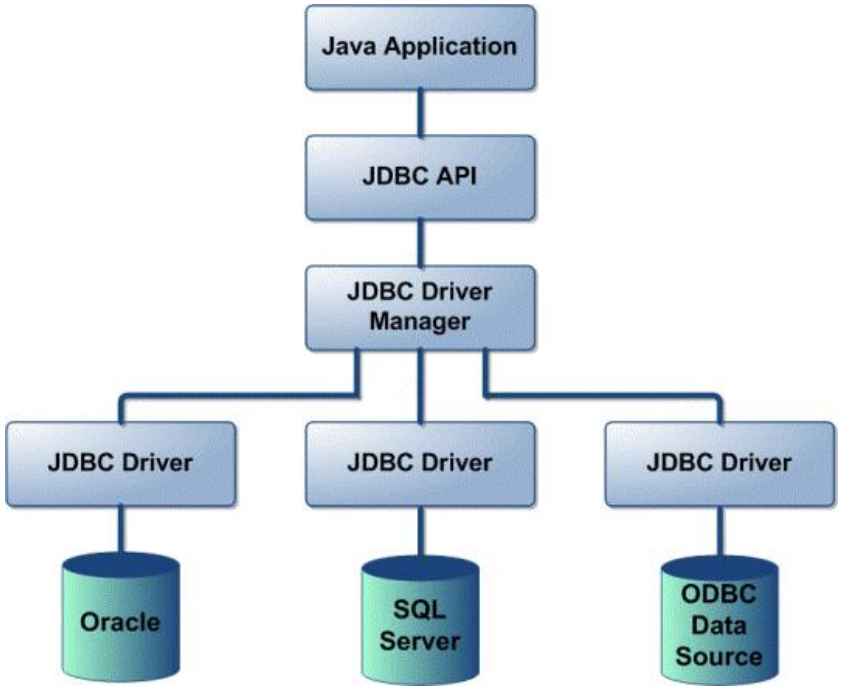
Active record, traditional CRUD

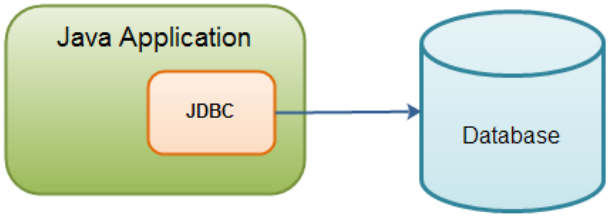


Event Sourcing in practice
https://ookami86.github.io/event-sourcing-in-practice/#title.md

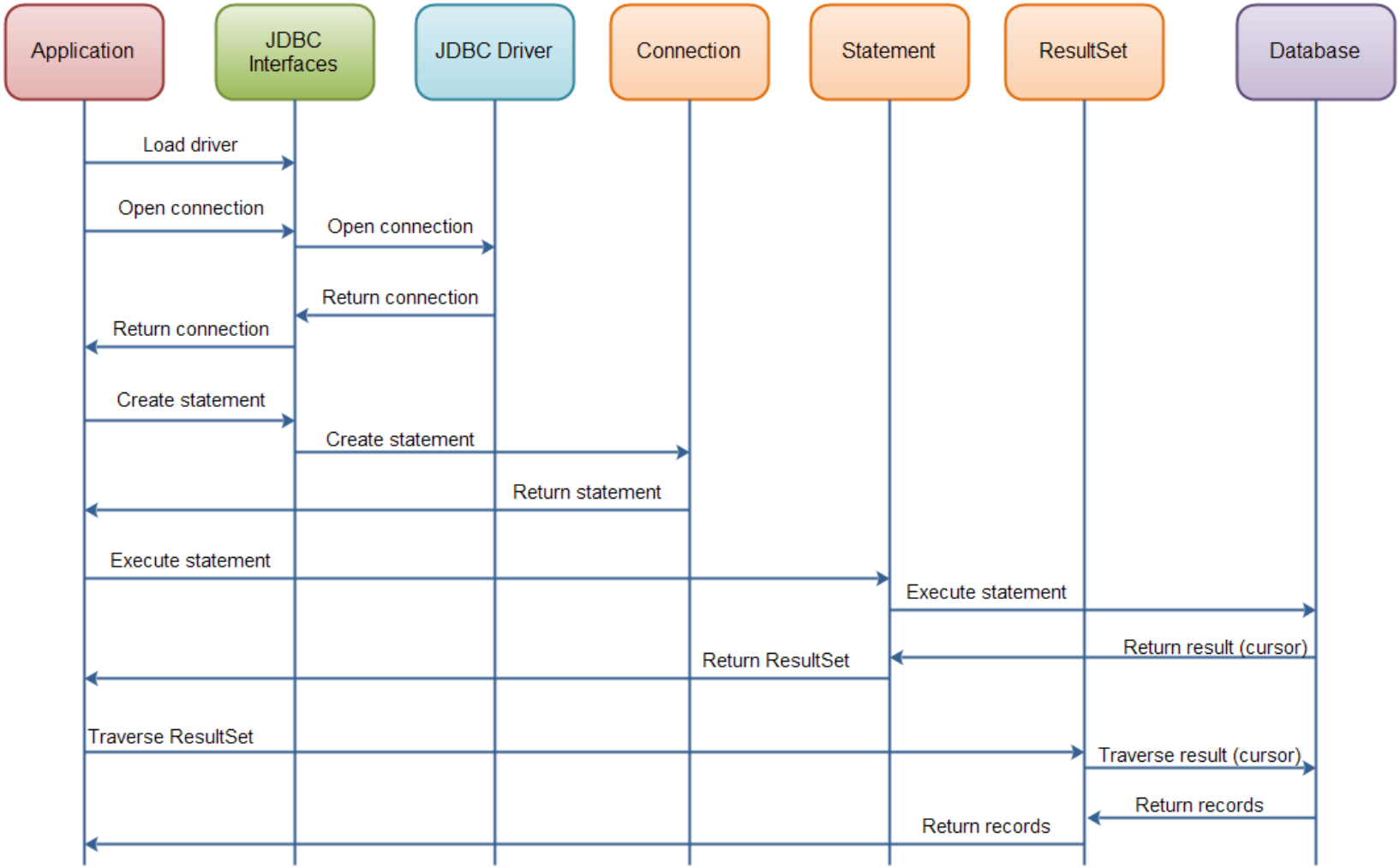
JDBC: java relational driver

- Although not the focus of the course, some details on the java ODBC driver (JDBC) are useful for configuring the drivers and connect to relational databases





JDBC



http://tutorials.jenkov.com/jdbc/overview.html

May need to configure driver…


```java
public class MySQLAccess {
  private Connection connect = null;
  private Statement statement = null;
  private PreparedStatement preparedStatement = null;
  private ResultSet resultSet = null;

  public void readDataBase() throws Exception {
    try {
      // This will load the MySQL driver, each DB has its own driver
      Class.forName("com.mysql.jdbc.Driver");
      // Setup the connection with the DB
      connect = DriverManager
          .getConnection("jdbc:mysql://localhost/feedback?"
              + "user=sqluser&password=sqluserpw");

      // Statements allow to issue SQL queries to the database
      statement = connect.createStatement();
      // Result set get the result of the SQL query
      resultSet = statement
          .executeQuery("select * from feedback.comments");
      writeResultSet(resultSet);
```

http://www.vogella.com/tutorials/MySQLJava/article.html

… to use JPA with relational databases


```java
public class MySQLAccess {
  private Connection connect = null;
  private Statement statement = null;
  private PreparedStatement preparedStatement = null;
  private ResultSet resultSet = null;

  public void readDataBase() throws Exception {
    try {
      // This will load the MySQL driver, each DB has its own driver
      Class.forName("com.mysql.jdbc.Driver");
      // Setup the connection with the DB
      connect = DriverManager
          .getConnection("jdbc:mysql://localhost/feedback?"
              + "user=sqluser&password=sqluserpw");

      // Statements allow to issue SQL queries to the database
      statement = connect.createStatement();
      // Result set get the result of the SQL query
      resultSet = statement
          .executeQuery("select * from feedback.comments");
      writeResultSet(resultSet);
```

http://www.vogella.com/tutorials/MySQLJava/article.html

Some References

Java JDBC

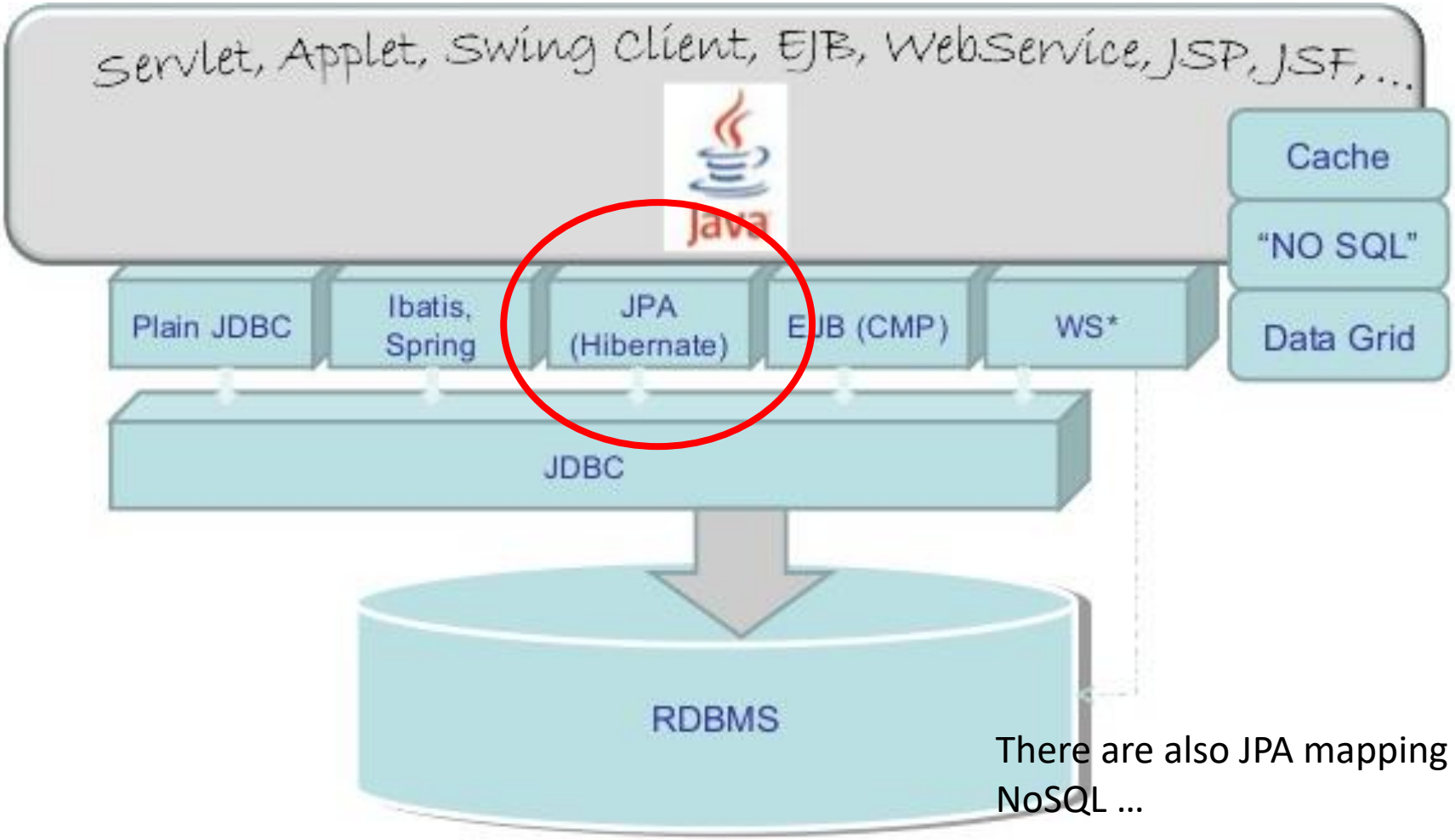
- Trail: JDBC(TM) Database Access
  - http://docs.oracle.com/javase/tutorial/jdbc/
- MySQL and Java JDBC - Tutorial
  - http://www.vogella.com/tutorials/MySQLJava/article.html
- Java JDBC
  - http://tutorials.jenkov.com/jdbc/index.html
- JDBC Tutorial
  - http://www.tutorialspoint.com/jdbc/
- JDBC Tutorial
  - http://www.mkyong.com/tutorials/jdbc-tutorials/

JPA: Java Persistence API

- API i.e. interfaces, NOT an implementation
- Basic methods/semantics for
  - Creating and managing data entities
  - defined in the javax.persistence package
  - the Java Persistence Query Language (JPQL)
- Mapping for relation and non relational models
  - object/relational metadata (ORM)
  - Recently to no-SQL persistence solutions
- JPA 2.0 is the work of the JSR 317 Expert Group.

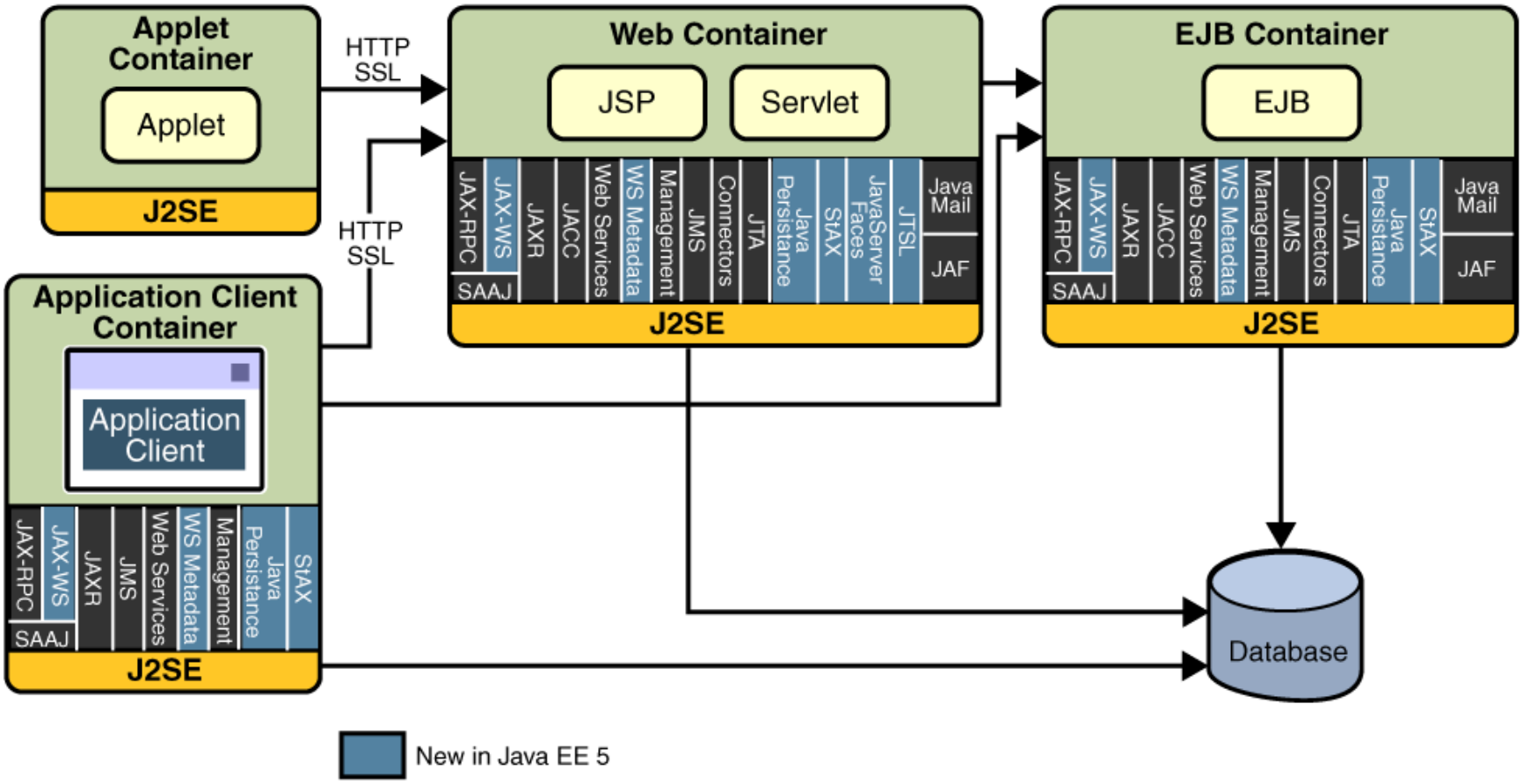
Chapter 32 - Introduction to the Java Persistence API @ Java Tutorial
http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html
http://en.wikipedia.org/wiki/Java_Persistence_API

Java applications and databases



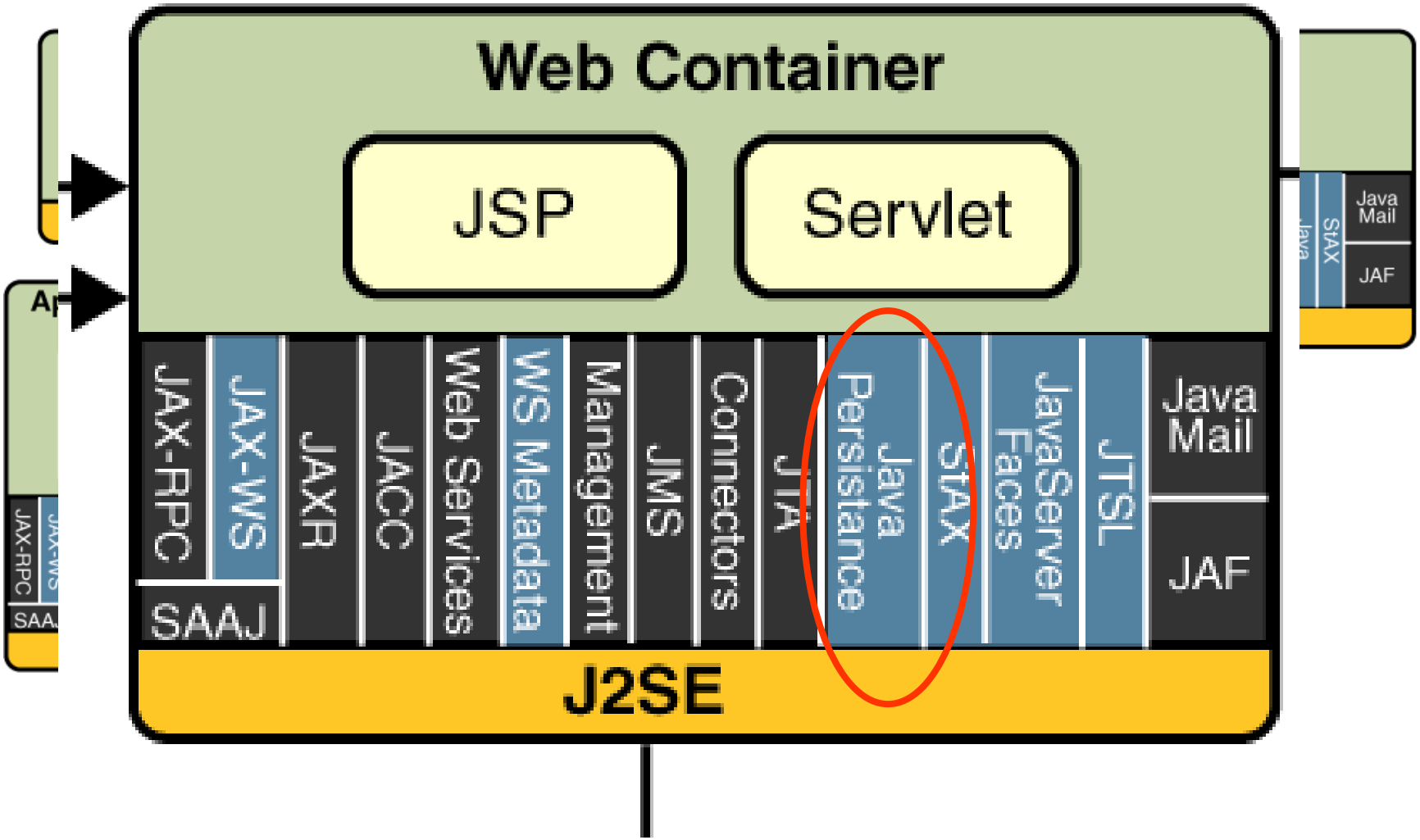
There are also JPA mapping to NoSQL …

JPA is supported on J2SE



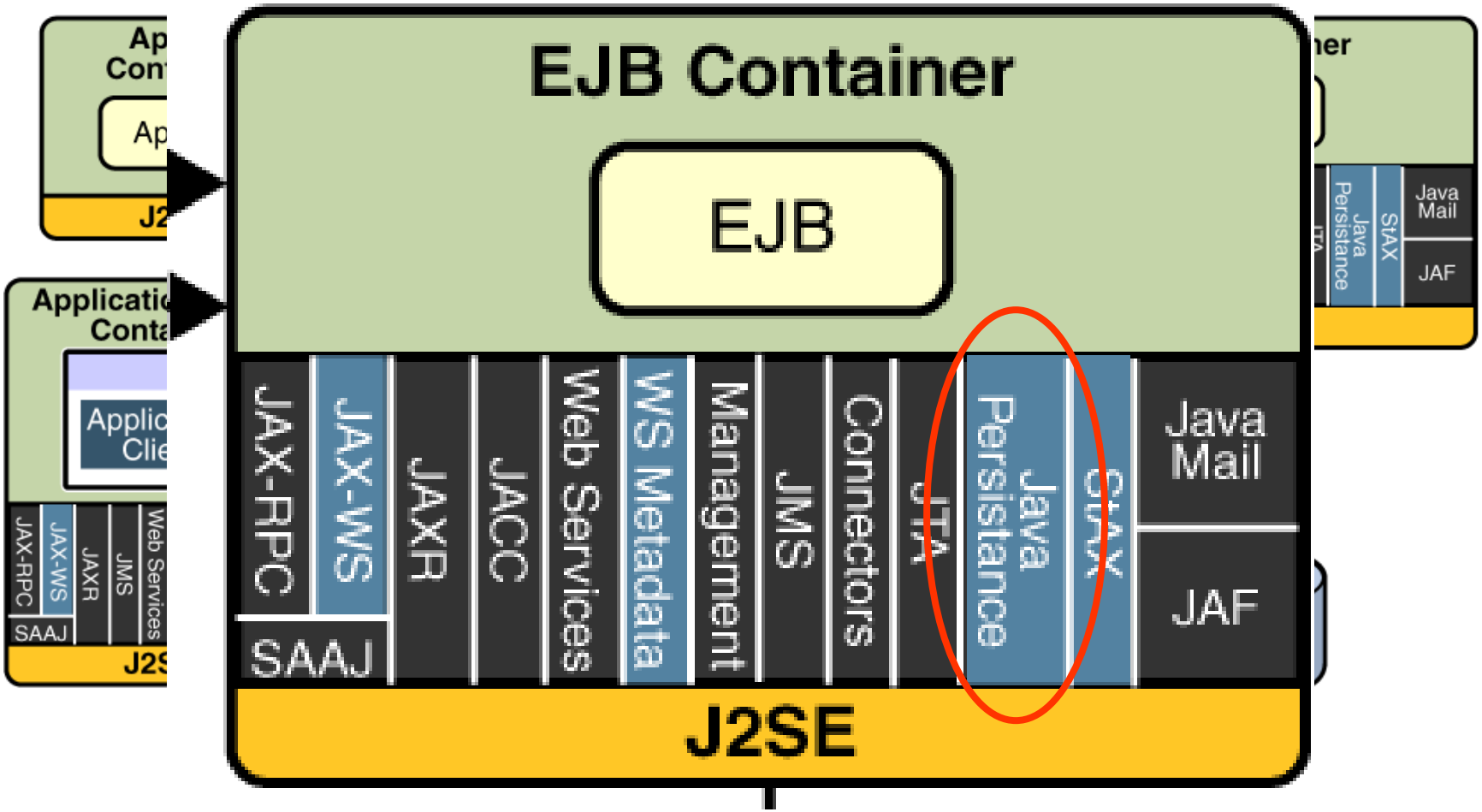
http://java.sun.com/javaee/5/docs/tutorial/doc/

JPA is supported on J2SE



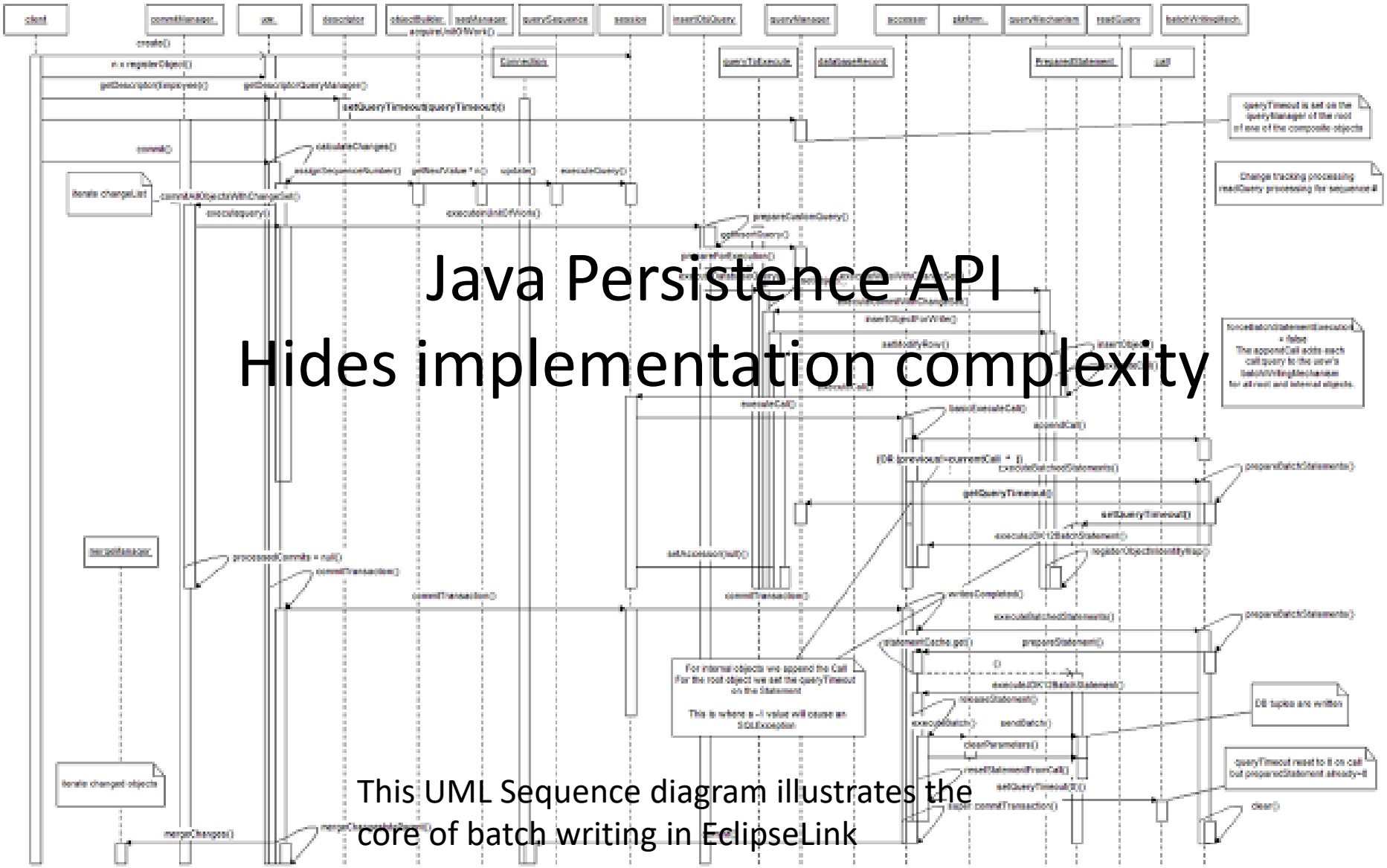
http://java.sun.com/javaee/5/docs/tutorial/doc/

JPA is supported on J2SE



http://java.sun.com/javaee/5/docs/tutorial/doc/

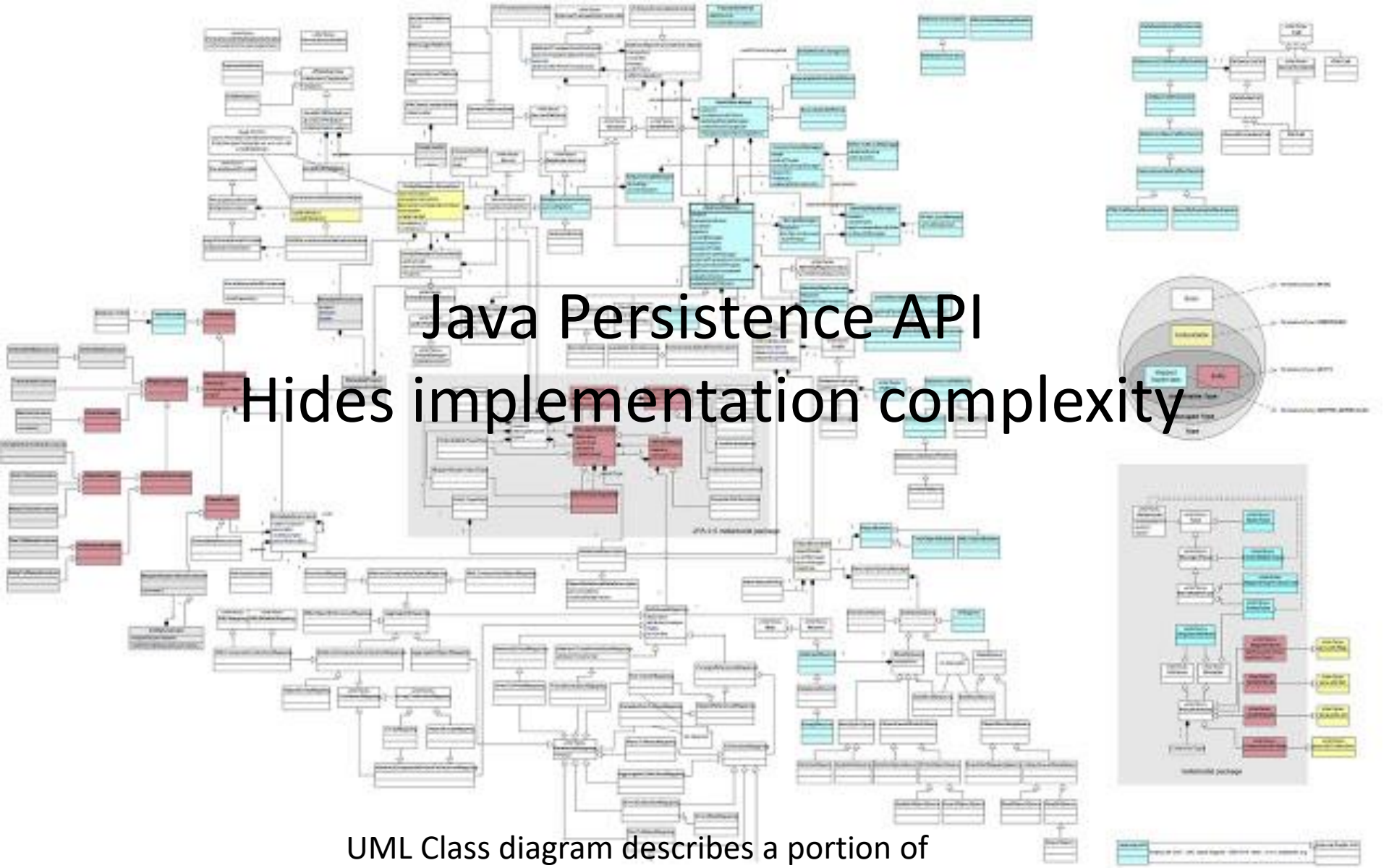
Java Persistence API
Hides implementation complexity



This UML Sequence diagram illustrates the core of batch writing in EclipseLink

http://eclipsejpa.blogspot.pt/2010/11/architecture.html

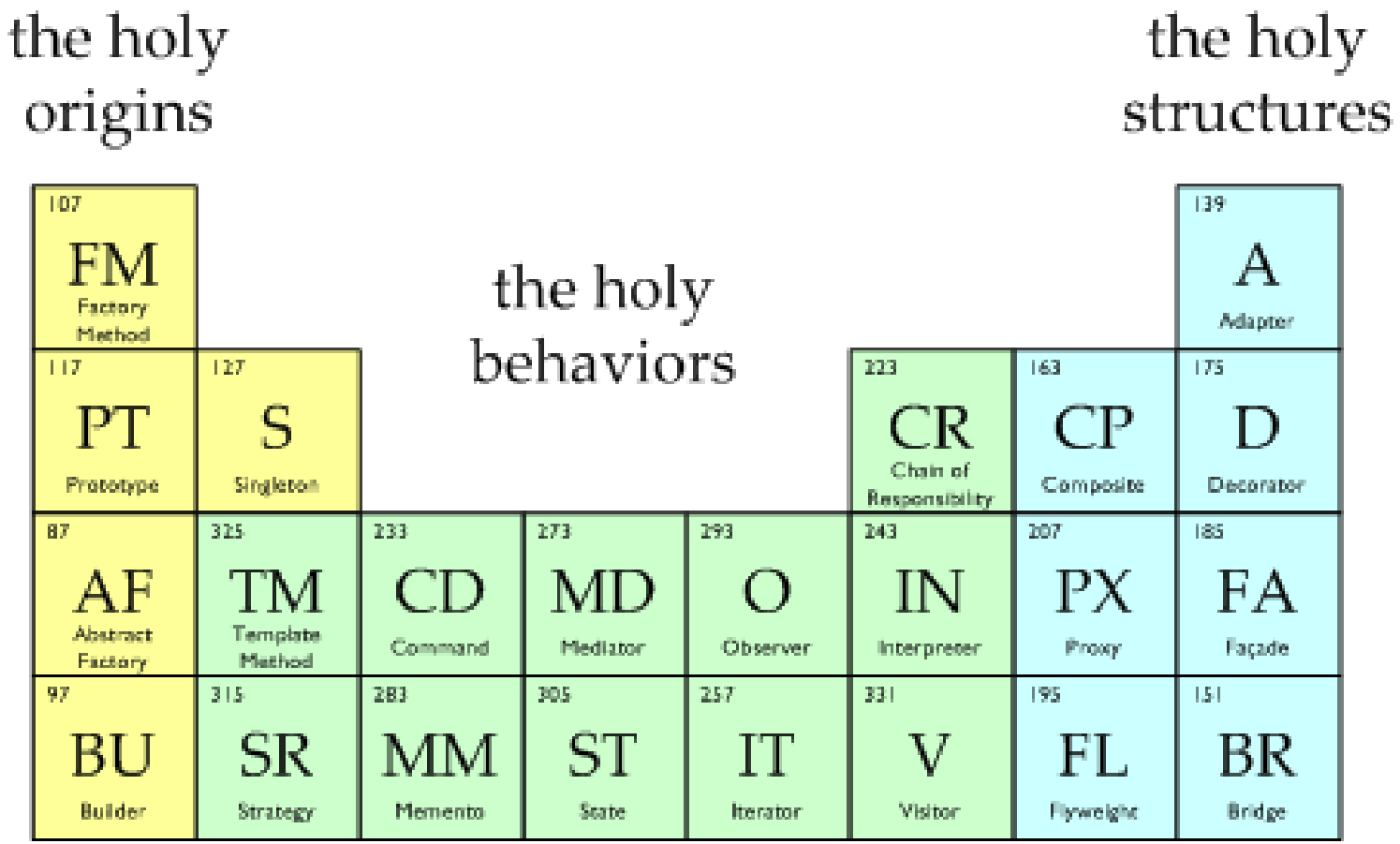
Java Persistence API
Hides implementation complexity



UML Class diagram describes a portion of the [EclipseLink](#) 2.0 API

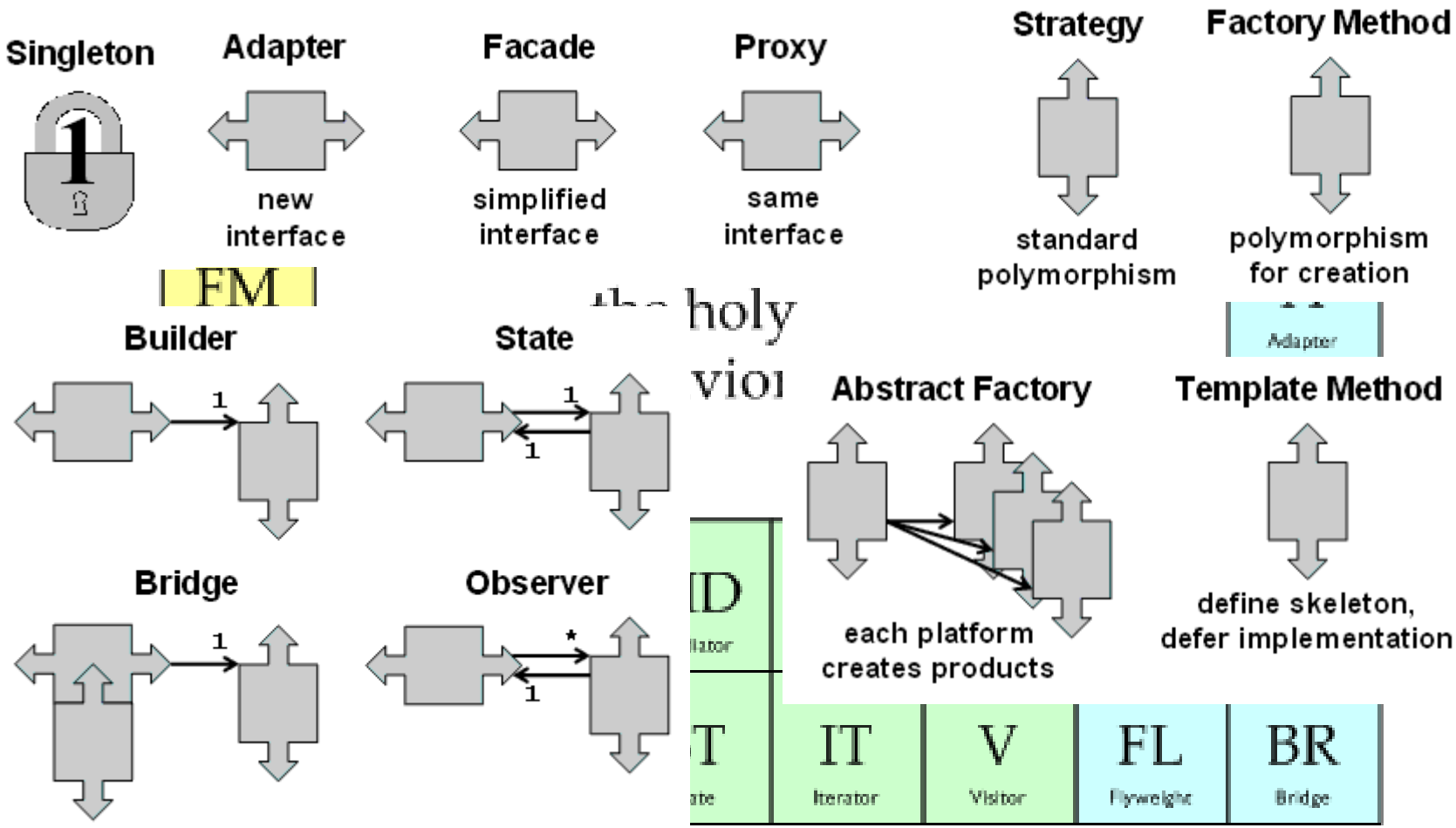
http://eclipsejpa.blogspot.pt/2010/11/architecture.html

The Sacred Elements of the Faith



http://www.vincehuston.org/dp/

The Sacred Elements of the Faith

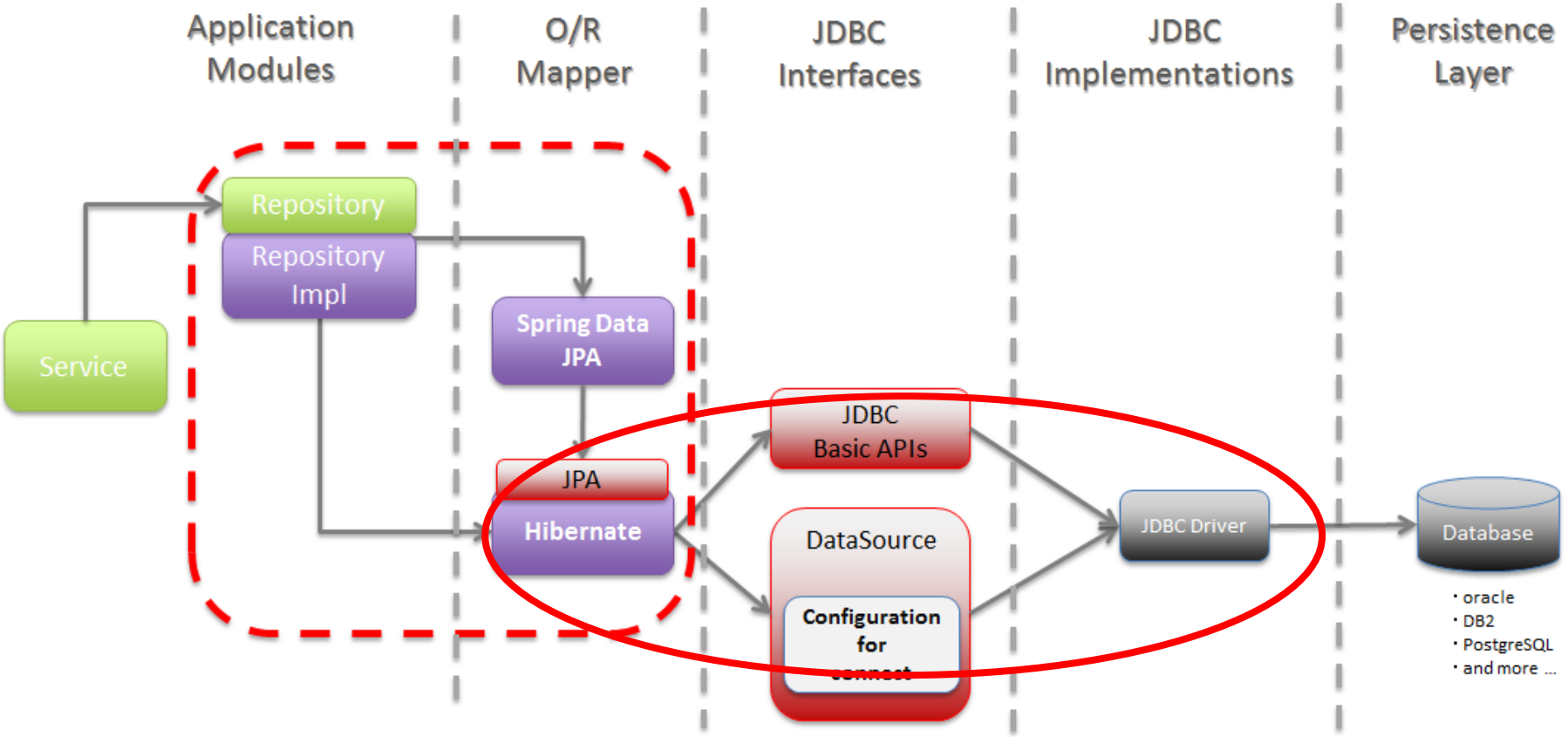


http://www.vincehuston.org/dp/

Some relevant details

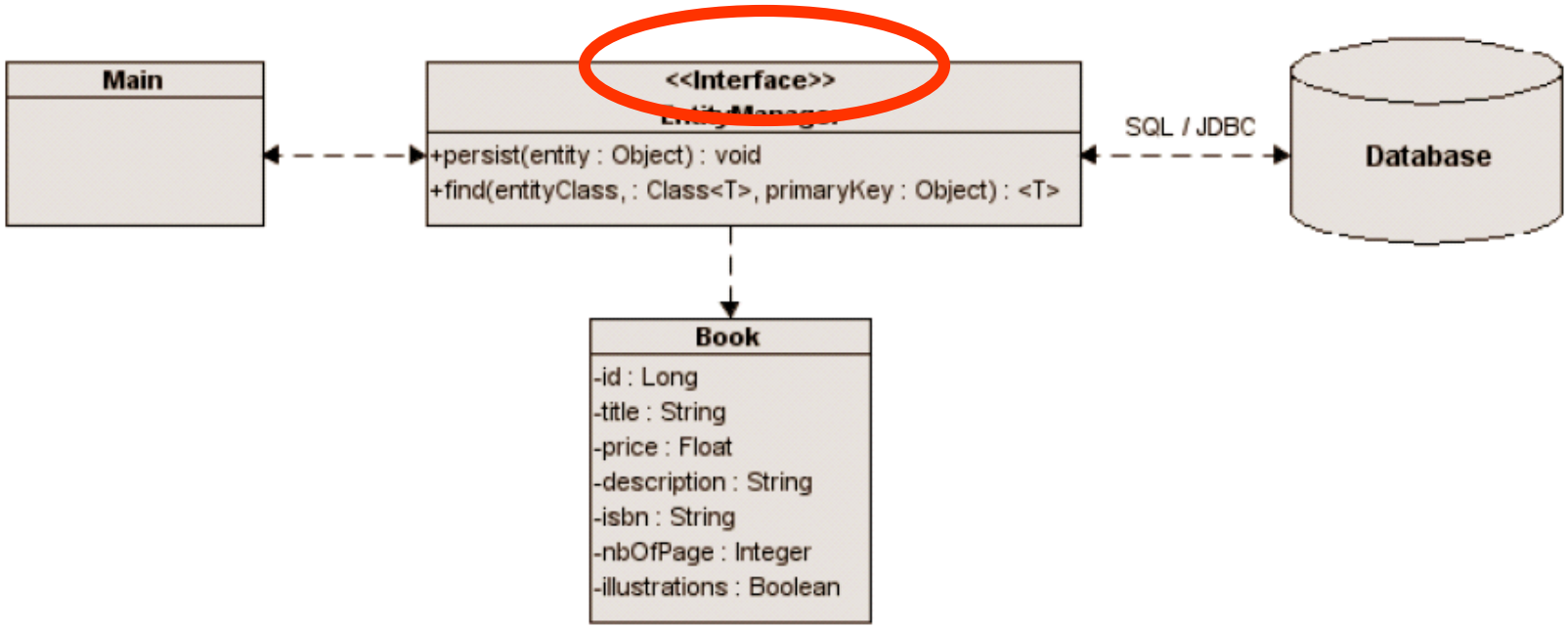
- Several detailed tutorials
  - JPA Tutorial - JPA Introduction
    - http://www.java2s.com/Tutorials/Java/JPA/index.htm
  - Tutorial points
    - https://www.tutorialspoint.com/jpa/
  - See “Persistence in Java” notes https://goo.gl/ERgWXY
- Focus on architecture and rationale
  - Persistence context and EntityManager
- ORM, ODM , OGM ….
  - Entities
  - Relation
  - queries
  - Fecthing
  - cascading

Persistence context and EntityManager



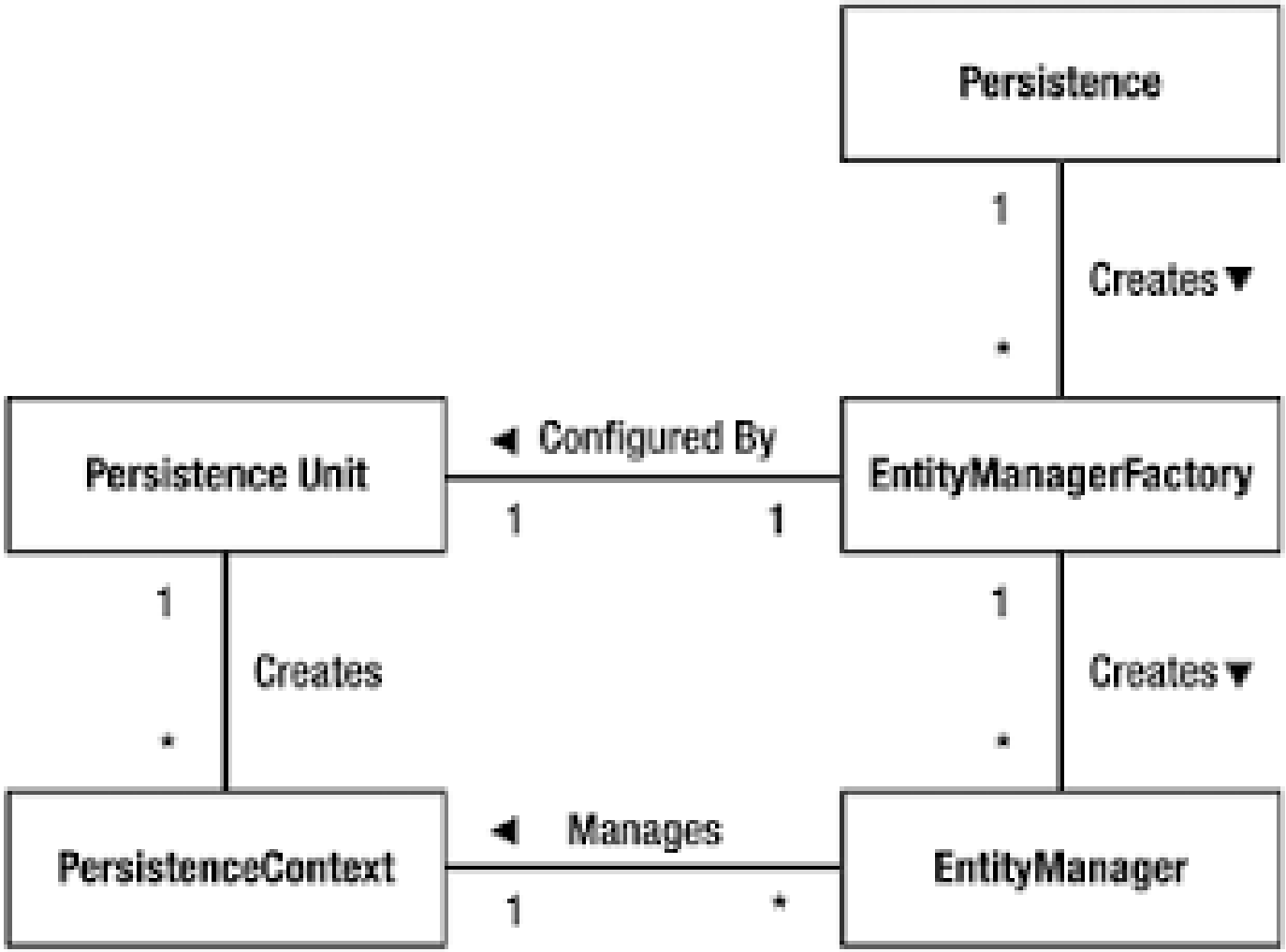
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.architecture

JPA EntityManager interfaces

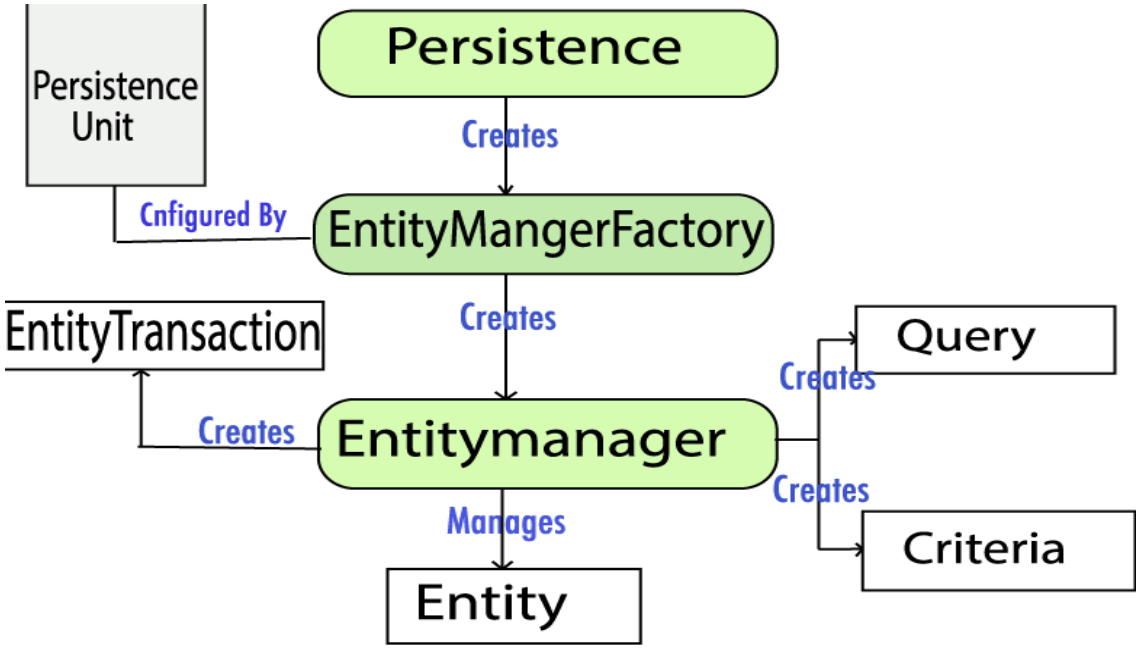



```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

Factories and context

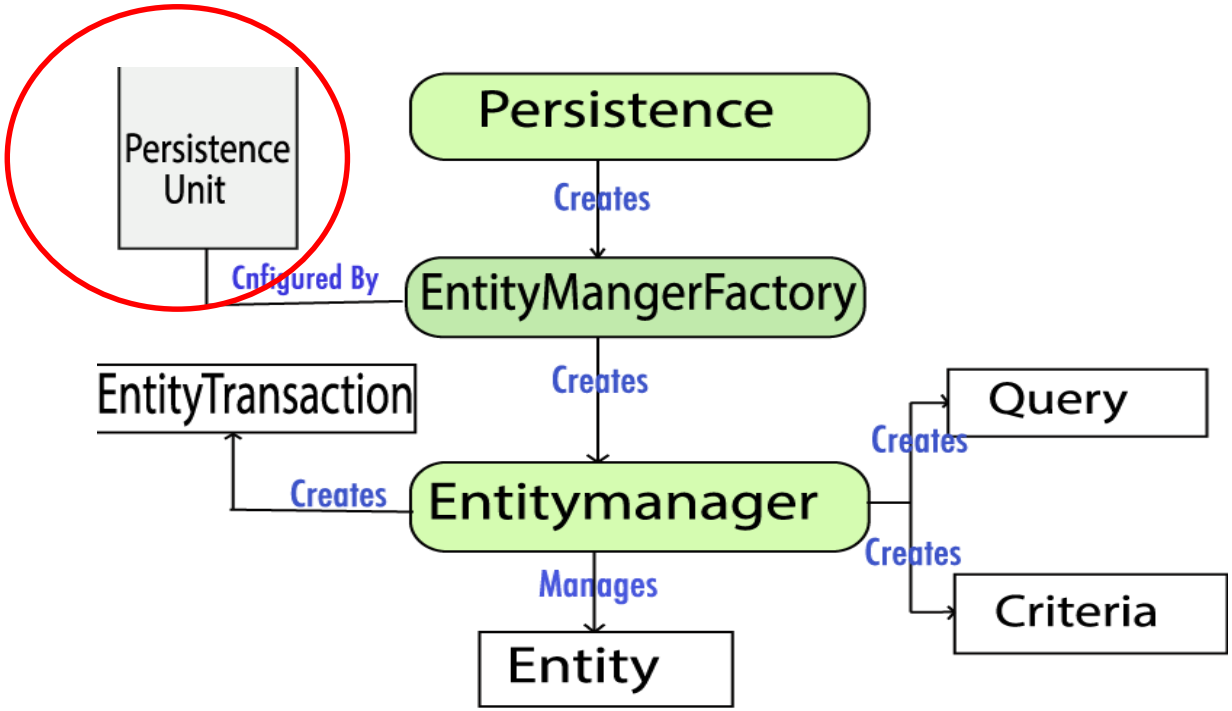


Factories and context



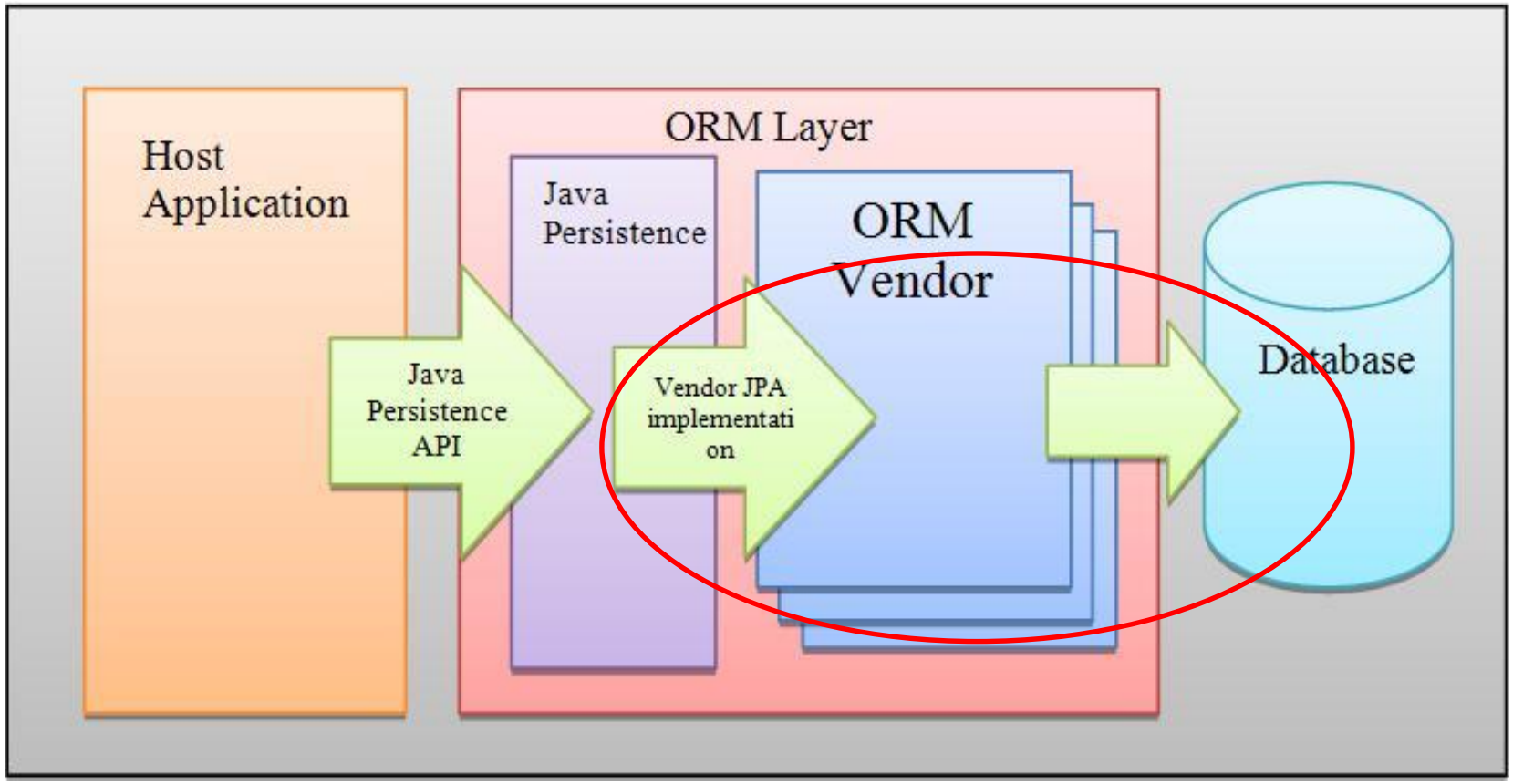

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

Factories and context




```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

Persistence Unit/Context



http://www.npersistence.org/

Several supporting frameworks

http://www.eclipse.org/eclipselink/

http://www.datanucleus.org/

Vendors supporting JPA 2.0

- BatooJPA
- DataNucleus (formerly JPOX)
- EclipseLink (formerly Oracle TopLink)
- JBoss Hibernate
- ObjectDB
- OpenJPA
- IBM, via its OpenJPA-based Feature Pack for OSGi Applications and JPA 2.0 for WebSphere Application Server
- Versant JPA (not relational, object database)

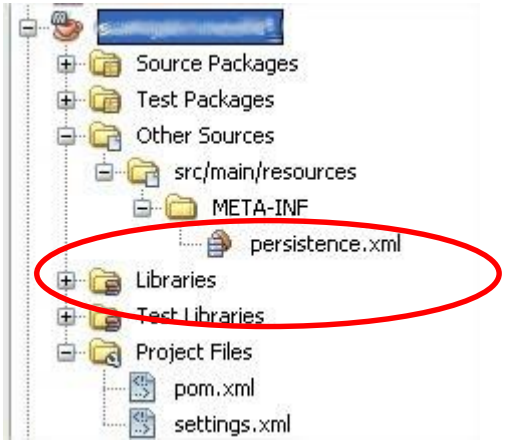
http://www.hibernate.org/

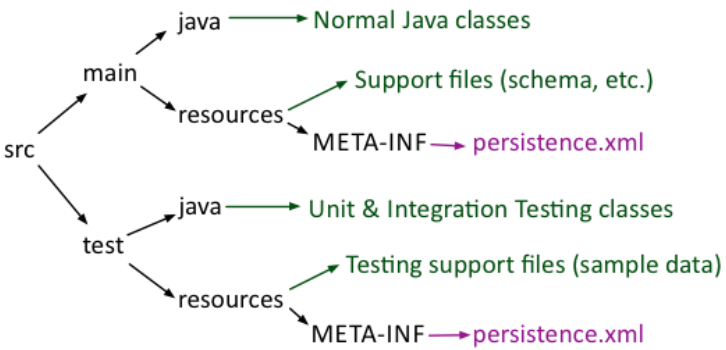
http://openjpa.apache.org/

http://www.objectdb.com/

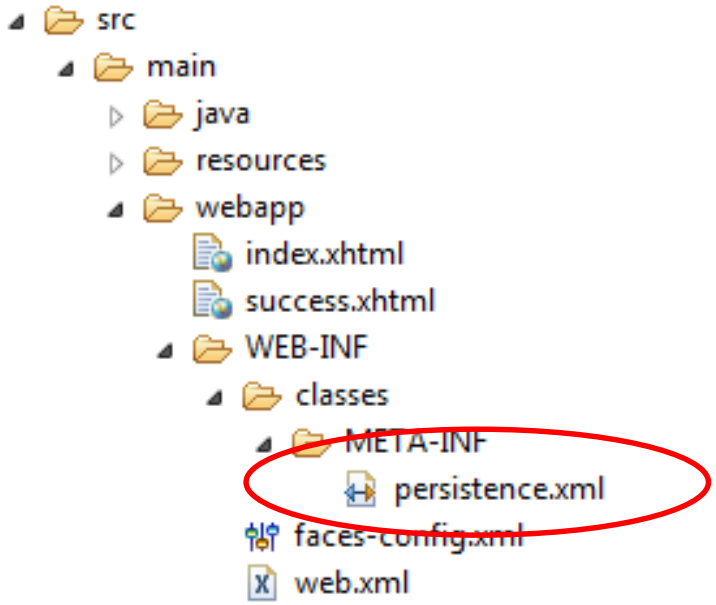
Persistence.xml

Java application





Web application



https://www.tutorialspoint.com/jpa/jpa_entity_managers.htm
http://www.objectdb.com/java/jpa/entity/persistence-unit

Persistence.xml: configuring the persistence context

```
<persistence>
   <persistence-unit name="EmployeeService" transaction-
   type="RESOURCE_LOCAL">
     <class>examples.model.Employee</class>
     <properties>
       <property name="toplink.jdbc.driver"
                 value="org.apache.derby.jdbc.ClientDriver"/>
       <property name="toplink.jdbc.url"
        value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
       <property name="toplink.jdbc.user" value="APP"/>
       <property name="toplink.jdbc.password" value="APP"/>
     </properties>
   </persistence-unit>
</persistence>
```

With Netbeans most of configurations can be done using the IDE

Persistence.xml: Configuring a persistence-unit


```
<persistence>
   <persistence-unit name="EmployeeService" transaction-
   type="RESOURCE_LOCAL">
     <class>examples.model.Employee</class>
     <properties>
       <property name="toplink.jdbc.driver"
                 value="org.apache.derby.jdbc.ClientDriver"/>
       <property name="toplink.jdbc.url"
        value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
       <property name="toplink.jdbc.user" value="APP"/>
       <property name="toplink.jdbc.password" value="APP"/>
     </properties>
   </persistence-unit>
</persistence>
```

With Netbeans most of configurations can be done using the IDE

JUST a quick stop

Persistence.xml: Configuring a persistence-unit

TRANSACTIONS

```
<persistence>
    <persistence-unit name="EmployeeService" transaction-
    type="RESOURCE_LOCAL">
      <class>examples.model.Employee</class>
      <properties>
        <property name="toplink.jdbc.driver"
                  value="org.apache.derby.jdbc.ClientDriver"/>
        <property name="toplink.jdbc.url"
         value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
        <property name="toplink.jdbc.user" value="APP"/>
        <property name="toplink.jdbc.password" value="APP"/>
      </properties>
    </persistence-unit>
</persistence>
```

With Netbeans most of configurations can be done using the IDE

Remember?

vogella Tutorials Services Products Books Company Donate Contact us Search

JPA 2.0 with EclipseLink - Tutorial

Lars Vogel

Version 2.2

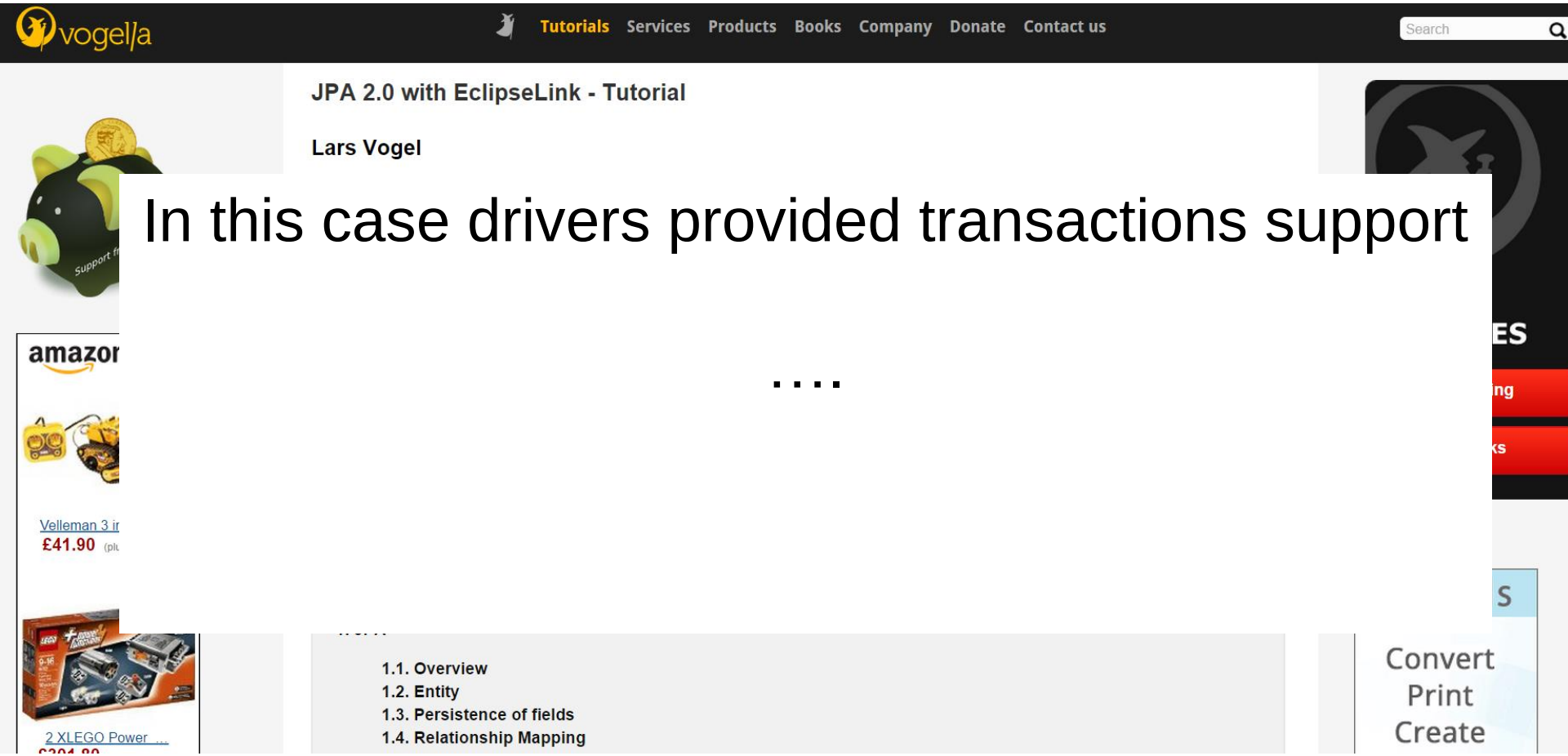
16.03.2012

JPA with Eclipselink

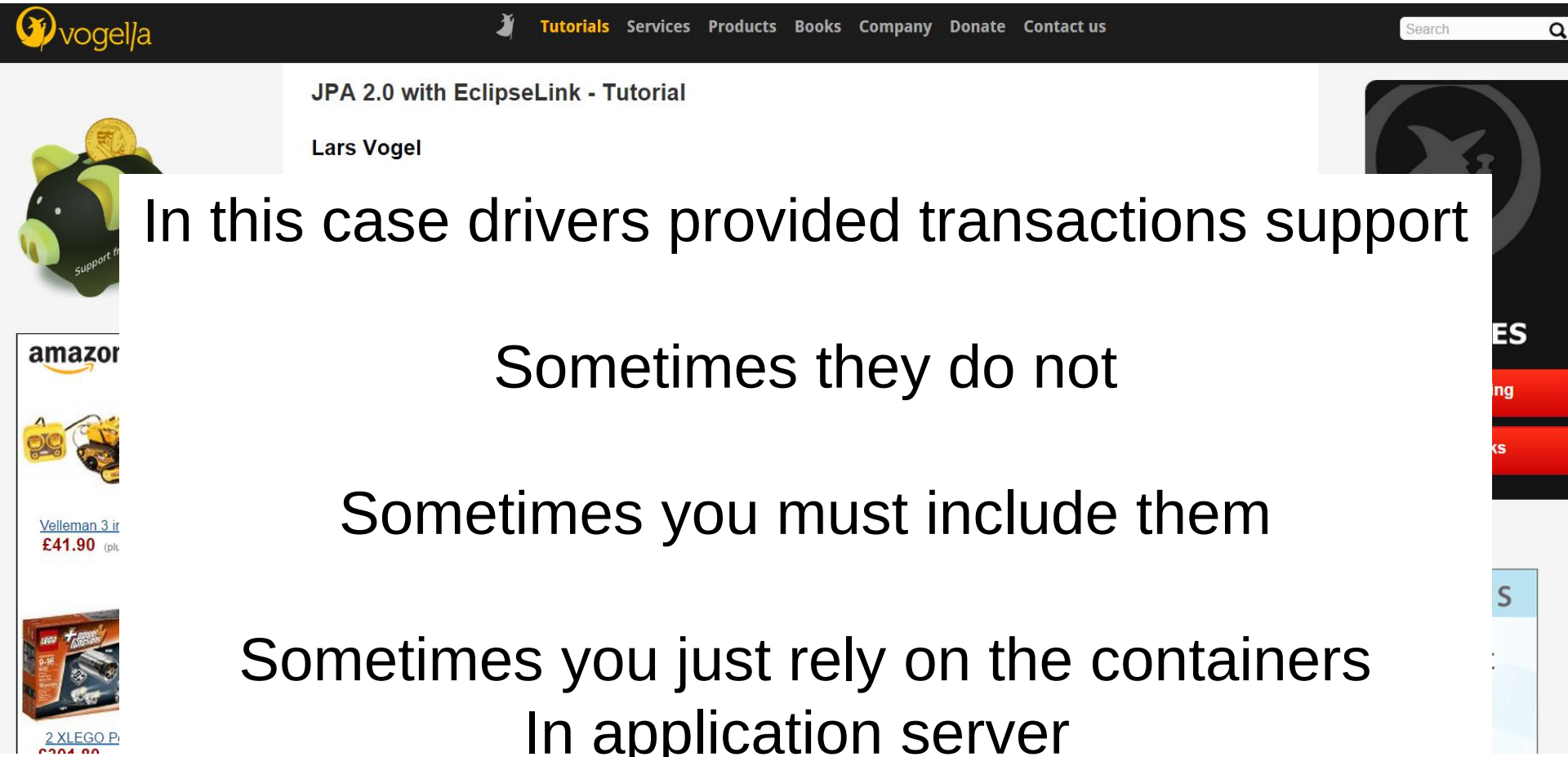
This tutorial explains how to use EclipseLink, the reference implementation for the Java Persistence API (JPA). The usage of EclipseLink is demonstrated for stand-alone Java applications (outside the Java EE environment). The EclipseLink 2.3.X implementation is used for this tutorial.

Table of Contents

Remember?


vogella
Tutorials Services Products Books Company Donate Contact us
Search
JPA 2.0 with EclipseLink - Tutorial
Lars Vogel
In this case drivers provided transactions support
....
1.1. Overview
1.2. Entity
1.3. Persistence of fields
1.4. Relationship Mapping
Convert
Print
Create

Remember?



In this case drivers provided transactions support

Sometimes they do not

Sometimes you must include them

Sometimes you just rely on the containers
In application server

In the persistence.xml JPA configuration file, you can have a line like:

```
<persistence-unit name="com.nz_war_1.0-SNAPSHOTPU" transaction-type="JTA">
```

or sometimes:

```
<persistence-unit name="com.nz_war_1.0-SNAPSHOTPU" transaction-type="RESOURCE_LOCAL">
```

My question is:

What is the difference between `transaction-type="JTA"` and `transaction-type="RESOURCE_LOCAL"` ?

JPA 2 | EntityManagers, Transactions and everything around it
http://www.kumaranuj.com/2013/06/jpa-2-entitymanagers-transactions-and.html

Container vs Application Managed

Container managed entity managers (EJB, Spring Bean, Seam component)

- Injected into application
- Automatically closed
- JTA transaction – propagated

```
transaction-type="JTA">
```

Application managed entity managers

> Used outside of the JavaEE 5 platform
> Need to be explicitly created
  - Persistence.createEntityManagerFactory()
> RESOURCE_LOCAL transaction – not propagated
> Need to explicitly close entity manager

```
transaction-type="RESOURCE_LOCAL">
```

SUN TECH DAYS 2008–2009

http://stackoverflow.com/questions/1962525/persistence-unit-as-resource-local-or-jta

JPA Best Practices

Transactions: a snipet

```
// WITH TRANSACTIONs SUPPORT
EntityManagerFactory emf = PersistenceManager.getInstance().getEntityManagerFactory();
EntityManager em = emf.createEntityManager();

try {
  EntityTransaction t = em.getTransaction();
  try {
    t.begin();
    ...
    t.commit();
  } finally {
    if (t.isActive()) t.rollback();
  }
} finally {
  em.close();
}
// NO TRANSACTIONS SUPPORT
EntityManagerFactory emf = PersistenceManager.getInstance().getEntityManagerFactory();
EntityManager em = emf.createEntityManager();

try {
  ...
} finally {
  em.close();
}
```

http://javanotepad.blogspot.pt/2007/05/jpa-entitymanagerfactory-in-web.html

Transaction & JTA

Java Transaction API (JTA)

Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

The JTA specification was developed by Sun Microsystems in cooperation with leading industry partners in the transaction processing and database system arena. See JSR 907. Also see the Java Transaction Service (JTS) page.

Downloads

Download the JTA specification, interfaces, and Javadoc files.

Java Transaction API Specification 1.1 Maintenance Release
Download

Class Files 1.1 Download
Javadocs 1.1 Download

Java Transaction API Specification 1.0.1 B
Maintenance Release of Maintenance Reviews
Download

Class Files 1.0.1B Download
Javadocs 1.0.1B Download

Java Transaction API Specification 1.0.1

Baeldung Start Here Courses ▾ Guides ▾ About ▾

Spring Data Annotations

Last modified: September 2, 2018

by Attila Fejér

Persistence Spring Spring Data

I just announced the new *Spring Boot 2* material, coming in REST With Spring:

>> CHECK OUT THE COURSE

+ This article is part of a series:

1. Introduction

Spring Data provides an abstraction over data storage technologies. Therefore, our business logic code can be much more independent of the underlying persistence implementation. Also, Spring simplifies the handling of implementation-dependent details of data storage.

In this tutorial, we'll see the most common annotations of the Spring Data, Spring Data JPA, and Spring Data MongoDB projects.

2. Common Spring Data Annotations

2.1. *@Transactional*

When we want to **configure the transactional behavior of a method**, we can do it with:

```
@Transactional
void pay() {}
```

If we apply this annotation on class level, then it works on all methods inside the class. However, we can override its effects by applying it to a specific method.

It has many configuration options, which can be found in this article.

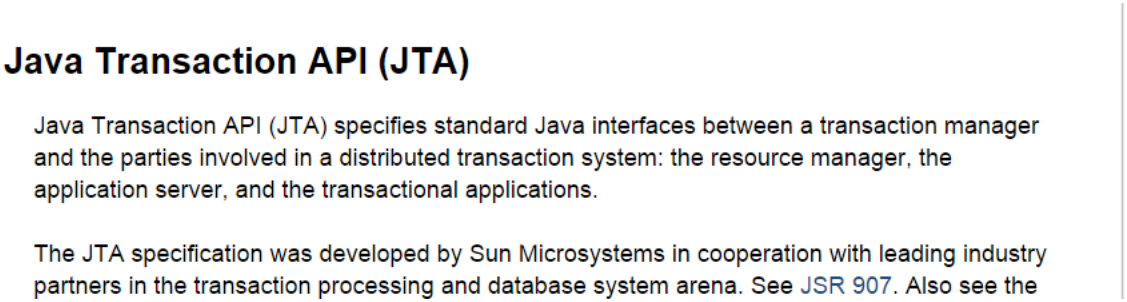
2.2. *@NoRepositoryB*

Sometimes we want to create repository interfaces with the only goal of providing common methods for the child repositories.

Of course, we don't want Spring to create a bean of these repositories since we won't inject them anywhere.

https://www.baeldung.com/spring-data-annotations

Transaction & JTA



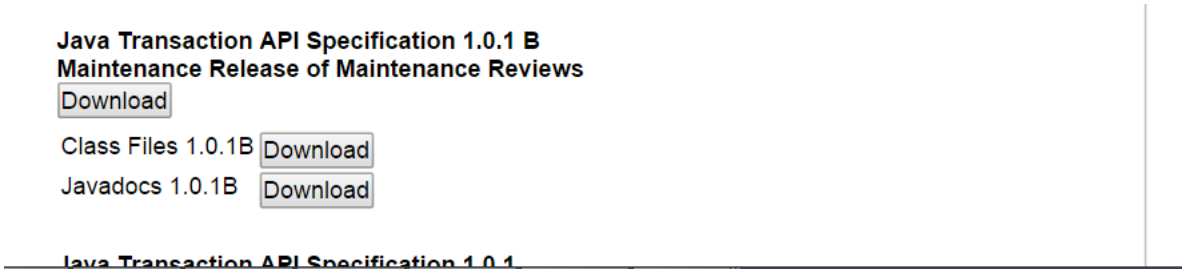
Java Transaction API (JTA)

Java Transaction API (JTA) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.

The JTA specification was developed by Sun Microsystems in cooperation with leading industry partners in the transaction processing and database system arena. See JSR 907. Also see the

NOT the focus of the course BUT

Most of the times an issue to find the proper implementation



Java Transaction API Specification 1.0.1 B
Maintenance Release of Maintenance Reviews
Download

Class Files 1.0.1B Download

Javadocs 1.0.1B Download

Java Transaction API Specification 1.0.1

Forgetting transactions

Persistence.xml: Configuring the managed entities


```
<persistence>
   <persistence-unit name="EmployeeService" transaction-
   type="RESOURCE_LOCAL">
     <class>examples.model.Employee</class>
     <properties>
       <property name="toplink.jdbc.driver"
                 value="org.apache.derby.jdbc.ClientDriver"/>
       <property name="toplink.jdbc.url"
        value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
       <property name="toplink.jdbc.user" value="APP"/>
       <property name="toplink.jdbc.password" value="APP"/>
     </properties>
   </persistence-unit>
</persistence>
```

With Netbeans most of configurations can be done using the IDE

Persistence.xml: Configuring the database connection


```
<persistence>
   <persistence-unit name="EmployeeService" transaction-
   type="RESOURCE_LOCAL">
     <class>examples.model.Employee</class>
     <properties>
       <property name="toplink.jdbc.driver"
                 value="org.apache.derby.jdbc.ClientDriver"/>
       <property name="toplink.jdbc.url"
        value="jdbc:derby://localhost:1527/EmpServDB;create=true"/>
       <property name="toplink.jdbc.user" value="APP"/>
       <property name="toplink.jdbc.password" value="APP"/>
     </properties>
   </persistence-unit>
</persistence>
```

With Netbeans most of configurations can be done using the IDE

Persistence.xml: properties are driver related

```xml
<?xml version="1.0" encoding="utf-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0">

    <persistence-unit name="hikePu" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ogm.jpa.HibernateOgmPersistence</provider>

        <properties>
            <property name="hibernate.ogm.datastore.provider" value="neo4j_embedded" />
            <property name="hibernate.ogm.datastore.database" value="HikeDB" />
            <property name="hibernate.ogm.neo4j.database_path" value="target/test_data_dir" />
        </properties>
    </persistence-unit>
```

```xml
...
<properties>
    <property name="hibernate.ogm.datastore.provider" value="mongodb" />
    <property name="hibernate.ogm.datastore.database" value="HikeDB" />
    <property name="hibernate.ogm.datastore.host" value="mongodb.mycompany.com" />
    <property name="hibernate.ogm.datastore.username" value="db_user" />
    <property name="hibernate.ogm.datastore.password" value="top_secret!" />
</properties>
...
```

Persistence.xml: properties are driver related

```
<persistence>
   <persistence-unit name="EmployeeService" transaction-
   type="RESOURCE_LOCAL">
     <class>examples.model.Employee</class>
     <properties>
       <property name="toplink.jdbc.driver"
     </properties>
   </persistence-unit>
</persistence>
```

See “Persistence in Java” – some examples and links on drivers
https://goo.gl/ERgWXY

What about Spring boot?

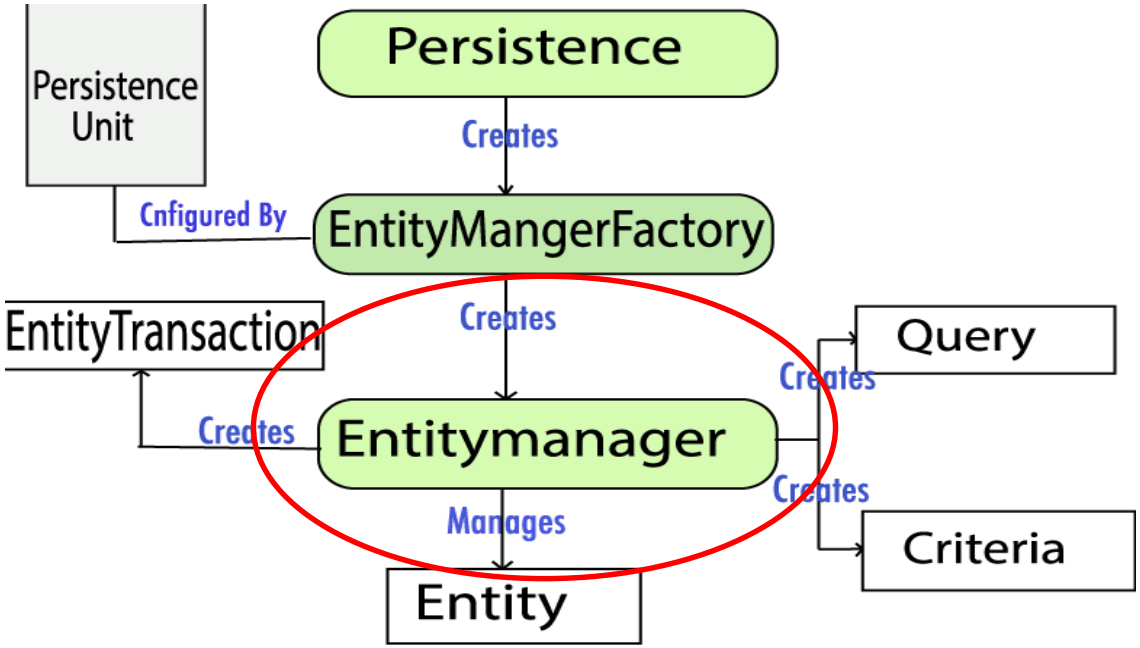
- The configurations
  - do not use persistence.xml
  - Use **application.properties**
  - Mostly same information
  - Different looks

```xml
<persistence-unit name="org.hibernate.tutorial.jpa" transaction-type="RESOURCE_LOCAL">
    <description>
        Persistence unit for the JPA tutorial of the Hibernate Getting Started Guide
    </description>
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>org.halyph.sessiondemo.Event</class>

    <properties>
        <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
        <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/jpatestdb" />
        <property name="javax.persistence.jdbc.user" value="root" />
        <property name="javax.persistence.jdbc.password" value="root" />

        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.hbm2ddl.auto" value="create" />
    </properties>

</persistence-unit>
```


```
logging.level.org.springframework.web=DEBUG
# Database
spring.datasource.driver = com.mysql.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/test3
spring.datasource.username = root
spring.datasource.password = root



spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.
```

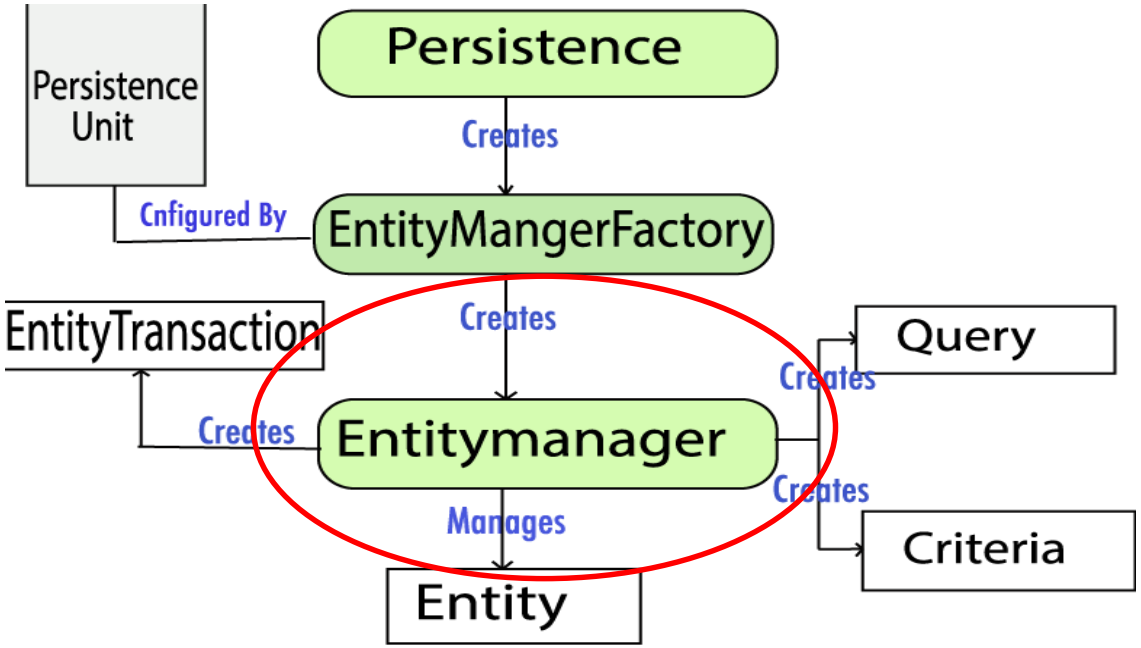
Spring boot jpa mysql example
http://candidjava.com/tutorial/spring-boot-jpa-mysql-example/
https://gist.github.com/halyph/2990769

EntityManager: JPA entry point




```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

EntityManager: JPA entry point




```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

Entity Manager has an interface

- `void persist(Object entity)`
  - Makes an instance managed and persistent
- `<T> T find(Class<T> entityClass, Object primaryKey)`
  - Searches for an entity of the specified class and
- `primary key<T> T getReference(Class<T> entityClass,Object primaryKey)`
  - Gets an instance, whose state may be lazily fetched
- `void remove(Object entity)`
  - Removes the entity instance from the persistence context and from the underlying database
- `<T> T merge(T entity)`
  - Merges the state of the given entity into the current persistence context
- `void refresh(Object entity)`
  - Refreshes the state of the instance from the database, overwriting changes made to the entity, if any
- `void flush()`
  - Synchronizes the persistence context to the underlying database
- `void clear()`
  - Clears the persistence context, causing all managed entities to become detached
- `void clear(Object entity)`
  - Removes the given entity from the persistence Context boolean contains(Object entity) Checks whether the instance is a managed entity instance belonging to the current persistence context

Can use entity managers

```java
@Component
public class UserEntityManagerCommandLineRunner implements CommandLineRunner {

 private static final Logger log = LoggerFactory.getLogger(UserEntityManagerCommandLineRunner.class);

 @Autowired
 private UserService userService;

 @Override
 public void run(String...args) {

  log.info("-------------------------------");
  log.info("Adding Tom as Admin");
  log.info("-------------------------------");
  User tom = new User("Tom", "Admin");
  userService.insert(tom);
  log.info("Inserted Tom" + tom);
```

```java
@Repository
@Transactional
public class UserService {

 @PersistenceContext
 private EntityManager entityManager;

 public long insert(User user) {
  entityManager.persist(user);
  return user.getId();
 }

 public User find(long id) {
  return entityManager.find(User.class, id);
 }

 public List < User > findAll() {
  Query query = entityManager.createNamedQuery(
   "query_find_all_users", User.class);
  return query.getResultList();
 }
```

Introduction to JPA Using Spring Boot Data
https://dzone.com/articles/introduction-to-jpa-using-spring-boot-dat

Can use entity managers

```java
@Component
public class UserEntityManagerCommandLineRunner implements CommandLineRunner {

 private static final Logger log = LoggerFactory.getLogger(UserEntityManagerCommandLineRunner.class);

 @Autowired
 private UserService userService;

 @Override
 public void run(String...args) {

  log.info("-----------------------");
  log.info("Adding Tom as Admin");
  log.info("-------------------------------");
  User tom = new User("Tom", "Admin");
  userService.insert(tom);
  log.info("Inserted Tom" + tom);
```

The code below is very simple. CommandLineRunner interface is used to indicate that this bean has to be run as soon as the Spring application context is initialized (similar to startup annotation in JavaEE).

```java
public long insert(User user) {
 entityManager.persist(user);
 return user.getId();
}

public User find(long id) {
 return entityManager.find(User.class, id);
}

public List < User > findAll() {
 Query query = entityManager.createNamedQuery(
  "query_find_all_users", User.class);
 return query.getResultList();
}
```

Introduction to JPA Using Spring Boot Data
https://dzone.com/articles/introduction-to-jpa-using-spring-boot-dat

@Repositories, no entity manager

```java
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);
}
```

```java
public interface JpaRepository<T, ID extends Serializable> extends
  PagingAndSortingRepository<T, ID> {

    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> save(Iterable<? extends T> entities);

    void flush();

    T saveAndFlush(T entity);

    void deleteInBatch(Iterable<T> entities);
}
```

CrudRepository, JpaRepository, and PagingAndSortingRepository in Spring Data
https://www.baeldung.com/spring-data-repositories

The entity

```java
@Entity
@NamedQuery(query = "select u from User u", name = "query_find_all_users")
public class User {

 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;

 private String name; // Not perfect!! Should be a proper object!
 private String role; // Not perfect!! An enum should be a better choice!

 protected User() {}

 public User(String name, String role) {
  super();
  this.name = name;
  this.role = role;
 }

 public Long getId() {
  return id;
 }
```

Introduction to JPA Using Spring Boot Data
https://dzone.com/articles/introduction-to-jpa-using-spring-boot-data-jpa

```java
@Component
public class UserEntityManagerCommandLineRunner implements CommandLineRunner {

 private static final Logger log = LoggerFactory.getLogger(UserEntityManagerCommandLineRunner.class);

 @Autowired
 private UserService userService;

 @Override
 public void run(String...args) {

  log.info("--------------------------------");
  log.info("Adding Tom as Admin");
  log.info("--------------------------------");
  User tom = new User("Tom", "Admin");
  userService.insert(tom);
```

```java
@Repository
@Transactional
public class UserService {

 @PersistenceContext
 private EntityManager entityManager;

 public long insert(User user) {
  entityManager.persist(user);
  return user.getId();
 }

 public User find(long id) {
  return entityManager.find(User.class, id);
 }
```

```java
package com.example.h2.user;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository < User, Long > {}
```


```java
@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner {

 private static final Logger log = LoggerFactory.getLogger(UserRepositoryCommandLineRunner.class);

 @Autowired
 private UserRepository userRepository;

 @Override
 public void run(String...args) {
  User harry = new User("Harry", "Admin");
  userRepository.save(harry);
  log.info("------------------------------");
  log.info("Finding all users");
  log.info("------------------------------");
  for (User user: userRepository.findAll()) {
   log.info(user.toString());
  }
 }
```

```java
package com.example.h2.user;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository < User, Long > {}
```

Entity (User) and key (long)


```java
@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner {

 private static final Logger log = LoggerFact

 @Autowired
 private UserRepository userRepository;

 @Override
 public void run(String...args) {
  User harry = new User("Harry", "Admin");
  userRepository.save(harry);
  log.info("------------------------------");
  log.info("Finding all users");
  log.info("------------------------------");
  for (User user: userRepository.findAll()) {
   log.info(user.toString());
  }
 }
```

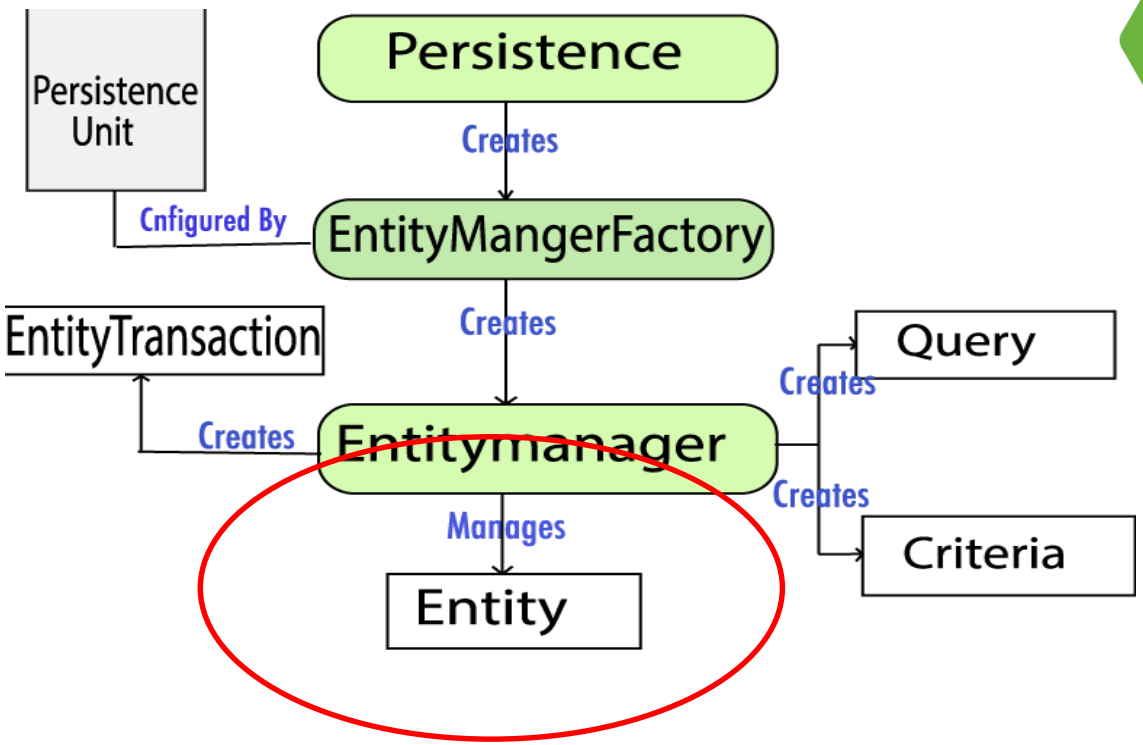
```java
@Repository                                                    );
@Transactional
public class UserService {

 @PersistenceContext
 private EntityManager entityManager;

 public long insert(User user) {
  entityManager.persist(user);
  return user.getId();
 }

 public User find(long id) {
  return entityManager.find(User.class, id);
 }
```

Have similar in JavaEE (not standard)

```java
public abstract class AbstractFacade<T> {
    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass,
    }
```

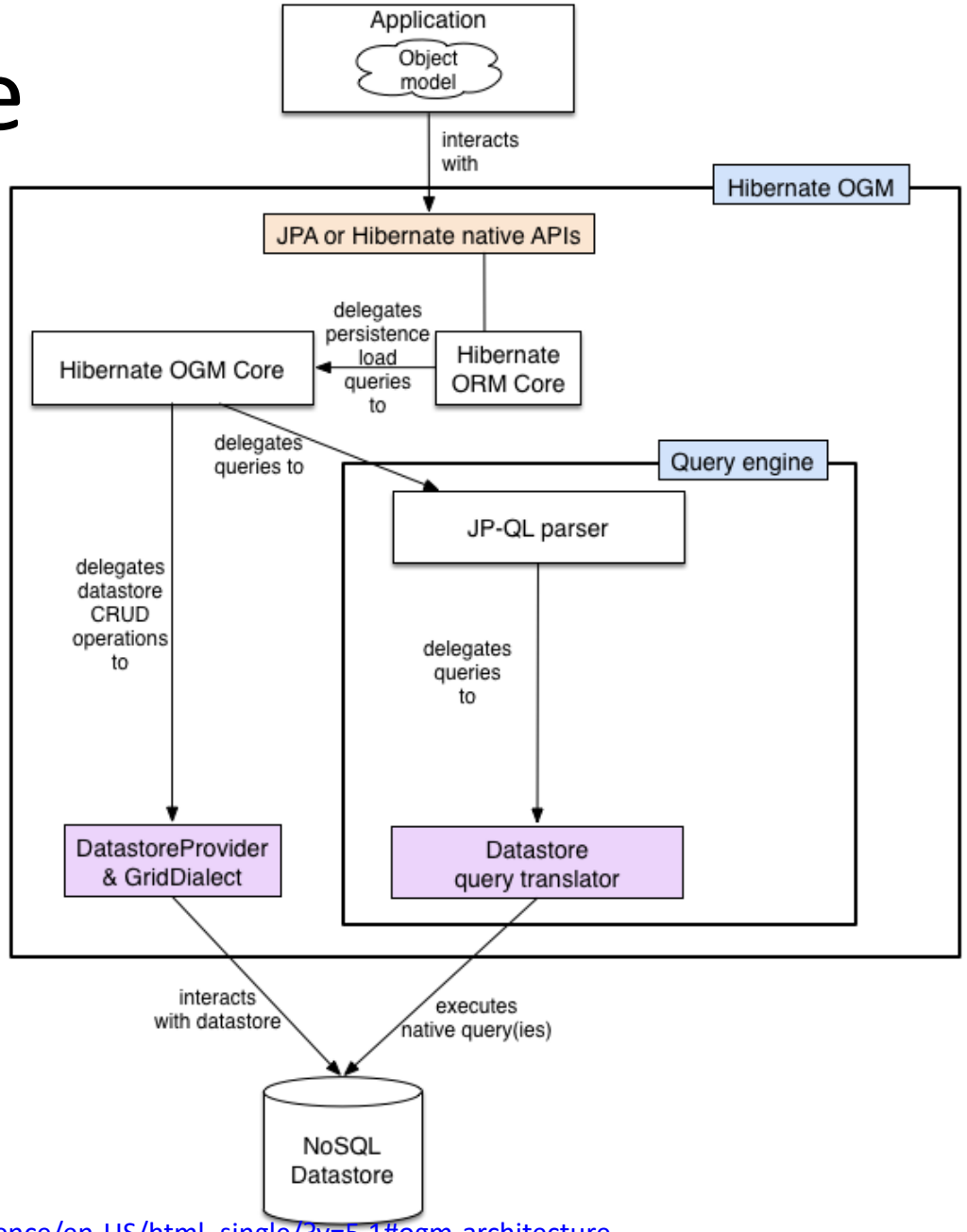
```java
@Stateless
public class CustomerFacade extends AbstractFacade<Customer> {
    @PersistenceContext(unitName = "AffableBeanPU")
    private EntityManager em;

    protected EntityManager getEntityManager() {
        return em;
    }

    public CustomerFacade() {
        super(Customer.class);
    }

}
```

Entity: JPA information




```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

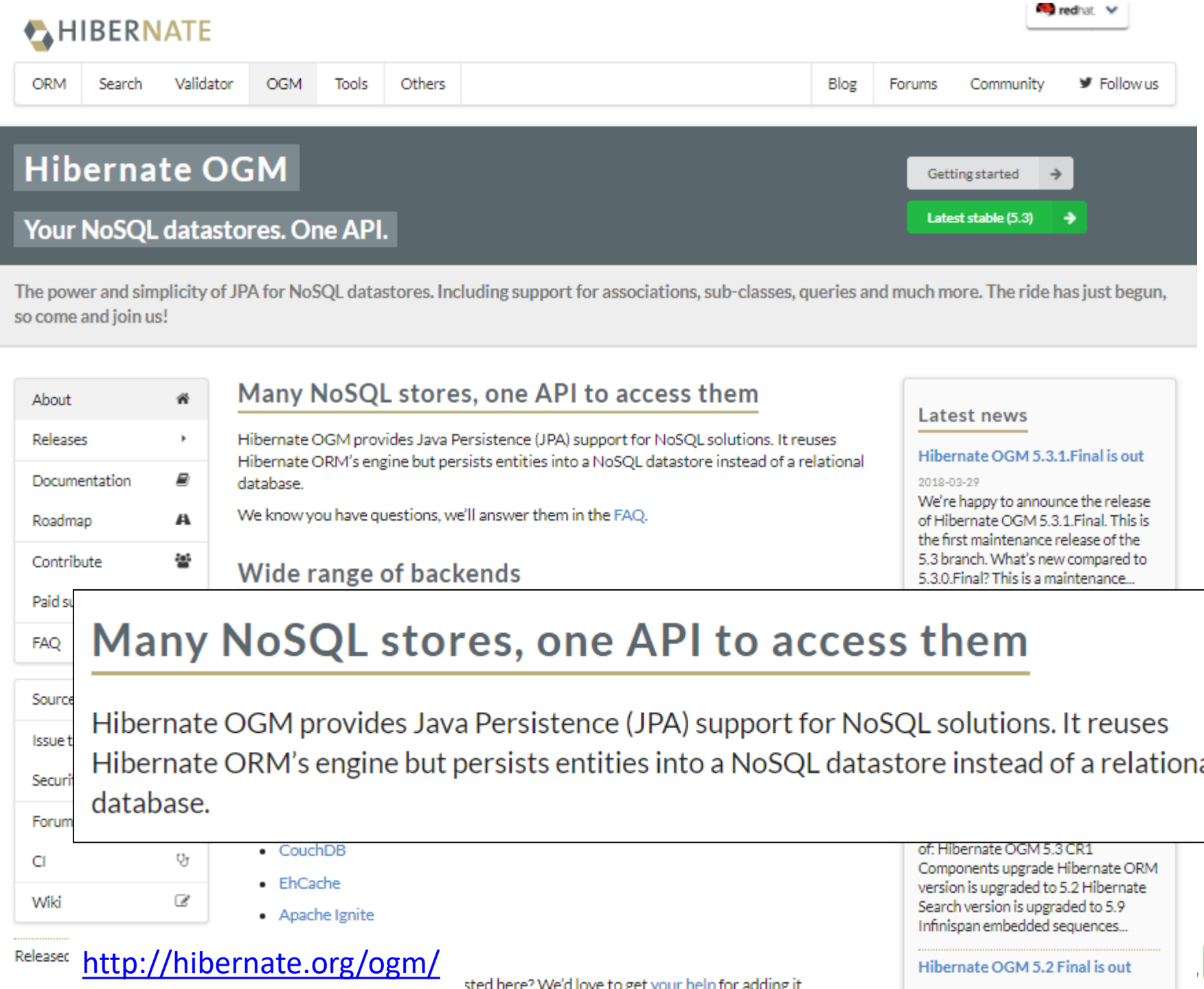
Architecture



https://docs.jboss.org/hibernate/stable/ogm/reference/en-US/html_single/?v=5.1#ogm-architecture

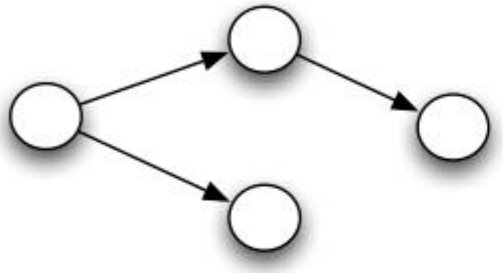
JPA is not only ORM

- Hibernate OGM is a good example
  - a.k.a. Hibernate for NoSQL (no only SQL)
  - https://docs.jboss.org/hibernate/ogm/4.0/reference/en-US/html_single/#preface
- ORM – object relational mapping
  - Relational databases
- OGM – object graph mapping
  - neo4J, …
- ODM – object document mapping
  - mongoDB …



http://hibernate.org/ogm/

Entities



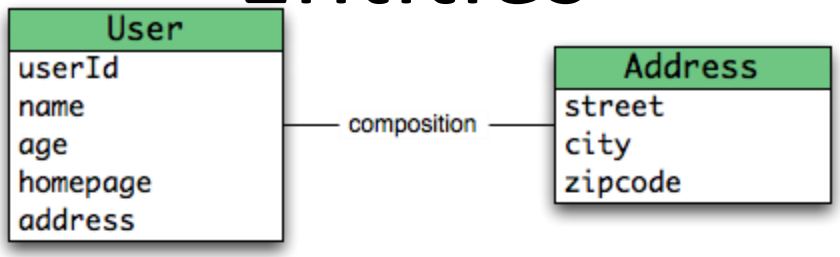
```
{ "user" : {
  "id": "124",
  "name": "Emmanuel",
  "addresses" : [
    { "city": "Paris", "country": "France" },
    { "city": "Atlanta", "country": "USA" }
  ]
}
```

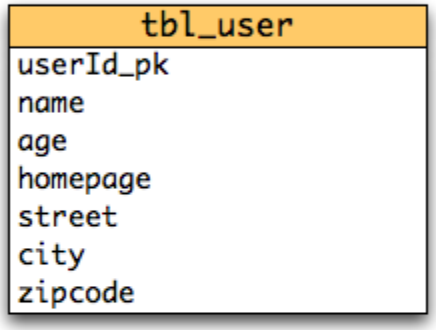
key	value
123	Address@23
126	“Booya”

1	Things	A	foo	B	bar	C	baz	
2	Things	C	bam	E	coh	People	A	Emmanuel
3	Languages	A	C	B	Java	C	Ceylon	

https://docs.jboss.org/hibernate/stable/ogm/reference/en-US/html_single/?v=5.1#ogm-architecture

Entities

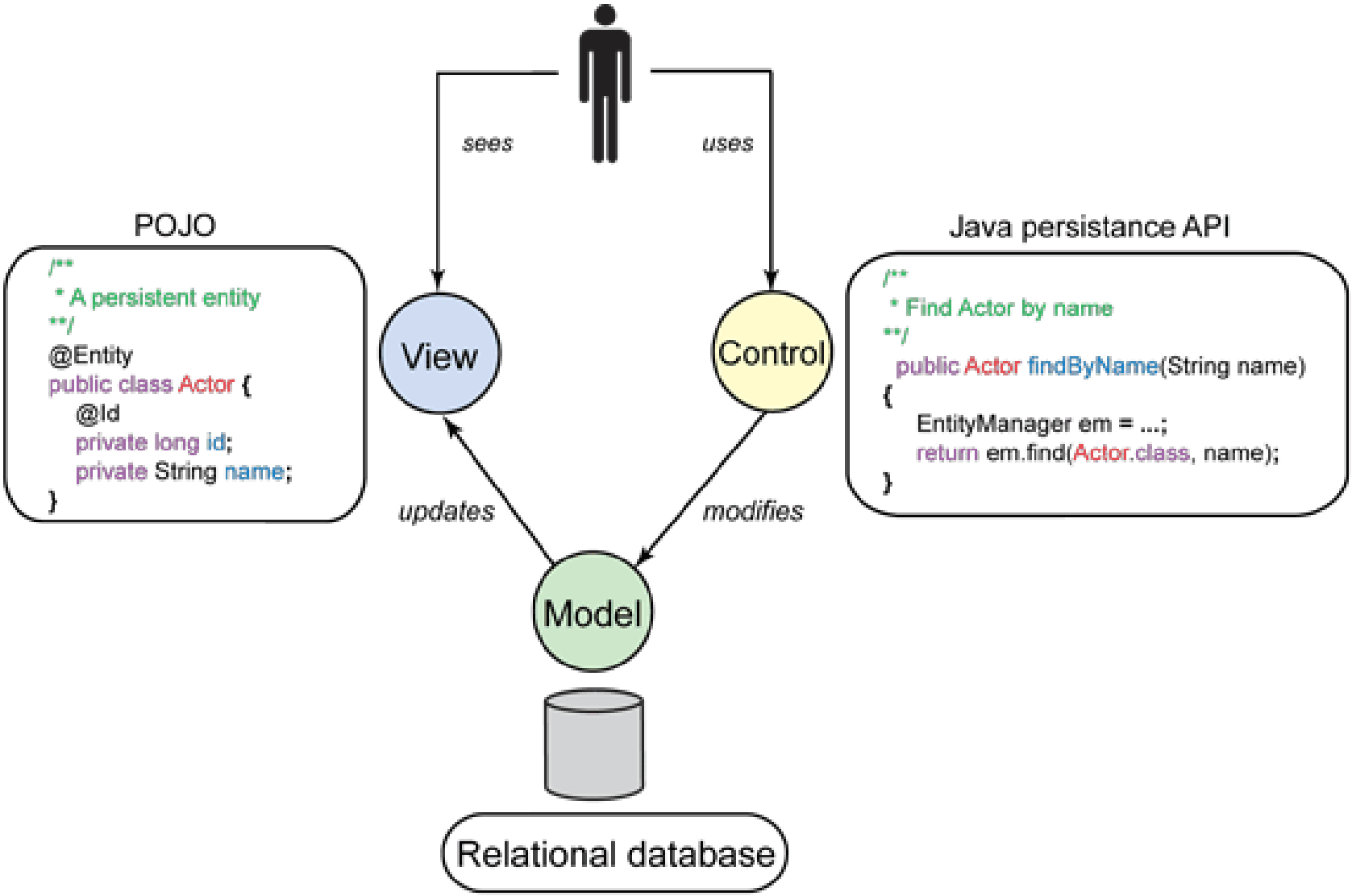




key	value
tbl_user,userId_pk,1	{userId_pk=1, name="Emmanuel", age=33, homepage="http://emmanuelbernard.com", address.street="rue de la Paix", address.city="Paris", address.zipcode="75008"}

https://docs.jboss.org/hibernate/stable/ogm/reference/en-US/html_single/?v=5.1#ogm-architecture

MVC approach



Entities

Primary Key

- JPA ID Annotation
- JPA ID Auto Generator
- JPA ID GenerationType
- JPA ID SequenceGenerator
- JPA ID Table Generator

```
@Entity
public class Person {
  @Id
  @GeneratedValue
  private Long id;
  private String name;
  private String surname;
```

Introduction to JPA Using Spring Boot Data

Object relational impedance mismatch involves databases with many-to-many relationships that comparatively have mismatched data. Read on a more in-depth introduction!

by Ranga Karanam · Aug. 08, 17 · Database Zone · Tutorial

This guide will help you understand what JPA is and set up a simple JPA example using Spring Boot.

You will learn:

- What is JPA?
- What is the problem solved by JPA?
- What are the alternatives to JPA?
- What is Hibernate and how does it relate to JPA?
- What is Spring Data JPA?
- How to create a simple JPA project using Spring Boot Data JPA Starter.

Tools you will need:

- Maven 3.0+ (build tool)
- Your favorite IDE; we will use Eclipse.
- JDK 1.8+
- In memory database H2

What Is Object Relational Impedance Mismatch?

Java is an object-oriented programming language. In Java, all data is stored in objects.

Typically, relational databases are used to store data (these days, a number of other NoSQL data stores are also becoming popular, but we will stay away from them for now). Relational databases store data in tables.

The way we design objects is different from the way the relational databases are designed. This results in an impedance mismatch.

- Object-oriented programming consists of concepts like encapsulation, inheritance, interfaces, and polymorphism.
- Relational databases are made up of tables with concepts like normalization.

Examples of Object Relational Impedance Mismatch

Introduction to JPA Using Spring Boot Data
https://dzone.com/articles/introduction-to-jpa-using-spring-boot-data-jpa

Example 1

The task table below is mapped to the task table. However there are mismatches in column names.

DZone Database Zone

Over a million developers have joined DZone. Log In / Sign Up

REFCARDZ GUIDES ZONES JOBS | Agile AI Big Data Cloud Database DevOps Integration IoT Java Microservices Open Source Performance Security Web Dev

Introduction to JPA Using Spring Boot Data

Object relational impedance mismatch involves databases with many-to-many relationships that comparatively have mismatched data. Read on a more in-depth introduction!

by Ranga Karanam · Aug. 08, 17 · Database Zone · Tutorial

Like (17) Comm

Join the DZone comm

MariaDB TX, proven in prod enterprise — a modern data

This guide will help you

You will learn:

- What is JPA?
- What is the problem
- What are the altern
- What is Hibernate a
- What is Spring Data
- How to create a sim

Tools you will need:

- Maven 3.0+ (build t
- Your favorite IDE; w
- JDK 1.8+
- In memory databas

What Is Object

Java is an object-oriente

Typically, relational data are also becoming popul tables.

The way we design objec an impedance mismatch.

- Object-oriented pro polymorphism.
- Relational database

Examples of Object Relational Impedance Mismatch

Let's consider a simple example with employees and tasks.

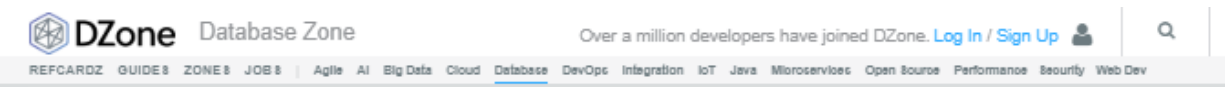
```
@Entity
@Table(name = "Task")
public class Task {
 @Id
 @GeneratedValue
 private int id;

 @Column(name = "description")
 private String desc;

 @Column(name = "target_date")
 private Date targetDate;

 @Column(name = "is_done")
 private boolean isDone;

}
```

```
CREATE TABLE task
  (
     id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
     description VARCHAR(255),
     is_done     BOOLEAN,
     target_date TIMESTAMP,
     PRIMARY KEY (id)
  )
```

Introduction to JPA Using Spring Boot Data
https://dzone.com/articles/introduction-to-jpa-using-spring-boot-data-jpa

Example 1

The task table below is mapped to the task table. However there are mismatches in column names.



Introduc Data

Object relational i that comparatively

by Ranga Karan

Like (17) Co

Join the DZone co

MariaDB TX, proven in enterprise — a modern

This guide will help y

You will learn:

- What is JPA?
- What is the prob
- What are the alt
- What is Hiberna
- What is Spring D
- How to create a

Tools you will need

- Maven 3.0+ (bui
- Your favorite ID
- JDK 1.8+
- In memory data

What Is Obje

Java is an object-orie

Typically, relational d are also becoming po tables.

The way we design o an impedance misma

- Object-oriented polymorphism.
- Relational datab

Examples of C

```
    //Some other code

@ManyToMany
private List<Task> tasks;
}

public class Task {

    //Some other code

@ManyToMany(mappedBy = "tasks")
private List<Employee> employees;
}
```

```
CREATE TABLE employee
  (
     id            BIGINT NOT NULL,
     OTHER_COLUMNS
  )


  CREATE TABLE employee_tasks
  (
     employees_id BIGINT NOT NULL,
     tasks_id     INTEGER NOT NULL
  )

  CREATE TABLE task
  (
     id          INTEGER GENERATED BY DEFAULT AS IDENTITY,
     OTHER_COLUMNS
  )
```

Introduction to JPA Using Spring Boot Data

https://dzone.com/articles/introduction-to-jpa-using-spring-boot-data-jpa

Example 1

The task table below is mapped to the task table. However there are mismatches in column names.

details

Nikos' Java blog

quick & easy Java tutorials

Home

Essential object-relational mapping in JPA

42 Votes

The Java Persistence API is the standard framework for persistence in both Java SE and Java EE platforms. It is the outcome of the collaborative work of the industry's leading vendors in object-relational mapping, EJB and JDO, including a remarkable contribution from the open source community.

In this post we summarize, one-by-one, the essential relationships between entities. We examine each relationship from the prespectives of "Domain Model" and "Relation Model", and provide the associated construction in JPA. Finally, for each relationship we present an example from the real world.

The relationships are divided in two categories. "For daily use" presents the correct way to use common relationships in our application development activities. On the other hand, "Containing pitfalls" identify common mistakes.

For daily use:

1. One-to-One (one direction)
2. One-to-One (both directions)
3. Many-to-One (one direction)
4. Many-to-One (both directions)
5. One-to-Many (one direction)
6. One-to-Many (both directions)
7. Many-to-Many (both directions)

Containing pitfalls:

Nikos Pougounias is a senior Analyst-Programmer specialized in Java Enterprise Edition and Object Oriented Design. During the past years, he has been contributing to major portfolios under the European Commission.

Contact me about this blog via e-mail.

SEARCH

Search

VISITORS

o 444,799 hits

TOP POSTS (24H)

o Free SCJP Mock exams
o Hibernate dalam 10 menit
o Essential object-relational mapping in JPA

CATEGORIES

o Application Server (3)

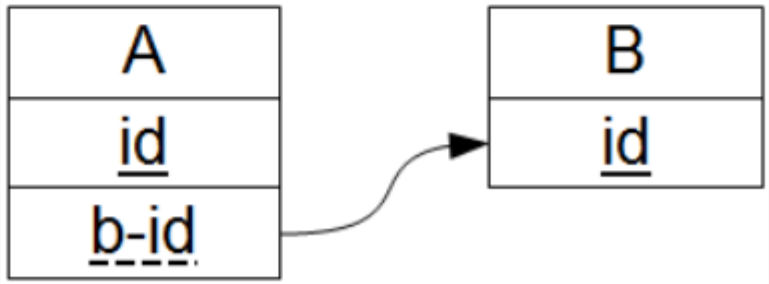
https://nikojava.wordpress.com/2011/08/04/essential-jpa-relationships/

details

One-to-One (one direction)

Domain model of an one-to-one unidirectional relationship

```
@Entity
class A {
   @Id int id;

   @OneToOne
   B b;
}

@Entity
class B {
   @Id int id;
}
```



Relation model of an one-to-one unidirectional relationship

https://nikojava.wordpress.com/2011/08/04/essential-jpa-relationships/

Nikos' Java blog

Home

Essential o

The Java Persis
both Java SE a
work of the ind
and JDO, incluc
community.

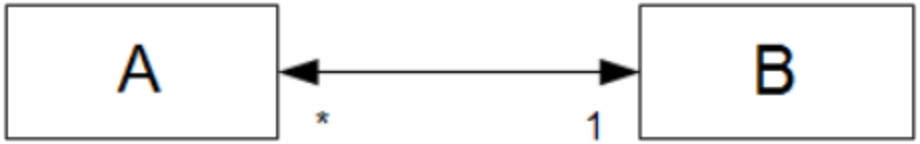
In this post we
between entitie
"Domain Model
construction in
from the real w

The relationshi
the correct way
development a
common mistal

For daily use:

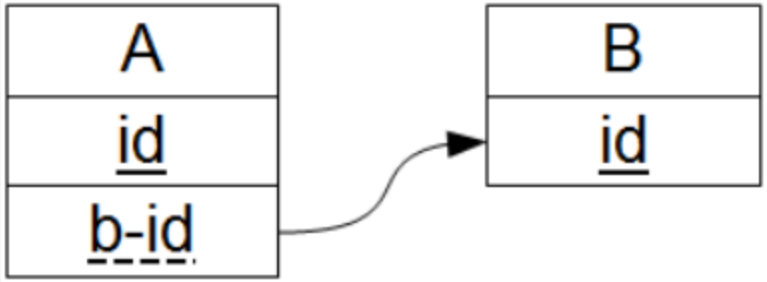
1. One-to-
2. One-to-
3. Many-to
4. Many-to
5. One-to-
6. One-to-
7. Many-to

Many-to-One (both directions)



Domain model of a many-to-one bidirectional relationship

```
@Entity
class A {
   @Id int id;

   @ManyToOne
   B b;
}

@Entity
class B {
   @Id int id;

   @OneToMany(mappedBy="b")
   Collection<A> listOfA;
}
```



o Application Server (3)

https://nikojava.wordpress.com/2011/08/04/essential-jpa-relationships/

ls

JPA Entity Listeners And Callback Methods

March 12, 2014 by Amr Mohammed — Leave a Comment

Before JPA 2.1 the concept of Entity Listeners (Injectable EntityListeners) wasn't defined until JPA 2.1 released. **The entity listener concept allows the programmer to designate a methods as a lifecycle callback methods to receive notification of entity lifecycle events. A lifecycle callback methods can be defined on an entity class, a mapped superclass, or an entity listener class associated with an entity or mapped superclass.**

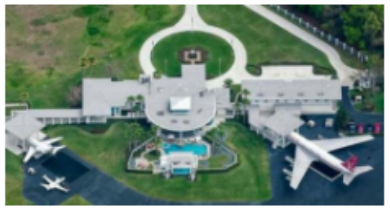
Default entity listeners are entity listeners whose callback methods has been called by all entities that are defined in a certain persistence context (unit). The lifecycle callback methods and entity listeners classes can be defined by using metadata annotations or by an XML descriptor.

Entity Listener (Default Listener) Using Entity Listener Class

If you've looked before for the introduction examples for eclipselink (See EclipseLink Tutorial), it's time to listening about events that thrown by lifecycle of the entities. Let's look at the first entity listener that would be listening for each defined entities in the persistence.xml.

PersistenceContextListener.java

```
package net.javabeat.eclipselink.data;

import javax.persistence.PostPersist;
import javax.persistence.PostRemove;
import javax.persistence.PostUpdate;
import javax.persistence.PrePersist;
```



JPA Entity Listeners And Callback Methods
http://javabeat.net/jpa-entity-listeners-callback-methods/

Home >> Java EE >> JPA Entit

JPA Entity Listene

March 12, 2014 by Amr Mohamm

Before JPA 2.1 the concept c
until JPA 2.1 released. **The e**
a methods as a lifecycle cal
events. A lifecycle callback
superclass, or an entity list

Default entity listeners are
all entities that are defined
methods and entity listener
by an XML descriptor.

Entity Listener (Defau

If you've looked before for t
Tutorial), it's time to listeni
look at the first entity lister
persistence.xml.

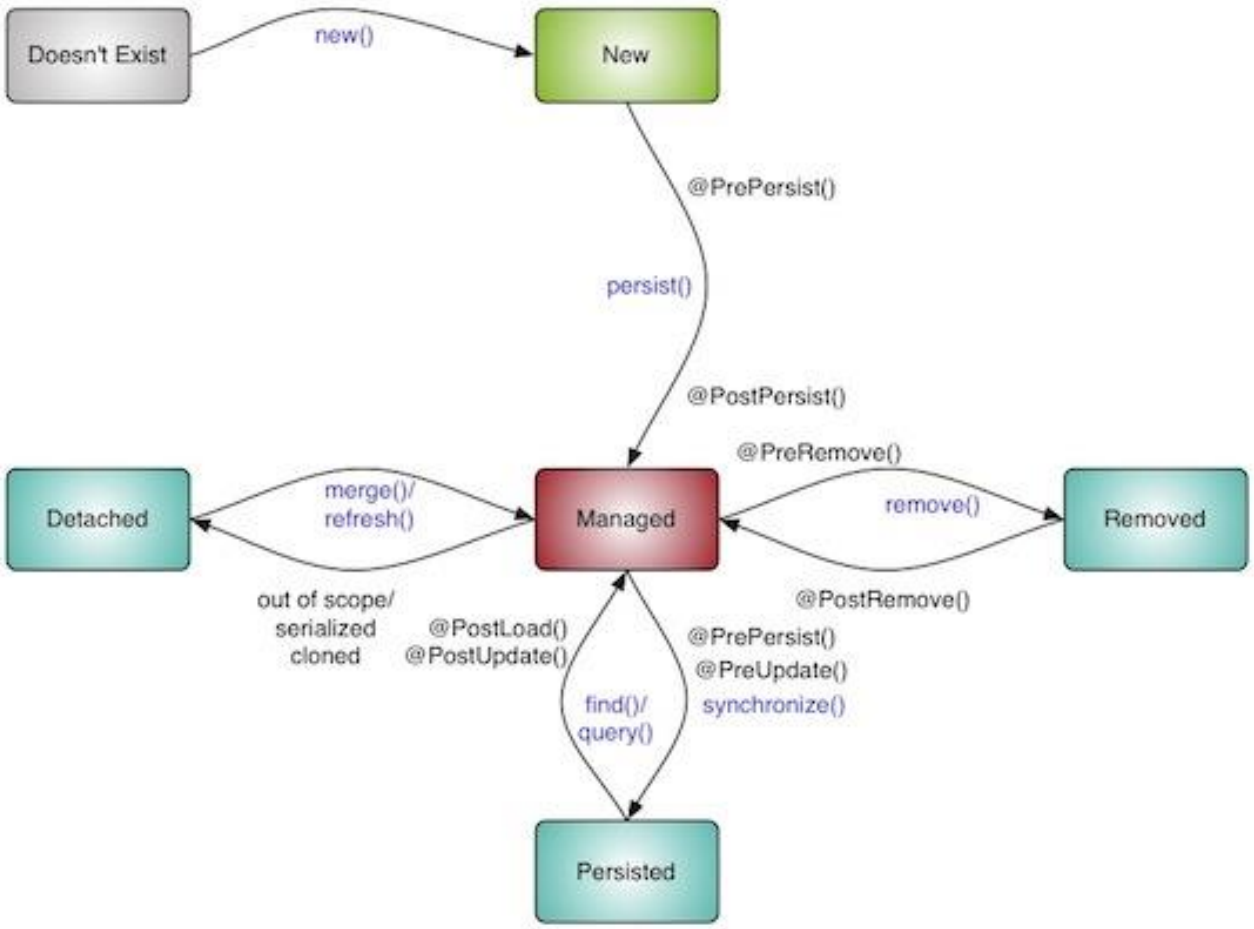
PersistenceContextListene

```
package net.javabeat.ec

import javax.persistenc
import javax.persistenc
import javax.persistenc
import javax.persistenc
```

```
public class PersistenceContextListener {
 @PrePersist
 public void prePersist(Object object) {
 System.out.println("PersistenceContextListener :: Method PrePersist
Invoked Upon Entity :: " + object);
 }

 @PostPersist
 public void postPersist(Object object){
 System.out.println("PersistenceContextListener :: Method PostPersist
Invoked Upon Entity :: " + object);
 }

 @PreRemove
 public void PreRemove(Object object){
 System.out.println("PersistenceContextListener :: Method PreRemove
Invoked Upon Entity :: " + object);
 }

 @PostRemove
 public void PostRemove(Object object){
 System.out.println("PersistenceContextListener :: Method PostRemove
Invoked Upon Entity :: " + object);
 }

 @PreUpdate
 public void PreUpdate(Object object){
 System.out.println("PersistenceContextListener :: Method PreUpdate
Invoked Upon Entity :: " + object);
```

JPA Entity Listeners And Callback Methods
http://javabeat.net/jpa-entity-listeners-callback-methods/

Entities have a lifecycle



Finding entities

```java
@Entity
@Table(name="EMP")
public class Person {
  @Id
  @Column(name = "EMP_ID")
  private long id;

  @Basic
  private String name;
  private String surname;
```

```java
public void test(){
  Person p1 = new Person("Tom", "Smith");
  p1.setId(1L);
  p1.setPicture("asdf".getBytes());
  Person p2 = new Person("Jack", "Kook");
  p2.setId(2L);
  p1.setPicture("java2s.com".getBytes());

  save(p1);
  save(p2);

  listAll();

  Person emp =  em.find(Person.class, 1L);
  if (emp != null) {
    em.remove(emp);
  }
  listAll();
}
```

oneToOne

```java
@Entity
public class Professor {
  @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;
  private String name;
  private long salary;

  @OneToOne
  @JoinColumn(name="PSPACE_ID")
  private ParkingSpace parkingSpace;

  public int getId() {
      return id;
  }
```

```java
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy=GenerationType
  private long id;

  private String name;
  @OneToOne
  @JoinColumn(name="DEPT_ID")
  private Department department;

  public Person() {}
```

```java
@Entity
public class Department {
  @Id
  @GeneratedValue(strategy=Gene
  private long id;

  private String name;

  @OneToOne(mappedBy="department")
  private Person person;
```

Relation: ManyToOne / OneToMany

```java
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;

  @ManyToOne (cascade=CascadeType.ALL)
  private Department department;
```

```java
@Entity
public class Department {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;

  @OneToMany(mappedBy="department",cascade=CascadeType.ALL)
  private Set<Person> persons = new HashSet<Person>();
```

Bi-directional relationships

- In JPA,
  - The application is responsible for that
  - the responsibility of the application, or the object model to maintain relationships.
  - E.g. If your application adds to one side of a relationship, then it must add to the other side.
- Can be “hidden” in add and set methods
- Verify if the relationship is ok
  - **If is ok, avoid infinite loops**
  - If not , update it

@OnetoMany: setters and add

```
public class Phone {
    private Employee owner;
    ...
    public void setOwner(Employee employee) {
        this.owner = employee;
        if (!employee.getPhones().contains(this)) {
            employee.getPhones().add(this);
        }
    }
    ...
}

public class Employee {
    private List phones;
    ...
    public void addPhone(Phone phone) {
        this.phones.add(phone);
        if (phone.getOwner() != this) {
            phone.setOwner(this);
        }
    }
    ...
}
```

@OnetoMany: setters and add

```
public class Phone {
    private Employee owner;
    ...
    public void setOwner(Employee employee) {
        this.owner = employee;
        if (!employee.getPhones().contains(this)) {
            employee.getPhones().add(this);
        }
    }
    ...
}

public class Employee {
    private List phones;
    ...
    public void addPhone(Phone phone) {
        this.phones.add(phone);
        if (phone.getOwner() != this) {
            phone.setOwner(this);
        }
    }
    ...
}
```

Avoid never ending persistence loops on bidirectional relations while Sure that data is store as it is in memory

@OnetoMany: setters and add

```
public class Phone {
    private Employee owner;
    ...
    public void setOwner(Employee employee) {
        this.owner = employee;
        if (!employee.getPhones().contains(this)) {
            employee.getPhones().add(this);
        }
    }
    ...
}

public class Employee {
    private List phones;
    ...
    public void addPhone(Phone phone) {
        this.phones.add(phone);
        if (phone.getOwner() != this) {
            phone.setOwner(this);
        }
    }
    ...
}
```

Avoid never ending persistence loops on bidirectional relations while Sure that data is store as it is in memory

JPA Implementation Patterns: Bidirectional Assocations

By Vincent Partington March 16, 2009 JPA, JPA implementation patterns 33 Comments

Last week we started our search for JPA implementation patterns with the Data Access Object pattern. This week we continue with another hairy subject.

JPA offers the `@OneToMany`, `@ManyToOne`, `@OneToOne`, and `@ManyToMany` annotations to map associations between objects. While EJB 2.x offered container managed relationships to manage these associations, and especially to keep bidirectional associations in sync, JPA leaves more up to the developer.

The Setup

Let's start by expanding the `Order` example from the previous blog with an `OrderLine` object. It has an id, a description, a price, and a reference to the order that contains it:

```
@Entity
public class OrderLine {
        @Id
        @GeneratedValue
        private int id;

        private String description;
```

JPA Implementation Patterns: Bidirectional Assocations
http://blog.xebia.com/jpa-implementation-patterns-bidirectional-assocations/

Taking this idea to its logical conclusion we end up with these methods for the `OrderLine` class:

```
public Order getOrder() { return order; }

public void setOrder(Order order) {
        if (this.order != null) { this.order.internalRemoveOrderLine(this); }
        this.order = order;
        if (order != null) { order.internalAddOrderLine(this); }
}
```

And these methods for the `Order` class:

```
public Set getOrderLines() { return Collections.unmodifiableSet(orderLines); }

public void addOrderLine(OrderLine line) { line.setOrder(this); }
public void removeOrderLine(OrderLine line) { line.setOrder(null); }

public void internalAddOrderLine(OrderLine line) { orderLines.add(line); }
public void internalRemoveOrderLine(OrderLine line) { orderLines.remove(line); }
```

These methods provide a POJO-based implementation of the CMR logic that was built into EJB 2.x. With the typical POJOish advantages of being easier to understand, test, and maintain

```
private int id;

private String description;
```

MAKE SURE THAT MEMORY IMAGE is COHERENT WITH PERSISTENCE

JPA Implementation Patterns: Bidirectional Assocations
http://blog.xebia.com/jpa-implementation-patterns-bidirectional-assocations/

ManyToMany

- **Avoid Many to Many**
  - Inconsistent relational implementations
  - Too coupled to Relational model

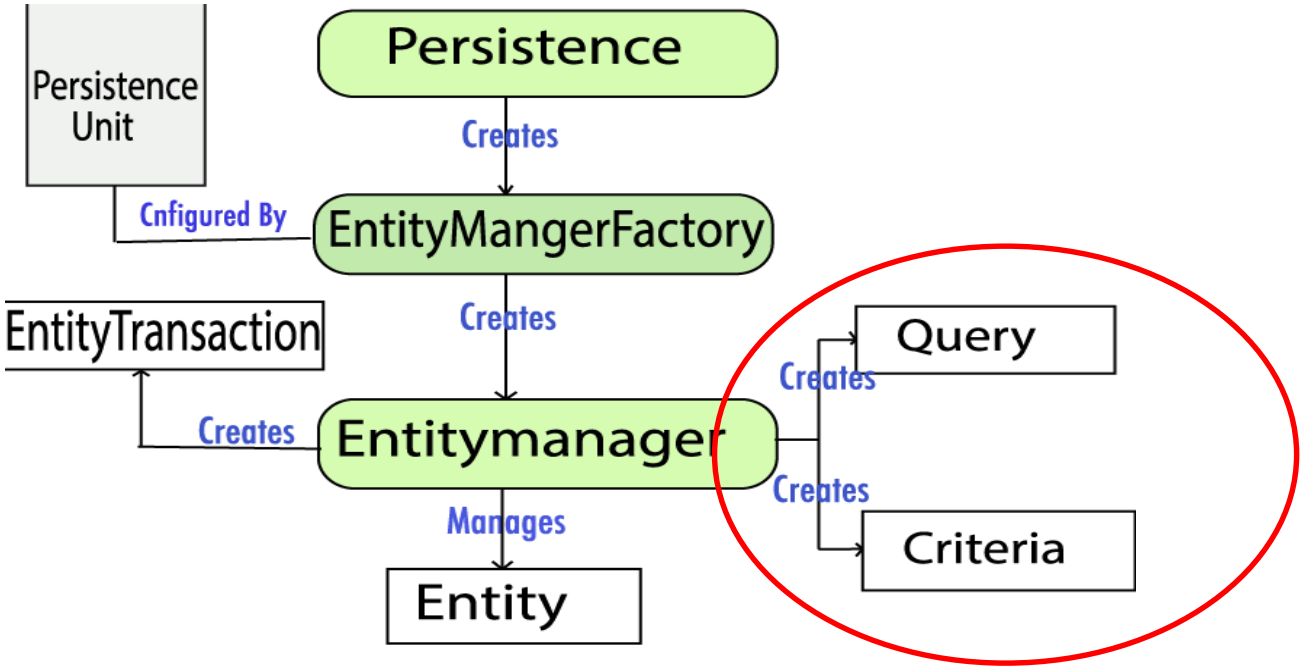
```java
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;

  @ManyToMany
  @JoinTable(name="PERSON_DEPT",
  joinColumns=@JoinColumn(name="PERSON_ID"),
  inverseJoinColumns=@JoinColumn(name="DEPT_ID"))
  private Set<Department> departments = new HashSet<De
```

```java
@Entity
public class Department implements java.io.Serializable{
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;

  @ManyToMany
  private Set<Person> persons = new HashSet<Person>();
```

Entities relationships

- Inheritance is possible but not advisable
- Relation many 2 many
  - Possible but not advisable
  - Implementations not uniform
  - Not a OO concepts
- Annotations support relational parameters
  - Avoid them

Queries




```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("chapter02PU");
EntityManager em = emf.createEntityManager();
em.persist(book);
```

Queries and JPQL

- A very similar to SQL
  - Object flavour
  - We will present some examples
  - For detailed description see references
- An example
  - The simplest JPQL query selects all the instances of a single entity:

```
SELECT b FROM Book b
```

  - To restrict the result, add search criteria where you can use the WHERE clause as follows:

```
SELECT b FROM Book b WHERE b.title = "H2G2“
```

  - Note the familiar object dot (.) notation.

Cascade

```
public void test() {
  Employee emp = new Employee();
  emp.setName("Tom");
  emp.setId(1);

  Address addr = new Address();
  addr.setId(1);
  addr.setStreet("street");
  addr.setCity("city");
  addr.setState("state");
  emp.setAddress(addr);
  em.persist(emp);

}
```

```
@Entity
public class Employee {
    @Id
    private int id;
    private String name;

    @ManyToOne(cascade=CascadeType.PERSIST)
    Address address;
```

http://www.java2s.com/Tutorials/Java/JPA/6005__JPA_Persist_With_Relationship_Cascade.htm

cascade

```
public void test() {
   Employee emp = new Employee();
   emp.setName("Tom");
   emp.setId(1);

   Phone ph = new Phone();
   ph.setEmployee(emp);
   ph.setNumber("1234567890");

   ParkingSpace ps = new ParkingSpace();
   ps.setEmployee(emp);
   ps.setLocation("Dept");

   emp.setParkingSpace(ps);
   emp.getPhones().add(ph);
   em.persist(emp);
   emp = em.find(Employee.class, 1);
   em.remove(emp);

}
```

```
@Entity
public class Employee {
    @Id
    private int id;
    private String name;

    @OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})
    @JoinColumn(name="PSPACE_ID")
    ParkingSpace parkingSpace;

    @OneToMany(cascade={CascadeType.PERSIST, CascadeType.REMOVE},
               mappedBy="employee")
    Collection<Phone> phones;
```

http://www.java2s.com/Tutorials/Java/JPA/6010__JPA_Remove_With_Relationships_Cascade.htm

Queries

Basic Query
- JPA Query Intro
- JPA Query Select Two Entities
- JPA Query Select Properties
- JPA Query Named Parameter
- JPA Query Entity Parameter
- JPA Query Number Indexed Parameters
- JPA Query Date P
- JPA Query Distinc
- JPA Query Distinc
- JPA Query AND C
- JPA Query OrderE
- JPA Query OrderE
- JPA Query Like
- JPA Query Like Es
- JPA Query Is Emp
- JPA Query Not En
- JPA Query Betwe
- JPA Query Groupl
- JPA Query All
- JPA Query ANY
- JPA Query IN
- JPA Query In One
- JPA Query Exists
- JPA Query AVG S
- JPA Query AVG J
- JPA Query COUN
- JPA Query Count
- JPA Query Functi

Join Query
- JPA Query Join Fetch
- JPA Query Join ManyToOne
- JPA Query Join OrderBy
- JPA Query Join Three Entities
- JPA Query Left Join

Advanced Query
- JPA Query new Object
- JPA Query Member OF
- JPA Query Object Function
- JPA Query Paging
- JPA Query Result Output
- JPA TypedQuery
- JPA Query Named
- JPA Named Query ParamTypes
- JPA Query Several Named Queries
- JPA Query Collections Size
- JPA Query Retrieve ManyToOne Map
- JPA Query Native Query Result Class

```
List<Professor> l = em.createQuery("SELECT e FROM Professor e").getResultList();
for(Professor p:l){
  System.out.println(p);
}
```

http://www.java2s.com/Tutorials/Java/JPA/4000__JPA_Query_Intro.htm

Spring Data JPA - Reference Documentation

Oliver Gierke · Thomas Darimont · Christoph Strobl · Mark Paluch · Jay Bryant – Version 2.1.0.RELEASE, 2018-09-21

Table of Contents

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

```java
@Entity
public class Employee {

  private @Id @GeneratedValue Long id;
  private String firstName, lastName, description;

  private Employee() {}

  public Employee(String firstName, String lastName, String description) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.description = description;
  }
}
```

This defines a simple JPA entity with a few fields. The following code shows a simple repository definition:

```java
public interface EmployeeRepository extends CrudRepository<Employee, Long> {

  Employee findByFirstName(String firstName);

  List<Employee> findByLastName(String lastName);
}
```

This interface extends Spring Data's `CrudRepository` and defines the type (`Employee`) and the id type (`Long`). Put this code inside a Spring Boot application with `spring-boot-starter-data-jpa` like this:

```java
@SpringBootApplication
public class MyApp {

  public static void main(String[] args) {
    SpringApplication.run(MyApp.class, args);
  }
}
```

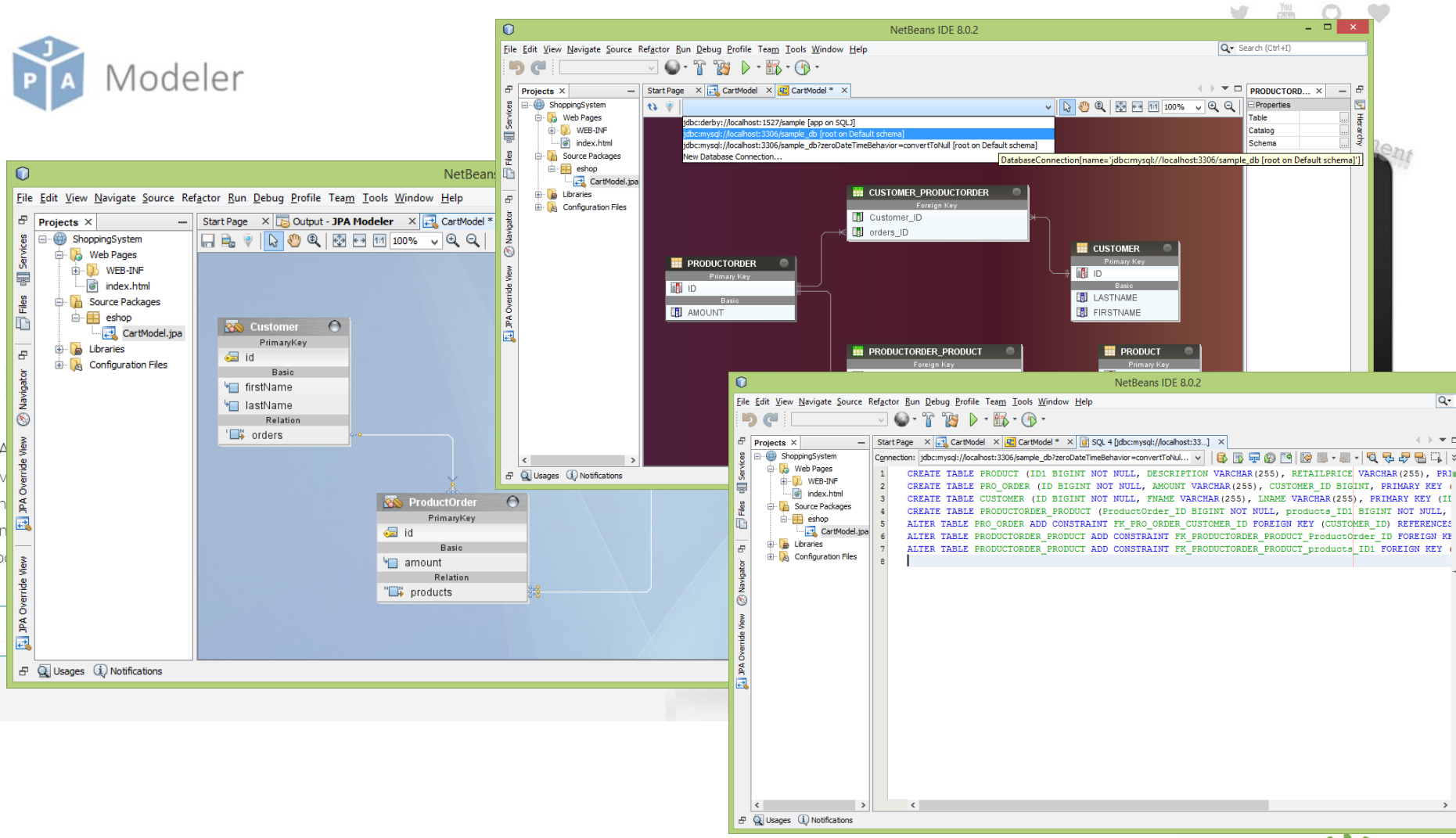
Launch your app and Spring Data (having been autoconfigured by Boot, SQL or NoSQL) will automatically craft a concrete set of operations:

http://projects.spring.io/spring-data/

JPA modeler: nice plugin

http://jpamodeler.github.io/

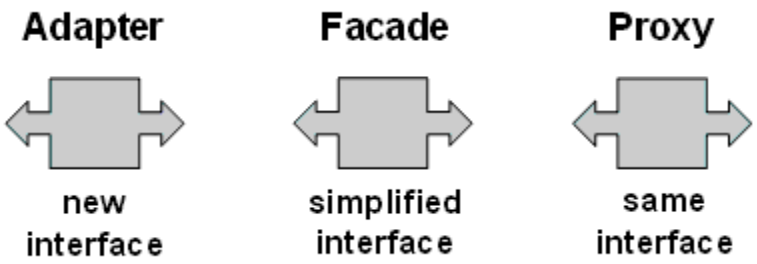
JPA modeler: nice plugin



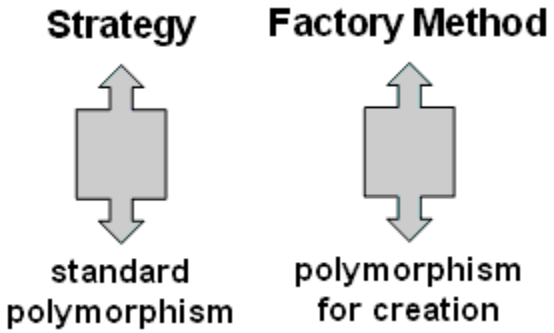
http://jpamodeler.github.io/

Persistence in J2EE

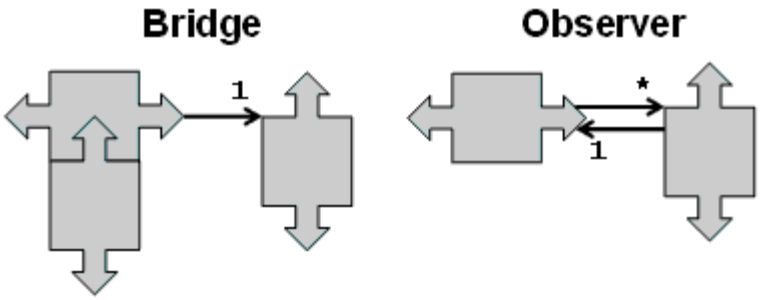
- patterns that we already saw in other contexts



Access the resources
is through a specific interface
(JPA, JDBC)



The interaction managment is delegated
(EntityManager) that deals with specifities (e.g. Drivers



Tracking the changes in state of the entities
(e.g. Entity lifecycle)

Some final considerations

- The objective of JPA
  - Decouple Domain model / objects from persistence solution
  - ORM, OGM, ODM ….
- Still not a science
  - Depends on drivers, own architecture and frameworks,…
  - Need to know the technology timeline
    - Toplink, EclipseLink, ….
- Avoid relational related annotations
  - Too coupled to relational model

Skipped

- Transactions and configurations

Fetching strategies

Annotation	Default Fetching Strategy
@OneToOne	EAGER
@ManyToOne	EAGER
@OneToMany	LAZY
@ManyToMany	LAZY

Fetching: lazy & eager

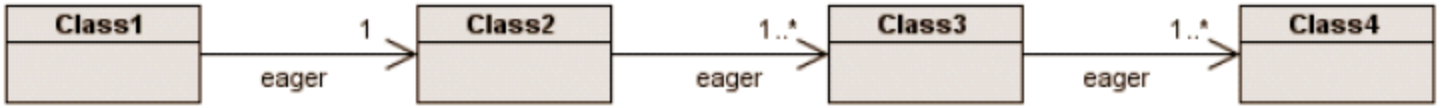
```
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;
  @OneToOne(fetch=LAZY)
  private Department department;
```

- **EAGER**:
  - Convenient, but slow
  - FetchType.EAGER = Loads ALL relationships
- **LAZY**:
  - FetchType.LAZY = Doesn't load the relationships unless explicitly "asked for" via getter
  - More coding, but much more efficient
- Not specific for relational models but more common in ORM

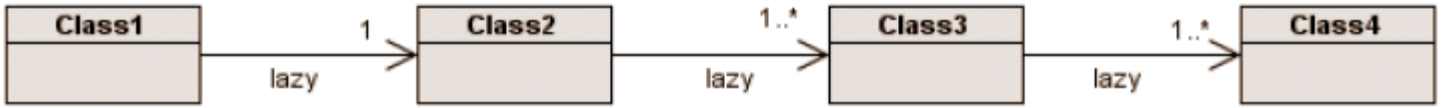
http://byobi.com/2013/03/analyzing-sql-server-object-dependencies-with-nodexl/#!prettyPhoto

http://hibernate.org/orm/2.x/162

Fetching

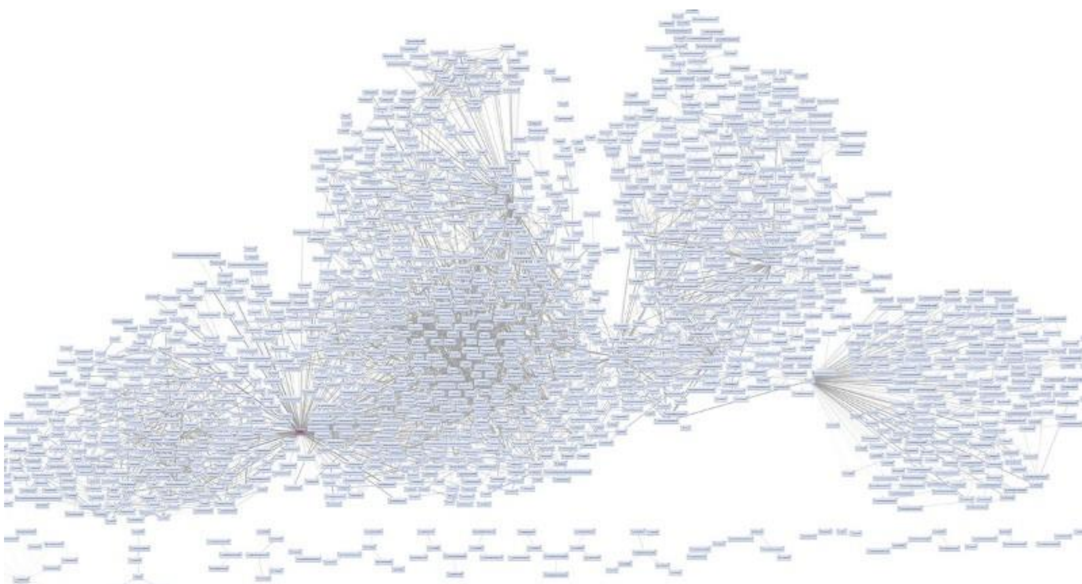


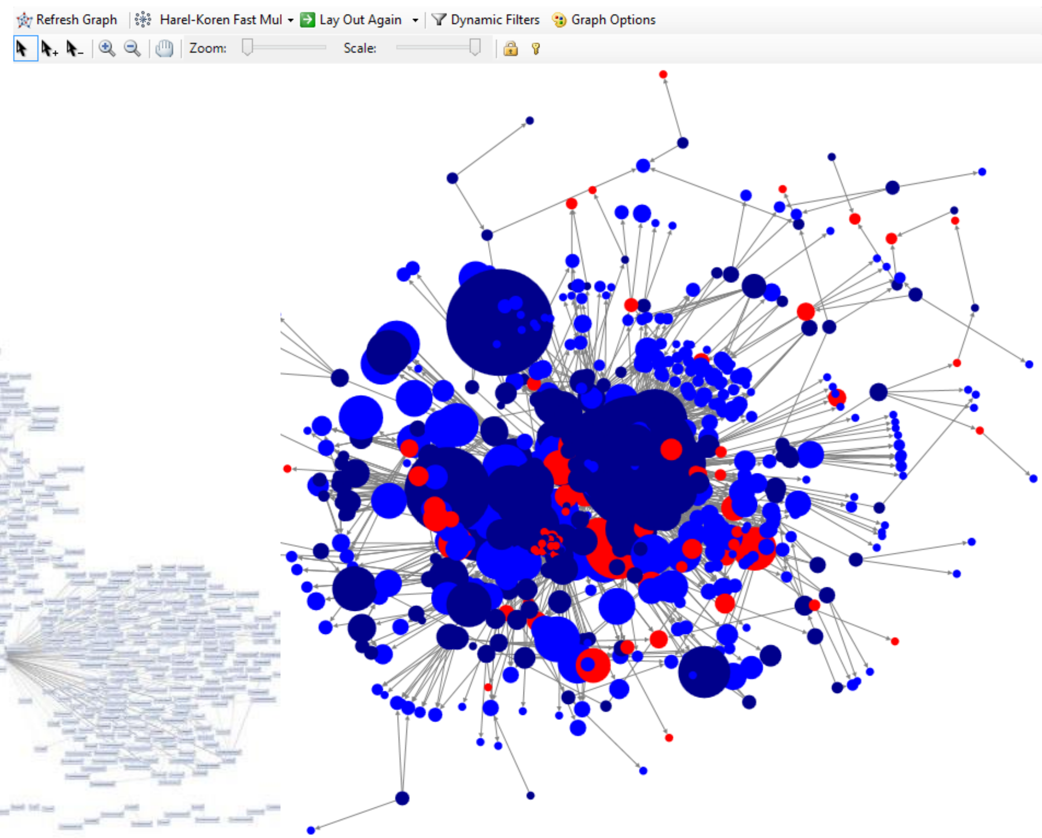
class1.getClass2().getClass3().getClass4()



Fetching: lazy & eager

```java
@Entity
public class Person {
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;

  private String name;
  @OneToOne(fetch=LAZY)
  private Department department;
```





http://byobi.com/2013/03/analyzing-sql-server-object-dependencies-with-nodexl/#!prettyPhoto
http://biztech.shepardos.com/?p=161

JPA – Lazy Loading and Eager Loading

March 13, 2014 by Amr Mohammed — Leave a Comment

Eclipselink 2.1 is a persistence provider runtime of the Java Persistence API 2.1 specification. **JPA specification defines two major strategies of loading data (Lazy and Eager)**. The EAGER strategy is a requirement on the persistence provider runtime that data must be eagerly fetched. The LAZY strategy is a hint to the persistence provider runtime that data should be fetched lazily when it is first time accessed.

The Lazy loading is commonly used to defer the loading of the attributes or associations of an entity or entities until the point at which they are needed, meanwhile the eager loading is an opposite concept in which the attributes and associations of an entity or entities are fetched explicitly and without any need for pointing them.

Determine The Loading Strategy

EclipseLink 2.1 provides you different ways to mention your desired loading strategy.

- Basic Annotation: It provides a fetchType attribute that can be used for determining the strategy to which the attribute(s) that should be loaded eagerly or lazily.
- @OneToOne, @ManyToOne, @OneToMany and @ManyToMany: Every associations annotations provide fetchType attribute that can be used for determining the strategy to which the attributes(s) that should be loaded eagerly or lazily.

Lazy Fetch Type With @OneToOne and @ManyToOne

If you've noted before @OneToOne association example, you've absolutely seen the association between Employee and Address entities adhered a bidirectional OneToOne. It's the time to give more attention about Eager and Lazy loading in that association.

VLAD MIHALCEA'S BLOG

Teaching is my way of learning

Home
High-Performance Java Persistence
Hibernate
Spring
FlexyPool
Presentations
Trainings
About
Archive

EAGER fetching is a code smell

DECEMBER 15, 2014 / VLADMIHALCEA

Follow @vlad_mihalcea 3,967 followers

Introduction

Hibernate fetching strategies can really make a difference between an application that barely crawls and a highly responsive one. In this post, I'll explain why you should prefer query based fetching instead of global fetch plans.

Fetching 101

Hibernate defines four association retrieving strategies:

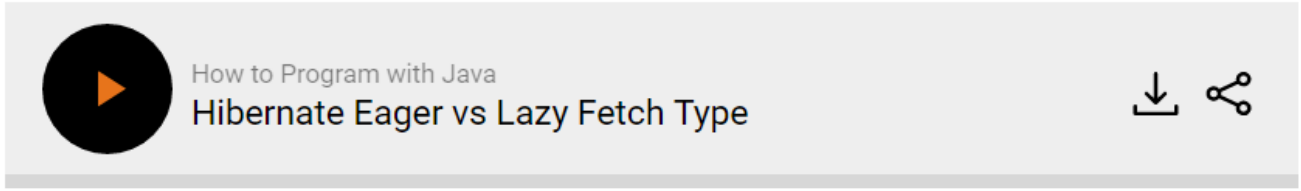
Fetching Strategy	Description
Join	The association is OUTER JOINED in the original SELECT statement
Select	An additional SELECT statement is used to retrieve the associated entity(entities)

l SELECT statement is used to retrieve the whole
llection. This mode is meant for to-many

EAGER fetching is a code smell
https://vladmihalcea.com/2014/12/15/eager-fetching-is-a-code-smell/

Hibernate Eager vs Lazy Fetch Type

by TREVOR PAGE *on* OCTOBER 25, 2013



Background

The topic of Fetch types in Hibernate is a fairly advanced topic, as it requires you to have a decent understanding of the Java programming language, as well as have had exposure to SQL and the Hibernate framework for Java. In this post I'm going to assume that you posses this knowledge, if you do not, then I suggest you follow along with my podcasts available here.

What the heck is a Fetch Type?

CATEGORIES

Advanced Tips
Assignment
Hibernate
HTML
Intermediate Tips
Introduction
JavaScript
Mailing List
Podcast
SQL
Uncategorized

Search for:
Search

LEARN JAVA FAST

Adapting and Learning

Thoughts on software development and life in general

Monday, January 11, 2016

JPA Pitfalls / Mistakes

Labels: java, JPA

Tweet

From my experience, both in helping teams and conducting training, here are some pitfalls/mistakes I have encountered that caused some problems in Java-based systems that use JPA.

- Requiring a public no-arg constructor
- *Always* using bi-directional associations/relationships
- Using `@OneToMany` for collections that can become huge

Requiring a Public No-arg Constructor

Yes, a JPA `@Entity` requires a zero-arguments (or default no-args) constructor. **But this can be made `protected`. You do not have to make it `public`.** This allows better object-oriented modeling, since you are *not* forced to have a publicly accessible zero-arguments constructor.

> The entity class must have a no-arg constructor. The entity class may have other constructors as well. The no-arg constructor must be public **or protected**. [emphasis mine]
>
> - from Section 2.1 of the *Java Persistence API 2.1 Specification (Oracle)*

If the entity being modeled has some fields that need to be initialized when it is created, this should be done through its constructor.

Let's say we're modeling a hotel room reservation system. In it, we probably have entities like room, reservation, etc. The reservation entity will likely require start and end dates, since it would not make much sense to create one without the period of stay. Having the start and end dates included as arguments in the reservation's constructor would allow for a better model. Keeping a *protected* zero-arguments constructor would make JPA happy.

NOTE: Some JPA providers may overcome a missing no-arg constructor by adding one at build time.

```
1  @Entity
2  public class Reservation { ...
3   public Reservation(
4     RoomType roomType, DateRange startAndEndDates) {
5    if (roomType == null || startAndEndDates == null) {
6     throw new IllegalArgumentException(...);
7    } ...
```

Popular Posts

Maker-Checker Design Concept

Quantifying Domain Model versus Transaction Script

Decorating Error Pages with SiteMesh 2.4.x

Domain-Driven Design: Referencing Aggregates and Using DTOs

Domain-Driven Design: Accounting Domain Model

About Me

LORENZO DEE

Loves root beer and milkshake. Drinks coffee. Software developer, trainer, architect, speaker, father, husband, entrepreneur, manager

View my complete profile

I'm also a proud member of:

Java Code Geek

Twitter

Links

Journey of a Thousand Miles

Unicornfree with Amy Hoy

InfoQ

Presentation Zen

Followers

Seguidores (8)

Seguir

Blog Archive

JPA Pitfalls / Mistakes

http://lorenzo-dee.blogspot.pt/2016/01/jpa-pitfalls-mistakes.html

Final comments

Persistence and data storage

- A huge area
- Quality attributes and trade offs
  - Number of users, transactions, redudancy
- Technology
  - Concurrency and transactions
  - Models
    - NOSQL (a.k.a. Non relational) –
      - Google's BigTable, Amazon's Dynamo and Cassandra used by Facebook and others. (source wikipedia)
    - Object oriented databases
    - …
- Not the scope of this course

JPA is not Hibernate

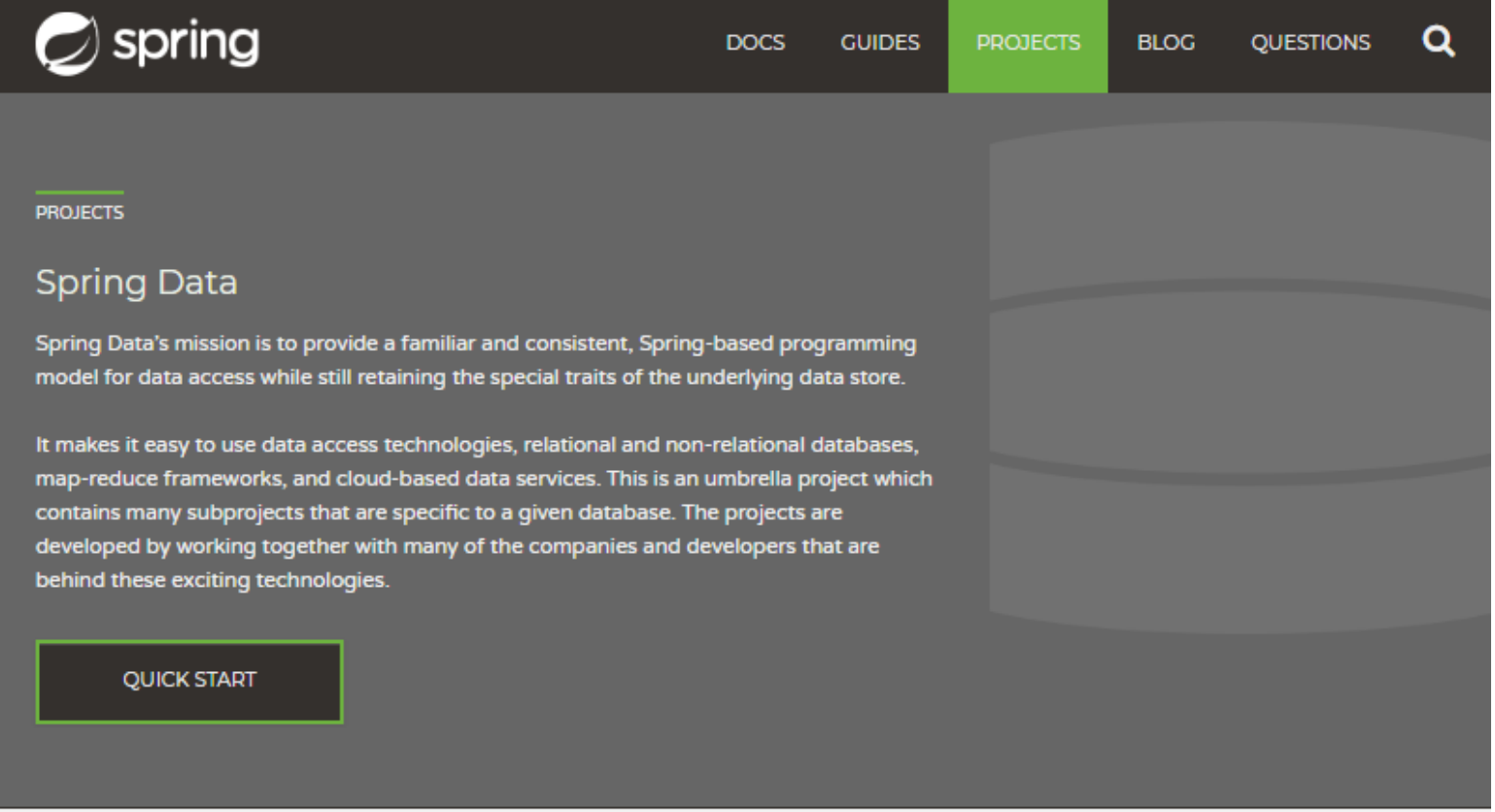
- Hibernate
  - Popular ORM **implementation**
  - was/is popular ORM solution
  - Introduced annotations
  - Strong JPA implemenation extendings also to NoSQL
- JPA is an **API**
  - **Supported by Java SE**
  - Initially focus on relational models
  - Evolved and incorporated Hibernate features/design
  - Community inputs
- BUT
  - JPA is Not Hibernate

Persistence in J2EE

- Mostly based on relational model
  - JDBC
  - New NoSQL mapping namely hibernate OMG
- May need to transform object to model
  - JPA – an “automated” way
  - Adapter – a good practice
  - Most documentation on Object relation mapping
  - Not suitable to object to NoSQL

Persistence in SpringBoot

- Spring Data
  - Support JPA
  - Support relational (JDBC) and NoSQL
  - https://www.baeldung.com/spring-data
- Same patterns, different looks
  - *repository
  - https://www.baeldung.com/spring-data-repositories
- More annotations
  - https://www.baeldung.com/spring-data-annotations



Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers

Spring Data

RELEASE

Moore-BUILD-SNAPSHOT SNAPSHOT

Lovelace-RELEASE

Lovelace-BUILD-SNAPSHOT SNAPSHOT

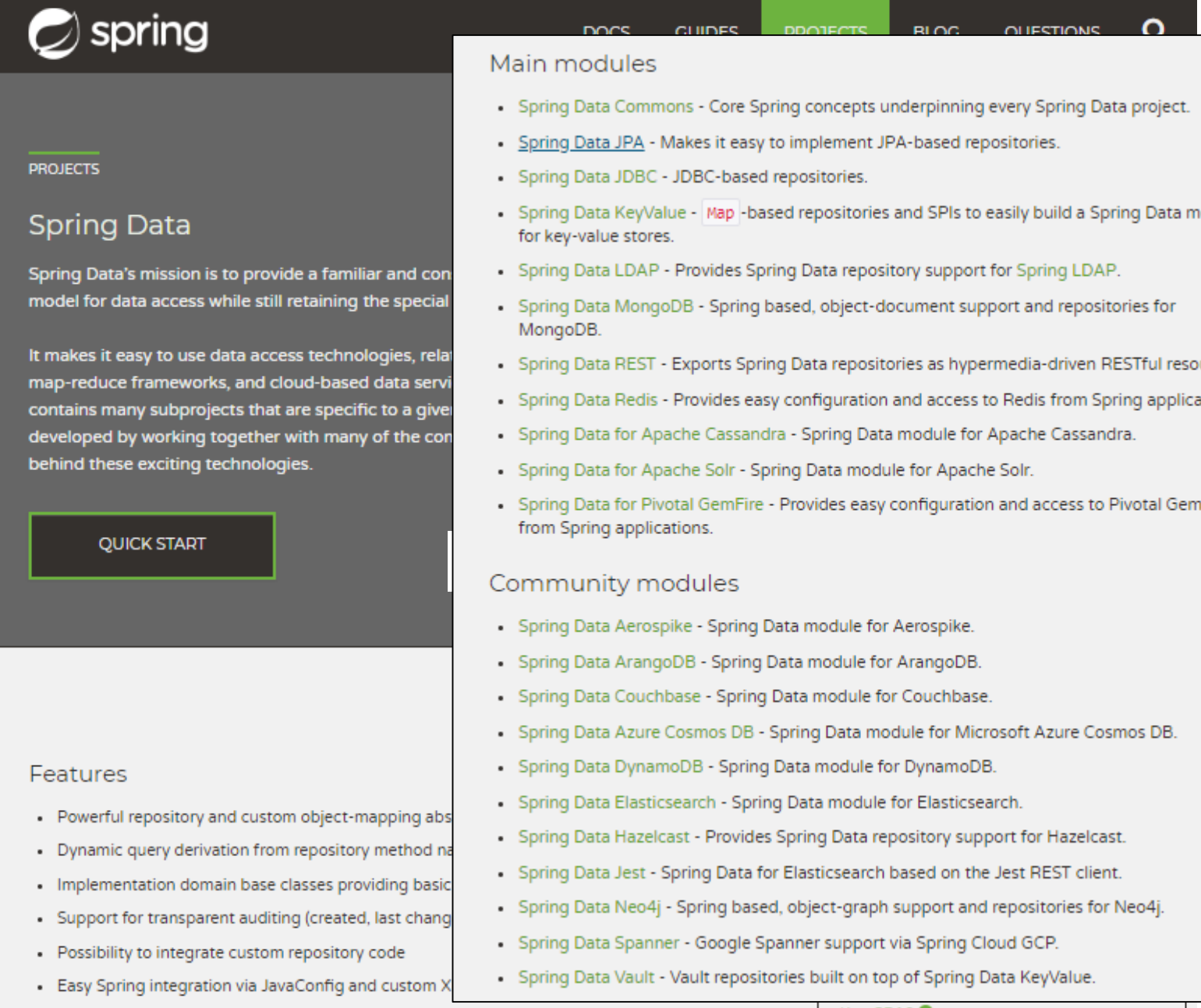
Kay-SR10

Kay-BUILD-SNAPSHOT SNAPSHOT

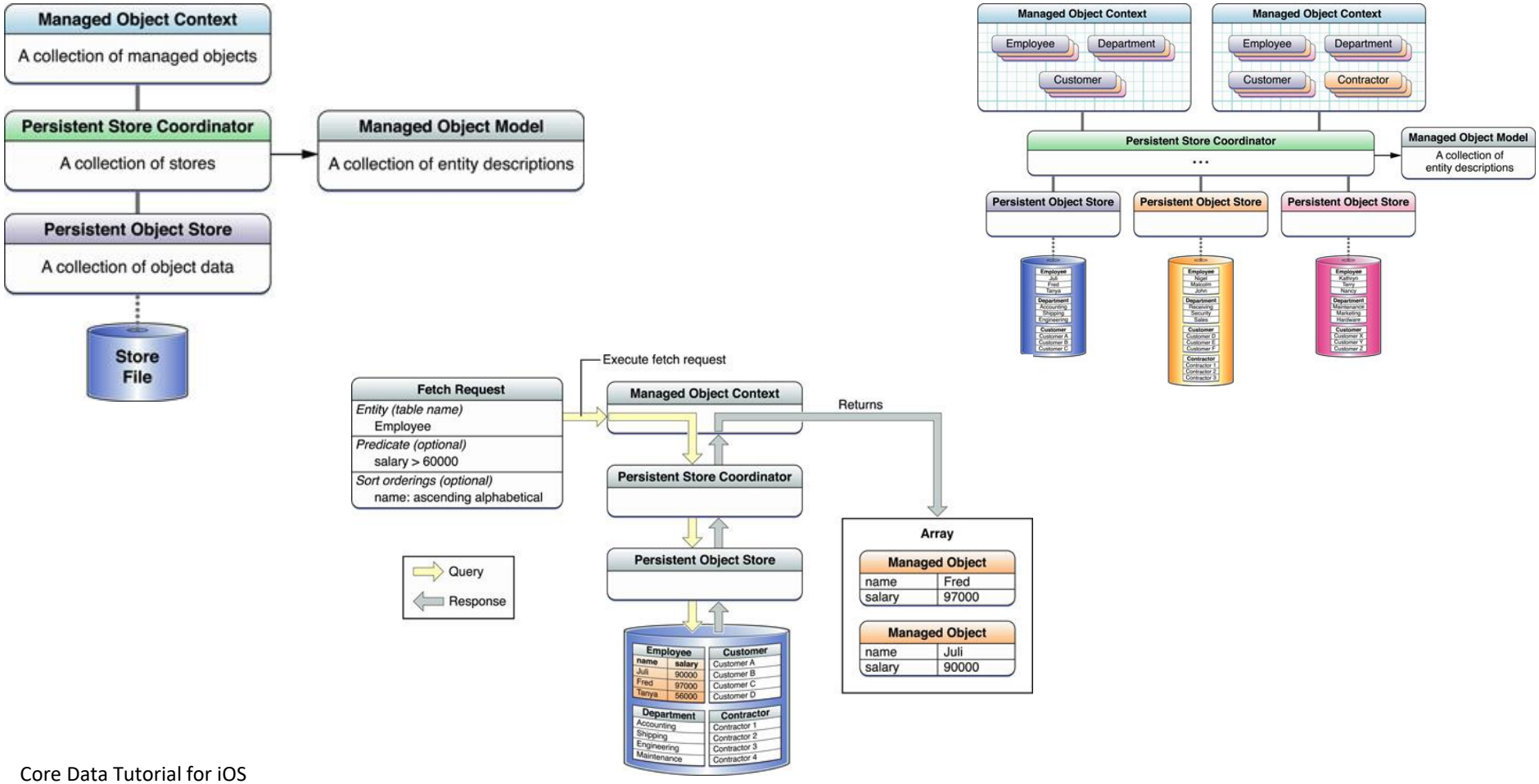
Ingalls-SR15

http://projects.spring.io/spring-data/

Main modules


spring
DOCS
GUIDES
PROJECTS
BLOG
QUESTIONS
PROJECTS
Spring Data
Spring Data's mission is to provide a familiar and con
model for data access while still retaining the special
It makes it easy to use data access technologies, rela
map-reduce frameworks, and cloud-based data servi
contains many subprojects that are specific to a give
developed by working together with many of the con
behind these exciting technologies.
QUICK START
Main modules
Spring Data Commons - Core Spring concepts underpinning every Spring Data project.
Spring Data JPA - Makes it easy to implement JPA-based repositories.
Spring Data JDBC - JDBC-based repositories.
Spring Data KeyValue - Map -based repositories and SPIs to easily build a Spring Data module for key-value stores.
Spring Data LDAP - Provides Spring Data repository support for Spring LDAP.
Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.
Spring Data Redis - Provides easy configuration and access to Redis from Spring applications.
Spring Data for Apache Cassandra - Spring Data module for Apache Cassandra.
Spring Data for Apache Solr - Spring Data module for Apache Solr.
Spring Data for Pivotal GemFire - Provides easy configuration and access to Pivotal GemFire from Spring applications.
Community modules
Spring Data Aerospike - Spring Data module for Aerospike.
Spring Data ArangoDB - Spring Data module for ArangoDB.
Spring Data Couchbase - Spring Data module for Couchbase.
Spring Data Azure Cosmos DB - Spring Data module for Microsoft Azure Cosmos DB.
Spring Data DynamoDB - Spring Data module for DynamoDB.
Spring Data Elasticsearch - Spring Data module for Elasticsearch.
Spring Data Hazelcast - Provides Spring Data repository support for Hazelcast.
Spring Data Jest - Spring Data for Elasticsearch based on the Jest REST client.
Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.
Spring Data Spanner - Google Spanner support via Spring Cloud GCP.
Spring Data Vault - Vault repositories built on top of Spring Data KeyValue.
Features
Powerful repository and custom object-mapping abs
Dynamic query derivation from repository method na
Implementation domain base classes providing basic
Support for transparent auditing (created, last chang
Possibility to integrate custom repository code
Easy Spring integration via JavaConfig and custom X
Advanced integration with Spring MVC controllers
Kay-SR10
Kay-BUILD-SNAPSHOT
SNAPSHOT
Ingalls-SR15
http://projects.spring.io/spring-data/
Main modules

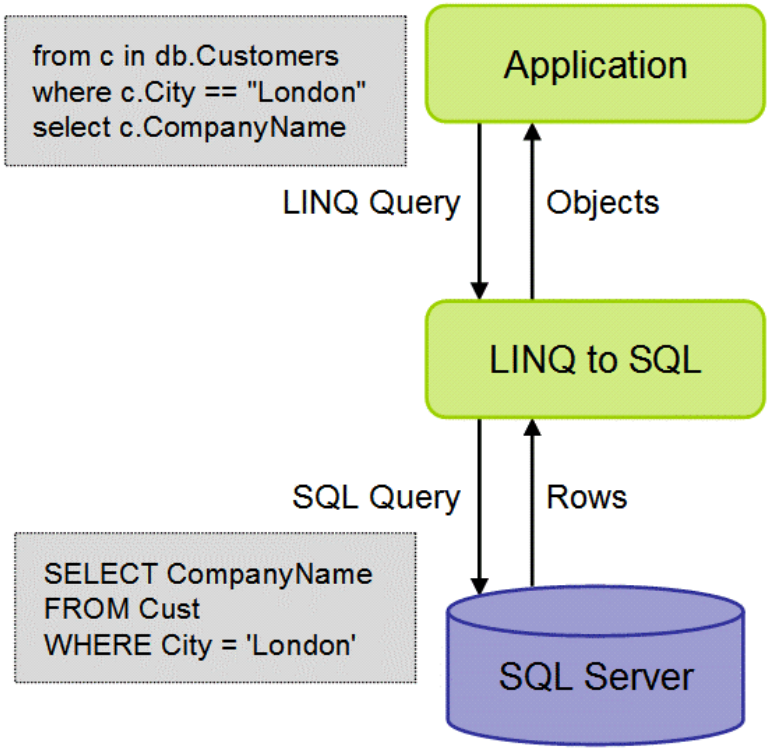
iOS CoreData: similar ....

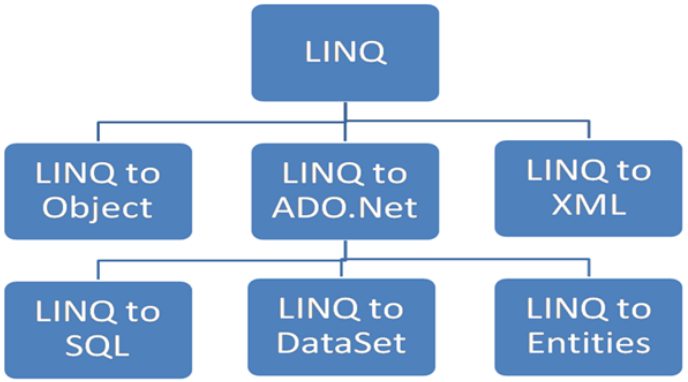


Core Data Tutorial for iOS
http://developer.apple.com/library/ios/#documentation/DataManagement/Conceptual/iPhoneCoreData01/Introduction/Introduction.html

Introduction to Core Data Programming Guide
http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html

LiNQ from .NET: similar ....





LINQ: .NET Language-Integrated Query
http://msdn.microsoft.com/en-us/library/bb308959.aspx
http://msdn.microsoft.com/en-us/library/bb397926.aspx

The END