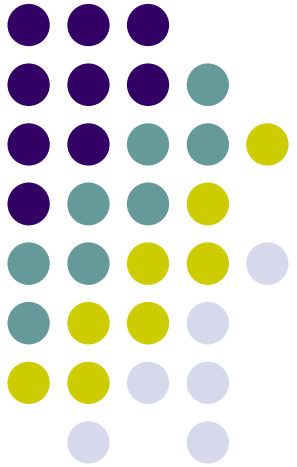
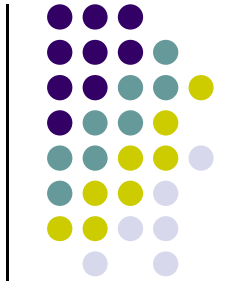
Persistencia de Objetos con Hibernate



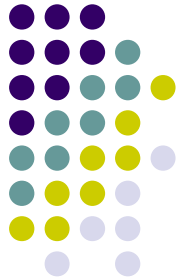
Contexto actual



Actualmente en el mercado existen ya algunas bases de datos orientadas a objetos pero a nivel comercial no suelen ser una opción aún. Las empresas buscan el respaldo y la confianza de productos probados y consolidados a lo largo de un tiempo aceptable.

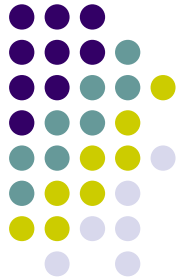
Las bases de datos relacionales aún están en su pleno apogeo, todavía librando algunas batallas respecto a su performance cuando de alto volumen transaccional se habla.

Contexto actual



En los últimos 20 años se ha invertido muchísimo dinero y esfuerzo en producir bases de datos relacionales en pro de obtener mayor rendimiento, también se han desarrollado infinitas herramientas dedicadas a su uso (reporting, modelado, administración, etc.) mas cientos de miles de sistemas que utilizan las organizaciones de hoy basados en BD relacionales. Esto es un gran freno para las tecnologías de 00DB.

Contexto actual



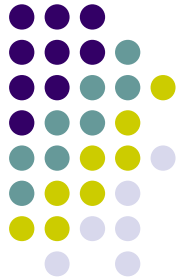
De la misma manera los lenguajes orientados a objetos son la promesa y la alternativa mas elegida a la hora de nuevos desarrollos y reingenierías de sistemas. De esta forma en los escenarios actuales se suele trabajar con lenguajes orientados a objetos y se utilizan bases de datos relacionales, la combinación de estas tecnologías es un tema ampliamente discutido y de estas discusiones han surgido diferentes corrientes arquitectónicas.

Esta mezcla de tecnologías basadas en diferentes paradigmas plantea vanos problemas importantes:

1. *Se debe diseñar un sistema orientado a objetos y la base de datos debe ser creada en base a este modelo?*

2. *Se debe crear un modelo de datos y diseñar un modelo de objetos dependiente de la estructura de datos?*

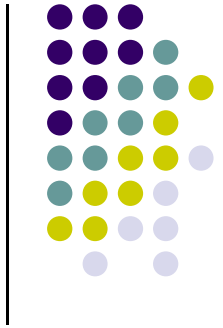
Posibles soluciones OO



- Clases de acceso.
En este tipo de estrategia y respecto al acceso a datos es comúnmente asociado con nombres como DAO (Data Access Object), DAC (Data Access Component). Para el acceso a datos esta estrategia delega las responsabilidades de acceder y manejar el código SQL (tanto en queries simples como en el acceso a Store procedures) a ciertas clases.

- ORM (Object Relation Mapper)
En este tipo de estrategia o de generación dinámica de SQL, se basa en archivos de mapeos. Es la estrategia más flexible, usa frameworks específicos diseñados para tener una mayor independencia con la DB. Consecuencias de esta estrategia:
Es poco invasiva, nos permite trabajar casi sin cambios en nuestro modelo.

¿Qué es Hibernate?

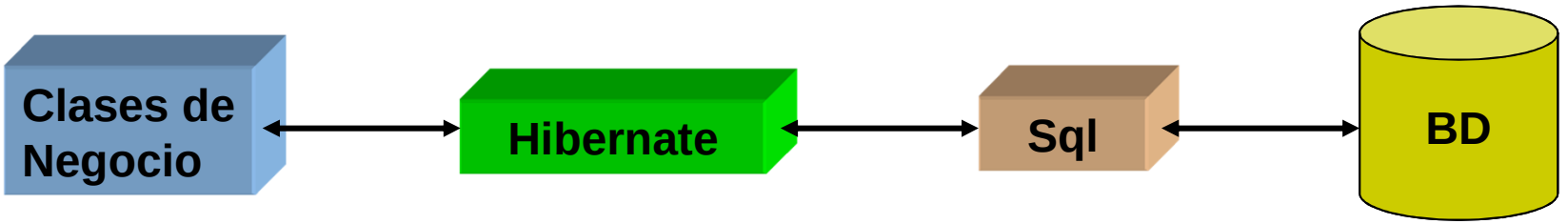


-Hibernate es un producto considerado una capa (persistence layer), framework, u object-relational mapper, tanto para JAVA (Hibernate) como para .NET (NHibernate) Este producto ya posee un nivel importante de maduración en el mercado en el orden de los 6 años. Es provisto por JBOSS, empresa que del grupo RED HAT (proveedora de una versión Linux con soporte para empresas) brindando una serie de características adicionales que le permiten mantener un liderazgo en este tipo de productos junto a los productos de otros grandes vendors como TOP Link (de Oracle), EJB (nativo de J2EE de Sun MicroSystem) entre otros.

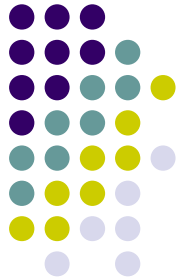
-Básicamente Hibernate permite desarrollar clases persistentes basándose en Java y en el paradigma OO (incluyendo asociaciones, herencias, polimorfismo, composición y las colecciones de framework Java) sin necesidad de trabajar directamente con la base de datos y con sql.

¿Qué es Hibernate?

Hibernate es el puente entre nuestra aplicación y la BBDD, sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.



Características

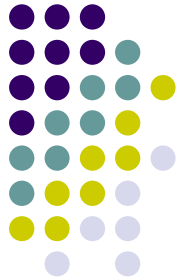


Relaciones 1-1 / Relaciones 1-n / Relaciones n-m.

Inherente a las colecciones de objetos, permite mantener esta convención de diseño manteniendo independiente su implementación en la base de datos ya sea teniendo referencias distintas tablas o a tablas de relación en el caso de relaciones n-m.

Uni-direccionalidad, Bi-Direccionalidad en 1-n, n-m en todas las asociaciones. Incluye la posibilidad de configurar y administrar las referencias entre los objetos asociados.

Características



Lazy Load en relaciones 1-n, n-m.

Permite configurar de que manera de van a ir instanciando y accediendo los objetos miembros de colecciones en aquellos objetos que las contengan.

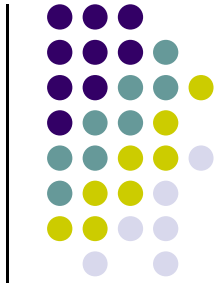
Updates y Deletes en Cascada.

Puede administrar el mantenimiento de algunas de los requerimientos de integridad referenciar en la base desde el modelo y viceversa.

Mapeo de herencias.

Con tres tipos de estrategias:
1) tabla única para toda la generalización
2) Mapeo uno a uno clase – tabla para toda la generalización con “foreing key” para vinculación de extensión de clases concretas
3) solo clases concretas a tablas.

Características



Estrategias de generación de IDs

Dado que es necesario mantener e identificar la identidad de los objetos aún persistidos con su relación de las tuplas que componen el estado interno en la base. Provee algoritmos de generación de Id: HiLo, Identidad, Secuencia. Soporte de Id compuestos, etc.

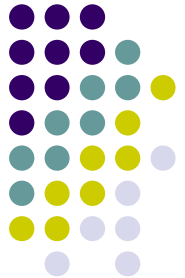
Soporte para múltiples bases.

Permite y facilita el intercambio de principales vendores como DB2, SQL Server, Oracle entre otras sin necesidad de cambiar nada de nuestro moleo.

Lenguaje de consulta de objetos HQL propietario

Para casos de aplicaciones de mediano rango que necesiten mayor accesibilidad al código de consultas, es fácil de asimilar para el desarrollador.

¿Qué ventaja tiene Hibernate?

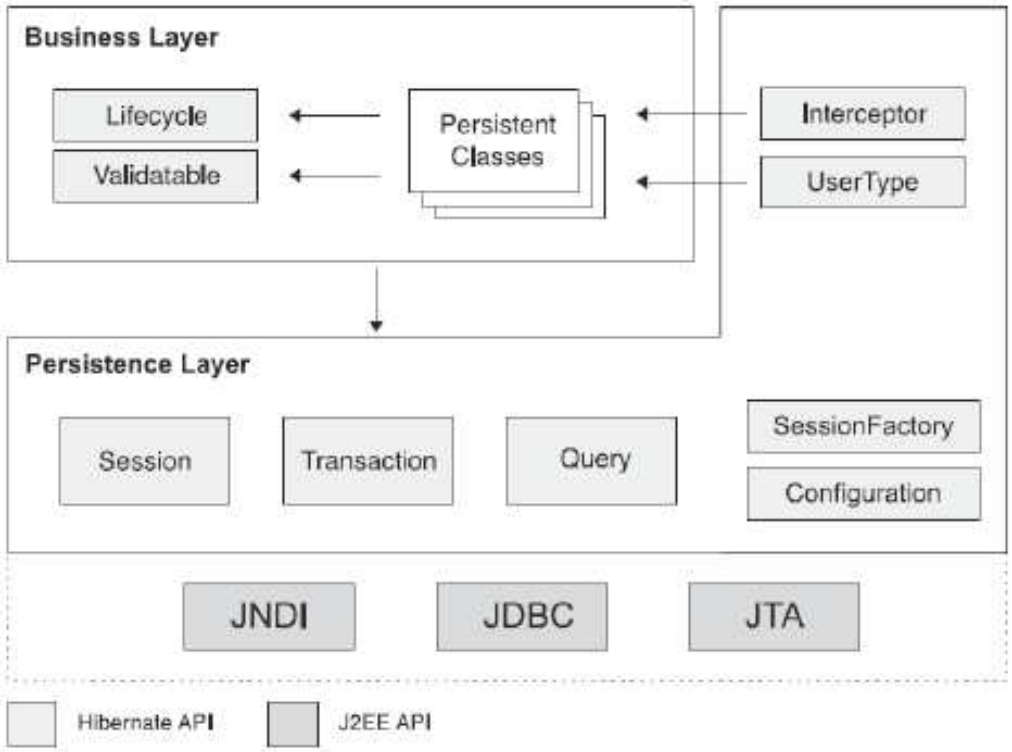


Permite crear desde un modelo de alto nivel que es independiente de la plataforma (Platform Independent Model – PIM), el cual es un modelo de dominio, totalmente independiente de hibernate, a un modelo de bajo nivel de abstracción que es específico de la plataforma (Platform Specific Model – PSM), donde contiene información que es especifico para hibernate.

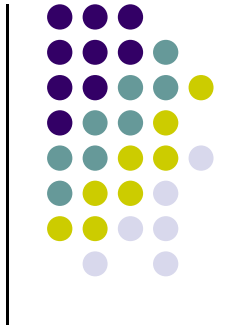
Reducir los tiempos de desarrollo y mantenimiento de nuestras aplicaciones

Arquitectura de Hibernate

La siguiente Figura muestra los roles de las interfaces Hibernate más importantes en las capas de persistencia y de negocio de una aplicación J2EE. La capa de negocio está situada sobre la capa de persistencia, ya que la capa de negocio actúa como un cliente de la capa de persistencia.



Arquitectura de Hibernate



INTERFACES

• ***Session*** interfaz primaria en toda aplicación Hibernate. Instancia "poco pesada, su creación y destrucción es muy "barata". (Creación y destrucción de varias sesiones)

• ***SessionFactory*** permite obtener instancias *Session*. Típicamente hay una

única *SessionFactory* para toda la aplicación. Una por cada base de datos.

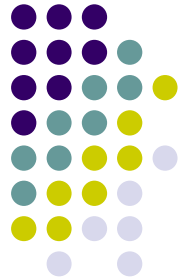
• ***Configuration*** se utiliza para configurar y "arrancar" Hibernate.

• ***Query*** permite realizar peticiones a la base de datos y controla cómo se ejecuta. Se escriben en **HQL** o en el dialecto SQL nativo

• ***Type*** hace corresponder un tipo Java con un tipo de una columna de la base de datos.

Todas las propiedades persistentes de las clases persistentes, incluyendo las asociaciones, tienen un tipo Hibernate correspondiente. Esto lo hace altamente flexible y extendible. Incluso se permiten tipos definidos por el usuario (interfaz *UserType* y *CompositeUserType*).

Archivo de configuración inicial de Hibernate

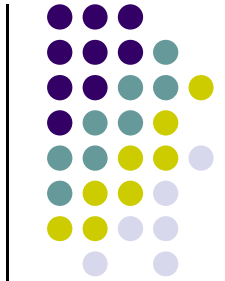


Hibernate trabaja muy bien en ambientes de producción con la mayoría de las aplicaciones de J2EE y con servidores de aplicación.

Hibernate facilita la utilización de objetos tipo POJO (Plain Old Java Object) para aprovechar la capa de persistencia de las clases (propiedades accesibles en las clases a través de los métodos setter y setter).

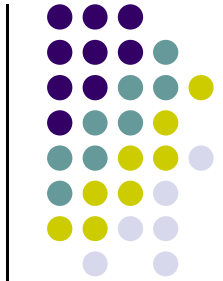
Para comenzar a utilizar hibernate lo primero que tenemos que realizar en configurar con que base de datos queremos trabajar y cuales son nuestros archivos de mapeo (opcionalmente podemos utilizar anotaciones).

Archivo de configuración inicial de Hibernate



```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>

      <!-- Database connection settings -->
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost/testdb</property>
      <property name="connection.username">root</property>
      <property name="connection.password">root</property>

      <!-- SQL dialect -->
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">true</property>

      <!-- Drop and re-create the database schema on startup -->
      <property name="hbm2ddl.auto">update</property>

      <mapping resource="edu/hibernate/bo/Cliente.hbm.xml"/>
      ……..
   </session-factory>
</hibernate-configuration>
```

Mapeo de Objetos a la BD



- **Clase Modelo**

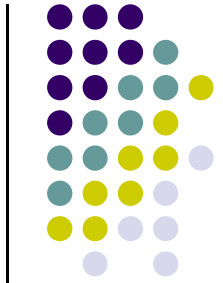
```
package de.gloegl.road2hibernate;

import java.util.Date;

public class Event {
    private String title;
    private Date date;
    private Long id;

    public Long getId() {
        return id;
    }

    private void setId(Long id) {
        this.id = id;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

- **Archivo XML de Mapeo**

Cada clase persistente tiene que tener un ID de atributo, esta propiedad es usada para distinguir los objetos persistentes, este concepto es llamado Database Identity

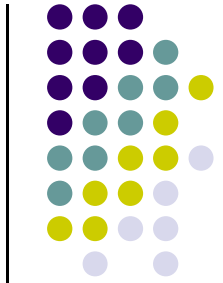
```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

        <class name="de.gloegl.road2hibernate.Event" table="EVENTS">
                <id name="id" column="uid" type="long">
                        <generator class="increment"/>
                </id>
                <property name="date" type="timestamp"/>
                <property name="title" column="eventtitle"/>
        </class>

</hibernate-mapping>
```

Hola Mundo con Hibernate



```
package edu.hibernate.bo;

public class Producto {

   private long id;
   private String descripcion;
   private double precio;

   public Producto() {}

   public Producto(String descipcion, double precio) {
        this.descripcion = descipcion;
        this.precio = precio;
   }

   public String getDescripcion() {
         return descripcion;
   }

   public long getId() {
       return id;
   }

   public double getPrecio() {
       return precio;
   }

   public void setDescripcion(String descripcion) {
       this.descripcion = descripcion;
   }

   public void setId(long id) {
     ……... etc
}
```

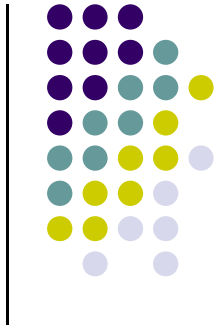
Hola Mundo con Hibernate



Archivo Producto.hbm.xml en el mismo directorio de la clase

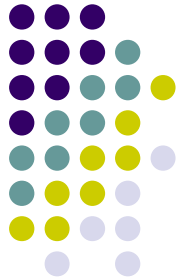
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="edu.hibernate.bo.Producto">

        <id name="id" column=“id" >
            <generator class="native" />
        </id>

        <property name="descripcion"  column="descripcion"/>
        <property name="precio" column=“precio"/>

    </class>
</hibernate-mapping>
```

Hola Mundo con Hibernate

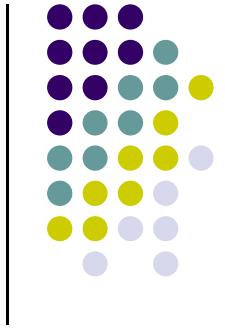


```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>

      <!-- Database connection settings -->
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost/testdb</property>
      <property name="connection.username">root</property>
      <property name="connection.password">admin</property>

      <property name="connection.pool_size">5</property>

      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

      <property name="show_sql">true</property>

      <!-- Drop and re-create the database schema on startup -->
      <property name="hbm2ddl.auto">update</property>

      <mapping resource="edu/hibernate/bo/Producto.hbm.xml"/>
      ……..
   </session-factory>
</hibernate-configuration>
```

Guardar un nuevo objeto

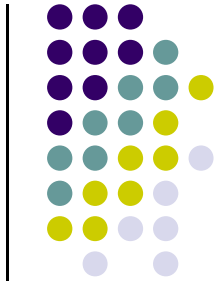


```
// Crea una factoría de Sesiones a partir del archivo hibernate.cfg.xml
// configurar y crear una única instancia del SessionFactory
Configuration  config = new Configuration();
SessionFactory sessionFactory = config.configure().buildSessionFactory();

Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Producto p = new Producto();
p.setDescripcion(“Monitor LCD 17");
p.setPrecio(1420);
session.save(p);

tx.commit();
session.close();
```



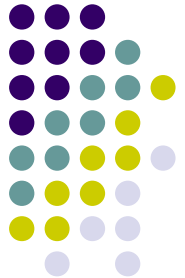
Trabajando con Hibernate

Guardar un nuevo objeto



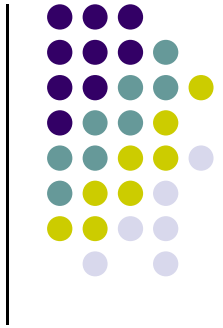
```
// Crea una factoría de Sesiones a partir del archivo hibernate.cfg.xml
// configurar y crear una única instancia del SessionFactory
Configuration  config = new Configuration();
SessionFactory sessionFactory = config.configure().buildSessionFactory();

Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

User user = new User();
user.setNombre(“Juan");
user.setApellido(“Perez");
session.save(user);

tx.commit();
session.close();
```

Recuperar un Objeto por OID



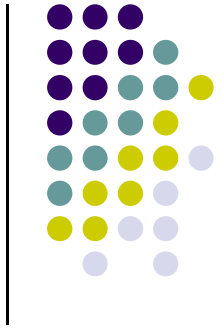
```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

int userID = 1234;
User user = (User) session.get(User.class, new Long(userID));

tx.commit();
session.close();
```

Actualizar un Objeto



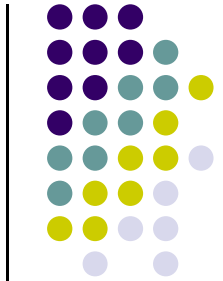
```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

int userID = 1234;
User user = (User) session.get(User.class, new Long(userID));

user.setNombre(“Maria");
session.update(user);

tx.commit();
session.close();
```

Borrar un objeto



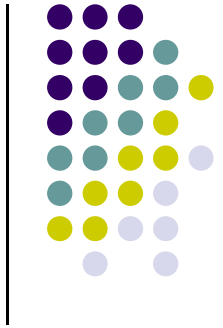
```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

int userID = 1234;
User user = (User) session.get(User.class, new Long(userID));
session.delete(user);

tx.commit();
session.close();
```

Consulta de objetos: HQL



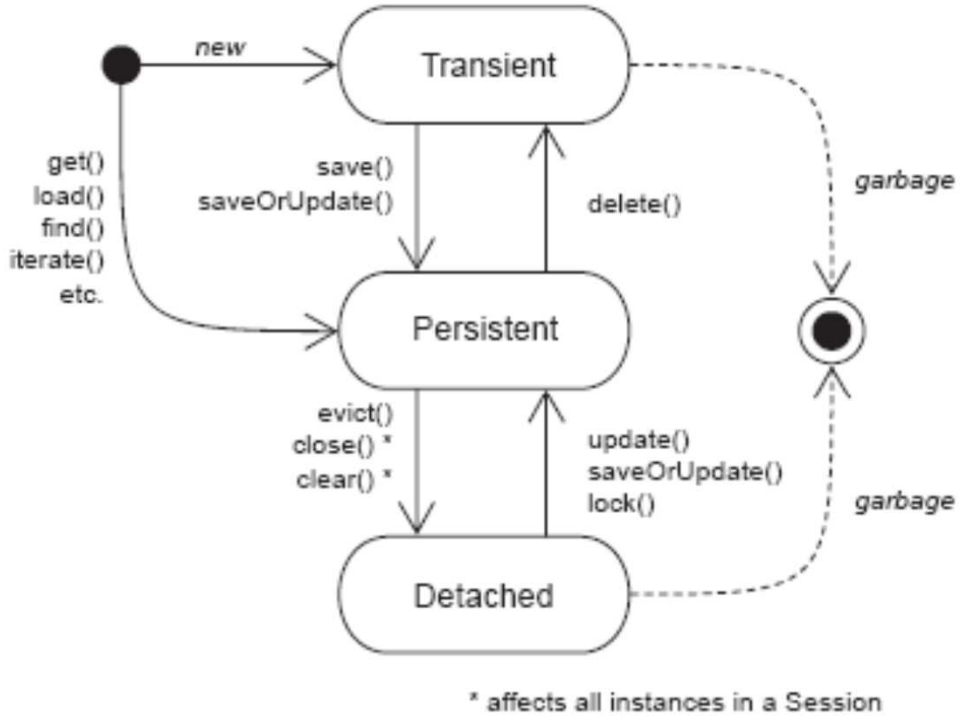
```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Query query = session.createQuery("from Producto as producto where
      producto.precio > 10");

List productos = query.list();

Iterator iterator = productos.iterator();
while (iterator.hasNext()) {
      Producto p = (Producto) iterator.next();
      System.out.println(p.getId());
      System.out.println(p.getDescripcion());
      System.out.println(p.getPrecio());
      System.out.println("");
}
tx.commit();
session.close();
```

Lock sobre un Objeto

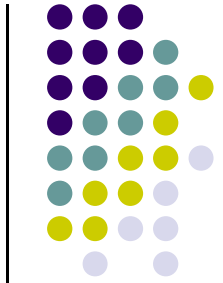


```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

int userID = 1234;
User user = (User) session.get(User.class, new Long(userID), LockMode.UPGRADE)

user.setNombre(“Maria");
session.update(user);

tx.commit();
session.close();


// LockMode.WRITE
// LockMode.UPGRADE
// LockMode.READ
```

Ciclo de vida de un Objeto Persistente



Mapeo de asociaciones



Uno-a-Uno

- **Mantenido con Foreign Keys en la Base de Datos**

Uno-a-Mucho / Muchos-a-uno

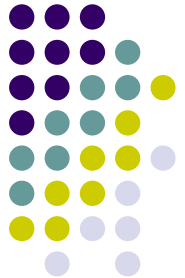
- **Objecto en el lado ‘uno’**
- **Colección en el lado ‘muchos’**

Muchos-a-Muchos

- **Usa una tabla de ‘mapping’ en la base de datos.**

- **Ver ejemplos.**

Métodos



Uno-a-Uno

- **Mantenido con Foreign Keys en la Base de Datos**

Uno-a-Mucho / Muchos-a-uno

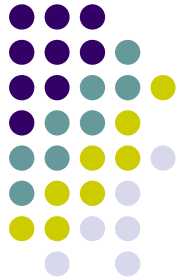
- **Objecto en el lado ‘uno’**
- **Colección en el lado ‘muchos’**

Muchos-a-Muchos

- **Usa una tabla de ‘mapping’ en la base de datos.**

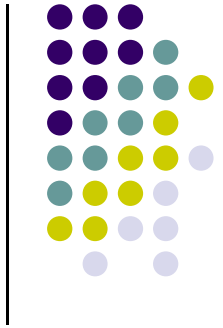
- **Ver ejemplos.**

Annotations - Intro



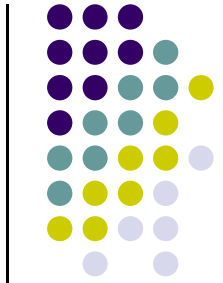
```
@Entity
public class Flight  {
  private Long id;

  @Id
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }
}
```

Annotations - Intro



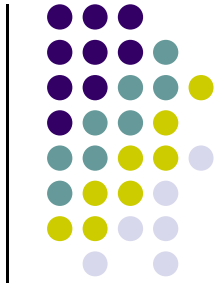
```
@Entity
@Table(name=“tablaFligth")
public class Flight  {
  private Long id;

  @Id
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }
}
```

Annotations - Column



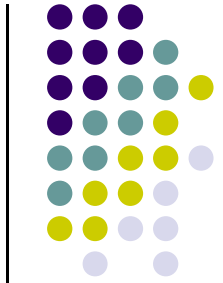
```
@Entity
public class Flight  {

  @Version
  @Column(name="OPTLOCK")
  public Integer getVersion() { ... }
}
```

Annotations - Column



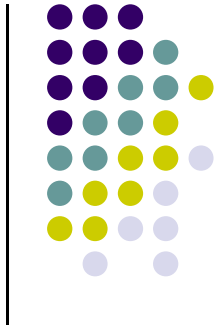
```
@Transient
public String getLengthInMeter() { ... } //transient property
String getName() {... } // persistent property

@Basic
public int getLength() { ... } // persistent property
@Basic(fetch = FetchType.LAZY)

@Temporal(TemporalType.TIME)
java.util.Date getDepartureTime() { ... } // persistent property

@Enumerated(EnumType.STRING)
Starred getNote() { ... } //enum persisted as String in database
```

Annotations - Column



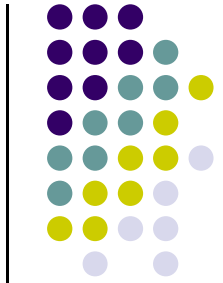
```
@Entity
public class Flight  {
...
@Column(updatable = false, name = "flight_name",
nullable = false, length=50)
public String getName() { ... }
```

Annotations - Column



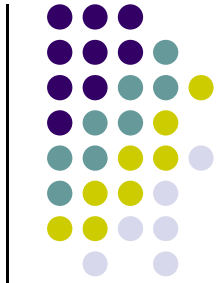
```
@Column(
name="columnName",
boolean unique() default false,
boolean nullable() default true,
boolean insertable() default true,
boolean updatable() default true,
String columnDefinition() default "",
String table() default "",
int length() default 255,
int precision() default 0, // decimal precision
int scale() default 0 // decimal scale
```

Annotations - Embedded



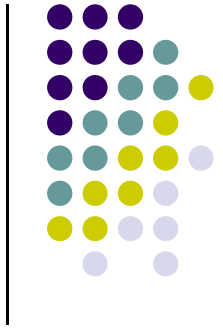
```
@Entity
public class Person {

@Embedded
@AttributeOverrides( {
@AttributeOverride(name="iso2", column =
@Column(name="bornIso2") ),
@AttributeOverride(name="name", column =
@Column(name="bornCountryName") )
} )
private Country bornIn;
...
}
```

Annotations - Embedded



```
@Embeddable
public class Country  {

private String iso2;
@Column(name="countryName")
private String name;

public String getIso2() { return iso2; }
public void setIso2(String iso2) { this.iso2 = iso2; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
...
}
```

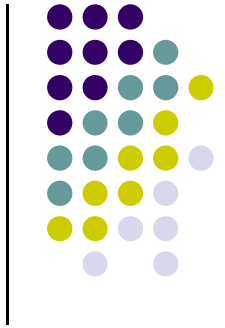
Annotations - ID



```
@Id @GeneratedValue
public Long getId() { ... }
```

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
public Long getId() { ... }
```

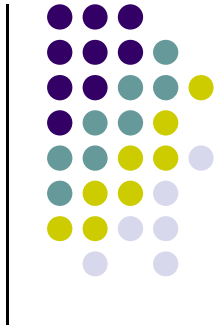
Annotations - ID



```
@Id
@GeneratedValue(strategy=GenerationType.SEQUENC
E, generator="SEQ_STORE")
public Integer getId() { ... }
```

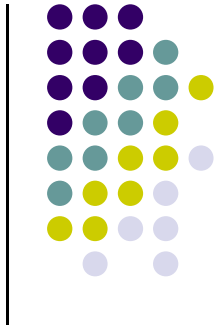
AUTO
TABLE
IDENTITY
SEQUENCE

Annotations - ID



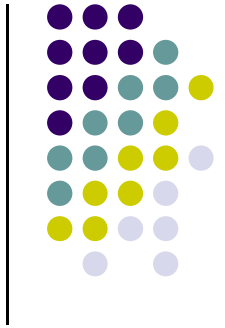
```
<table-generator name="EMP_GEN"
table="GENERATOR_TABLE"
pk-column-name="key"
value-column-name="hi"
pk-column-value="EMP"
allocation-size="20"/>

@javax.persistence.TableGenerator(
name="EMP_GEN",
table="GENERATOR_TABLE",
pkColumnName = "key",
valueColumnName = "hi"
pkColumnValue="EMP",
allocationSize=20
)
```

Annotations - ID



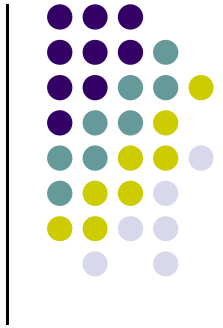
```
<sequence-generator name="SEQ_GEN"
sequence-name="my_sequence"
allocation-size="20"/>
//and the annotation equivalent

@javax.persistence.SequenceGenerator(
name="SEQ_GEN",
sequenceName="my_sequence",
allocationSize=20
)
```

Annotations - ID



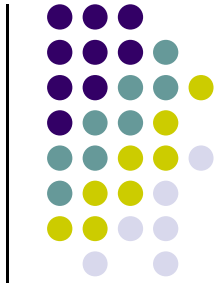
```
@Entity
@javax.persistence.SequenceGenerator(
name="SEQ_STORE",
sequenceName="my_sequence"
)

public class Store  {
  private Long id;
  @Id
@GeneratedValue(strategy=GenerationType.SEQUEN
CE, generator="SEQ_STORE")
  public Long getId() { return id; }
  }
```

Annotations - Herencia



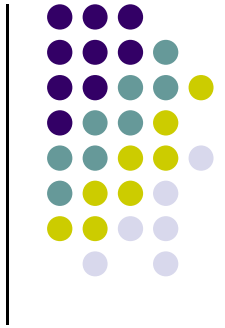
```
@Entity
@Inheritance(strategy =
InheritanceType.TABLE_PER_CLASS)
public class Flight  {
```

Annotations - Herencia



```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="planetype",
discriminatorType=DiscriminatorType.STRING
)
@DiscriminatorValue("Plane")
public class Plane { ... }

@Entity
@DiscriminatorValue("A320")
public class A320 extends Plane { ... }
```

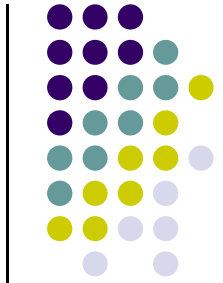
Annotations - Herencia



```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Boat  { ... }
```

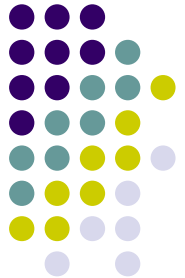
```
@Entity
@PrimaryKeyJoinColumn(name="BOAT_ID")
public class AmericaCupClass extends Boat { ... }
```

Annotations - OneToOne



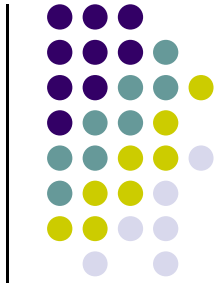
```
@Entity
public class Body {

  @Id
  public Long getId() { return id; }

  @OneToOne(cascade = CascadeType.ALL)
  @PrimaryKeyJoinColumn
  public Heart getHeart() {
    return heart;
  }
...
}
```

Annotations - OneToOne



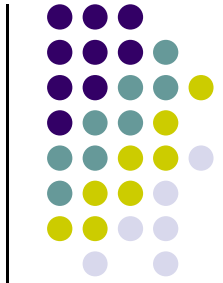
```
@Entity
public class Heart {

  @Id
  public Long getId() { ...}
  }
```

Annotations - OneToOne



```
@Entity
public class Customer {
  @OneToOne(cascade = CascadeType.ALL)
  @JoinColumn(name="passport_fk")
  public Passport getPassport() {
...
}

@Entity
public class Passport {
  @OneToOne(mappedBy = "passport")
  public Customer getOwner() {
...
}
```

Annotations - OneToOne



```
@Entity
public class Customer implements Serializable {
  @OneToOne(cascade = CascadeType.ALL)
  @JoinTable(name = "CustomerPassports",
joinColumns = @JoinColumn(name="customer_fk"),
inverseJoinColumns = @JoinColumn(name="passport_fk")
)
  public Passport getPassport() {
...
}
```

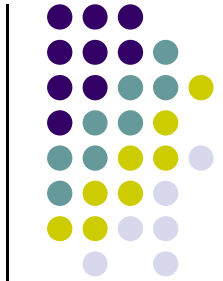
Annotations - ManyToOne

```
@Entity()
public class Flight implements Serializable {
  @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
  @JoinColumn(name="COMP_ID")
  public Company getCompany() {
    return company;
  }
...
}
```

Annotations - ManyToOne

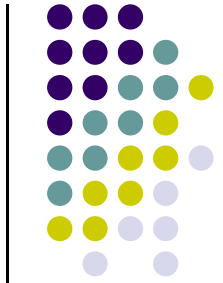
```
@Entity
public class Flight {

  @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
  @JoinTable(name="Flight_Company",
  joinColumns = @JoinColumn(name="FLIGHT_ID"),
  inverseJoinColumns = @JoinColumn(name="COMP_ID")
)
  public Company getCompany() {
  return company;
 }
...
}
```

Annotations - Collections



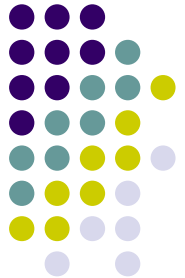
Tipo	Java Collection	
Bag	List, Collection	@OneToMany, @ManyToMany
List	List, Collection	@OneToMany, @ManyToMany
Set	Set	@OneToMany, @ManyToMany
Map	Map	

Annotations - OneToMany



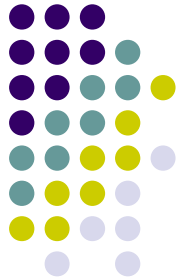
```
@Entity
public class City {
  @OneToMany(mappedBy="city")
  @OrderBy("streetName")
  public List<Street> getStreets() {
    return streets;
  }
...
}

@Entity
public class Street {
  public String getStreetName() {
    return streetName;
  }
  @ManyToOne
  public City getCity() {
    return city;
  }
...
}
```

Annotations - OneToMany



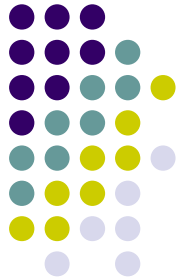
```
@Entity
public class Software {
  @OneToMany(mappedBy="software")
  @MapKey(name="codeName")
  public Map<String, Version> getVersions() {
     return versions;
  }
...
}

@Entity
@Table(name="tbl_version")
public class Version {
  public String getCodeName() {...}
  @ManyToOne
  public Software getSoftware() { ... }
...
}
```

Annotations - OneToMany



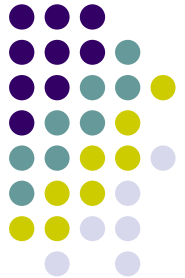
```
@Entity
public class Troop {
  @OneToMany(mappedBy="troop")
  public Set<Soldier> getSoldiers() {
...
}

@Entity
public class Soldier {
  @ManyToOne
  @JoinColumn(name="troop_fk")
  public Troop getTroop() {
...
}
```

Annotations - ManyToMany



```
@Entity
public class Store {
  @ManyToMany(cascade = {CascadeType.PERSIST})
  public Set<Customer> getCustomers() {
...
  }
}

@Entity
public class Customer {
  @ManyToMany(mappedBy="customers")
  public Set<Store> getStores() {
...
  }
}
```

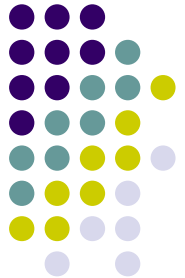
Annotations - FetchType



FetchType.Lazy => Select
FetchType.EAGER => Join

```
@ManyToMany(fetch=FetchType.EAGER)
public Set<Localidad> getLocalidades() {
   return localidades;
}

@ManyToMany(fetch=FetchType.LAZY)
@LazyCollection(LazyCollectionOption.EXTRA)
public Set<Localidad> getLocalidades() {
   return localidades;
}
```

Annotations - Query

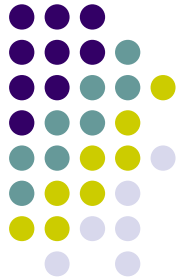


```
@Entity
@NamedQuery(name="night.moreRecentThan", query="select n from Night n where
n.date >= :date")
public class Night {
...
}

public class MyDao {
  doStuff() {
    Query q = s.getNamedQuery("night.moreRecentThan");
    q.setDate( "date", aMonthAgo );
    List results = q.list();
    ...
  }
...
}
```

Annotations - SqlQuery

```
@NamedNativeQuery(name="implicitSample", query="select s.* from
SpaceShip as s",
resultClass=SpaceShip.class)
public class SpaceShip {
```

Conclusiones



• Consideramos que esta herramienta es una excelente elección para aplicaciones de hasta mediana complejidad y volumen transaccional

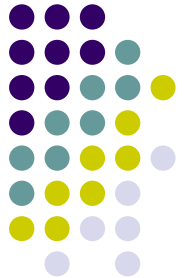
• Su implementación permite conservar y preservar el paradigma de objetos (POJOS tanto para Java como para .NET)

• Su curva de aprendizaje no es tan empinada con respecto al aprendizaje necesario que los desarrolladores deben aprender para su manejo.

• Posee mecanismos para manejo transaccional adecuados y su overhead de trabajo para incorporarlos es aceptable. Entre 10% y 30% mas lento que el sql directo

• Implica invertir un tiempo en el mapeo de las clases del modelo de dominio a las tablas.

Conclusiones



• Si bien el trabajo de mapeo consiste en tiempo, se aportan estrategias adecuadas para manejarlo

• Permite independencia de la base de datos con la que se opera.

• Permite alguna independencia de los IDE a utilizar interactuando con los más importantes.

• Es open source, permitiendo reducir los costos de desarrollo.

• Posee ya algunos años de evolución que avalan su aceptación en el mercado.

• Permite variar su configuración interna dando alternativas a cuestiones de performance, posibilita el HQL (lenguaje SQL propietario) aunque ninguna de estas dos opciones son recomendables, en términos de mantener el sistema puro o lo mas cercano posible al paradigma de objetos.