

Principles and Techniques of DBMS 7

Object-Relation Mapping I

Haopeng Chen

***RE**liable, **IN**telligent and **Scalable** Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

<http://reins.se.sjtu.edu.cn/~chenhp>

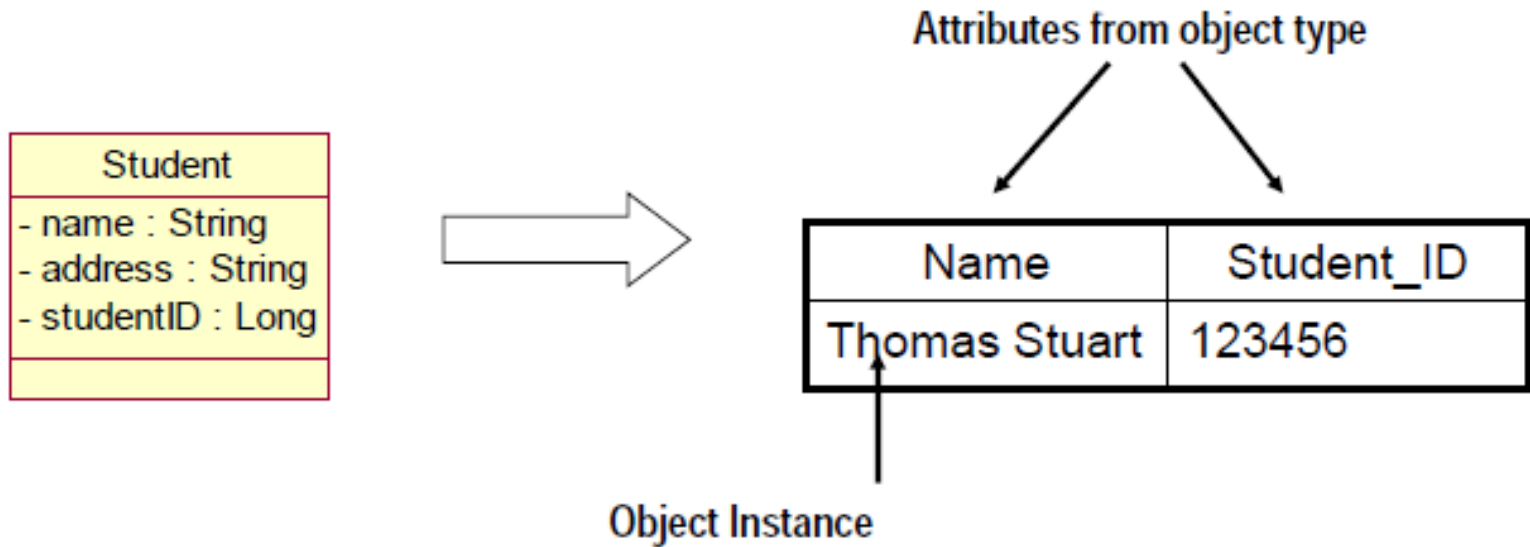
e-mail: chen-hp@sjtu.edu.cn

- O/R mapping
 - Basic O/R Mapping
 - Relationship Mapping
 - Work with Objects
 - Id Mapping
 - Id Generator

- The term Object/Relational Mapping refers to
 - the technique of mapping data from an object model representation to a relational data model representation (and visa versa)
- Hibernate is an Object/Relational Mapping solution for Java environments.

Mapping Persistent Classes to Tables

- In a relational database
 - Every row is regarded as an object
 - A column in a table is equivalent to a persistent attribute of a class



- Event.class

```
import java.util.Date;
public class Event {
    private Long id;
    private String title;
    private Date date;

    public Event() {}
    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; }
}
```

- Event.hbm.xml

```
<hibernate-mapping package="Sample.Entity">
  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="native"/>
    </id>
    <property name="date" type="timestamp"
      column="EVENT_DATE"/>
    <property name="title"/>
  </class>
</hibernate-mapping>
```

- hibernate.cfg.xml

```
<hibernate-configuration>
```

```
  <session-factory>
```

```
    <!-- Database connection settings -->
```

```
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
    <property name="connection.url">jdbc:mysql://localhost:3306/test</property>
```

```
    <property name="connection.username">root</property>
```

```
    <property name="connection.password">12345678</property>
```

```
    <!-- JDBC connection pool (use the built-in) -->
```

```
    <property name="connection.pool_size">1</property>
```

```
    <!-- SQL dialect -->
```

```
    <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
```

```
    <!-- Enable Hibernate's automatic session context management -->
```

```
    <property name="current_session_context_class">thread</property>
```

- hibernate.cfg.xml

```
<!-- Disable the second-level cache -->
<property name="cache.provider_class">
    org.hibernate.cache.internal.NoCacheProvider
</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="Sample/Entity/User.hbm.xml"/>

</session-factory>

</hibernate-configuration>
```



```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new Configuration().configure().buildSessionFactory(); }
        catch (Throwable ex) {
            // Make sure you log the exception, as it might be
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

```
@WebServlet("/EventServlet")
public class EventServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        .....
        try {
            String title = (String) request.getParameter("title");
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            String datestr = (String) request.getParameter("date");
            Date date=sdf.parse(datestr);
            Session session =
                HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Event t = new Event();
            t.setDate(date);
            t.setTitle(title);
            session.save(t);
            session.getTransaction().commit();
        }
    }
}
```

```
out.println("<h1>The event has been inserted!</h1><br>");

session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
List events = session.createQuery("from Event").list();
session.getTransaction().commit();
for (int i = 0; i < events.size(); i++) {
    Event theEvent = (Event) events.get(i);
    out.println("id: " + theEvent.getId() + "<br>" + "title: "
        + theEvent.getTitle() + "<br>" + "date: " + theEvent.getDate()
        + "<br><br>");
}
```

.....

- Person.class

```
package org.hibernate.tutorial.domain;
```

```
public class Person {
```

```
    private Long id;
```

```
    private int age;
```

```
    private String firstname;
```

```
    private String lastname;
```

```
    public Person() {}
```

```
//Accessor methods for all properties, private setter for 'id'
```

```
.....
```

```
    private Set events = new HashSet();
```

```
    public Set getEvents() { return events; }
```

```
    public void setEvents(Set events) { this.events = events; }
```

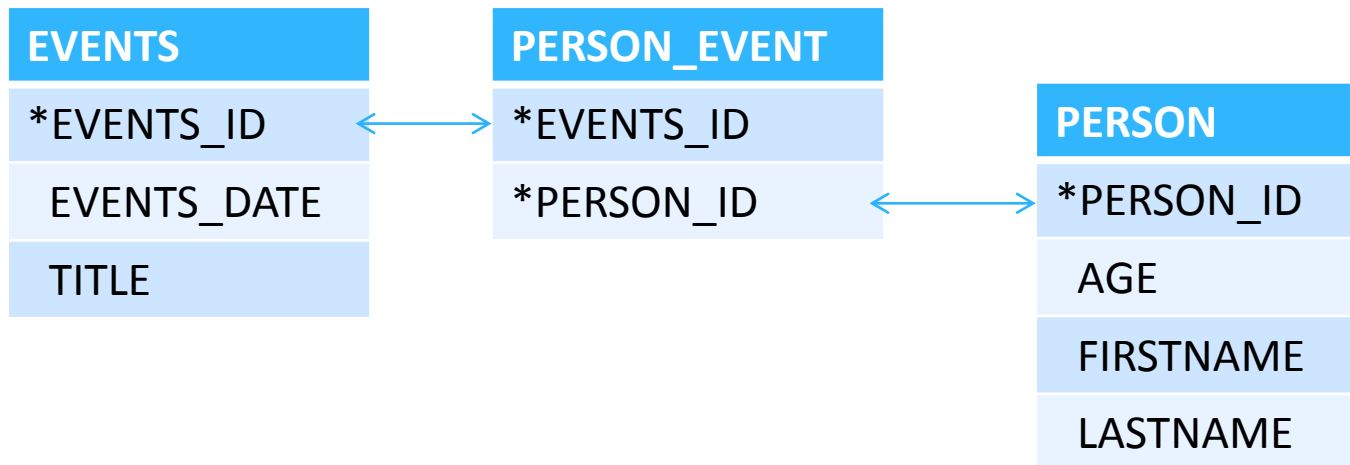
```
}
```

- Person.hbm.xml

```
<class name="Person" table="PERSON">
  <id name="id" column="PERSON_ID">
    <generator class="native"/>
  </id>
  <property name="age"/>
  <property name="firstname"/>
  <property name="lastname"/>
  <set name="events" table="PERSON_EVENT">
    <key column="PERSON_ID"/>
    <many-to-many column="EVENT_ID" class="Event"/>
  </set>
</class>
```

- Hibernate's configuration

```
<mapping resource="org/hibernate/tutorial/domain/Event.hbm.xml"/>
<mapping resource="org/hibernate/tutorial/domain/Person.hbm.xml"/>
```

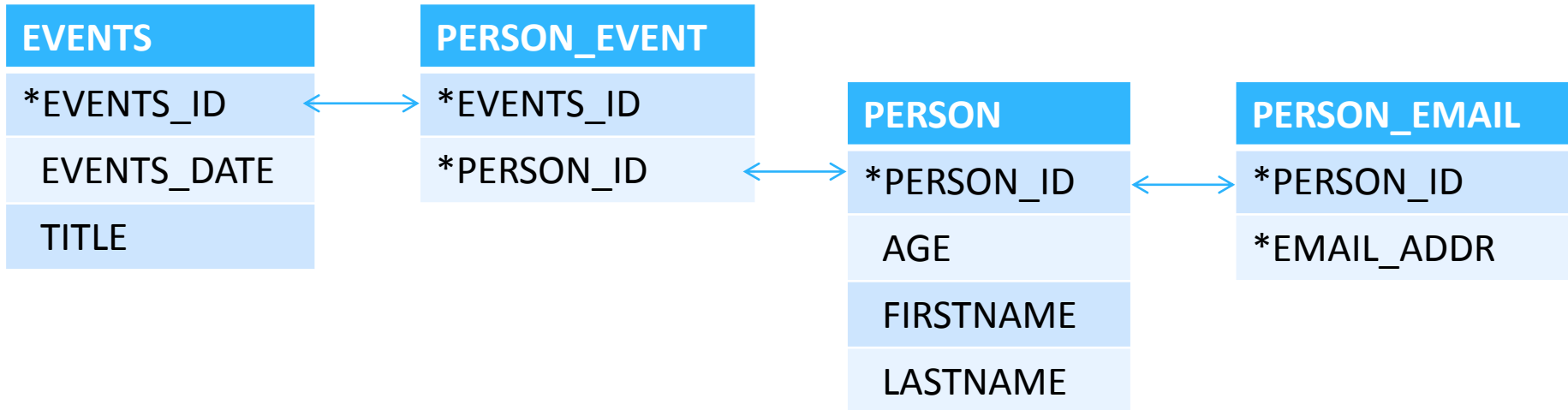


```
@WebServlet("/PersonServlet")
public class PersonServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        .....
        String event = (String) request.getParameter("event");
        String person = (String) request.getParameter("person");
        long eventId = new Long(event);
        long personId = new Long(person);

        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Person aPerson = (Person) session.createQuery(
            "select p from Person p left join fetch p.events where p.id = :pid").
            setParameter("pid", personId).uniqueResult();
        Event anEvent = (Event) session.load(Event.class, eventId);
        session.getTransaction().commit();

        aPerson.getEvents().add(anEvent);

        session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        session.update(aPerson);
        session.getTransaction().commit();
    }
}
```



- Person.class

```
public class Person {
    private Long id;
    private int age;
    private String firstname;
    private String lastname;
    public Person() {}
    // Accessor methods for all properties, private setter for 'id'

    private Set events = new HashSet();
    public Set getEvents() { return events; }
    public void setEvents(Set events) { this.events = events; }

    private Set emailAddresses = new HashSet();
    public Set getEmailAddresses() { return emailAddresses; }
    public void setEmailAddresses(Set emailAddresses)
    { this.emailAddresses = emailAddresses; }
}
```

- Person.hbm.xml

```
<class name="Person" table="PERSON">
  <id name="id" column="PERSON_ID">
    <generator class="native"/>
  </id>
  <property name="age"/>
  <property name="firstname"/>
  <property name="lastname"/>
  <set name="events" table="PERSON_EVENT">
    <key column="PERSON_ID"/>
    <many-to-many column="EVENT_ID" class="Event"/>
  </set>
  <set name="emailAddresses" table="PERSON_EMAIL">
    <key column="PERSON_ID"/>
    <element type="string" column="EMAIL_ADDRESS"/>
  </set>
</class>
```

```
private void addEmailToPerson(Long personId, String emailAddress) {  
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    Person aPerson = (Person) session.load(Person.class, personId);  
    session.getTransaction().commit();  
  
    session = HibernateUtil.getSessionFactory().getCurrentSession();  
    session.beginTransaction();  
    session.refresh(aPerson);  
    aPerson.getEmailAddresses().add("new@new.com");  
    session.getTransaction().commit();  
}
```

- **Event.class**

```
public class Event {
    private Long id;
    private String title;
    private Date date;
    public Event() {}
    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; }

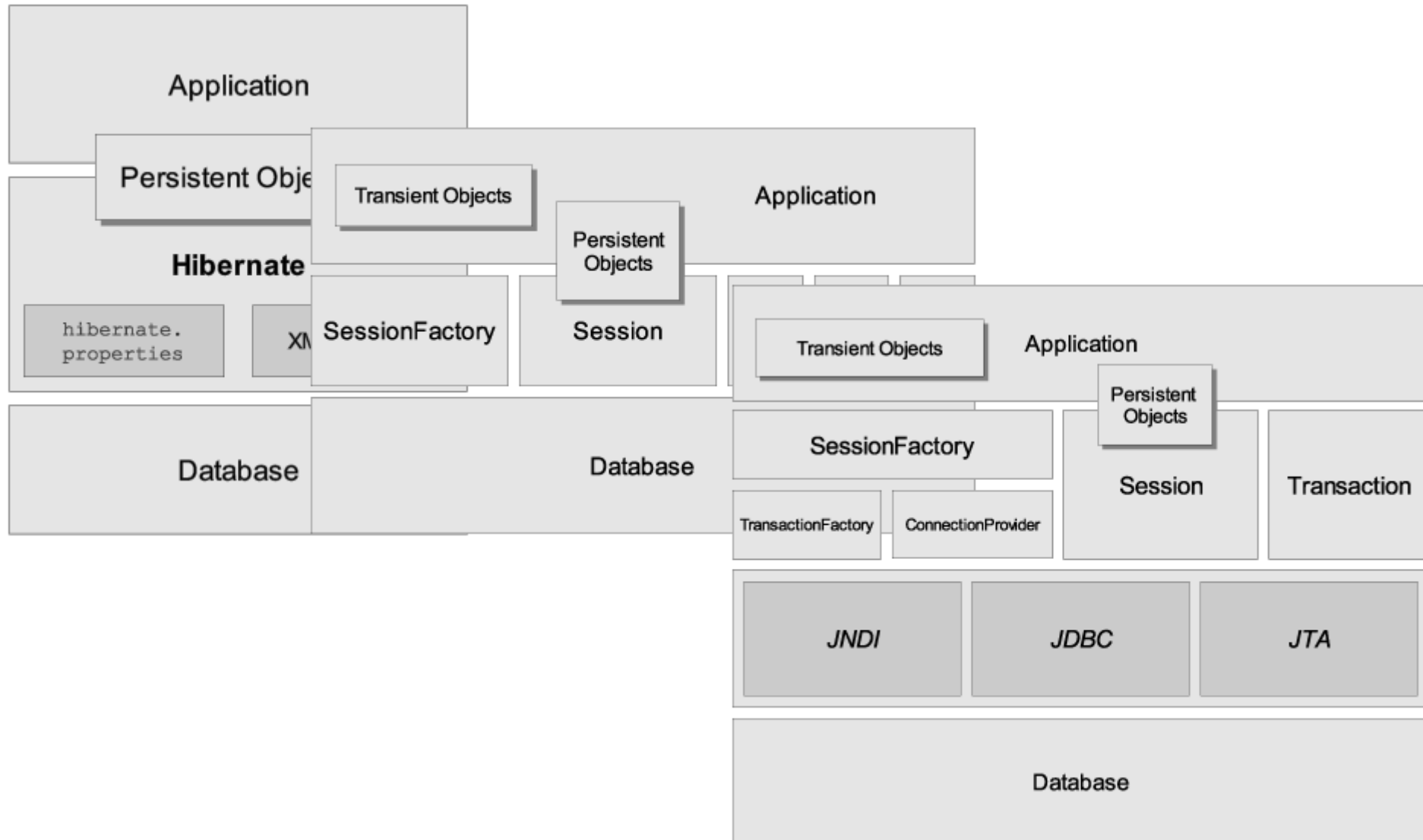
    private Set participants = new HashSet();
    public Set getParticipants() { return participants; }
    public void setParticipants(Set participants) {
        this.participants = participants;
    }
}
```

- Event.hbm.xml

```
<hibernate-mapping package="org.hibernate.tutorial.domain">
  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="native"/>
    </id>
    <property name="date" type="timestamp"
      column="EVENT_DATE"/>
    <property name="title"/>
    <set name="participants" table="PERSON_EVENT" inverse="true">
      <key column="EVENT_ID"/>
      <many-to-many column="PERSON_ID" class="Person"/>
    </set>
  </class>
</hibernate-mapping>
```

```
private void addEmailToPerson(Long personId, String emailAddress) {
    Session session =
        HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Person aPerson = (Person) session.load(Person.class, personId);
    session.getTransaction().commit();

    session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    session.refresh(anEvent);
    Set participants = anEvent.getParticipants();
    Iterator iter = participants.iterator();
    while(iter.hasNext()){
        Person thePerson = (Person)iter.next();
        out.println("Participant: " + thePerson.getFirstname() + " " +
            thePerson.getLastname() + "<br><br>");
    }
    session.getTransaction().commit();
}
```



- SessionFactory (org.hibernate.SessionFactory)
- Session (org.hibernate.Session)
- Persistent objects and collections
- Transient and detached objects and collections
- Transaction (org.hibernate.Transaction)(Optional)
- ConnectionProvider
(org.hibernate.connection.ConnectionProvider)(Optional)
- TransactionFactory
(org.hibernate.TransactionFactory)(Optional)
- *Extension Interfaces*

- An entity is a regular Java object (aka POJO) which will be persisted by Hibernate.

```
@Entity
```

```
public class Flight implements Serializable {
```

```
    Long id;
```

```
    @Id
```

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

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

```
}
```

```
@Entity
@Table(name="TBL_FLIGHT",
       schema="AIR_COMMAND",
       uniqueConstraints=
           @UniqueConstraint(
               name="flight_number",
               columnNames={"comp_prefix","flight_number"} ) )
public class Flight implements Serializable {

    @Column(name="comp_prefix")
    public String getCompagnyPrefix() {
        return companyPrefix;
    }

    @Column(name="flight_number")
    public String getNumber() { return number; }
}
```

```
name="ClassName"  
table="tableName"  
discriminator-value="discriminator_value"  
mutable="true|false"  
schema="owner"  
catalog="catalog"  
proxy="ProxyInterface"  
dynamic-update="true|false"  
dynamic-insert="true|false"  
select-before-update="true|false"  
polymorphism="implicit|explicit"  
where="arbitrary sql where condition"  
persister="PersisterClass"  
batch-size="N"  
optimistic-lock="none|version|dirty|all"  
lazy="true|false"  
entity-name="EntityName"  
check="arbitrary sql check condition"  
rowxml:id="rowid"  
subselect="SQL expression"  
abstract="true|false"  
node="element-name"
```

```
@Entity
public class Person {
    @Id Integer getId() { ... }
    ...
}
```

```
<id
  name="propertyName"
  type="typename"
  column="column_name"
  unsaved-value="null|any|none|undefined|id_value"
  access="field|property|ClassName" >
  node="element-name|@attribute-name|element/@attribute|."
  <generator class="generatorClass"/>
</id>
```

- id as a property using a component type

`@Entity`

```
class User {  
    @EmbeddedId  
    @AttributeOverride(  
        name="firstName", column=@Column(name="fld_firstname")  
    )  
    UserId id;  
    Integer age;  
}
```

`@Embeddable`

```
class UserId implements Serializable {  
    String firstName;  
    String lastName;  
}
```

- id as a property using a component type

```
@Entity
class Customer {
    @EmbeddedId
    CustomerId id;
    boolean preferredCustomer;

    @MapsId("userId")
    @JoinColumns({
        @JoinColumn(name="userfirstname_fk",
                    referencedColumnName="firstName"),
        @JoinColumn(name="userlastname_fk",
                    referencedColumnName="lastName")
    })
    @OneToOne
    User user;
}
```

```
@Embeddable
class CustomerId implements Serializable {
    UserID userId;
    String customerNumber;
    //implements equals and hashCode
}
```

- Multiple id properties without identifier type

```
@Entity
class Customer implements Serializable {
    @Id
    @OneToOne
    @JoinColumn({
        @JoinColumn(name="userfirstname_fk",
            referencedColumnName="firstName"),
        @JoinColumn(name="userlastname_fk",
            referencedColumnName="lastName")
    })
    User user;

    @Id
    String customerNumber;
    boolean preferredCustomer;
    //implements equals and hashCode
}
```

- Multiple id properties with a dedicated identifier type

```
@Entity
@IdClass(CustomerId.class)
class Customer implements Serializable {
    @Id
    @OneToOne
    @JoinColumn({
        @JoinColumn(name="userfirstname_fk", referencedColumnName="firstName"),
        @JoinColumn(name="userlastname_fk", referencedColumnName="lastName")
    })
    User user;

    @Id
    String customerNumber;
    boolean preferredCustomer;
}

class CustomerId implements Serializable {
    UserId user;
    String customerNumber
    //implements equals and hashCode
}
```


- IDENTITY
- SEQUENCE (called seqhilo in Hibernate)
- TABLE (called MultipleHiLoPerTableGenerator in Hibernate)
- AUTO

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    Integer getId() { ... };
}
```

```
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    Integer getId() { ... };
}
```

- HIBERNATE - Relational Persistence for Idiomatic Java,
 - http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/#preface



Thank You!