*Web*开发技术

Web Application Development

第11课
WEB后端框架-O/R映射II

Episode Eleven

O/R Mapping

陈昊鹏
chen-hp@sjtu.edu.cn

Overview

- O/R mapping
  - Inheritance strategy
  - Work with object

Modeling Inheritance

- A data model does not support modeling inheritance in a direct way
- Three Strategies
  - table per subclass
  - table per concrete class
  - table per class hierarchy

Modeling Inheritance

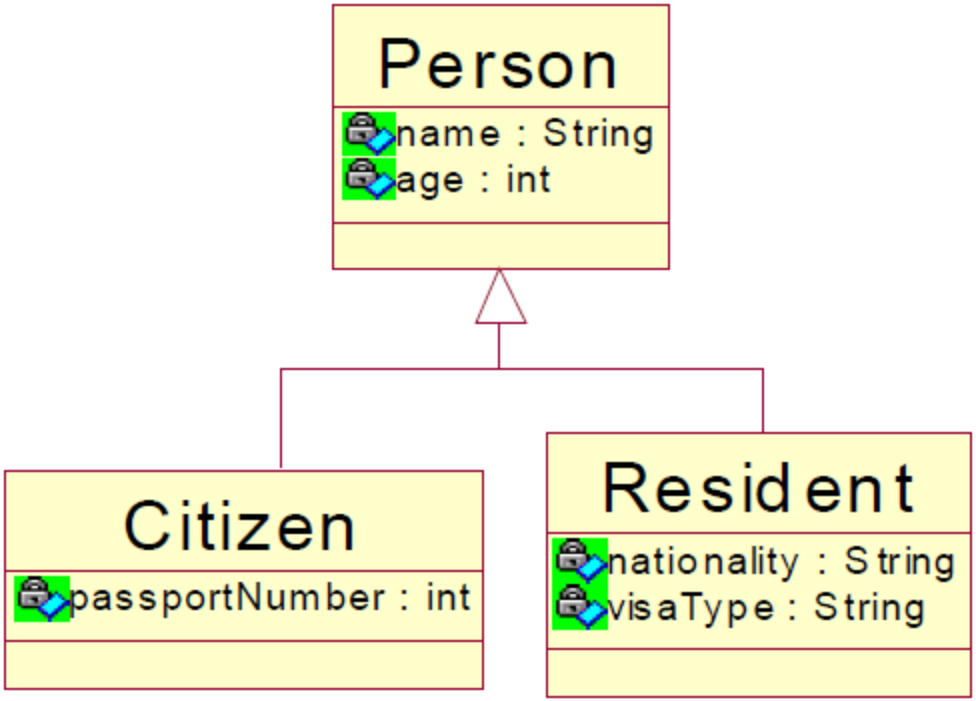
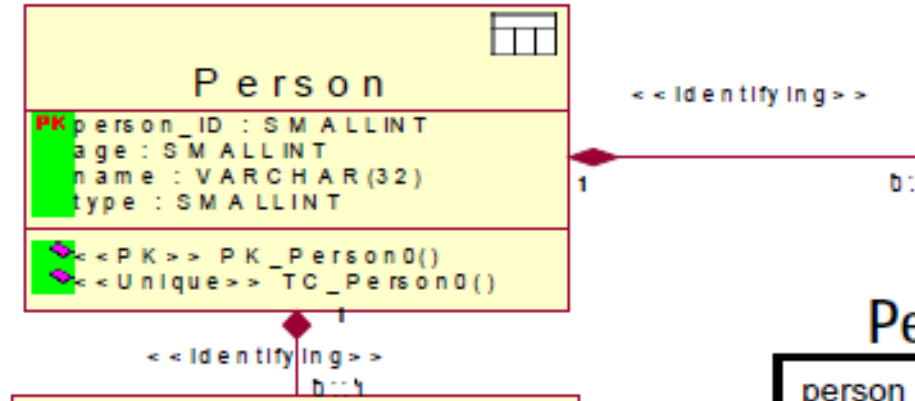


Table per subclass

REliable, INtelligent & Scalable Systems



Person table

person_ID	type	name	age
1	citizen	John	19
2	resident	Mike	73
3	person	Julie	20

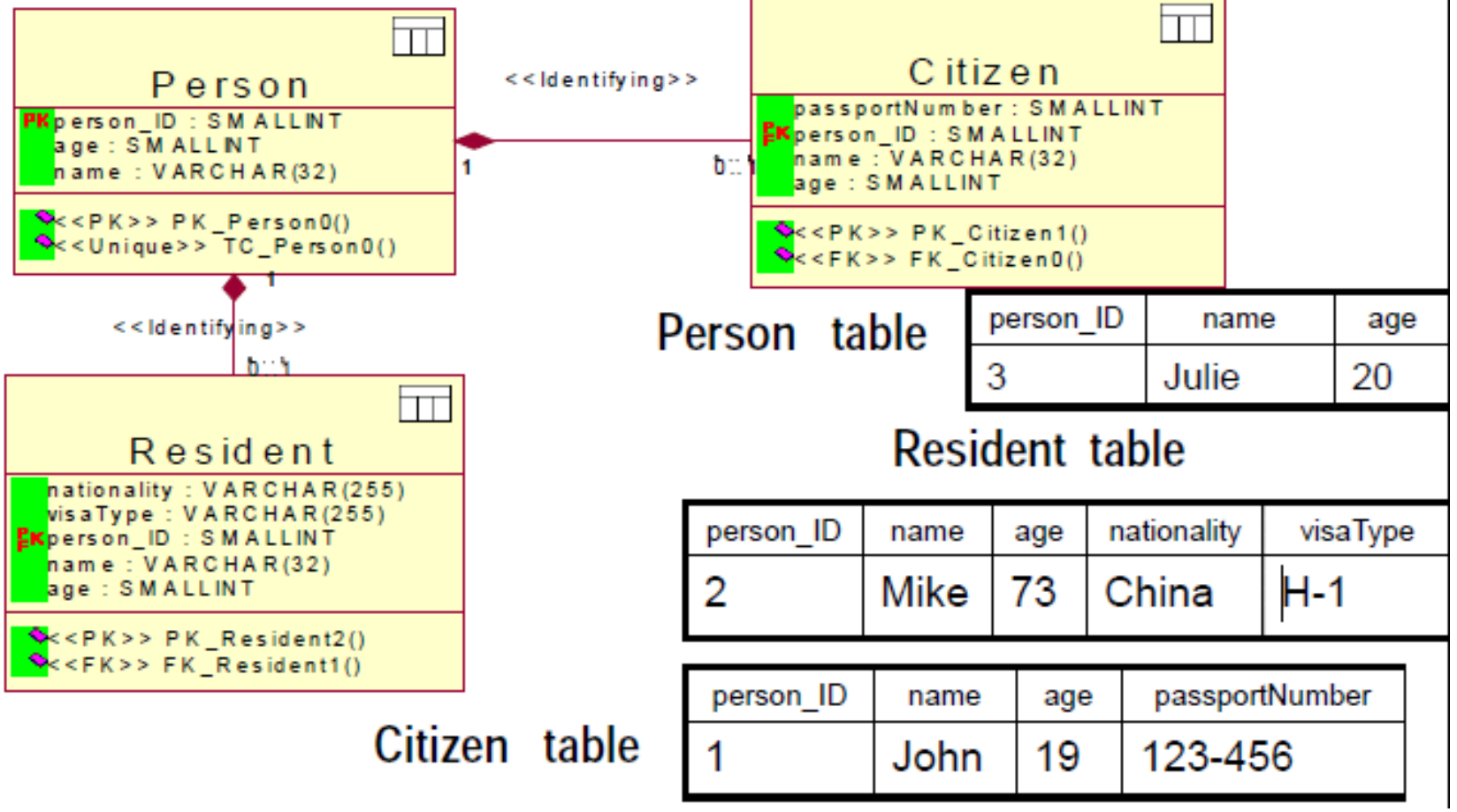
Citizen table

person_ID	passportNumber
1	123-456

Resident table

person_ID	nationality	visaType
2	China	H-1

Table per concrete class



Person table

person_ID	name	age
3	Julie	20

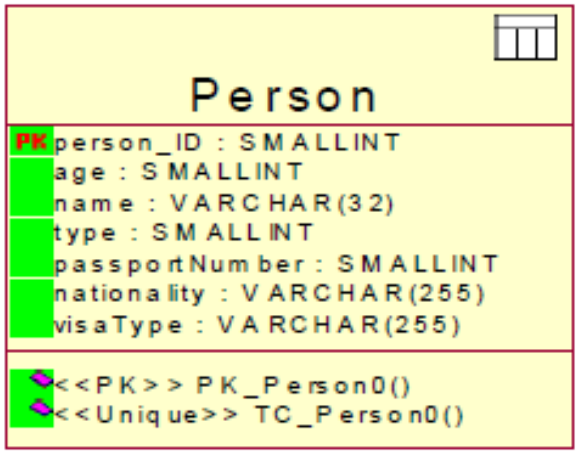
Resident table

person_ID	name	age	nationality	visaType
2	Mike	73	China	H-1

Citizen table

person_ID	name	age	passportNumber
1	John	19	123-456

Table per class hierarchy



Person table

person_ID	type	name	age	passportNumber	nationality	visaType
1	citizen	John	19	123-456		
2	resident	Mike	73		China	H-1
3	person	Julie	20			

Inheritance strategy

- Single table per class hierarchy strategy

planes
*ID
DISC
type
manufacturer
capacity
comfort

ID	DISC	type	manufacturer	capacity	comfort
1	2	320	Airbus	320	NULL
2	1	777	Boeing	NULL	Great
3	0	78	II	NULL	NULL

Inheritance strategy

- Single table per class hierarchy strategy

```
public class Plane {
 private Long id;
 private String type;
 private String manufacturer;

 public Plane() {}

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

 public String getType() { return type; }
 public void setType(String type) { this.type = type; }

 public String getManufacturer() { return manufacturer; }
 public void setManufacturer(String manufacturer) {
    this.manufacturer = manufacturer;
 }

}
```

Inheritance strategy

- Single table per class hierarchy strategy

```
public class Airbus extends Plane{
 private String capacity;

 public Airbus() {}

 public String getCapacity() { return capacity; }
 private void setCapacity(String capacity) { this.capacity = capacity; }

}


public class Boeing extends Plane{
 private String comfort;

 public Boeing() {}

 public String getComfort() { return comfort; }
 private void setComfort(String comfort) { this.comfort = comfort; }

}
```

Inheritance strategy

- Single table per class hierarchy strategy

```
<hibernate-mapping package="Sample.Entity"
                   discriminator-value="0“ >
    <class name="Plane" table="planes">
        <id name="id" column="ID">
            <generator class="native"/>
        </id>
        <discriminator column="DISC" type="string"/>
        <property name="type"/>
        <property name="manufacturer"/>

        <subclass name="Airbus" discriminator-value="1">
            <property name="capacity" />
        </subclass>
        <subclass name="Boeing" discriminator-value="2">
            <property name="comfort" />
        </subclass>
    </class>
</hibernate-mapping>
```

Inheritance strategy

- Single table per class hierarchy strategy

```
String manufacturer = (String) request.getParameter("manufacturer");

Session session =
         HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
List planes = session.createQuery(
         "from Plane as p where p.manufacturer = :manu").
          setParameter("manu", manufacturer).list();
session.getTransaction().commit();

for (int i = 0; i < planes.size(); i++) {
     Plane thePlane = (Plane) planes.get(i);
     out.println("id: " + thePlane.getId() + "<br>" +
                 "type: " + thePlane.getType() + "<br>" +
                 "manufacturer: " + thePlane.getManufacturer() +
                 "<br><br>");
}
```

Inheritance strategy

- Single table per class hierarchy strategy

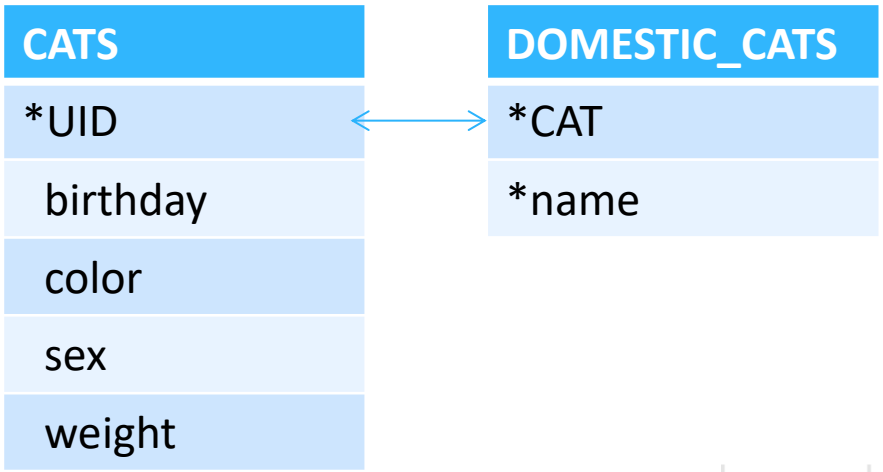
```
Airbus airbus = new Airbus();
airbus.setManufacturer("Airbus");
airbus.setType("320");
airbus.setCapacity("320");
session.save(airbus);

Boeing boeing = new Boeing();
boeing.setManufacturer("Boeing");
boeing.setType("777");
boeing.setComfort("Great");
session.save(boeing);

Plane plane = new Plane();
plane.setManufacturer("Il");
plane.setType("78");
session.save(plane);
```

Inheritance strategy

- Joined subclass strategy



UID	birthday	color	sex	weight
3	2017-04-16 00:00:00	blue	male	20

CAT	name
3	Doraemon

Inheritance strategy

- Joined subclass strategy

```
public class Cat {

        private Long uid;
        private Date birthday;
        private String color;
        private String sex;
        private int weight;

        public Cat() {}

        public Long getUid() { return uid; }
        private void setUid(Long uid) { this.uid = uid; }

        public Date getBirthday() { return birthday; }
        public void setBirthday(Date birthday) { this.birthday = birthday; }

        public String getColor() { return color; }
        public void setColor(String color) { this.color = color; }

        public String getSex() { return sex; }
        public void setSex(String sex) { this.sex = sex; }

        public int getWeight() { return weight; }
        public void setWeight(int weight) { this.weight = weight; }

}
```

Inheritance strategy

- Joined subclass strategy

```
public class DomesticCat extends Cat {
     private String name;

      public DomesticCat() {}

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

Inheritance strategy

- Joined subclass strategy

```
<hibernate-mapping package="Sample.Entity">
    <class name="Cat" table="CATS">
        <id name="uid" column="UID">
            <generator class="native"/>
        </id>
        <property name="birthday" type="date"/>
        <property name="color"/>
        <property name="sex"/>
        <property name="weight"/>
        <joined-subclass name="DomesticCat" table="DOMESTIC_CATS">
           <key column="CAT"/>
           <property name="name" type="string"/>
        </joined-subclass>
    </class>
</hibernate-mapping>
```

Inheritance strategy

- Joined subclass strategy

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();
List cats = session.createQuery("from Cat").list();
session.getTransaction().commit();

for (int i = 0; i < cats.size(); i++) {
     Cat theCat = (Cat)cats.get(i);
     out.println("id: " + theCat.getUid() + "<br>" +
                 "birthday: " + theCat.getBirthday() + "<br>" +
                 "color: " + theCat.getColor() + "<br>" +
                 "weight: " + theCat.getWeight() + "<br>");
     if (theCat instanceof DomesticCat) {
               DomesticCat aCat = (DomesticCat)theCat;
               out.println("name: " + aCat.getName() + "<br>");
     }
     out.println("<br>");
}
```

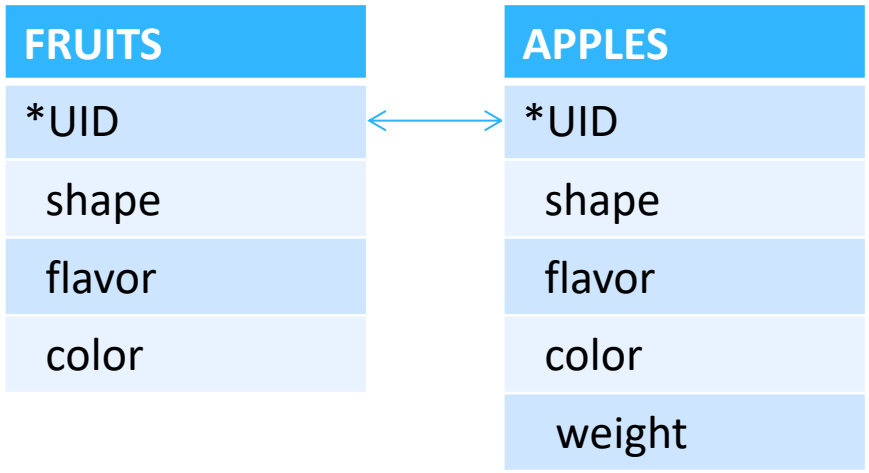
Inheritance strategy

- Joined subclass strategy

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
List cats = session.createQuery("from Cat").list();
DomesticCat domesticCat = new DomesticCat();
Date birthday = new Date();
domesticCat.setUid(new Long(3));
domesticCat.setBirthday(birthday);
domesticCat.setColor("blue");
domesticCat.setName("Doraemon");
domesticCat.setSex("male");
domesticCat.setWeight(20);
session.save(domesticCat);
session.getTransaction().commit();
```

Inheritance strategy

- Table per class strategy



UID	shape	flavor	color
NULL	NULL	NULL	NULL

UID	weight	shape	flavor	color
5	9	round	so-so	yellow

Inheritance strategy

- Table per class strategy

```
public class Fruit {
 private Long uid;
 private String shape;
 private String flavor;
 private String color;

 public Fruit() {}

 public Long getUid() { return uid; }
 public void setUid(Long uid) { this.uid = uid; }

 public String getShape() { return shape; }
 public void setShape(String shape) { this.shape = shape; }

 public String getFlavor() { return flavor; }
 public void setFlavor(String flavor) { this.flavor = flavor; }

 public String getColor() { return color; }
 public void setColor(String color) { this.color = color; }
}
```

Inheritance strategy

- Table per class strategy

```
public class Apple extends Fruit {

 private int weight;

 public Apple() {}

 public int getWeight() { return weight; }
 public void setWeight(int weight) { this.weight = weight; }

}
```

Inheritance strategy

- Table per class strategy

```
<hibernate-mapping package="Sample.Entity">
    <class name="Fruit" table="FRUITS">
        <id name="uid" column="UID"> </id>
        <property name="shape"/>
        <property name="flavor"/>
        <property name="color"/>
        <union-subclass name="Apple" table="APPLES">
           <property name="weight"/>
        </union-subclass>
    </class>
</hibernate-mapping>
```

Inheritance strategy

- Table per class strategy

```
@WebServlet("/FruitServlet")
public class FruitServlet extends HttpServlet {
  protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
    ……
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Apple apple = new Apple();
    apple.setUid(new Long(5));
    apple.setShape("round");
    apple.setFlavor("so-so");
    apple.setColor("yellow");
    apple.setWeight(9);
    session.save(apple);
```

Inheritance strategy

- Table per class strategy

```
@WebServlet("/FruitServlet")
public class FruitServlet extends HttpServlet {
  protected void processRequest(HttpServletRequest request,
                                HttpServletResponse response)
          throws ServletException, IOException {
    ……
    List fruits = session.createQuery("from Fruit").list();
    session.getTransaction().commit();

    for (int i = 0; i < fruits.size(); i++) {
       Fruit theFruit = (Fruit)fruits.get(i);
       out.println("id: " + theFruit.getUid() + "<br>" +
                  ……
       if (theFruit instanceof Apple) {
            Apple aFruit = (Apple)theFruit;
            out.println("weight: " + aFruit.getWeight() + "<br>");
       }
```

Hibernate object states

- Hibernate defines and supports the following object states:
  - *Transient*
    - an object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session.
  - *Persistent*
    - a persistent instance has a representation in the database and an identifier value.
  - *Detached*
    - a detached instance is an object that has been persistent, but its Session has been closed.

Making objects persistent


```
DomesticCat fritz = new DomesticCat();
fritz.setColor(Color.GINGER);
fritz.setSex('M');
fritz.setName("Fritz");
Long generatedId = (Long) sess.save(fritz);

DomesticCat pk = new DomesticCat();
pk.setColor(Color.TABBY);
pk.setSex('F');
pk.setName("PK");
pk.setKittens( new HashSet() );
pk.addKitten(fritz);
sess.save( pk, new Long(1234) );
```

Or

```
sess.persist( pk, new Long(1234) );
```

Loading an object


```
Cat fritz = (Cat) sess.load(Cat.class, generatedId);

// you need to wrap primitive identifiers
long id = 1234;
DomesticCat pk = (DomesticCat) sess.load( DomesticCat.class, new Long(id) );

Cat cat = new DomesticCat();
// load pk's state into cat
sess.load( cat, new Long(pkId) );
Set kittens = cat.getKittens();

Cat cat = (Cat) sess.get(Cat.class, id);
 if (cat==null) {
   cat = new Cat();
   sess.save(cat, id);
}
return cat;

sess.save(cat);
sess.flush(); //force the SQL INSERT
sess.refresh(cat); //re-read the state (after the trigger executes)
```

Executing queries


```
 List cats = session.createQuery(
    "from Cat as cat where cat.birthdate < ?")
    .setDate(0, date)
    .list();
List mothers = session.createQuery(
    "select mother from Cat as cat join cat.mother
                        as mother where cat.name = ?")
    .setString(0, name)
    .list();
List kittens = session.createQuery(
    "from Cat as cat where cat.mother = ?")
    .setEntity(0, pk)
    .list();
Cat mother = (Cat) session.createQuery(
    "select cat.mother from Cat as cat where cat = ?")
    .setEntity(0, izi)
    .uniqueResult();
Query mothersWithKittens = (Cat) session.createQuery(
    "select mother from Cat as mother left join fetch mother.kittens");
Set uniqueMothers = new HashSet(mothersWithKittens.list());
```

Iterating results


```
Iterator kittensAndMothers = sess.createQuery(
     "select kitten, mother from Cat kitten join kitten.mother mother")
    .list()
    .iterator();
while ( kittensAndMothers.hasNext() ) {
    Object[] tuple = (Object[]) kittensAndMothers.next();
    Cat kitten = (Cat) tuple[0];
    Cat mother = (Cat) tuple[1];
    ....
}

Iterator results = sess.createQuery(
     "select cat.color, min(cat.birthdate), count(cat) from Cat cat " + "group by cat.color")
     .list()
     .iterator();
while ( results.hasNext() ) {
    Object[] row = (Object[]) results.next();
    Color type = (Color) row[0];
    Date oldest = (Date) row[1];
    Integer count = (Integer) row[2];
    .....
}
```

Bind parameters


```
//named parameter (preferred)
Query q = sess.createQuery("from DomesticCat cat
                            where cat.name = :name");
q.setString("name", "Fritz");
Iterator cats = q.iterate();

//positional parameter
Query q = sess.createQuery("from DomesticCat cat
                            where cat.name = ?");
q.setString(0, "Izi");
Iterator cats = q.iterate();

//named parameter list
List names = new ArrayList();
names.add("Izi");
names.add("Fritz");
Query q = sess.createQuery("from DomesticCat cat
                            where cat.name in (:namesList)");
q.setParameterList("namesList", names);
List cats = q.list();
```

Pagination and Scrollable iteration


```
Query q = sess.createQuery("from DomesticCat cat");
q.setFirstResult(20);
q.setMaxResults(10);
List cats = q.list();

Query q = sess.createQuery("select cat.name, cat from DomesticCat cat " + "order
by cat.name");
ScrollableResults cats = q.scroll();
if ( cats.first() ) {
    // find the first name on each page of an alphabetical list of cats by name
    firstNamesOfPages = new ArrayList();
    do {
        String name = cats.getString(0);
        firstNamesOfPages.add(name);
    } while ( cats.scroll(PAGE_SIZE) );
    // Now get the first page of cats
    pageOfCats = new ArrayList();
    cats.beforeFirst();
    int i=0;
    while( ( PAGE_SIZE > i++ ) && cats.next() )
       pageOfCats.add( cats.get(1) );
}
cats.close()
```

Criteria queries and native SQL

REliable, INtelligent & Scalable Systems

```
Criteria crit = session.createCriteria(Cat.class);
crit.add( Restrictions.eq( "color", eg.Color.BLACK ) );
crit.setMaxResults(10);
List cats = crit.list();


List cats = session.createSQLQuery("SELECT {cat.*} FROM CAT
{cat} WHERE ROWNUM<10")
    .addEntity("cat", Cat.class)
    .list();
List cats = session.createSQLQuery(
    "SELECT {cat}.ID AS {cat.id}, {cat}.SEX AS {cat.sex}, " +
    "{cat}.MATE AS {cat.mate}, {cat}.SUBCLASS AS
{cat.class}, ... " +
    "FROM CAT {cat} WHERE ROWNUM<10")
    .addEntity("cat", Cat.class)
    .list()
```

Modifying objects


```
DomesticCat cat = (DomesticCat) sess.load( Cat.class, new Long(69) );
cat.setName("PK");
sess.flush(); // changes to cat are automatically detected and persisted

// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catId);
Cat potentialMate = new Cat();
firstSession.save(potentialMate);

// in a higher layer of the application
cat.setMate(potentialMate);

// later, in a new session
secondSession.update(cat); // update cat
secondSession.update(mate); // update mate
or
secondSession.merge(cat); // merge cat
secondSession.merge(mate); // merge mate
```

Automatic state detection


```
// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catID);

// in a higher tier of the application
Cat mate = new Cat();
cat.setMate(mate);

// later, in a new session
secondSession.saveOrUpdate(cat);
// update existing state (cat has a non-null id)
secondSession.saveOrUpdate(mate);
// save the new instance (mate has a null id)
```

Deleting persistent objects


```
sess.delete(cat);
```

Replicating object


```
//retrieve a cat from one database
Session session1 = factory1.openSession();
Transaction tx1 = session1.beginTransaction();
Cat cat = session1.get(Cat.class, catId);
tx1.commit();
session1.close();

//reconcile with a second database
Session session2 = factory2.openSession();
Transaction tx2 = session2.beginTransaction();
session2.replicate(cat, ReplicationMode.LATEST_VERSION);
tx2.commit();
session2.close();
```

- ReplicationMode:
  - IGNORE
  - OVERWRITE
  - EXCEPTION
  - LATEST_VERSION

Flushing the Session

- *flush*, occurs by default at the following points:
  - before some query executions
  - from `org.hibernate.Transaction.commit()`
  - from `Session.flush()`

- The SQL statements are issued in the following order:
  - all entity insertions in the same order the corresponding objects were saved using `Session.save()`
  - all entity updates
  - all collection deletions
  - all collection element deletions, updates and insertions
  - all collection insertions
  - all entity deletions in the same order the corresponding objects were deleted using `Session.delete()`

Flushing the Session


```
sess = sf.openSession();
Transaction tx = sess.beginTransaction();
sess.setFlushMode(FlushMode.COMMIT);
// allow queries to return stale state

Cat izi = (Cat) sess.load(Cat.class, id);
izi.setName(iznizi);

// might return stale data
sess.find("from Cat as cat left outer join cat.kittens kitten");

// change to izi is not flushed!
...
tx.commit(); // flush occurs
sess.close();
```

Transitive persistence

- For each basic operation of the Hibernate session there is a corresponding cascade style
- including `persist()`, `merge()`, `saveOrUpdate()`, `delete()`, `lock()`, `refresh()`, `evict()`, `replicate()`
  - `CascadeType.PERSIST`
  - `CascadeType.MERGE`
  - `CascadeType.REMOVE`
  - `CascadeType.REFRESH`
  - `CascadeType.DETACH`
  - `CascadeType.ALL`

Transitive persistence


```
@Entity
public class Customer {
    private Set<Order> orders;
    @OneToMany(cascade=CascadeType.ALL, orphanRemoval=true)
   public Set<Order> getOrders() { return orders; }
   public void setOrders(Set<Order> orders) {
       this.orders = orders;
   }
   ...
}

@Entity
public class Order { ... }

Customer customer = em.find(Customer.class, 1l);
Order order = em.find(Order.class, 1l);
customer.getOrders().remove(order);
//order will be deleted by cascade
```

Project

- To reconstruct your project by using Hibernate
  - Mapping your tables into classes.
  - Accessing DB by manipulating objects.
  - Primary Keys of users, books and records are Auto Incremented.

References

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

- *Web*开发技术
- *Web Application Development*

Thank You!