Hibernate Training

TechFerry Infotech Pvt. Ltd.
(http://www.techferry.com/)

Conversations

- Introduction to Hibernate
- Hibernate in Action
- Object Relational Mapping (ORM)
    - Association Mappings
    - Inheritance Mappings
- HQL (Hibernate Query Language)
    - Joining Associations in HQL
- Spring Hibernate Integration

Hello Hibernate

Inherent differences in Object and Relational Model:

- Java Objects have associations
- RDBMS tables have relations with foreign keys

Questions to consider:

- How do we implement inheritance in RDBMS tables?
- Are your Form beans (to be used on views) different from entity beans? Do you do data transfer from one type of bean to another?
- Do you manually associate objects because data is retrieved from RDBMS using join queries?
- How much time programmers spend on persistence and data retrieval tasks?

Can all this boilerplate persistence code be automated?

Why Hibernate?

- Open Source persistence technology
  - relieve developers from majority of common data persistence related programming tasks
- ORM framework
  - follow natural Object-oriented idioms including inheritance, polymorphism, association, composition, and the Java collections framework.
- Comprehensive Query Facilities:
  - support for Hibernate Query Language (HQL), Java Persistence Query Language (JPAQL), Criteria queries, and "native SQL" queries; all of which can be scrolled and paginated to suit your exact performance needs.

Why Hibernate?

- **High Performance:**
  - lazy initialization, many fetching strategies
  - optimistic locking with automatic versioning/ time stamping
  - Hibernate requires no special database tables or fields and generates much of the SQL at system initialization time instead of runtime.
- **Reliability and Scalability:**
  - proven by the acceptance and use by tens of thousands of Java developers
  - designed to work in an application server cluster and deliver a highly scalable architecture

Hibernate in action

Code Demo....

- Annotations: @Entity, @Table, @Id, @Column, @GeneratedValue,

Methods:

- persist() vs save()
- update vs saveOrUpdate()
- load() vs get()
- createQuery().list()
- delete()

Hibernate in action

- Concurrency Control: @Version
- Sorting: @OrderBy, @Sort
- Pagination
- Lazy vs Eager Fetching: fetch = FetchType.EAGER
- @Transient, @Lob

Reference:

- http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/
- http://www.techferry.com/articles/hibernate-jpa-annotations.html

Association Mappings

Types of Associations:

- @OneToOne
- @ManyToOne
- @OneToMany
- @ManyToMany

Relationship Types:

- Uni-directional
- Bi-directional

RDBMS Implementations:

- Shared Primary Key
- Foreign Key
- Association Table

@OneToOne

- @PrimaryKeyJoinColumn - associated entries share the same primary key.
- @JoinColumn & @OneToOne mappedBy attribute - foreign key is held by one of the entities.
- @JoinTable and mappedBy - association table

- Persist two entities with shared key: @MapsId

@ManyToOne

- @JoinColumn - foreign key is held by one of the entities.
- @JoinTable - association table

@OneToMany

- mappedBy attribute for bi-directional associations with ManyToOne being the owner.
- OneToMany being the owner or unidirectional with foreign key - try to avoid such associations but can be achieved with @JoinColumn
- @JoinTable for Unidirectional with association table

@ManyToMany

- @JoinTable - association table.
- mappedBy attribute for bi-directional association.

Mapping Inheritance

- table per class hierarchy
  - single table per Class Hierarchy Strategy: the <subclass> element in Hibernate
- table per class/subclass
  - joined subclass Strategy: the <joined-subclass> element in Hibernate
- table per concrete class
  - table per Class Strategy: the <union-class> element in Hibernate

Table per class hierarchy- Single Table

```
@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 { ... }
```

Table per class/subclass -joined subclass Strategy

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

@Entity
@PrimaryKeyJoinColumn
public class Ferry extends Boat { ... }
```

Table per concrete class

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

Note: This strategy does not support the IDENTITY generator strategy: the id has to be shared across several tables. Consequently, when using this strategy, you should not use AUTO nor IDENTITY.

Inheritance Mapping Reference:
http://docs.jboss.org/hibernate/core/3.3/reference/en/html/inheritance.html

HQL

Creating Query:

Query hqlQuery = session.createQuery("from Category c where c.name like 'Laptop%'");

Method Chaining:

List results = session.createQuery("from User u order by u.name asc").setFirstResult(0).setMaxResults(10).list();

Named Parameters:

String queryString = "from Item item where item.description like :searchString";
List result = session.createQuery(queryString).setString("searchString", searchString).list();

HQL Contd...

Positional Parameters:

```
String queryString = "from Item item "
                     + "where item.description like ? "
                     + "and item.date > ?";
List result = session.createQuery(queryString).setString(0, searchString)
.setDate(1, minDate).list();
```

Binding Entity Parameters:

```
session.createQuery("from Item item where item.seller = :seller")
.setEntity("seller", seller).list();
```

HQL Operators and Keywords

=, <>, <, >, >=, <=, between, not between, in, and not in.

```
from Bid bid where bid.amount between 1 and 10
from Bid bid where bid.amount > 100
from User u where u.email in ( "foo@hibernate.org", "bar@hibernate.org" )
```

Keywords: null,not null, like, not like, upper(), lower(), and, or

```
from User u where u.email is null
from User u where u.email is not null
from User u where u.firstname like "G%"
from User u where u.firstname not like "%Foo B%"
from User u where lower(u.email) = 'foo@hibernate.org'
from User user where ( user.firstname like "G%" and user.lastname like "K%" )
        or user.email in ( "foo@hibernate.org", "bar@hibernate.org" )
```

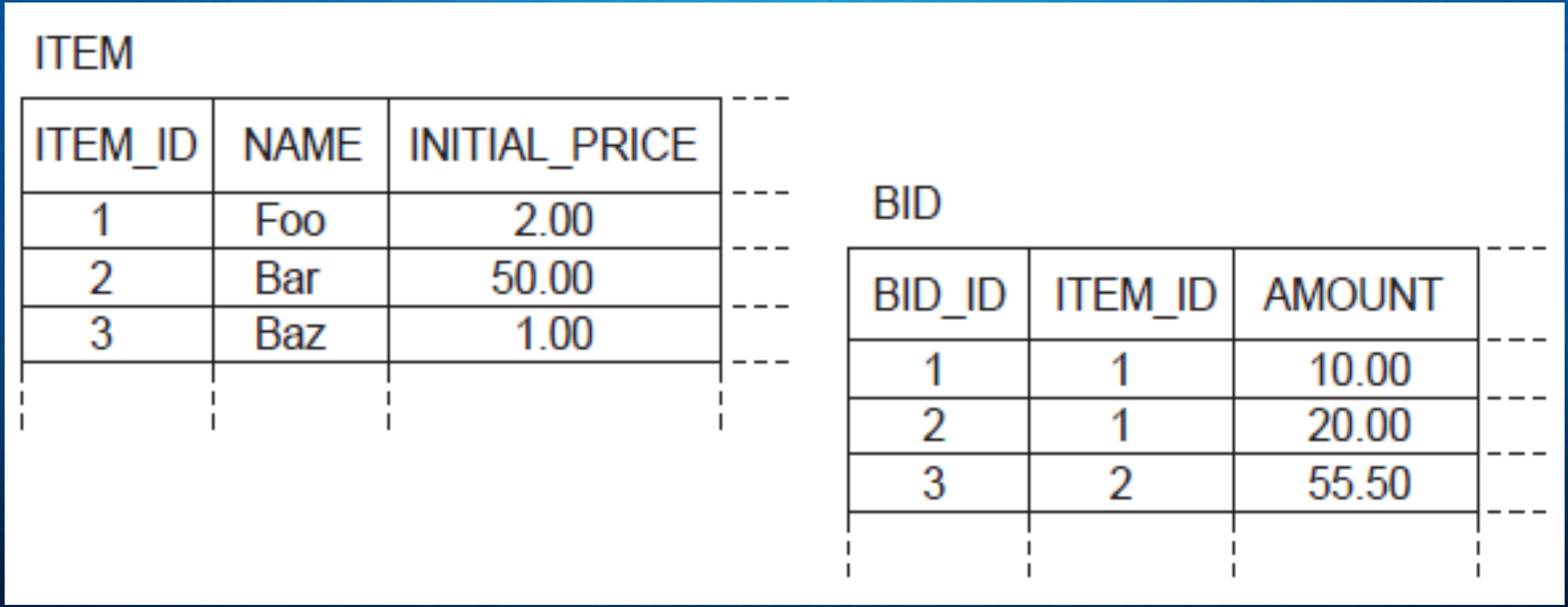
Other keywords

Keywords: group by, having, order by, count(), avg(), distinct

```
select item.id, count(bid), avg(bid.amount)
from Item item
join item.bids bid
where item.successfulBid is null
group by item.id
having count(bid) > 10

select distinct item.description from Item item
```

HQL - Joining Associations



ITEM

ITEM_ID	NAME	INITIAL_PRICE
1	Foo	2.00
2	Bar	50.00
3	Baz	1.00

BID

BID_ID	ITEM_ID	AMOUNT
1	1	10.00
2	1	20.00
3	2	55.50

In Hibernate queries, you don't usually specify a join condition explicitly. Rather, you specify the name of a mapped Java class association.

Example: item.bids, bid.item

HQL Joins

HQL provides four ways of expressing (inner and outer) joins:

- An ordinary join in the from clause
- A fetch join in the from clause
- An implicit association join
- A theta-style join in the where clause

Ordinary Join in the from clause

```
from Item item
join item.bids bid
where item.description like '%gc%'
and bid.amount > 100
```

```
Query q = session.createQuery("from Item item join item.bids bid");
Iterator pairs = q.list().iterator();

while ( pairs.hasNext() ) {
Object[] pair = (Object[]) pairs.next();
Item item = (Item) pair[0];
Bid bid = (Bid) pair[1];
}
```

Ordinary Joins Contd..

```
select item
from Item item
join item.bids bid
where item.description like '%gc%'
and bid.amount > 100
```

```
Query q = session.createQuery("select i from Item i join i.bids b");
Iterator items = q.list().iterator();
while ( items.hasNext() ) {
Item item = (Item) items.next();
}
```

Fetch Joins

```
from Item item
left join fetch item.bids
where item.description like '%gc%'

from Bid bid
left join fetch bid.item
left join fetch bid.bidder
where bid.amount > 100
```

- Hibernate currently limits you to fetching just one collection eagerly. You may fetch as many one-to-one or many-to-one associations as you like.
- If you fetch a collection, Hibernate doesn’t return a distinct result list.

Implicit Joins

```
from Bid bid where bid.item.description like '%gc%'
```

Implicit joins are always directed along many-to-one or one-to-one associations, never through a collection-valued association (you can’t write item.bids.amount).

```
from Bid bid
where bid.item.category.name like 'Laptop%'
and bid.item.successfulBid.amount > 100
```

Implicit Joins Contd..

```
from Bid bid
join bid.item item
where item.category.name like 'Laptop%'
and item.successfulBid.amount > 100
```

```
from Bid as bid
join bid.item as item
join item.category as cat
join item.successfulBid as winningBid
where cat.name like 'Laptop%'
and winningBid.amount > 100
```

Theta Style Joins

When the association is not defined.

from User user, LogRecord log where user.username = log.username

```
Iterator i = session.createQuery(
"from User user, LogRecord log " +
"where user.username = log.username"
)
.list().iterator();
while ( i.hasNext() ) {
Object[] pair = (Object[]) i.next();
User user = (User) pair[0];
LogRecord log = (LogRecord) pair[1];
}
```

Spring Hibernate Integration

- Injecting Hibernate SessionFactory in @Repository classes.
- Spring's HibernateTemplate
- JPA EntityManager

Thank You and Questions?