Enterprise JavaBeans 3.0
Entity Beans

Luca Cavallaro
cavallaro@elet.polimi.it
credit to: Matteo Miraz
miraz@elet.polimi.it

19/12/2007

EJB Query Language

- Similar to SQL
  - Modified to handle objects
- Contatiner maps an EJBQL query to one or more SQL queries
- Better portability: independent from vendor specific features
- Less expressive power: some advanced constructs are missing
  - But we can do native queries

Query API

- EntityManager
  - `createQuery(String ejbQL): Query`
  - `createNamedQuery(String name): Query`
  - `createNativeQuery(String sql): Query`
  - `createQuery(String sql, Class resultClass): Query`
- Query
  - `List getResultList()`
  - `Object getSingleResult()`
  - `setMaxResults(int n); //paginazione`
  - `setFirstResult(int n);//paginazione`
  - `Query setParameter(String name, Object value)`

Example

```
try {
  Query q = em.createQuery("FROM Customer c
  WHERE c.firstName='Mario' AND
  c.lastName='Rossi' ");
  Customer c = (Customer)
  q.getSingleResult();
} catch (EntityNotFoundException exc) {
  /* non sono state trovate entità */
} catch (NotUniqueResultException exc) {
 /* è stata trovata più di un'entità */
}
```

Example (2)

```
Query q = em.createQuery(“FROM Customer c
  WHERE c.firstName=‘Mario’ AND
  c.lastName=‘Rossi’ ”);
Collection<Customer> coll =
  (Collection<Customer>) q.getResultList();

for(Customer r : coll) {
  System.out.println(r.getFirstName());
  …
}
```

Parameters

- EJB QL allows parametric queries
  - Specify parameters by name

```
Query q = em.createQuery("FROM Customer c WHERE
  c.firstName=:firstName AND  c.lastName=:lastName ");
q.setParameter("firstName", "mario");
q.setParameter("lastName", "rossi");
Collection coll = q.getResultList();
```

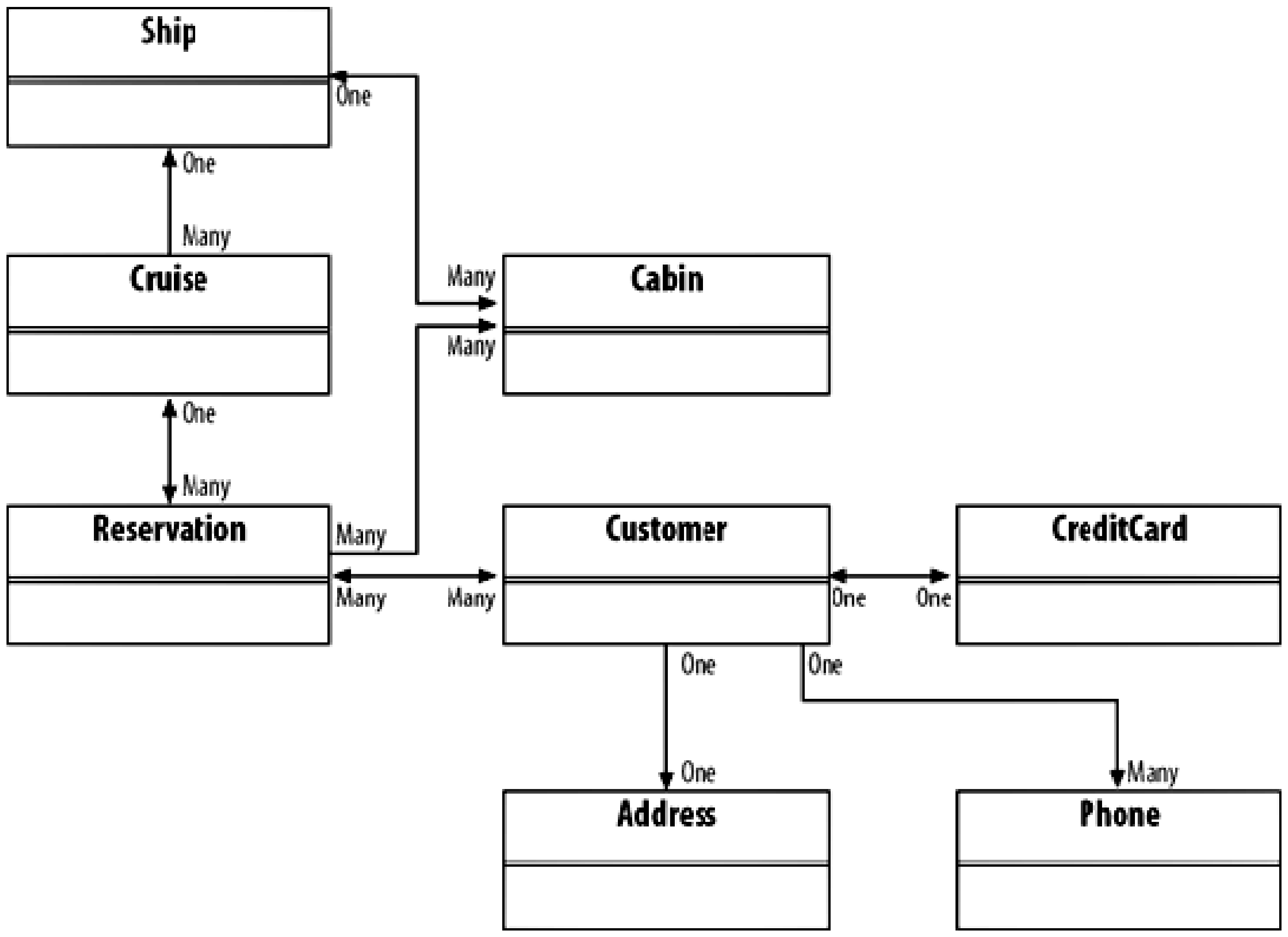
  - Specify parameters by position

```
Query q = em.createQuery("FROM Customer c WHERE
  c.firstName=?1 AND  c.lastName=?2 ");
q.setParameter(1, "mario");
q.setParameter(2, "rossi");
Collection coll = q.getResultList();
```

Date come parametri e FlushMode

- When we use java.util.Date (or Calendar)parameters, we have to specify the format
  - We can use methods as
    ```
    setParameter(String name, Date date, TemporalType type)
    ```
  - TemporalType is an enum
    ```
    {DATE, TIME, TIMESTAMP }
    ```
- As a default behavior data are written on the DB as a query is performed
  - Sometimes we want a different behavior
  ```
  query.setFlushMode(FlushModeType.COMMIT);
  ```

Example



Ship
One
One
Many
Cruise
Many
Cabin
Many
One
Many
Reservation
Many
Customer
CreditCard
Many
Many
One
One
One
One
One
Many
Address
Phone

EJB QL

- SQL like language
  - Extended to consider objects!

- Entities are identified by name
  - `@Entity public class Customer { … }`
    `“SELECT c FROM Customer AS c”`
  - `@Entity(name=“foo”) public class Customer { … }`
    `“SELECT c FROM foo AS c”`
  - Watch out for name clashing!
  - `@Table does not affect names`

EJB QL

- We can select
  - Objects

```
String q = “SELECT OBJECT (c) FROM Customer AS c”;
Query que = em.createQuery(q);
Collection<Customer> c = que.getResultList();
```

  - Fields

```
q = “SELECT c.firstName, c.lastName FROM Customer AS c”;
Query que = em.createQuery(q);
Collection<Object[]> c = que.getResultList();
```

  - Embedded oobjects or objects in *OneToOne* o *ManyToOne relations*

```
q = “SELECT c.customer.address FROM CreditCard AS c”;
Query que = em.createQuery(q);
Collection<Address> c = que.getResultList();
```

EJB QL: navigation

- Why can't we have direct access to *OneToMany* or *ManyToMany objects*???
  - What happens in Java:

```
@Entity public class Ship { …
  @OneToMany Collection<Cabin> cabin; …
}
@Entity public class Cabin { … String name; … }

"SELECT c.cabin.name FROM Cabin AS c"
```

in Java :

ship.getCabin().getName()

getCabin returns a Collection!!!

EJB QL and *toMany relations

- Get all customers that made a reservation

  `“SELECT c.reservation FROM Customer AS c”` *ILLEGALE!*

- EJB QL uses different JOIN operators
  - IN:

    `“SELECT r FROM Customer AS c, IN (c.reservation) r”`
  - INNER JOIN:

    `“SELECT r FROM Customer AS c INNER JOIN c.reservation r”`
  - Can be nested

  Select all addresses of people who traveled on ships

  ```
  “SELECT c.address FROM Cruise AS cr JOIN cr.reserv r JOIN
    r.customer c”
  ```

WHERE

- Similar to SQL, filters objects

```
SELECT c FROM Customer c WHERE
   c.creditCard.creditCompany.name = 'VISA'
```

- Precedences:
  - Navigation (.)
  - Arithmetics (+, - unari; *, /; +, -)
  - Comparison (=, >, >=, <, <=, <>, LIKE, BETWEEN, IN, IS NULL, IS EMPTY, MEMBER OF)
  - Logical operators (NOT; AND; OR)

Entity comparison

- Assume that between customers and reservations holds a unidirectional(from reservation to customer) many to many relation
  - How can we select a customer's reservations?

    ```
    SELECT r FROM Reservation AS r JOIN r.customers AS
      cust WHERE cust = :customerParameter
    ```

  - The comparison is performed on primary keys

WHERE: BETWEEN and IN

- BETWEEN
  - Used only for byte, short, int, long, double and float

    -
      ```
      SELECT s FROM Ship s
      WHERE s.tonnage BETWEEN 80000.00 AND 130000.00
      ```

    -
      ```
      SELECT s FROM Ship s
      WHERE s.tonnage NOT BETWEEN 80000.00 AND 130000.00
      ```
- IN
  - Different from IN in FROM clause(which means JOIN)

    -
      ```
      SELECT c FROM Customer c
      WHERE c.address.state IN ('Italia', 'Svizzera')
      ```

WHERE: IS NULL e IS EMPTY

- IS NULL
  - `SELECT c FROM Customer c`
    `WHERE c.address IS NULL`
  - `SELECT c FROM Customer c`
    `WHERE :city IS NOT NULL AND c.address.city = :city`

- IS EMPTY
  - Verifies if a *toMany relation is empty
  - `SELECT crs FROM Cruise crs`
    `WHERE crs.reservations IS EMPTY`
  - `SELECT crs FROM Cruise crs JOIN crs.reservation r`
    `WHERE crs.reservations IS NOT EMPTY (NON SI FA!!)`

WHERE: MEMBER OF and LIKE

- MEMBER OF
  - `SELECT r FROM Reservation r, Customer c WHERE c = :cust AND c MEMBER OF r.customers`
- LIKE
  - Similar to SQL
  - `SELECT c FROM Customer c WHERE c.lastName LIKE ‘Anton%’`

ORDER BY

- Similar to SQL
  - `SELECT c FROM Customer c`
    `ORDER BY c.lastName ASC, c.firstName DESC`
  - `SELECT c FROM Customer c`
    `ORDER BY c.address.zip ASC`

Simple and aggregate functions

- Simple functions
  - `LOWER, UPPER, TRIM, CONCAT, LENGTH, LOCATE, SUBSTRING`
  - `ABS, SQRT, MOD`
  - `CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP`
- Aggregate functions:
  - `COUNT, MAX, MIN, AVG, SUM`

GROUP BY e HAVING

- similar to SQL

  -
    ```
    SELECT cr.name, COUNT(res)
    FROM Cruise cr JOIN cr.reservation res
    GROUP BY cr.name
    HAVING COUNT(res) > 10
    ```

Subqueries

- Come in SQL…
  - `SELECT count(res) FROM Reservation res WHERE res.amountPaid > (SELECT avg(r.amountPaid) FROM Reservation r)`
- ALL
  - `SELECT c FROM Customer c JOIN c.reservations r WHERE r.amountPaid >= ALL ( SELECT  r.amountPaid FROM Reservation r )`
  - `SELECT cr FROM Cruise cr WHERE 0 < ALL ( SELECT res.amountPaid FROM cr.reservations res)`
- ANY / SOME
  - `SELECT cr FROM Cruise cr WHERE 0 = ANY ( SELECT res.amountPaid FROM cr.servation res)`
- EXIST
  - `SELECT cr FROM Cruise cr WHERE EXISTS ( SELECT res FROM cr.reservations WHERE res.amountPaid = 0)`

updates and deletions

- UPDATE
  - `UPDATE Reservation res SET res.amountPaid = (res.amountPaid + 10) WHERE res.amountPaid < 100;`
- DELETE
  - `DELETE FROM Reservation res WHERE res.amountPaid < 0`

- These query can generate inconsistencies:
  - Execute EntityManager.flush().

Native queries

- Scalar values
  - Query `createNativeQuery(String sql)`

    Results are managed as Scalar values in EJB QL (Collection, array of Object)

- Entity Native Queries
  - Query `createNativeQuery(String sql, Class entity)`
  - We have to select all coloumns representing entity fields

```
em.createNativeQuery(“SELECT p.phone_PK, p.phone_number,
  p.type, FROM PHONE AS p”, Phone.class);
```

Named Queries

- Queries can be assigned a name

- EJB QL
  - `@NamedQuery(name="foo", value="SELECT c FROM Customer c WHERE c.firstName = :name")`
  - `Query q = em.createNamedQuery("foo")`
- SQL
  - `@NamedNativeQuery(name="foo1", value="SELECT * FROM Customer")`
  - `Query q = em.createNamedNativeQuery("foo1")`

Riferences

- Burke & Monson-Haefel. *Enterprise JavaBeans 3.0.* O Reilly, fifth edition 2006.
- Ball et. al. *The Java EE 5 tutorial*. Sun Microsystems 2006. http://java.sun.com/javaee/5/docs/tutorial/doc/