Enterprise systems

Behzad Bordbar
Lecture 9
Recap

• Database Interaction Patterns
  • Configuration
  • SessionFactory
  • Session
  • Begin Transaction
  • Interact with DB
  • Commit

• Objects states: Transient, Persistent, Detached
Recap

• Identity
  • Identity in Java vs in Database, why important?
  • Studied various methods of establishing equality
  • Business key (e.g. email) 1) Override equals() and 2) Override hashCode()

• Summarized the points about Hibernate object that u should pay attention: default constructor, mutator/accessor, implement serializable, use surrogate key, its mutator private, leave update to hibernate,…
The following method when applied to an object \( O \), they will apply to any objects referenced by \( O \):

- `session.get` creates a new persistent obj by id from the db. It returns null if no such object in db.
- `session.load` is similar except, if there was no such object in the db it throws exception.
- `session.delete` makes the database row corresponding to a persistent (or even a detached!) object to be deleted—the object becomes transient.
- `session.save` on a transient item will assign it an id and make it persistent: Similar to `INSERT`
Sessions

- `session.update` reattaching a detached object. Similar to SQL UPDATE
- `session.lock` reattaches the object (to the session) without checking or updating the database (assumes db and application are fully sync) Generally, do not use this method
- `session.saveOrUpdate` clear from name
- `session.merge` checks for a persistent object with the same identifier in the session. If it finds one, it copies the data from the detached object onto the persistent one, Otherwise it creates a new persistent object from the data
- Check the API if you are not sure!
Querying

If we know the object id:

\[ x = z.getY().getX() \]

If we don’t know we need to execute a query.

1) use `session.createQuery` to return Query object.

2) HQL to write a query (HQL similar to SQL)

3) Execute Query object by invoking `list()` or `iterate()` to return a list of results or an iterator over the results.
Example of HQL

session.createQuery("from customer cust where cust.city=:cityName")
  .setString("cityName", "Birmingham")
  .list();

Returns a list of Customers.

- Method chaining
- `select` is optional. Can write “select fr...
- `customer cust` instead of “?” in PreparedStatement in SQL
- `from/where` used for projection
What does the following do?

Query query = createQuery("select cust, sa " + "from Customer cust, SalesAgent sa " + "where cust.city = sa.city");
So, you can write

```java
// get an iterator
ArrayList results = query.iterate();
while ( results.hasNext() ){
    Object[] row = (Object[]) results.next();
    Customer cust = (Customer) row[0];
    SalesAgent sa = (SalesAgent) row[1];

    // print the result
    System.out.format("Customer: %20s, Sales Agent: %20s\n", cust.getName(),
    sa.getName());
}
```
A little bit of HQL

You can use binary operations with exact semantics as SQL

=, <> , <, >, >=, <=, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL,

Binary operators for HQL collections

IS [NOT] EMPTY, [NOT] MEMBER [OF]

Following returns index of corresponding element from the collection

IS [NOT] EMPTY, [NOT] MEMBER [OF]

For further details see the manual
Pagination

Used when a query results in a large number of rows. Google gives only one page at a time.

If a query object results in ordering of the rows, we can use:

- `setFirstResult()` and
- `setMaxResults()`
int pageSize = 10;
int pageNo = 2;  // page 3
Query query =
createQuery("select cust, sa " +  
    "from Customer cust,
    SalesAgent sa " + 
    "where cust.city = sa.city" + 
    "order by cust.name asc, sa.name asc");  // order
query.setFirstResult(pageNo * pageSize);
query.setMaxResults(pageSize);
List customerSalesagentList = query.list();
Cascading Persistence

when an object is made persistent, the objects it refers to are also made persistent.

cascade, lets us control how much automatically persisted, deleted etc.

...<many-to-one name="nextMessage" cascade="all" column="next_message_id"/>

...
Cascading Persistence

Value for cascade are:

\texttt{none}: no automatic action on the referenced object takes place (default behaviour)

\texttt{persist, merge, lock, evict, :}

Cascade any persist(), merge(), lock(), evict() operation across this relationship.
Similarly

replicate, replace, save-update:
Cascade any replicate(), replace() … operation across this relationship.

delete: automatically removes the reference to the object(s) when delete() is called on the referencing object—note it is not deleted.
Cascading Persistence

delete-orphan: automatically delete any object for whom the reference has been removed from the referencing object. This option is only available for one-to-many and one-to-one relationships.

all: cascade all operations except delete-orphan.

all-delete-orphan: cascade all operations, and take the action of delete-orphan as well.

For further info see Hibernate manual
Delegate the synchronisation with DB to a Transaction object to commit or abort (rollback)

What if more than one transactions interact with the data?
Different levels of locking

In Hibernate four levels of locking considered

• Serializable
• Read Uncommitted (dirty read)
• Repeatable Read
• Read Committed (default with postgresql JDBC)
Serializable

Very resources intensive-no concurrency:
one transition must had completely finishes
(committed or rolled back) before the other
starts

For example a SELECT query sees only the
data before transactions begins. It does not
see the uncommitted data or changes
committed by other concurrent transactions

Now, if other concurrent transaction commits,
the transactions will rollback! Start again
Read Committed

Default used in PostgreSQL JDBC connection

One transaction can not see any value which has been written by another transaction if that other transaction has not yet committed.

Three concurrency problems:
• Lost Updates
• Unrepeatable Reads
• Phantom Read
Lost Updates

Transactions: tx1
  read row A
  Write to row A
  Commit

   tx2
  read row A
  write to row A
  Commit

So value written by tx1 is lost, hence related transitions has not happened
Lost Updates

In effect the old data is lost. Versioning will solve this problem (two different versions of the same row - with an identifier)
Unrepeatable Reads

Reading is not repeatable!

Transactions: tx1

read row A

Write to row A

Commit

read row A

The second read produces different value

This can cause problem.

Caching and versioning can take care of this.
Phantom Read

Transactions: tx1          tx2

    SELECT sth

    DELETE a row

    Commit

Same SELECT

tx1 gets a different result!

This is a problem as there is no versioning info left to work with
Phantom Read

Common problem with pagination
Run a SELECT and retrieve the first batch
Another transaction deletes a row from the second batch
We end up with the second set of result wrong—you never see the deleted result.
Be aware of this—although you can’t do much!
Dirty read

Suppose two transactions
1\textsuperscript{st}: book flight and hotel to Birmingham- say.opodo
1.1 Book flight to BHX
1.2 Book hotel
2\textsuperscript{nd}: book car at BHX (local car hire)- not.opodo

Scenario:
Flight is booked (1.1 commit)- so car is hired (2 commit)
No hotel is available- so 1.1 compensated (not Rollback)
Hence a car hired for a customer who will not arrive. Not much can be done- manual checking!?!