POLE STAR
www.polestarglobal.com

Purplefinder Enterprise Platform
Squeryl ORM and DSL

Peter Potts

16th February 2011

Resources

- Book: None
- Website: http://squeryl.org
- An example application in PEP R5: http://repository.enterprise.purplefinder.com
- Available on public Maven repositories.

ORM

- Object-relational mapping.
- An object is encapsulation of entity with attributes and behaviors.
- A programming technique to convert between related tables of data to objects.
- Example: JPA 2, Hibernate, Squeryl.

Java Persistence API (JPA)

- Pros:
  - Widely used.
  - Comprehensive.
- Cons:
  - Unnecessarily complex lifecycle management.
  - Unnecessarily complex transaction management.
  - JPQL is not type safe.
  - Criteria query API is unreadable.
  - Uses extreme Java features.

Squeryl

- Pros:
  - Simple yet sufficient.
  - Type-safe yet readable (DSL).
  - Uses standard Scala features.
- Cons:
  - New.
  - Foreign key aware.

DSL

- Domain-specific language is a language specific to the domain, namely querying a relational database.
- JPQL is readable but type-unsafe:

```
"select count(p) from Person p"
```

- JPA 2 criteria is type-safe but unreadable:

```
CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
CriteriaQuery<Long> criteriaQuery = criteriaBuilder.createQuery(Long.class);
Root<Person> root = criteriaQuery.from(Person.class);
Expression<Long> expression = criteriaBuilder.count(root);
criteriaQuery.select(expression);
TypedQuery<Long> typedQuery = entityManager.createQuery(criteriaQuery);
Long count = typedQuery.getSingleResult();
```

- Squeryl is type-safe and readable:

```
from(artists)(artist => compute(count)).single.measures
```

Persistent entity

- A generic persistent entity.
- Surrogate key.
- Optimistic concurrency control.

```
import org.squeryl.{Optimistic, KeyedEntity}

trait PersistentEntity extends KeyedEntity[Int] with Optimistic {
  val id: Int = 0
}
```

Defining a Schema

```
import org.squeryl.PrimitiveTypeMode._
import org.squeryl.Schema

case class Artist(var firstName: String, var lastName: String) extends PersistentEntity
case class Song(val title: String, val artistId: Int = 0) extends PersistentEntity

object MusicDatabase extends Schema {
  val artists = table[Artist]
  val songs = table[Song]

 on(artists)(artist => declare(
    artist.lastName defaultsTo ("!"),
    columns(artist.id, artist.firstName, artist.lastName) are (indexed)
  ))

  on(songs)(song => declare(
    song.title is (unique, indexed, dbType(varchar(64))),
    song.albumId is (indexed)
  ))
}
```

Sessions

```
import org.squeryl.Session
import org.squeryl.adapters.H2Adapter
import java.sql.DriverManager

class H2Session(memory: Boolean = false) extends Session {
  Class.forName("org.h2.Driver")

  def connection = if (memory)
    DriverManager.getConnection("jdbc:h2:mem:", "", "")
  else
    DriverManager.getConnection("jdbc:h2:target/database", "sa", "")

  def databaseAdapter = new H2Adapter
}
```

Transactions

- New transaction whatever:

```
transaction {…}
```

- New transaction if needed:

```
inTransaction {…}
```

Insert, Update, Delete

- Insert

```
val artist = artists.insert(new Artist("Herby", "Hamcock"))
```

- Update

```
artist.lastName = "Hancock"
artists.update(artist)
```

- Delete

```
artists.delete(artist.id)
```

Select

- Select statement is lazy and immutable.

```
class Artist(var firstName: String, var lastName: String) extends PersistentEntity {
  def songs = from(MusicDatabase.songs)(song => where(song.artistId === id)
    select(song))
}
```

Relations

```
class Artist(var firstName: String, var lastName: String) extends PersistentEntity {
  lazy val songs: OneToMany[Song] = MusicDatabase.artistToSongs.left(this)
}

class Song(val title: String, val artistId: Int = 0) extends PersistentEntity {
 lazy val artist: ManyToOne[Artist] = MusicDatabase.artistToSongs.right(this)
}

object MusicDatabase extends Schema {
  val artists = table[Artist]
  val songs = table[Song]

  val artistToSongs = oneToManyRelation(artists, songs).via(
    (artist, song) => artist.id === song.artistId)

  artistToSongs.foreignKeyDeclaration.constrainReference(onDelete cascade)
}
```

Pagination

```
from(artists)(artist =>
  where(artist.lastName like "%a%")
  select (artist)
  orderBy (artist.lastName asc)).page(1, 2)
```