11ª Emisión

DIPLOMADO

# Desarrollo de Sistemas con Tecnología Java

## Módulo 11

## Persistencia con Jakarta

*Dr. Omar Mendoza González*

*omarmendoza564@aragon.unam.mx*

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

DGTIC

50 AÑOS Educación Continua 1971 - 2021

## Objetivo

- El participante integrará a sus aplicaciones de tipo empresarial los mecanismos apropiados de persistencia para almacenar la información en una base de datos.

# Operaciones en cascada

- Las operaciones en cascada de una relación son aquellas efectuadas en una entidad que deben ser aplicadas a las entidades relacionadas.

- JPA contempla cinco tipos distintos, referidos a la transición o cambio de estado de la entidad que contiene la relación que se identifican mediante el tipo enumerado ***CascadeType***.
  - PERSIST
  - REFRESH
  - REMOVE
  - MERGE
  - DETACH

# Operaciones en cascada

- REFRESH
  - Propaga el método refresh que «refresca» las entidades del contexto de persistencia con el contenido actual de la base de datos.
  - Su uso, poco común, tiene sentido cuando puede haber cambios en la BDque no se tengan en las entidades *managed*.
- DETACH
  - Desliga en cascada todas las entidades relacionadas.
- REMOVE.
  - Borra las entidades relacionadas.

# Operaciones en cascada

- El atributo de cascada, en todas las anotaciones de relaciones lógicas *@OneToOne, @OneToMany, @ManyToOne y @ManyToMany*, define la lista de operaciones del entity manager que se conectarán en cascada.

```
@Entity public class Employee {
    // ...
    @ManyToOne(cascade=CascadeType.PERSIST)
    Address address;
    // ...
}
```

# Operaciones en cascada

- Los ajustes en cascada son unidireccionales.

- Esto significa que deben establecerse *explícitamente en ambos lados de una relación* si se pretende el mismo comportamiento para ambas situaciones.

# Cascade Remove

- En realidad, solo hay dos casos en los que la operación *remove()* en cascada tiene sentido, relaciones de 1:1 y 1:N en las que existe una clara relación padre-hijo.

- No se puede aplicar a ciegas a todas las relaciones porque las entidades de destino también pueden participar en otras relaciones o pueden tener sentido como entidades independientes.

- Se debe tener cuidado al usar la opción de cascada REMOVE.

# Cascade Remove

```
@Entity
public class Employee {
   // ...
   @OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})
   ParkingSpace parkingSpace;
   @OneToMany(mappedBy="employee",
            cascade={CascadeType.PERSIST, CascadeType.REMOVE})
   Collection<Phone> phones;
         // ...
}
```

# Detachment y Merge

- Una ***Detached Entity*** es aquella que ya no está asociada con un contexto de persistencia y este ya no realiza el seguimiento de la entidad.

- Los cambios realizados en la entidad no se conservarán en la BD, pero la aplicación aún puede utilizar todo el estado que había en la entidad cuando se desconectó.

- Lo opuesto al desapego es Merge es el proceso mediante el cual un entity manager integra el estado de una ***Detached Entity*** en un contexto de persistencia.

# Detachment y Merge

- Cualquier cambio en el estado de la entidad que se haya realizado en la ***Detached Entity*** sobrescribe los valores actuales en el contexto de persistencia.

- Cuando la transacción se confirma, esos cambios se mantendrán.

- Merge permite que las entidades se cambien "fuera de línea" y luego se incorporen esos cambios más adelante.

# Detachment y Merge

- La operación ***merge()*** se usa para fusionar el estado de una entidad separada en un contexto de persistencia.

- El método es fácil de usar y solo requiere la instancia de la entidad separada como argumento.

```
public void updateEmployee(Employee emp) {
   em.merge(emp);
   emp.setLastAccessTime(new Date());
}
```

# API de criterios

- Es una API para crear consultas con objetos Java, como alternativa a la creación de series para consultas JPQL ( Java Persistence Query Language).

- Da soporte a las consultas de creación dinámicamente en tiempo de ejecución y también a la capacidad de crear consultas de tipo seguro que el compilador puede verificar.

- El compilador no puede verificar la corrección de las consultas JPQL y debe verificarse en tiempo de ejecución durante las pruebas.

# API de criterios

- Una consulta JPQL de ejemplo que devuelve una lista de empleados con menos de cinco años de servicio

SELECT e FROM Employee e WHERE e.serviceyears < 5

- La consulta de criterios equivalente

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> c = cb.createQuery(Employee.class);
Root<Employee> e = c.from(Employee.class);
c.where(cb.lt(e.get(Employee_.serviceyears), 5));
TypedQuery tq = em.createQuery(c);
List result = c.getResultList();
```

# API de criterios

- La interfaz de ***CriteriaBuilder*** es la principal puerta de entrada a ***Criteria API*** y actúa como una fábrica para los diversos objetos que se vinculan entre sí para formar una definición de consulta.

- El objeto ***CriteriaQuery*** forma el caparazón de la definición de consulta y generalmente contiene los métodos que coinciden con las cláusulas de consulta de QL de Persistencia de Jakarta

# API de criterios

| Jakarta Persistence QL Clause | Criteria API Interface | Method |
|---|---|---|
| SELECT | CriteriaQuery | select() |
| | Subquery | select() |
| FROM | AbstractQuery | from() |
| WHERE | AbstractQuery | where() |
| ORDER BY | CriteriaQuery | orderBy() |
| GROUP BY | AbstractQuery | groupBy() |
| HAVING | AbstractQuery | having() |

# API de criterios

| Jakarta Persistence QL Operator | CriteriaBuilder Method |
|---|---|
| AND | and() |
| OR | or() |
| NOT | not() |
| = | equal() |
| <> | notEqual() |
| > | greaterThan(),gt() |
| >= | greaterThanOrEqualTo(),ge() |
| < | lessThan(),lt() |
| <= | lessThanOrEqualTo(),le() |
| BETWEEN | between() |
| IS NULL | isNull() |

# API de criterios

| Jakarta Persistence QL Aggregate Function | CriteriaBuilder Method |
|---|---|
| AVG | avg() |
| SUM | sum(),sumAsLong(),sumAsDouble() |
| MIN | min(),least() |
| MAX | max(),greatest() |
| COUNT | count() |
| COUNT DISTINCT | countDistinct() |

# Contacto

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx