RESÚMEN UNIDAD 3: MAPEO OBJETO-RELACIONAL

Solución: Mapeo Objeto-Relacional (ORM)

El ORM es una técnica de programación que aborda este problema, permitiendo la traducción entre el sistema de tipos de datos utilizado en programación orientada a objetos y el utilizado en bases de datos relacionales.

Proceso del ORM:

- Convierte valores de objetos en formas que pueden ser almacenadas en bases de datos.
- Preserva propiedades y relaciones de objetos, haciendo que estos objetos sean persistentes.

En el modelo relacional, cada fila en la tabla se mapea a un objeto y cada columna a una propiedad.

ORM permite tomar un objeto Java y hacerlo persistente, carga el objeto de la base de datos a memoria y permite hacer consultas a las tablas de la base de datos.

Ventajas de ORM:

1. **Reducción del Tiempo de Desarrollo:**
   - Agiliza la creación del modelo a partir del esquema de la base de datos.
2. **Abstracción de la Base de Datos:**
   - Permite interactuar con objetos en lugar de estructuras de base de datos.
3. **Reutilización y Persistencia de Objetos:**
   - Facilita la reutilización de código y persistencia de objetos con métodos como Orm.Save y Orm.Load.
4. **Lenguaje Propio para Consultas:**
   - Ofrece un lenguaje propio simplificando las consultas.
5. **Independencia de la Base de Datos:**
   - Favorece la creación de aplicaciones independientes de la base de datos.
6. **Incentivo a la Portabilidad y Escalabilidad:**
   - Mejora la portabilidad y escalabilidad de los programas de software.

Desventajas de ORM:

1. **Tiempo de Aprendizaje:**
   - Su correcta utilización requiere tiempo de aprendizaje debido a su complejidad.
2. **Menor Rendimiento:**
   - Puede generar aplicaciones ligeramente más lentas por la transformación de consultas.
3. **Complejidad del Sistema:**
   - La utilidad de ORM disminuye con la complejidad del sistema relacional.

HERRAMIENTAS ORM MÁS UTILIZADAS:

Hibernate - Herramienta ORM:

- **Tipo:** Mapeo Objeto-Relacional (ORM)
- **Plataforma:** Java (también disponible en .Net como NHibernate)
- **Función Principal:** Facilita el mapeo de atributos entre bases de datos relacionales y modelos de objetos en aplicaciones.
- **Configuración:** Utiliza archivos declarativos (XML) o anotaciones en beans de entidades para establecer relaciones.
- **Licencia:** Software libre, distribuido bajo los términos de la licencia GNU LGPL.

Java Persistence API (JPA):

- **Descripción:** Especificación de Sun Microsystems para la persistencia de objetos Java en bases de datos relacionales.
- **Plataforma:** Desarrollada para la plataforma JEE, integrada en el estándar de EJB 3.0 y parte de la Java Specification Request JSR 220.
- **Requisitos:** Requiere J2SE 1.5 o superior, aprovechando características como anotaciones y genéricos de Java.

iBatis:

- **Descripción:** Framework de persistencia desarrollado por Apache Software Foundation.
- **Licencia:** Código libre.
- **Método de Persistencia:** Similar a Hibernate, utiliza ficheros de mapeo XML para persistir información de objetos en bases de datos relacionales.

La herramienta ORM Hibernate es desarrollada por Oracle, el openJPA es una especificación de Sun Microsystems y la herramienta iBatis es desarrollado por Apache software Foundation.

Archivo de configuración de Hibernate se llama Hibernate.cfg.xml

Configuración Hibernate en NetBeans:

1. **Creación de Archivo de Configuración:**
    - Utilizando el asistente en NetBeans, se configura el archivo de Hibernate.
    - Se elige la conexión a la base de datos de una lista registrada en el IDE.
2. **Generación Automática:**
    - El archivo de configuración generado incluye detalles de la conexión y bibliotecas de Hibernate.
    - Puede editarse a través del editor interactivo o directamente en código XML.
3. **Contenido del Archivo de Configuración:**
    - Contiene información sobre la base de datos que la aplicación va a utilizar.
    - Si hay varias bases de datos, se deben crear archivos de configuración separados.
4. **Archivos en Hibernate:**
    - **Hibernate.properties:** Define aspectos del gestor de la base de datos y conexiones.
    - **Archivos de Emparejamiento (Mapping):** (*.hbm.xml) Establecen la relación entre propiedades, tablas y columnas en la base de datos.

Archivos de Mapeo en Hibernate:

1. **Correspondencia Bean-Tabla:**
    - Los archivos de mapeo indican la relación entre el bean y una tabla de la base de datos.
    - Al realizar tareas que requieran acceso a la base de datos, se obtiene una conexión JDBC.
2. **Nomenclatura de Archivos:**
    - Se nombran usando el nombre de la clase seguido de la extensión "hbm.xml".
3. **Ingeniería Inversa:**
    - Posibilidad de crear archivos de mapeo basados en tablas de la base de datos seleccionadas.
    - Utiliza el archivo Hibernate.reveng.xml para modificar la configuración predeterminada, especificando el esquema de la base de datos, filtrando tablas y asignando tipos JDBC a los tipos de Hibernate.
4. **Extracción de Tablas:**
    - Para extraer una tabla específica, se define el archivo POJO Nombre_Clase.hbm.xml, describiendo la relación entre clases y tablas, así como propiedades y columnas.

Mapeo en Hibernate: Colecciones, Relaciones y Herencia

1. **Mapeo de Colecciones:**
    - Diversos mapeos para colecciones según modelos relacionales.
    - Uso de elementos de mapeo específicos (p. ej., <set>) dependiendo del tipo de interfaz (Set, etc.).
    - Las colecciones pueden manejar datos de diversos tipos, requiriendo una columna índice en la tabla de colección.
2. **Mapeo de Relaciones:**
    - Las relaciones utilizan transacciones y deben reflejar el tipo de relación en ambos modelos (objeto y relacional).
    - Se emplean identificadores de objetos (OID) como llave primaria en la tabla relacionada, actuando como clave foránea en la tabla principal.
3. **Mapeo de Herencia:**
    - Desafío ya que las bases de datos relacionales no admiten herencia directamente.
    - Tres mapeos principales: a una sola tabla, completa en tablas separadas, o cada clase en tablas concretas.
    - La elección se basa en rendimiento y escalabilidad del modelo.

La persistencia de objetos implica su capacidad de guardarse y recuperarse. En JDO, las clases persistentes implementan javax.jdo.PersistenceCapable, permitiendo la conversión y el mantenimiento en sistemas de bases de datos con el uso de un archivo de configuración XML.

Session y SessionFactory en Hibernate:

- **Session:**
  - Representa una unidad de trabajo con la base de datos en Hibernate.
  - Contiene una cola de sentencias SQL y una lista de objetos persistentes.
  - Forma el primer nivel de caché y define el alcance de un contexto de acceso a datos.
- **SessionFactory:**
  - Inicializa el entorno de Hibernate y proporciona objetos Session.
  - Abre una única unidad de trabajo para un almacén de datos específico.
  - Las sesiones deben cerrarse al completar una transacción.

Estados del Objeto en Hibernate:

1. **Transitorio (Transient):**
   - Objeto recién creado sin enlace al gestor de persistencia.
2. **Persistente:**
   - Objeto enlazado con la sesión, con cambios persistentes.
3. **Disociado (Detached):**
   - Objeto persistente en memoria después de finalizar la sesión, existiendo en Java y la base de datos.
4. **Borrado (Removed):**
   - Objeto marcado para eliminación en la base de datos, persistiendo en la aplicación Java hasta el cierre de la sesión.

Carga de Objetos en Hibernate:

- Métodos de Carga:
  - **load():**
    - Captura instancia persistente por identificador.
    - Acepta un objeto Class para cargar nueva instancia en estado persistente.
    - Lanza excepción si no existe la fila correspondiente en la base de datos.
    - Carga perezosa: intenta devolver un objeto proxy antes de la inicialización.
  - **get():**
    - Recupera objeto persistente por identificador.
    - Consulta la base de datos inmediatamente y devuelve null si no existe la fila.
- Diferencia Clave:
  - **load():** Lanza excepción ObjectNotFoundException si no existe la fila.
  - **get():** Devuelve null si no existe la fila correspondiente.

Para almacenar objetos persistentes en Hibernate:

1. **Creación y Almacenamiento:**
   - Instanciar objeto (estado transitorio).
   - Obtener sesión, iniciar transacción y usar save() para almacenar.
   - Realizar commit para sincronizar cambios.
2. **Actualización de Objetos:**
   - Actualizar directamente mientras la sesión esté abierta.
3. **Eliminación de Objetos:**
   - Borrar con Session.delete().
4. **Reasociación de Entidades:**
   - Hibernate soporta reasociación con Session.update() o Session.merge().

Resumen consultas SQL:

En Hibernate, el control de consultas SQL nativas se realiza con SQLQuery, obtenido a través de Session.createSQLQuery(). Pueden ejecutarse consultas básicas para obtener listas de escalares o consultas más complejas para obtener objetos entidades utilizando addEntity(). Ejemplos:

1. Lista de escalares:
   - `sess.createSQLQuery("SELECT * FROM Personas").list();`
   - `sess.createSQLQuery("SELECT ID,NOMBRE, EDAD FROM PERSONAS").list();`
2. Consulta de entidades:
   - `sess.createSQLQuery("SELECT * FROM PERSONAS").addEntity(Persona.class);`
   - `sess.createSQLQuery("SELECT ID,NOMBRE,EDAD FROM PERSONAS").addEntity(Persona.class);`

LENGUAJE HQL:
Hibernate utiliza el lenguaje de consulta HQL, similar a SQL pero orientado a objetos, considerando conceptos como herencia y asociación.
HQL es case-insensitive y devuelve resultados en forma de objetos, facilitando su uso.
Se ejecuta sobre el modelo de entidades definido en Hibernate, utilizando clases Java en lugar de tablas de la base de datos.

Resumen DE SENTENCIAS HQL:

- **Clausula FROM:** Consulta básica para mostrar todos los datos de una tabla, ej. `FROM Alumnos`.
- **Clausula SELECT:** Selecciona objetos y propiedades en el conjunto de resultados, ej. `SELECT alumno.nombre FROM Alumnos`.
- **Clausula WHERE:** Refina los resultados con condiciones, ej. `FROM Alumnos WHERE nombre='Francisco'`.
- **Funciones de Agregación:** Pueden usarse funciones como `avg`, `sum`, `max`, `count`, ej. `SELECT avg(alumnos.nota), sum(alumnos.nota) FROM Alumnos as alumnos`.
- **Expresiones:** Incluyen operadores matemáticos, comparaciones binarias, lógicos, funciones Java, etc.
- **Clausula ORDER BY:** Ordena la lista de resultados, ej. `FROM Alumnos ORDER BY nombre ASC`.
- **Clausula GROUP BY:** Agrupa resultados agregados por propiedades, ej. `SELECT avg(alumnos.nota) FROM Alumnos as alumnos GROUP BY alumnos.clase`.
- **Subconsultas:** Soporta subconsultas dentro de consultas, incluso subconsultas correlacionadas. Ejemplo: `FROM Alumnos WHERE nota > (SELECT avg(nota) FROM Alumnos)`.

CÓDIGO RELEVANTE :

Session session = HibernateUtil.getSessionFactory().openSession();

```
Transaction tx = null;
try
{
          tx = session.beginTransaction();
// Utilizar la Session para saveOrUpdate/get/delete/...tx.commit();
} catch (Exception e)
{
          if (tx != null)
          {
               tx.rollback();
               throw e;
          }
} finally {
          session.close();
}// Al finalizar la aplicación ...HibernateUtil.shutdown( );
```