T2-ACCESO-A-DATOS.pdf

bloodyraintatii

Acceso a datos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Acceso a Datos

Tema 9: Bases de datos orientadas a objetos

DEFINICIÓN DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

→ BBDD orientadas a objetos → las técnicas de bbdd se combinan con objetos.
→ ODBMS (Object Data Base Management System) → SGBD orientado a objetos.
→ Modelo de datos orientado a objetos (OODM) → es un tipo de modelo lógico que usa los objetos para representar y organizar datos en aplicaciones. Son modelos conceptuales y pueden representar complejidades que van más allá de los SGBD relacionales.
→ Base de datos orientada a objetos → colección de objetos definidos por un modelo orientado a objetos. Pueden extender la existencia de los objetos para que se almacenen indefinidamente, por lo que pueden seguir almacenados después de finalizar la aplicación, pudiendo recuperarlos más tarde y compartirlos por otras aplicaciones.

CARACTERÍSTICAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

→ Características de las bbdd orientadas a objetos:
- Los objetos tienen relación directa con el mundo real y no pierden su identidad ni integridad.
- Tienen un identificador de objeto generado por el sistema.
- Son fáciles de extender y de añadir nuevos tipos de datos y operaciones que se realizarán en ellos.
- Proporcionan encapsulación, ocultando entidades externas,
- Proporcionan propiedades de herencia.

→ Conceptos asociados a los objetos:
- Atributos/variables de instancia → características que suelen describir los objetos.
- Objeto → representación abstracta de una entidad del mundo real. Tiene identificador único, propiedades embebidas y capacidad de interactuar con otros objetos por sí mismo.
- Identidad → identificador externo (ID del objeto), creado por el sistema e inmutable.

SISTEMAS GESTORES DE BASES DE DATOS ORIENTADOS A OBJETOS

→ SGBD orientados a objetos → sistema gestor de almacenamiento de datos que soporta el modelado y la creación de los datos como objetos. Permite concurrencia y recuperación. Manipula objetos directamente sin tener que traducir filas y columnas.
→ Un SGBD tiene datos relacionados entre sí y una aplicación con acceso a la bbdd.
→ Características SGBD genérico: concurrencia, persistencia, recuperación de errores, gestión de almacenamiento, consultas.
→ Características SGBD orientado a objetos → abstracción, modularidad, jerarquía, encapsulación, tipología de objetos y las anteriores genéricas.

HERRAMIENTA OBJECTDB (CASO PRÁCTICO)

→ Con ObjectDB podemos gestionar una bbdd orientada a objetos.
→ Funcionalidades: herramienta visual donde realizar consultas, creador de diagnósticos, creador de copias maestro-esclavo y creador de copias de seguridad.

LENGUAJE DE CONSULTAS PARA OBJETOS

→ Lenguaje de consulta de objetos/Object Query Language (OQL) → lenguaje declarativo de tipología SQL para hacer consultas efectivas en bbdd y estructuras orientadas a objetos.
→ No tiene primitivas para modificar el estado de los objetos; las modificaciones se hacen a través de los métodos de los objetos.
→ Estructura básica de SELECT, FROM y WHERE.
→ Asociación → al borrar un objeto, los relacionados siguen existiendo.
→ Agregación → no se puede introducir o borrar un objeto de forma aislada.
→ Herencia → una clase hereda atributos y métodos de otra.
→ Especialización → proceso de definir subclases más específicas a partir de una general, agregando atributos o comportamientos únicos.
→ Generalización → proceso de crear una clase general a partir de otras específicas, extrayendo características comunes.
→ Las variables de tipo *iterador* se pueden nombrar de 3 formas distintas: *C in Customer*, *Customer C* o *Customer as C*. No es obligatorio tener la cláusula SELECT → *Customer* devolverá todos los objetos de este tipo con sus subtipos si tiene.
→ Una consulta OQL comienza con el nombre del objeto seguido de uno, varios o ningún atributo:

- *customer.empresa* → objeto de tipo Empresa.
- *customer.empresa.nombre* → String con el nombre de la empresa
- *customer.empresas_asociadas* → colección de un Set<Empresa>.

VENTAJAS EN LA UTILIZACIÓN DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

→ Ventajas de desarrollar una aplicación orientada a objetos:

- Mayor capacidad de modelado → se pueden encapsular comportamientos y estados, se pueden almacenar las relaciones de un objeto dentro de este y permite la herencia al poder formar objetos complejos.
- Flexibilidad importante → creación de nuevas categorías partiendo de otras ya existentes, se reduce la redundancia al poder aunar características en superclases y las clases u objetos son reusables.
- Facilidad de hacer consultas de forma práctica → es un lenguaje expresivo con acceso navegacional entre objetos y sus herencias.
- Rendimiento muy competitivo → es superior en las bbdd orientadas a objetos.

DESVENTAJAS EN LA UTILIZACIÓN DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

→ Posibles desventajas de las bbdd orientadas a objetos:

- Falta de un modelo de datos universal → no tiene y a la mayoría les falta una buena base teórica.
- Falta de experiencia → menos madurez que los sistemas de bbdd relacionales.
- Falta de estándares → no hay definidos.
- Competencia → los sistemas relacionales y los objeto-relacionales tienen un gran estándar aprobado, como SQL, y una base teórica sólida.
- Encapsulación casi obligada al hacer consultas → ya que se accede mediante el sistema de consultas.
- Relación con la teoría matemática → aún no tiene aprobación.

Tema 10: Bases de datos objeto relacionales

DEFINICIÓN DE BASE DE DATOS OBJETO RELACIONALES

→ BBDD objeto-relacional → bbdd relacional que evoluciona hacia algunas características del modelo de objetos, haciéndola una base de datos híbrida.
→ Soporta objetos, clases, herencia, etc. de los modelos orientados a objetos y tipos de datos o estructuras tabulares del modelo de datos relacional.
→ Resultado → modelo relacional de objetos que ofrece la intuición y la economía de una interfaz de objetos conservando su alta concurrencia y rendimiento de una relacional.
→ Se suelen usar en Java, C++, C#... y el SGBD más conocido es Oracle.

CARACTERÍSTICAS DE LAS BASES DE DATOS OBJETO RELACIONALES

→ Se podrán crear nuevos tipos de datos, dándole riqueza de dominios a las apps.
→ Los nuevos tipos de datos pueden ser tipos compuestos, por lo que podemos tener un método para convertirlos a ASCII y otro método inverso.
→ Tipos complejos: registros, listas, referencias, pilas, colas, arrays…
→ Se pueden hacer funciones con código en diferentes lenguajes: SQL, Java, C#...
→ Características de las bbdd objeto-relacionales:

- Mayor capacidad de expresión → para definir conceptos y diferentes asociaciones.
- Operadores personalizados → posibilidad de crear operadores específicos para consultas complejas, asignándoles un nombre y reutilizándolos.
- Encadenamiento y herencia en tipos de registro → soporta estructuras de datos complejas mediante encadenamiento y herencia.
- Facilita la reusabilidad → mediante bibliotecas de clases predefinidas.
- Comprobación de reglas de integridad → se pueden hacer mediante *triggers*.

TABLAS Y TIPOS DE OBJETOS: DEFINICIÓN

→ Crear un tipo de dato es definir cierto comportamiento de una agrupación de datos.
→ Para tipos de objetos → *object type*. | Para tipos de colecciones → *collection type*.
→ Un objeto representa una entidad del mundo real y tiene nombre (para identificar al tipo de objeto), atributos (para definir la estructura) y métodos (en PL/SQL, o C externamente).
→ La creación de un método en Oracle se hace junto a la creación de su tipología y debe llevar el tipo de compilación (como PRAGMA RESTRICT_REFERENCES) para evitar la manipulación de los diferentes datos o variables PL/SQL.
→ Creación de un tipo de dato:

```
CREATE TYPE coche_t AS OBJECT (
tipo vehiculo_t, → herencia
puertas NUMBER, → número
marca VARCHAR2(200)); → cadena de texto
```

TABLAS Y TIPOS DE OBJETOS: EXPLOTACIÓN

Tabla que almacena objetos con su propio ID:	Tabla cuyo tipo de dato es un objeto:
CREATE TABLE vehiculos OF vehiculo_t (numVehiculo PRIMARY KEY);	*CREATE TABLE vehiculos_antiguos (anio NUMBER, vehiculo vehiculo_t);*

→ La segunda tabla tiene una columna con un tipo de datos complejo y sin identidad de objeto.
→ Oracle también permite definir como tabla una columna con tipología de objeto o aquella que tiene el mismo número de columnas como atributos que almacena.

TIPOS DE COLECCIÓN: ARRAY
→ Las colecciones sirven para poder establecer relaciones de "uno a muchos" (1:N). Pueden ser en forma de array (Varray) o de tabla anidada.
→ Varray → elementos ordenados cuyo índice es su posición dentro del array, con un número máximo de elementos.

```
CREATE TYPE numeros AS VARRAY(10) OF NUMBER(10);
numeros ('6', '18', '75870');
```

→ Usos de un VARRAY:
- Definir la tipología de una columna de una tabla relacional.
- Definir la tipología de un atributo de un tipo objeto.
- Definir una variable del lenguaje PL/SQL.

→ Si el espacio que ocupa es mayor que 4000 bytes se almacena en una tabla aparte como un dato de tipo BLOB.

TIPOS DE COLECCIÓN: TABLAS ANIDADAS
→ Tabla anidada → lista de elementos de una misma tipología no ordenados y sin número máximo de elementos.
→ Se usa la expresión TABLE con SELECT, INSERT, DELETE y UPDATE.
→ Si la tabla anidada es de un tipo de objeto de usuario, también se puede ver como una tabla multicolumna, con una columna por cada atributo del objeto.

```
CREATE TYPE nombre_tipo AS TABLE OF tabla_tipo;
```

→ Usos de la definición de un tipo de objeto usuario:
- Para el tipo de datos de una tabla relacional.
- Para un atributo de un objeto de tipo usuario.
- Para una variable PL/SQL, un parámetro o una función que devuelva un tipo.

REFERENCIAS (??)
→ Tipo REF → atributo que guarda una referencia o enlace a un objeto del tipo definido y genera una relación entre ambos objetos.
→ Se usa para acceder a los objetos relacionados y actualizarlos, no para operar sobre ellos. Para usar una referencia o actualizarla, usaremos REF o NULL.

```
CREATE TYPE ordenes_t AS OBJECT(
ordennum NUMBER,
cliente REF clientes_t); → hace la referencia a un tipo de datos

CREATE TABLE ordenes_tab OF ordenes_t (
PRIMARY KEY (ordenesnum),
SCOPE FOR (cliente) IS clientes_tab); → restringe su dominio a una determinada tabla
```

HERENCIA DE TIPOS
→ Permite crear una jerarquía con niveles más especializados derivados del supertipo.

Tema 13: Explotación de bases de datos No-SQL

CONCEPTO Y CARACTERÍSTICAS PRINCIPALES DE LOS ÍNDICES

→ Índice → estructura de datos que mejora la velocidad de las operaciones de búsqueda y recuperación de información en una tabla. Funciona de manera similar a un índice en un libro, permitiendo localizar rápidamente los datos sin tener que revisar cada fila de la tabla.

→ Características de los índices:

- Grandes mejoras en las operaciones de lectura.
- Pueden penalizar las operaciones de escritura.
- Se almacenan en memoria.
- Por defecto se crea un índice único sobre el campo _id.
- No solo se indican los campos del índice sino el orden del mismo.
- Los operadores de negación no utilizan índices.
- Los emplean las querys y las "agregaciones".

db.collection.createIndex(*{ <field_1>: <index type>,* *<field_2>: <index type> … }* *{<options>})*	*db.usuarios.createIndex(* *{* *nombre: 1,* → orden ascendente *edad: -1* → orden descendente *},* *{* *unique: true* → índice único *}* *);*

TIPOLOGÍAS DE LOS ÍNDICES EN MONGODB (?)

→ Los índices pueden ser de varios tipos:

- Monoclave/simple → solo indexan por un campo de los documentos de la colección, solo afectando a un campo de búsqueda.

```
db.collection.createIndex( { <field>: <-1|1> } )
```

- Índices sobre entidades embebidas → las entidades y el orden deben ser iguales.
- Compuestos → indexan por varios campos de los documentos de la colección.

```
db.collection.createIndex( { <field_1>: <-1|1>,<field_2>: <-1|1> } )
```

- Hashed/multiclave → indexan de forma clave-valor para los valores de un array.
- Text → índices de texto completo en campos de cadena de texto.
- Geoespaciales → indexan por datos geográficos, como coordenadas espaciales.
- Unique/único → asegura que no haya valores duplicados en el campo indexado.

```
db.collection.createIndex({ <field>: <index_type> }, {unique: <true|false>} )
```

- Sparse → solo indexa documentos que tengan el valor especificado, no nulo.

```
db.collection.createIndex( { <field>: <index_type> }, {sparse: <true|false>} )
```

- Background → no bloquea a lectores/escritores al crearse en segundo plano.
- TTL (Time-To-Live) → elimina automáticamente docs tras un tiempo o en una fecha.
- Partial → solo indexa documentos que cumplan una condición específica.

Descansito &Coins

Gemini + 

¿Sabrías identificar en qué te puede ayudar Google Gemini para estudiar?

REGLAS

1. Observa las opciones disponibles
2. Responde como Gemini te ayuda a estudiar.
3. Gana Wuolah coins para descargas sin publi.

Fácil | 10

A. Sintetiza horas de investigación en minutos.

B. Convierte tus apuntes en un esquema visual.

C. Prepara un examen para autoevaluarte.

D. Convierte tus apuntes en podcasts.

E. Sube hasta 1.500 páginas y analiza textos largos.

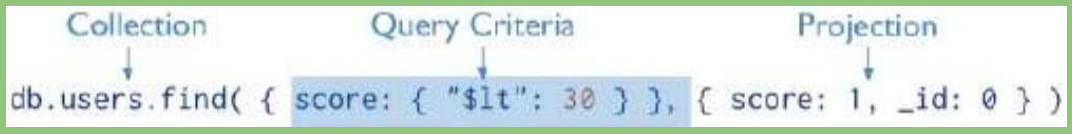
F. Todas las anteriores.

Gemini

OPERACIONES CON LOS ÍNDICES (¿)

→ Se pueden destacar las siguientes operaciones con los índices:

- Covered query → consulta que se resuelve contra un índice sin consultar datos de la colección →



- Borrado de índice → db.accounts.dropIndex({ "tax-id": 1 })
- Regeneración de índice → db.accounts.reIndex()
- Listado de índices → db.accounts.getIndexes()

CREACIÓN DE USUARIOS Y ROLES (¿?)

→ MongoDB tiene un control de acceso basado en roles (RBAC), otorgando rol/es a los usuarios para darles acceso o privilegios a los recursos de la bbdd.

→ Se debe garantizar un sistema de privilegios mínimos, dándoles los mínimos necesarios o imprescindibles a los usuarios.

→ Creación de un usuario → *db.createUser()*

→ Elementos de la creación → *user*, *pwd* (contraseña), *customData*, *roles*, *authenticationRestrictions* (array con las direcciones IP desde las que el usuario se puede conectar), *mechanisms* (array que especifica si tiene permiso para crear credenciales SCRAM) y *passwordDigestor* (String que indica si el servidor o el cliente traducen la contraseña).

→ Habilitar el control de acceso basado en roles → *--auth* o *security.authorization*.

→ Un privilegio se aplica sobre un recurso específico y las acciones sobre este.

→ Ver los privilegios de un rol → *rolesInfo* con *showPrivileges* y *showBuiltinRoles* a true.

IMPORTACIÓN DE DATOS

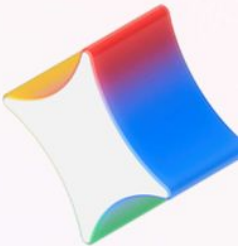
→ Se hace la importación de datos a través de Compass, una herramienta visual.

1. Nos conectamos a la base de datos, le damos a *Add Data* y a *Import File*.
2. Indicamos la ruta del fichero y el tipo (JSON o CSV). Si elegimos CSV, debemos especificar los campos a importar y sus tipos.
3. Configuramos las opciones de importación acorde a nuestro caso.
4. Al terminar la barra de proceso se mostrarán los datos importados.

EXPORTACIÓN DE DATOS

→ Vamos a hacer la exportación de datos con Compass también.

1. Nos conectamos a la bbdd y navegamos hasta la información o colección/es que queramos exportar.
2. Clickeamos el menú *Collection* y seleccionamos *Export Collection*.
3. Podemos exportar según una query determinada o la colección completa.
4. Seleccionamos el formato del fichero y le damos a *Export*.



Tema 14: Gestión de bases de datos nativas XML

BASES DE DATOS XML

→ BBDD XML nativas → bbdd que almacenan documentos y datos XML de forma muy eficiente, permitiendo que los datos se almacenen, consulten, combinen, indexen, etc.
→ Se basan en contenedores en vez de tablas, que pueden almacenar grandes cantidades de documentos XML o datos con alguna relación entre ellos.
→ Los contenedores pueden tener subcontenedores también.
→ La estructura de los datos XML en un contenedor no tiene por qué ser fija.
→ Son consultadas por expresiones Xpath en lugar de sentencias SQL.
→ Xpath → estándar mundial establecido por el W3C para navegar a través de documentos XML. Se basa en una representación de árbol del documento XML, seleccionando nodos según diferentes criterios.

VENTAJAS Y DESVENTAJAS DE LAS BASES DE DATOS XML CON RESPECTO A LAS RELACIONALES

→ Ventajas del uso de las bbdd XML:
- Pueden almacenar, mantener y consultar mayores cantidades de documentos XML.
- No es necesario configurar tablas así que no se necesita hacer diseños complicados antes de configurar la bbdd.
- Poder importar/exportar fácilmente hacia/desde otras apps con otros formatos.

→ Desventajas del uso de las bbdd XML:
- Son relativamente recientes por lo que la experiencia es aún limitada, al contrario que las bbdd relacionales, que son tecnología ya probada.
- Xpath no es muy fácil de aprender mientras que SQL está muy extendido.
- No todas las apps gestoras de datos son compatibles con estas bbdd.

GESTORES XML LIBRES Y COMERCIALES (no sé?)

→ De pago → Excelon XIS, GoXML DB, Infonyte DB.
→ Libres/de código abierto → Exist DB, X-Hibe DB, Tamino DB.

INSTALACIÓN Y CONFIGURACIÓN DE EXISTDB

→ Requisitos de sistema recomendados → JDK Java 8 y tener disponibles 128 MB de memoria caché y 1024 MB de memoria RAM.
→ Descargamos el archivo .jar y lo lanzamos. Seleccionamos ubicación y directorio del que cogerá los ficheros de información.
→ Configuramos la memoria y contraseña del usuario Admin.
→ Establecemos la cantidad de RAM que queremos darle al proceso Java y al caché.
→ Podemos agregar distintos paquetes a la instalación:
- Paquete “core” → requerido para poder “correr” la bbdd.
- Paquete “sources” → es preferible instalarlo pero no obligatorio.
- Paquete “app” → permite seleccionar algunas apps que se instalarán al arrancarlo por primera vez.

→ Finalizamos la instalación y ejecutamos el acceso directo o con los comandos *launcher.sh* o *launcher.bat*.
→ Vamos a localhost:8080 y ya tenemos la app lista.

ESTRATEGIAS DE ALMACENAMIENTO (NO?)
ESTABLECIMIENTO Y CIERRE DE CONEXIONES (NO?)
AGREGAR, MODIFICAR Y ELIMINAR RECURSOS (NO?)
MODIFICACIÓN DE CONTENIDOS XML (NO?)

TRANSACCIONES Y EXCEPCIONES (NO?)
→ Transacción → conjunto de declaraciones ejecutadas que forman una unidad de trabajo inseparable, de modo que se ejecutan todas o ninguna.
→ ExistDB admite transacciones compatibles con ACID → atomicidad, consistencia, aislamiento y durabilidad.
→ Se usa la clase XMLDBException para capturar todos los errores que ocurren al usar XML, ya sea con un código de error especificado o definido.

DESCARGA E INSTALACIÓN DE EXIST DB (CASO PRÁCTICO)
→ Vamos a la web oficial de la herramienta.
→ Hay diferentes formas de integración de la bbdd en nuestro sistema:
- Descargar e instalar la última versión.
- Descargar una imagen de *docker* y montarla.
- Dependencia *maven* con la bbdd para inyectarla en el proyecto que queramos.

→ En nuestro caso usamos la primera opción.

Tema 15: Programación de componentes de AAD

CONCEPTO Y CARACTERÍSTICAS DE UN COMPONENTE

→ Componente → unidad de software que encapsula un segmento de código con ciertas funciones.
→ Pueden ser visuales, cuyo estilo se da por el entorno de desarrollo e interfaz de usuario, o no visuales, con funciones similares a las bibliotecas remotas.
→ Características principales de los componentes del software:

- Es una unidad ejecutable que se puede instalar y utilizar independientemente.
- Puede interactuar y operar con otros componentes desarrollados por terceros.
- No tienen estado o no es visible desde "fuera".
- La programación orientada a componentes se puede asociar a los distintos engranajes electrónicos que forman un sistema más grande en conjunto.

→ Elementos que forman un componente:

- Atributos, operaciones y eventos → son la interfaz del componente, definiendo cómo interactúa con otros elementos (nivel sintáctico).
- Comportamientos → define cómo funciona internamente (nivel semántico).
- Protocolos y escenarios → representa cómo el componente se comunica e integra con otros componentes en diferentes situaciones.
- Propiedades → características específicas que definen al componente.

PROPIEDADES Y ATRIBUTOS DE UN COMPONENTE

→ Las propiedades de un componente determinan su estado y lo distinguen del resto. Se puede acceder a ellas y modificarlas de diferentes formas.
→ Las propiedades de los componentes se dividen en simples, indexadas, compartidas y restringidas.
→ Se pueden verificar y modificar las propiedades con los métodos *get()* y *set()*.

Tipo getNombrePropiedad()	*SetNombrePropiedad(Tipo valor)*

- Atributos simples → representan un solo valor (color, tamaño, etiqueta) ⬆
- Índices → conjunto de valores con el mismo tipo, pudiendo acceder a ellos por índice → *setPropertyName (int index, PropertyType value)*.
- Atributos compartidos → permiten notificar cambios en el atributo mediante eventos, usando métodos específicos para gestionar los receptores → *addPropertyChangeListener* y *removePropertyChangeListener*.
- Atributos restringidos → permiten cambios solo si otros componentes lo aprueban, usando métodos específicos para gestionar la validación → *addVetoableChangeListener* y *removeVetoableChangeListener*.

IMPLEMENTACIÓN DE UN COMPONENTE Y PROPAGACIÓN (CASO PRÁCTICO)

→ Realizar un componente de la clase PropertyChangeSupport que ejecute métodos propagados de *addPropertyChangeListener* y *removeProperyChangeListener*.

1. Importar la paquetería java.beans.
2. Crear un objeto de la clase mencionada con una lista de oyentes que lanzarán los diferentes eventos de cambio de propiedad.
3. Implementar los métodos envolviendo las diferentes llamadas de los métodos del objeto soportado.

EVENTOS

→ Control activo → un componente activa el funcionamiento de otro mediante llamadas directas (la forma más común de interacción).
→ Control reactivo → los componentes generan eventos (que representan solicitudes de invocación de operaciones) y otros componentes reaccionan a estos para realizar acciones.
→ En resumen, los eventos permiten una interacción reactiva entre componentes, donde uno genera eventos y otros reaccionan a ellos, en lugar de usar llamadas directas.

INTROSPECCIÓN

→ Introspección → mecanismo por el que se pueden descubrir las propiedades, los métodos y los eventos contenidos en el componente.
→ Los componentes admiten la introspección de dos formas:

- Convenciones de nomenclatura específicas → es una forma de introspección basada en patrones de nombres que permiten descubrir atributos, métodos o eventos automáticamente.
- Información clara → incluye la reflexión como una técnica para inspeccionar dinámicamente métodos, campos y constructores públicos durante la ejecución, sin necesidad de conocer la estructura interna de antemano.

→ En resumen, la introspección permite descubrir las características de un componente mediante convenciones de nombres o reflexión, lo que facilita su uso y comprensión en tiempo de ejecución.

PERSISTENCIA DEL COMPONENTE

→ La persistencia en EJB se logra con la biblioteca JPA de Java. Podemos encontrar esta especificación en Oracle.
→ JPA requiere JDK 1.5 o superior y conocimientos sólidos de POO, Java y SQL.
→ Clases principales de JPA:

- Persistence → proporciona métodos para obtener un *EntityManagerFactory*.
- EntityManagerFactory → genera objetos *EntityManager* usando el patrón Factory.
- EntityManager → interfaz principal para gestionar operaciones CRUD sobre objetos persistentes.
- Entity → anotación Java que representa entidades como registros en la bbdd.
- EntityTransaction → permite agrupar operaciones persistentes, con soporte para rollback en caso de error.
- Query → interfaz para realizar consultas usando JPQL o SQL.

→ Las anotaciones JPA (como javax.Persistence.*) permiten definir entidades y su persistencia, y son soportadas por IDEs como NetBeans, Eclipse e IntelliJ IDEA.
→ En resumen, JPA facilita la persistencia de objetos en bases de datos mediante clases y anotaciones que gestionan operaciones CRUD y transacciones.

PERSISTENCIA DE 2 OBJETOS (CASO PRÁCTICO)

→ Se hacen los objetos como normalmente en Java y después se persisten así (gestionando excepciones con un bloque try-catch también):

```
EntityManagerFactory emf =Persistence.createEntityManagerFactory("FactoryPersonas");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
em.persist(persona1);
```

```
em.persist(persona2);
em.getTransaction().commit();
em.close();
```

EMPAQUETADO DE COMPONENTES (no sé?)

→ Un módulo J2EE agrupa componentes del mismo tipo (como EJB) junto con sus descriptores de implementación, definiendo atributos de transacciones y seguridad.

→ Los módulos JavaEE sin descriptores de implementación de aplicaciones se pueden implementar como módulos separados.

→ Tipos de módulos J2EE para aplicaciones web EJB:

- Módulo EJB → contiene clases EJB y descriptores de despliegue, empaquetados en un archivo .jar.
- Módulo Web → incluye Servlets, JSP, archivos HTML, GIF y descriptores, empaquetados en un archivo .war (archivo web).
- Módulo de aplicación cliente → contiene clases y descriptores de la aplicación cliente, empaquetados en un archivo .jar.
- Módulo adaptador de recursos → incluye interfaces, clases, bibliotecas nativas y descriptores para adaptadores de recursos.

→ Buenas prácticas → en aplicaciones J2EE, es común separar el front-end web (módulo .war) y el back-end EJB (módulo .jar) dentro de un archivo .ear, aunque esto puede complicarse en aplicaciones simples.

→ En resumen, el empaquetado de componentes en J2EE organiza los elementos de una aplicación en módulos específicos, facilitando su implementación y gestión.

CRITERIOS

Tema 9

4a) Se han identificado las ventajas e inconvenientes de las bases de datos que almacenan objetos.

4c) Se ha gestionado la persistencia de objetos simples.

Tema 10

4d) Se ha gestionado la persistencia de objetos estructurados.

Tema 13

4c) Se ha gestionado la persistencia de objetos simples.

Tema 14

5a) Se han valorado las ventajas e inconvenientes de utilizar una base de datos nativa XML.

5b) Se ha instalado el gestor de base de datos.

Tema 15

6a) Se han valorado las ventajas e inconvenientes de utilizar programación orientada a componentes.

6b) Se han identificado herramientas de desarrollo de componentes.

6c) Se han programado componentes que gestionan información almacenada en ficheros.

6d) Se han programado componentes que gestionan mediante conectores información almacenada en bases de datos.

6e) Se han programado componentes que gestionan información usando mapeo objeto relacional.