CICLO FORMATIVO DE GRADO SUPERIOR: DESARROLLO DE APLICACIONES MULTIPLATAFORMA

2º CURSO

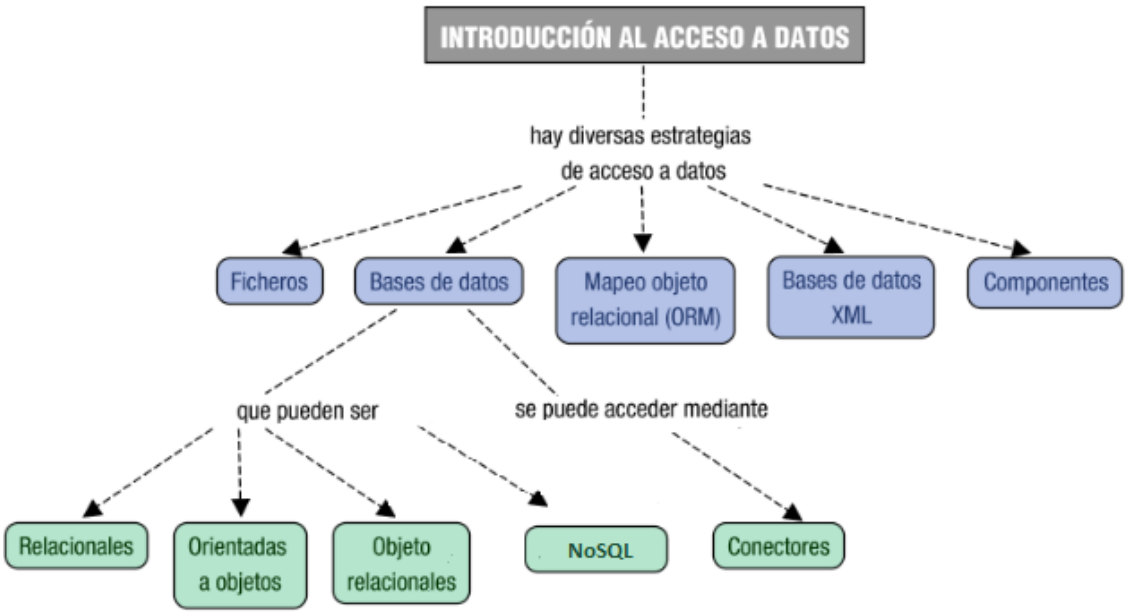
MODULO : ACCESO A DATOS

UT01: Introducción y Conceptos Básicos para el Acceso a Datos.

INDICE

UT01: INTRODUCCION Y CONCEPTOS BASICOS

1. MAPA CONCEPTUAL



2. INTRODUCCIÓN AL ACCESO A DATOS.

Podemos afirmar que en la inmensa mayoría de aplicaciones informáticas se pueden diferenciar, a grandes rasgos, en dos partes:

- Por un lado, **el programa** propiamente dicho, que realiza las operaciones deseadas con los datos necesarios.
- Por otro lado, **los datos** con los que opera le programa. Esos datos pueden ser obtenidos por el programa mediante diversos métodos: leídos mediante teclado, escaneados, leídos de algún soporte de almacenamiento secundario, etc.

En la mayoría de los casos, cuando programamos, nos interesa que el programa guarde los datos que le hemos introducido, o los resultados que dicho programa haya obtenido, de manera que si el programa termina su ejecución, los datos no se pierdan y puedan ser recuperados posteriormente, es decir, **persistan**. Una forma tradicional de hacer esto es mediante la utilización de ficheros o de bases de datos que se guardarán en un dispositivo de memoria no volátil (normalmente un disco).

Te habrás dado cuenta de que el almacenamiento en memoria RAM, mediante variables o vectores, es temporal, los datos se pierden cuando el programa termina. Quizás te habrá pasado alguna vez que, debido a un apagón eléctrico, has perdido el trabajo que estabas haciendo, que todavía no habías grabado. Los datos que se guardan en

almacenamiento secundario, como ficheros o bases de datos, se denominan datos persistentes, porque existen, o persisten más allá de la ejecución de la aplicación.

Ese almacenamiento secundario de datos que acabamos de mencionar, habitualmente suele consistir en una base de datos relacional, si bien, a veces, hay otros métodos de almacenamiento, y por tanto, métodos de acceso a esos datos. De conocer esos tipos de almacenamiento y cómo acceder a ellos es de lo que trata este módulo.

En esta unidad inicial, vas a ver una panorámica de los diversos métodos de **persistencia** que encontramos en el mercado.

3. ACCESO A DATOS.

Hay diversas estrategias de acceso a datos para gestionar la persistencia de los datos:

- Mediante **ficheros**.
- **Bases de datos**, que pueden ser:
  - Relacionales,
  - Orientadas a objetos,
  - Objeto-relacionales.
  - Bases de Datos NoSQL
- **Mapeo objeto relacional** (**ORM**).
- Bases de datos **XML** (eXtensible Markup Language).
- **Componentes**.

Al principio, en los primeros tiempos de la informática, los datos se guardaban en ficheros convencionales. Con el tiempo, y la experiencia de trabajar con dichos ficheros, se observaron los inconvenientes de los ficheros, y para intentar solucionar los inconvenientes que se observaron surgieron las **bases de datos, que entre otras ventajas permitían frente a los ficheros:**

- Eliminar el problema de la información redundante.
- Eliminar información inconsistente.
- Globalizar o centralizar la información.
- Garantizar el mantenimiento de la integridad en la información. Únicamente se almacena la información correcta.
- Independencia de datos. La independencia de datos implica una separación entre programas y datos, es decir, se pueden hacer cambios en la información que contiene la base de datos, o tener acceso a la base de datos de diferente manera, sin tener que hacer cambios en las aplicaciones o en los programas.

4. QUE METODO USAR PARA EL ACCESO A DATOS.

Posteriormente, con la aparición y expansión de la programación orientada a objetos, empezaron a surgir las Bases de datos orientadas a objetos, y también se ampliaron algunas bases de datos relacionales, añadiéndoles extensiones de orientación a objetos.

Entonces ¿qué método de acceso a datos es mejor? ¿Cuál deberías utilizar en la próxima aplicación que construyas?

Pues..., no hay una respuesta fácil para esas preguntas, no se puede afirmar que haya un método que sea el mejor de manera absoluta. Más bien, la cuestión es tener claro qué tipo de aplicación hay que construir y, según eso, estudiar qué tipo de sistema de almacenamiento será mejor usar: si una base de datos orientada a objetos, o una base de datos XML, etc.

Conociendo el funcionamiento de las diferentes alternativas podemos comparar sus prestaciones al problema de la persistencia concreto que se nos presente. Cada una de las tecnologías tiene su propio origen y filosofía para alcanzar el mismo fin y, por esta razón, no es fácil analizar sus ventajas y desventajas frente a las demás alternativas.

Por poner un ejemplo, lo más sencillo posible: si voy a crear una base de datos para guardar mi colección de vídeos, probablemente no me va a interesar utilizar una base de datos Oracle, sino un producto mucho más barato, y sencillo de instalar y mantener.

5. FICHEROS.

En las antiguas aplicaciones informáticas, antes de que surgieran las bases de datos, la información se guardaba en ficheros.

Así, por ejemplo, una aplicación que guardaba los datos de personas, almacenaba dichos datos en un fichero convencional cuyo contenido bien podía ser este:

Antonio Pérez Pérez 30 C/ Morales nº 11 Madrid Madrid
Feliciano Gómez Sander 25 C/ Terreros nº 121 Vitoria Vitoria
Arturo Bueno Hernández 46 C/ Cocoliso nº 43 Murcia Murcia

...

Esto tenía como efecto, que el programador de las aplicaciones que usaran ese fichero, tuviera que construir el programa conociendo detalladamente las posiciones de los datos, para saber desde qué posición hasta qué otra posición, se guardaba el nombre y apellidos, etc. Además, tendría que controlar si se guardan filas de datos duplicadas, y así un montón de inconvenientes. Por eso, cuando surgieron las bases de datos, se empezó a dejar de usar los ficheros convencionales.

Pero bien es cierto, que aún en las más modernas aplicaciones, a veces necesitamos un simple fichero para guardar información, como por ejemplo un fichero de configuración, o un fichero log. Es decir, no siempre nos hace falta una base de datos para almacenar la información.

En Java, como en otros lenguajes de programación, hay diversas clases para el manejo de ficheros, pues, como hemos dicho, a veces son muy útiles. Para guardar poca información, es mejor usarlos que usar otro método.

5.1. USO DE FICHEROS EN LA ACTUALIDAD.

Hay que tener en cuenta, que los ficheros en sí, para grabar la información del modo que poníamos como ejemplo anteriormente, en el ejemplo de las personas, ya no se usan. Pero, sí que se usan ficheros que guardan datos siguiendo un patrón o **estructura bien definida**, en otros métodos de almacenamiento, como por ejemplo en ficheros y en bases de datos **XML**.

De especial interés y uso cada vez más extendido, son los ficheros XML. Éstos son archivos de texto que por consiguiente no necesitan un software propietario para ser interpretados, como ocurre con la mayoría de los archivos binarios, y tienen normalmente la extensión xml.

También debemos tener en cuenta, que las bases de datos relativamente modernas, como son las **bases de datos XML**, guardan sus datos empleando ficheros xml.

Por eso, en muchas ocasiones se recurre a utilizar este tipo de soluciones, el uso de ficheros en vez de bases de datos, y en particular de ficheros XML cuando se necesita intercambiar información a través de varias plataformas de hardware o de software, o de varias aplicaciones. A veces se exporta de una base de datos a ficheros XML para trasladar la información a otra base de datos que leerá esos ficheros XML.

Por esta razón se emplea XML en tecnologías de comunicación como, por ejemplo, en **WML** (lenguaje de formato inalámbrico) y **WAP** (protocolo de aplicaciones inalámbricas).

La fácil estructuración de la información en los ficheros XML ha permitido que surjan muchas librerías de conversión de la información almacenada a otros formatos como a **PDF**, texto, hojas de cálculo, etc. Hay muchos productos propietarios y de código abierto.

6. BASE DE DATOS.

Tal y como habéis podido estudiar en el módulo de Bases de Datos del curso anterior, un sistema de bases de datos es:

- Un sistema de información orientado hacia los datos, que pretende recuperar y almacenar la información de manera eficiente y cómoda.
- Surge en un intento de resolver las dificultades del procesamiento tradicional de datos, teniendo en cuenta que los datos suelen ser independientes de las aplicaciones.

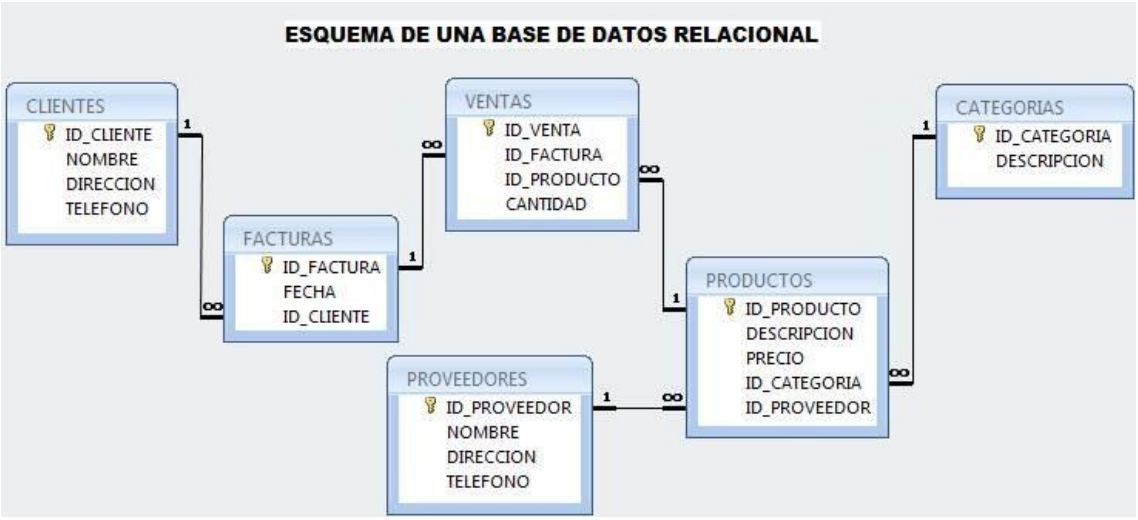
6.1. INTRODUCCION.

Las ventajas que aportan los sistemas de bases de datos respecto a los sistemas de archivos convencionales son:

- **Independencia** de los datos respecto de los procedimientos. El usuario tiene una visión abstracta de los datos, sin necesidad de ningún conocimiento sobre la implementación de los ficheros de datos, índices, etc. Esto supone un gran ahorro en los costes de programación, de forma que la modificación de la estructura de los datos no suponga un cambio en los programas y viceversa. Sin ella, el mantenimiento de la base de datos ocuparía el 50% de los recursos humanos dedicados al desarrollo de cualquier aplicación.
- Disminución de las **redundancias** y en consecuencia,
- Disminución de la posibilidad de que se produzca **inconsistencia** de datos.
- Mayor **integridad de los datos.**
- Mayor **disponibilidad** de los datos.
- Mayor **seguridad** de los datos.
- Mayor **privacidad** de los datos.
- Mayor **eficiencia** en la recogida, codificación y entrada en el sistema.
- Lo que se suele denominar interfaz con el pasado y futuro: una base de datos debe estar abierta a reconocer información organizada físicamente por otro software.
- **Compartición** de los datos. Los datos deben poder ser accedidos por varios usuarios simultáneamente, teniendo previstos procedimientos para salvaguardar la integridad de los mismos.

Podemos afirmar generalizando, que se usa un sistema de ficheros convencional **cuando la cantidad de datos a guardar es tan reducida** que no justifica las desventajas del uso de los sistemas de bases de datos. Por ejemplo, para guardar los datos del resultado de la instalación de un programa, usamos un fichero de texto, no se guardan los datos en una base de datos.

6.2. BASES DE DATOS RELACIONALES



Probablemente habrás cursado ya el módulo de bases de datos, si es así, e incluso si no lo es, quizás sepas que el **modelo de bases de datos relacional** fue propuesto en 1969 por **Edgar Codd**.

El propósito del modelo relacional es proporcionar un método declarativo para especificar datos y consultas. Así, en el diseño de la base de datos establecemos qué información contendrá dicha base de datos, luego recuperaremos la información que queramos, y dejamos al software del sistema gestor de la base de datos que se ocupe de: describir las estructuras de datos para almacenarlos, y gestionar los procedimientos de recuperación para obtener las consultas deseadas.

Las bases de datos relacionales son adecuadas para manejar grandes cantidades de datos, compartir datos entre programas, realizar búsquedas rápidas, etc. Pero tienen como desventaja fundamental que no presentan un buen modelo de las relaciones entre los datos, ya que todo se representa como tablas bidimensionales, o sea, en filas y columnas.

Podemos decir de las bases de datos relacionales que:

- Están muy extendidas.
- Son muy robustas.
- Permiten interoperabilidad entre aplicaciones.
- Permiten una forma de compartir datos entre aplicaciones.
- Son el común denominador de muchos sistemas y tecnologías. Una base de datos puede ser utilizada en programas realizados en Java, o en C++, etc.

Otros modelos tradicionales

Podemos encontrar tres etapas:

- Pre-Relacional: Jerarquico y en Red.
- Relacional
- Post-Relacional: Orientado a Objetos, NoSQL etc.

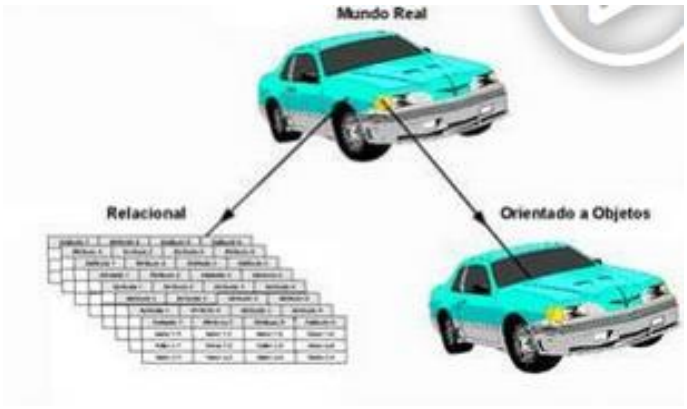
Cuando se estudian las bases de datos relacionales, se suelen mencionar también los modelos **jerárquico** y **en red**. Algunos sistemas antiguos utilizan todavía estas arquitecturas, en centros de datos donde se manejan grandes volúmenes de información y la complejidad de los sistemas hace prohibitivo el coste que supondría migrar a sistemas que utilizaran otro modelo como el relacional.

El modelo relacional fue el primer modelo de bases de datos definido de manera formal. Posteriormente, se formularon modelos informales para describir bases de datos jerárquicas (el modelo jerárquico) y bases de datos en red (el modelo en red). Ambas, las bases de datos jerárquicas y en red existían antes que las bases de datos relacionales, **pero fueron descritas como modelo después** de que el modelo relacional fuera definido.

6.3. BASES ORIENTADAS A OBJETOS

El origen de las Bases de datos orientadas a objetos (en adelante: BDOO) se debe básicamente a las siguientes razones:

- La existencia de problemas al representar cierta información y modelar ciertos aspectos del mundo real. Los modelos clásicos permiten representar gran cantidad de datos, pero las operaciones y representaciones que se pueden realizar sobre ellos son bastante simples.
- Pasar del modelo de objetos al modelo relacional, para almacenar la información, genera dificultades que en el caso de las BDOO no surgen, ya que el modelo es el mismo. Es decir, los datos de los programas escritos en lenguaje orientado a objetos se pueden almacenar directamente, sin conversión alguna, en las BDOO.



Los sistemas de bases de datos orientadas a objetos soportan un **modelo de objetos puro**, ya que no están basados en extensiones de otros modelos más clásicos como el relacional.

Por ello, una característica general es que **el lenguaje de programación y el esquema de la base de datos utilizan las mismas definiciones de tipos**.

El acceso a los datos puede ser más rápido con las bases de datos orientadas a objetos que con las bases de datos tradicionales porque no hay necesidad de utilizar las uniones o joins, que si se necesitan en los esquemas relacionales tabulares. Esto se debe a que un objeto puede recuperarse directamente sin una búsqueda, simplemente siguiendo punteros.

Cada vez más, las necesidades de las aplicaciones actuales con respecto a las bases de datos son:

- Soporte para objetos complejos y datos multimedia.
- Jerarquías de objetos o tipos y herencia.
- Gestión de versiones.
- Modelos extensibles mediante tipos de datos definidos por el usuario.
- Integración de los datos con sus procedimientos asociados.
- Manipulación navegacional (en vez de declarativa) y de conjunto de registros.

- Interconexión e interoperabilidad.

Como ejemplos de Sistemas Gestores de Bases de datos Orientados a Objetos podemos señalar:

- Jasmine
- ObjectStore
- GemStone

6.3.1. COMPARATIVA ENTRE BASES DE DATOS RELACIONALES Y ORIENTADAS A OBJETOS.

En numerosos bancos de pruebas realizados para comparar los sistemas de bases de datos orientados a objetos (ODBMS) y los sistemas de bases de datos relacionales (RDBMS), se ha mostrado que los ODBMS pueden ser superiores para ciertos tipos de tareas.

La explicación a esto es que muchas operaciones se realizan utilizando interfaces navegacionales más que declarativos, y el **acceso a datos navegacional** es normalmente implementado muy eficientemente.

Un buen argumento a favor de las bases de datos orientadas a objetos es la transparencia (no ensucia la construcción de una aplicación). El desarrollador así se debe preocupar de los objetos de su aplicación, en lugar de cómo los debe almacenar y recuperar de un medio físico.

Podemos decir que las **ventajas de un SGBDOO frente a las relacionales** son:

- Permiten mayor **capacidad de modelado**. El modelado de datos orientado a objetos permite modelar el mundo real de una manera mucho más fiel. Esto se debe a:
  - un objeto permite encapsular tanto un estado como un comportamiento
  - un objeto puede almacenar todas las relaciones que tenga con otros objetos
  - los objetos pueden agruparse para formar objetos complejos (herencia).
- **Extensibilidad**, debido a que:
  - Se pueden construir nuevos tipos de datos a partir de los ya existentes.
  - Podemos agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
  - Tenemos reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.
- Disposición de un **lenguaje de consulta más expresivo**. El acceso navegacional desde un objeto al siguiente es la forma más común de acceso a datos en un sistema gestor orientado a objetos. Mientras que SQL utiliza el acceso declarativo. El acceso navegacional es más adecuado para gestionar operaciones tales como consultas recursivas, etc.

- **Adaptación a aplicaciones** avanzadas de base de datos. Hay muchas áreas en las que las bases de datos relacionales no han tenido excesivo éxito, como es el caso de en sistemas de diseño CAD, CASE, sistemas multimedia, etc. en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.
- **Prestaciones**. Los sistemas gestores de bases de datos orientadas a objetos proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales. Aunque hay autores, que han argumentado que los bancos de prueba, usados en dichas pruebas, están dirigidos a aplicaciones de ingeniería donde los SGBDOO son más adecuados. También está demostrado, que los SGBDR tienen un rendimiento mejor que los SGBDOO en las aplicaciones tradicionales de bases de datos como el procesamiento de transacciones en línea (OLTP).
- **Reglas de acceso**. En las bases de datos relacionales, a los atributos se accede y se modifican a través de operadores relacionales predefinidos. En las orientadas a objetos se procede mediante las interfaces que se creen a tal efecto de las clases. Desde este punto de vista, los sistemas orientados a objetos dan independencia a cada objeto que el sistema relacional no permite.
- **Clave**. En el modelo relacional, las claves primarias generalmente tienen una forma representable en texto, sin embargo los objetos no necesitan una representación visible del identificador

6.3.2. DESVENTAJAS DE LAS BASES DE DATOS RELACIONALES FRENTE A LAS ORIENTADAS A OBJETOS.

Como desventajas o puntos débiles de las BBDDOO respecto a las relacionales podemos mencionar:

- La **reticencia del mercado**, tanto para desarrolladores como usuarios, a este tipo de bases de datos.
- **Carencia de un modelo de datos universal**. No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen de una base teórica. El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.
- **Carencia de experiencia**. Al ser una tecnología relativamente nueva, todavía no se dispone del nivel de experiencia del que se dispone para los sistemas relacionales.
- Panorama actual. Tanto los sistemas gestores de bases de datos **como los sistemas gestores de bases de datos objeto-relacionales están muy extendidos**. SQL es un estándar aprobado y ODBC y JDBC son estándares de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.
- **Dificultades en optimización**. La optimización de consultas necesita realizar una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.

6.4. BASES DE DATOS OBJETO RELACIONALES.

Entendemos Base de Datos Objeto Relacional (BDOR), una base de datos que ha evolucionado desde el modelo relacional a otro extendido que incorpora conceptos del paradigma orientado a objetos. Por tanto, **un Sistema de Gestión Objeto-Relacional (SGBDOR) contiene ambas tecnologías: relacional y de objetos.**

En una Base de Datos Objeto Relacional el diseñador puede crear sus propios tipos de datos y crear métodos para esos tipos de datos.

Por ejemplo, podríamos definir un tipo de la siguiente manera:

```
CREATE TYPE persona_t AS OBJECT (
nif VARCHAR2(9),
nombre VARCHAR2(30),
direccion VARCHAR2(40),
telefono VARCHAR2(15),
fecha_nac DATE);
```

Por poner un ejemplo, si consideramos la base de datos Oracle, debido a los requerimientos de las nuevas aplicaciones, el sistema de gestión de bases de datos relacional desde versión 8i fue extendido con conceptos del modelo de bases de datos orientadas a objetos. De esta manera, aunque las estructuras de datos que se utilizan para almacenar la información siguen siendo tablas, los diseñadores pueden utilizar muchos de los mecanismos de orientación a objetos para definir y acceder a los datos. Se reconoce el concepto de objetos, de tal manera que un objeto tiene un tipo, se almacena en cierta fila de cierta tabla y tiene un identificador único (OID). Estos identificadores se pueden utilizar para referenciar a otros objetos y así representar relaciones de asociación y de agregación.

La ventaja de este tipo de base de datos es que los usuarios pueden pasar sus aplicaciones actuales sobre bases de datos relaciones este nuevo modelo sin tener que reescribirlas. Más tarde, se pueden ir adaptando las aplicaciones y bases de datos para que utilicen las funciones orientadas a objetos.

Con las Bases de Datos Objeto-Relacional se amplía el modelo relacional destacando las siguientes aportaciones:

- Se aumentan la variedad en los tipos de datos,
    - se pueden crear nuevos tipos de datos que permitan construir aplicaciones complejas con una gran riqueza de dominios. Se soportan tipos complejos como: registros, conjuntos, referencias, listas, pilas, colas y vectores.

- Hay extensiones en el control de la Semántica de datos Objeto-Relacionales:
  - Se pueden crear procedimientos almacenados y funciones que tengan un código en algún lenguaje de programación, como por ejemplo: SQL, Java, C, etc.
- Se pueden compartir varias librerías de clases ya existentes, esto es lo que conocemos como reusabilidad.

Como productos comerciales de bases de Datos Objeto-Relacional podemos destacar:

- **DB2** Universal Database de IBM (International Business Machines).
- Universal Server de Informix (ahora de IBM),
- INGRES II, de Computer Associates.
- **ORACLE** de Oracle Corporation,
- SyBASE

Como productos de código abierto, destacamos **PostGreSQL**.

6.5. BASES DE DATOS NOSQL

Las Bases de Datos NoSQL se interpretan a veces como "lo que no es sql" y a veces como "no solo sql". Es todo aquello que **No** se adhiere al modelo relacional y las características que dan por sentadas las Bases de Datos Relacionales. En un sentido más estricto se refieren a aquellas Bases de Datos **NO basadas** en:

- Almacenamiento basado en tablas.
- SQL como lenguaje de consulta.
- Soporte para restricciones de integridad y transacciones.

Las características principales de las Bases de Datos NoSQL:

- El almacenamiento no está basado en tablas.
- No se manejan con SQL.
- Prima la disponibilidad pero no la consistencia. Es decir, lo que prima es disponibilidad básica, estado flexible y consistencia en el tiempo.

7. ACCESO A BASES DE DATOS MEDIANTE CONECTORES.

Para gestionar la información de las bases de datos hay unos estándares que facilitan a los programas informáticos manipular la información almacenada en ellas.

En Java existe un API basado en estos estándares que permite al desarrollar aplicaciones, que se pueda acceder a bases de datos relacionales: JDBC (Java Database Connectivity, conectividad de bases de datos de Java).

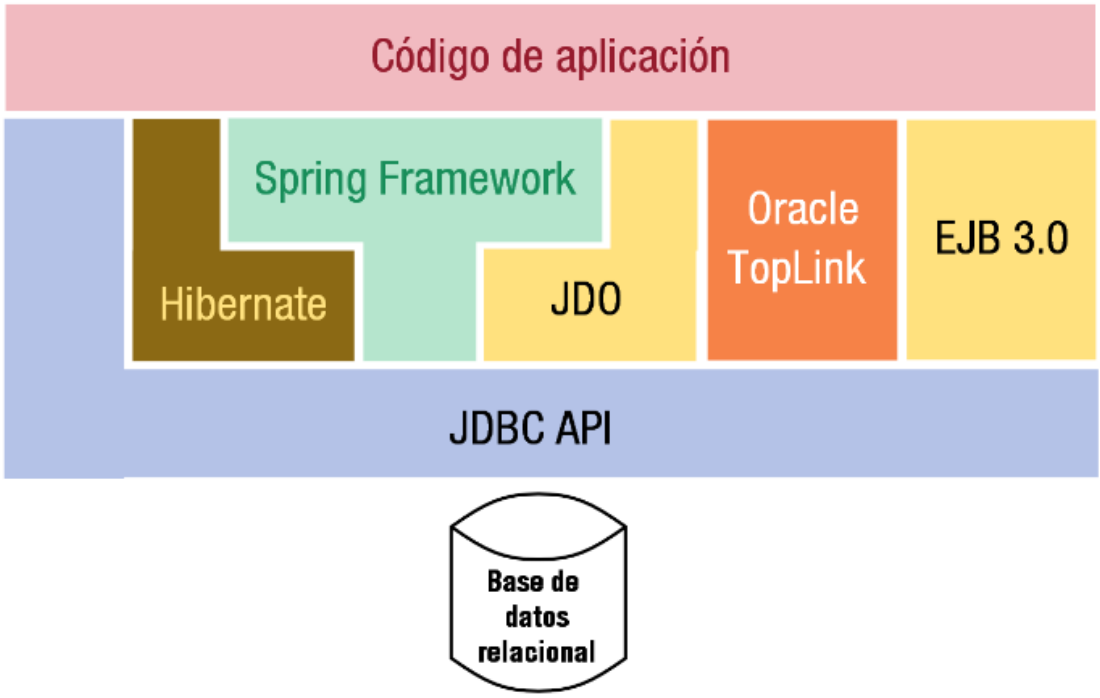
La mayoría de las aplicaciones importantes de una empresa están respaldadas por una arquitectura normalizada y optimizada de bases de datos relacionales. Tradicionalmente, dichas aplicaciones están basadas en sentencias SQL con las cuales se gestionan todos los datos que manejan.

Este modelo continúa teniendo una gran importancia estratégica y es la base para el continuo crecimiento del mapeo Objeto-Relacional (O/R) y está asociado a los mecanismos de persistencia.

Un **driver JDBC** es un **componente** software **que posibilita a una aplicación Java interaccionar con una base de datos.**

El **API JDBC** define interfaces y clases para escribir aplicaciones de bases de datos en Java realizando conexiones de base de datos.

Mediante JDBC el programador puede enviar sentencias SQL, y PL/SQL a una base de datos relacional. JDBC permite embeber SQL dentro de código Java.



La ilustración muestra una representación de los diferentes mecanismos de mapeo O/R y cómo se relacionan con el código de la aplicación y con los recursos de datos relacionados. Se observa claramente la función crítica que desempeña el driver JDBC puesto que está situado en la base de cada uno de los marcos de trabajo.

La ventaja de usar conectores JDBC es que independiza de la base de datos que utilice.

No obstante hay un trabajo de traducción para mapear los campos devueltos por cada consulta a la colección de objetos correspondiente. Y hay que trabajar las sentencias de actualización, inserción y eliminación para cada uno de los campos. Esto constituye una razón para tratar de buscar alternativas menos costosas en tiempo de desarrollo.

8. MAPEO OBJETO-RELACIONAL (ORM)

En los inicios de la programación, los programas se desarrollaban con la llamada programación imperativa. De hecho, y como sabes, la programación orientada a objetos surgió unos años después.

Con la expansión de la programación orientada a objetos, se da la circunstancia de que las bases de datos relacionales se siguen utilizando hoy en día de manera masiva, por lo que muchas aplicaciones se desarrollan con POO pero los datos, no se almacenan en bases de datos orientadas a objetos, sino en las bases de datos relacionales.

El sistema más extendido en las empresas hoy en día para guardar la información de sus aplicaciones es el uso de una **base de datos relacional**. Tradicionalmente, dichas aplicaciones están basadas en sentencias SQL con las cuales se gestionan todos los datos que manejan. Este sistema es la base para el continuo crecimiento del mapeo Objeto-Relacional (O/R) y está asociado a los mecanismos de persistencia.

Cuando se programan sistemas orientados a objetos, utilizando una base de datos relacional, los programadores invierten gran cantidad de tiempo en desarrollar los objetos persistentes, o sea, convertir los objetos del lenguaje de programación a registros de la base de datos. Igualmente, también pasan bastante tiempo implementando la operación inversa, es decir, convirtiendo los registros en objetos.

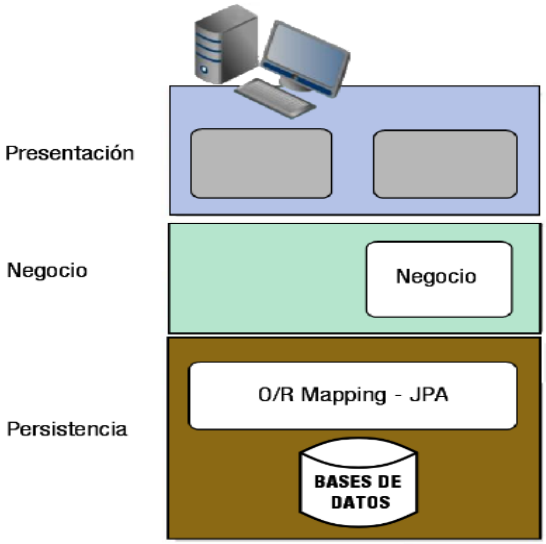
El **mapeo objeto-relacional** (Object-Relational Mapping, o ORM) consisten en una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el sistema utilizado en una base de datos relacional.

Cuando se trabajan con programación orientada a objetos y con bases de datos relacionales, es fácil observar que estos son dos paradigmas diferentes. El modelo relacional trata con relaciones y conjuntos, es de naturaleza matemática. Por el contrario, el paradigma orientado a objetos trata con objetos, atributos y asociaciones de unos con otros.

Cuando se requiere almacenar la información de los objetos utilizando una base de datos relacional se comprueba que hay un problema de compatibilidad entre estos dos paradigmas, el llamado **desfase objeto-relacional**.

Por ello, para ahorrar trabajo al programador, se puede utilizar un framework que se encargue de realizar estas tareas de modo transparente, de modo que el programador no tenga por qué usar JDBC ni SQL y la gestión del acceso a base de datos esté centralizada en un componente único permitiendo su reutilización.

8.1. CAPA DE PERSISTENCIA Y FRAMEWORK DE MAPEO.



La **capa de persistencia** de una aplicación es la pieza que permite almacenar, recuperar, actualizar y eliminar el estado de los objetos que necesitan persistir en un sistema gestor de datos.

En el caso del mapeo objeto-relacional, un **ORM** es una capa que permite relacionar objetos con un modelo de datos relacional, ocultando todo el mecanismo de conexión al **motor de base de datos,** y también no teniendo que escribir las sentencias SQL necesarias para la gestión de los datos.

La **capa de persistencia** traduce entre los dos modelos de datos: desde objetos a registros y desde registros a objetos. Así, si el programa quiere grabar un objeto, entonces llama al **motor de persistencia**, el motor de persistencia traduce el objeto a registros y llama a la base de datos para que guarde estos registros.

De este modo el programa sólo ve que puede guardar y recuperar objetos, como si estuviera programado para una base de datos orientada a objetos. Y la base de datos sólo ve que guarda y recupera registros como si el programa estuviera dirigiéndose a ella de forma relacional.

Dispones de múltiples alternativas como desarrollador en Java cuando pretendas trabajar con mapeadores O/R. Hay tres comunidades que están implicadas en el mundo de la persistencia O/R de Java de forma activa:

- **Organizaciones basadas en el estándar**: Las alternativas más importantes basadas en el estándar, son **EJB** 3.0 y **JDO**.
- **Comunidades código abierto (open source)**: Las comunidades open source incluyen importantes tecnologías, entre ellas **Hibernate** y el framework **Spring**.
- **Grupos comerciales:** entre las implementaciones comerciales se puede resaltar **TopLink**.

Cada uno de los mecanismos de mapeo O/R tiene una dependencia particular en el conector JDBC para poder comunicarse con la base de datos de una forma eficiente. Si el conector JDBC que participa en la comunicación no es óptimo, la posible gran

eficiencia de cualquier framework quedará debilitada. Por tanto, seleccionar el driver JDBC que mejor se adapte a la aplicación es esencial a la hora de construir un sistema eficiente en el que intervenga un mecanismo de mapeo O/R.

9. BASES DE DATOS XML.

Se define **XML** como: **eXtensible Markup Language** (lenguaje de marcado extensible). Es una especificación del World Wide Web Consortium (**W3C**).

Debido a las limitaciones de la tecnología de bases de datos relacionales están surgiendo nuevas clases de bases de datos como son las bases de datos XML.

Estas bases de datos almacenan los datos estructurados como XML sin la necesidad de traducir los datos a una estructura relacional o de objeto.

Podemos distinguir entre:

- **Bases de datos nativas XML.** Consiste en un modelo lógico para documentos XML. El Sistema Gestor de Base de Datos correspondiente almacena y recupera documentos de acuerdo a dicho modelo.
    - Productos ejemplo de esa filosofía son: eXist, Apache Xindice, Berkeley dbXML....
- **Bases de datos compatibles con XML** (XML-enabled): son bases de datos, generalmente objeto-relacionales, que admiten entradas de datos en forma de XML y proporcionan salidas en este mismo formato.
    - Podemos citar productos como: Oracle, DB2 (Viper),...

Las bases de datos XML nativas permiten trabajar con **XQL** (eXtensible Query Language), el cuál sirve un propósito similar a SQL en una base de datos relacional.

El trabajo con bases de datos XML nativas involucra dos pasos básicos:

- Describir los datos mediante Definiciones de Tipos de Datos (Document Type Definitions, DTD) o esquemas XML y
- Definir un nuevo esquema de base de datos XML nativa o Mapa de Datos a usar para almacenar y obtener datos.

10. DESARROLLO DE COMPONENTES

La expansión, en los últimos años, de Internet y el comercio electrónico ha llevado a la necesidad de adoptar nuevas tecnologías y nuevos requisitos de desarrollo en las aplicaciones informáticas.

Al principio de la informática, los ordenadores eran caros y la mano de obra de los programadores asequible. Con el paso del tiempo los ordenadores son baratos y la mano de obra de los programadores suele ser cara. Además, los requisitos de las aplicaciones informáticas son cada vez más complejos. Por ello, aparecieron las técnicas orientadas a

objetos, para, entre otras cosas, intentar programar de manera que el código que se desarrolle pueda ser reutilizable.

Un componente es una unidad de software que realiza una función bien definida y posee una interfaz bien definida. Posee interfaces especificadas contractualmente, pudiendo ser desplegado independientemente y puede interaccionar con otros componentes para formar un sistema.

Un interfaz es un punto de acceso a los componentes: permite a los clientes acceder a los servicios proporcionados por un componente.

La idea de dividir u organizar en trozos el software, o sea, dividirlo en componentes surge para reducir la complejidad del software. Así se permite la reutilización y se acelera el proceso de ensamblaje del software.

Los creadores de componentes pueden especializarse creando objetos cada vez más complejos y de mayor calidad.

La interoperabilidad entre componentes de distintos fabricantes:

- Aumenta la competencia,
- Reduce los costes y,
- Facilita la construcción de estándares.

Por tanto, así el software se hace cada vez más rápido, de mejor calidad y a menor coste incluso de mantenimiento.

Un componente software también debe especificar sus necesidades para funcionar correctamente, lo que se denominan las dependencias de contexto:

- Interfaces requeridas
- Entorno de ejecución: máquina necesaria, sistema operativo a utilizar, etc.

En el entorno Java, contamos con los **JavaBeans**.