2.- 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.
* **Mapeo objeto relacional**(ORM).
* Bases de datos XML(eXtensible Markup Language).
* **Componentes**.

2.1.- Qué estrategia o método de acceso a datos usar.

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.

3.- 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.

3.1.- Uso ficheros en la actualidad.

Actualmente se usan ficheros que guardan datos siguiendo un patrón o **estructura bien definida**, como por ejemplo en ficheros y en bases de datos XML.

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

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 ficheros XML en vez de bases de datos 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.

3.2.- Características de los ficheros

* Un fichero es un conjunto de bits almacenados en un dispositivo, no se va a perder cuando se apague el dispositivo, se pueden intercambiar entre dispositivos (copiar/mover).
* Tiene una gran característica, los datos almacenados no se eliminan al apagar el dispositivo, por lo que tienen un almacenamiento persistente a diferencia de la RAM.
* Tienen 3 secciones:
  + RUTA (PATH): es dónde se encuentra ubicado el fichero
  + NOMBRE: Cómo se llama el fichero.
  + EXTENSIÓN: Qué tipo de fichero es.
* Un fichero debe tener un nombre único en su ruta, pero pueden existir dos fichero con el mismo nombre y extensión en rutas diferentes.

En cuanto a las extensiones:

* La extensión determina que tipo de fichero es y por lo tanto como debe tratarlo el sistema. Dependiendo del tipo de fichero su información estará ordenada de un tipo diferente, el tratamiento de los bits será distinto en cada tipo de fichero.
* Es necesario tener el software adecuado para poder leerlo.
* Las extensiones suelen tener 3 letras, pero nos podemos encontrar extensiones de 2 y 4 letras.

¿Cómo es un fichero?

* Cada desarrollador diseña como el crear conveniente la estructura del fichero, siempre teniendo en cuenta que está formado por bloques de bytes en los que se guardan la información deseada.
* Estos bloques se denominan **registros** y gracias a la extensión y el software correspondiente el S.O. puede interpretar y mostrar de forma correcta el fichero.

Ficheros en Java

Para gestionar los ficheros vamos a utilizar dos librerías incluídas en el JDK

* Java.io 🡪 Java Input Output
* Java.nio 🡪 Java Non-blocking Input Output

4.- Bases de datos.

Una Base de datos es una colección de datos relacionados lógicamente entre sí, con una descripción y definición comunes y que están estructurados de una determinada manera. Es un **conjunto estructurado de datos** que representa entidades y sus interrelaciones, almacenados con la mínima redundancia y posibilitando el acceso a ellos eficientemente por parte de varias aplicaciones y usuarios.

Básicamente es:

* Un [*sistema de información*](https://www.adistanciafparagon.es/pluginfile.php/35606/mod_resource/content/3/AD01_Contenidos_Web/4_bases_de_datos.html#t2f6afd06-6d9f-96a7-5f8a-a245646c336f)*(1)*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.

(1) Cualquier sistema o subsistema de equipo computacional interconectado que se utiliza para obtener, almacenar, manipular, administrar, mover, controlar, desplegar, intercambiar, transmitir o recibir datos.

4.1.- Introducción.

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** (1).
* 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.

(1) Consiste en que los datos almacenados en una base datos sean correctos y completos. Cuando el contenido de una base de datos se modifica con sentencias SQL como INSERT, DELETE o UPDATE, la integridad de los datos almacenados puede perderse de muchas maneras diferentes. Pueden añadirse datos no válidos a la base de datos, tales como un pedido que especifica un producto que ya no existe en la base de datos.

4.2.- Bases de datos relacionales.

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.

4.3.- Bases de datos 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 oJOINS, 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.

4.4.- Comparativa entre bases de datos relacionales y orientadas a objetos.

En diversas pruebas realizadas 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 el acceso a datos navegacional es normalmente implementado muy eficientemente, que es el que usan muchas operaciones.

Las bases de datos orientadas a objetos es que no ensucia la construcción de una aplicación. El desarrollador se debe preocupar de los objetos de su aplicación, en lugar de como 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](https://www.adistanciafparagon.es/pluginfile.php/35606/mod_resource/content/3/AD01_Contenidos_Web/44_comparativa_entre_bases_de_datos_relacionales_y_orientadas_a_objetos.html#ta8d752c9-0782-7ae9-9f3e-223a7ae22a08) 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.
* **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.

Desventajas de las bases de datos orientadas a objetos frente a las relacionales.

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.

4.5.- 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*](https://www.adistanciafparagon.es/pluginfile.php/35606/mod_resource/content/3/AD01_Contenidos_Web/45_bases_de_datos_objetorelacionales.html#t6b73be5f-c499-1806-c391-3d760a195f1d) *(1)* 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.

(1) Un procedimiento almacenado (stored procedure en inglés) es un programa que se almacena físicamente en una base de datos.

5.- Acceso a bases de datos mediante conectores.

Para la gestión de las bases de datos existen estándares que facilitan la manipulación de la información almacenada en ellas.

En Java existe un API basado en estos estándares que permite a los desarrolladores, acceder a bases de datos relacionales: JDBC (Java Database Connectivity, conectividad de bases de datos de Java).

La mayoría de las aplicaciones están respaldadas por una arquitectura normalizada y optimizada de bases de datos relacionales. Dichas aplicaciones, mayormente, están basadas en lenguaje SQL con el que se gestionan los datos que manejan.

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.

El uso de conectores JDBC es independiente de la base de datos que utilice.

6.- Mapeo objeto relacional (ORM).

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 (1)* 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.

(1) Es un software que suele constar de módulos de software concretos, y que permite organizar y desarrollar un proyecto software. Normalmente incluye soporte de programas, librerías y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

6.1.- 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,
* Comunidades código abierto (open source) y
* Grupos comerciales.

Las comunidades open source incluyen importantes tecnologías, entre ellas Hibernate y el framework Spring.

Las alternativas más importantes basadas en el estándar, son EJB 3.0 y JDO.

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.