Centro de Estudios de Postgrado

Máster en Profesorado ESO, Bach., FP y EI

UNIVERSIDAD DE JAÉN

Centro de Estudios de Postgrado

Trabajo Fin de Máster

BASES DE DATOS ORIENTADAS A OBJETOS. PERSISTENCIA DE OBJETOS Y LENGUAJES

Alumno/a: **García Fernández, Carmen María**

Tutor/a: Prof. D. Carmen Martínez Cruz
Prof. D. Pedro González García
Dpto: Informática

Junio, 2016

RESUMEN

En otros tiempos donde Internet no estaba a la orden del día, los desarrolladores de aplicaciones no tenían porqué preocuparse para decidir la tecnología de base de datos que iban a utilizar.

La tecnología Relacional, que presenta un sencillo lenguaje de consulta, era la principal herramienta y el estándar de las bases de datos. Sólo unas pocas aplicaciones especializadas utilizaban Bases de Datos de Objetos, las cuales eran percibidas como lentas e incómodas.

Sin embargo, con la llegada de la Web, la tecnología de objetos se ha convertido en la corriente dominante. Tecnologías de desarrollo Web como Java, se orientan **objetos**, por lo que muchos desarrolladores vieron que tenía sentido asociarlas a las Bases de Datos Orientadas a Objetos. Asimismo, los objetos facilitan el desarrollo rápido puesto que son modulares y proporcionan un método más eficiente e intuitivo de dar forma al mundo real.

Uno de los problemas que se encuentran los programadores en los lenguajes orientados a objetos es la necesidad de almacenar y recuperar los datos de una forma eficiente y sencilla. Si se utilizan modelos relacionales se pierde parte de las ventajas de trabajar con Orientación a Objetos, puesto se debe hacer una transformación entre objetos y datos relacionales (y viceversa). Esta diferencia de paradigmas fue la que impulsó a crear los Sistemas Gestores de Bases de Datos Orientados a Objetos (SGBDOO) a fin de evitar las dificultades del paso del modelo de objetos al modelo relacional buscando la eficiencia y la sencillez.

Sin embargo, actualmente, los desarrolladores de aplicaciones de alto rendimiento combinan ambas tecnologías relacional y objetos, consiguiendo así una base de datos rápida, escalable, fiable y post-relacional que funcione sin problemas tanto como con objetos como con SQL.

En el Trabajo Final de Máster enfocamos la temática a la Bases de Datos de Objetos Puras, Persistencia de objetos y lenguajes. Para ello, llevamos a cabo un **Estudio Epistemológico** donde analizamos la evolución de las bases de datos, el estado del arte y tendencias futuras de las mismas. Por último, en **Proyección Didáctica** adaptamos el tema al módulo de **Programación** impartido en el CFGS Desarrollo de Aplicaciones Multiplataforma.

PALABRAS CLAVE

Internet; Bases de Datos; Tecnología Relacional; Tecnología de Objetos; Web; Objetos; Bases de Datos Orientadas a Objetos; Sistemas Gestores de Bases de Datos Orientados a Objetos; Estudio Epistemológico; Proyección Didáctica; Programación.

ABSTRACT

In other times where the Internet was not the order of the day, application developers did not need to worry to decide database technology that would be used. Relational technology that presents a simple query language was the main tool and standard databases. Only a few specialized applications using Object Databases, which were perceived as slow and cumbersome.

However, with the advent of the Web, object technology has become the mainstream. Web development technologies such as Java, objects are oriented, so many developers saw that it made sense to associate the object database. Also, objects facilitate rapid development since they are modular and provide a more efficient and intuitive method of shaping the real world.

One of the problems that programmers found in object-oriented languages is the need to store and retrieve data efficiently and easily. If relational models are used part of the advantages of working with object orientation, position should make a transformation between objects and relational data (and vice versa) is lost. This difference in paradigms was prompted to create the database management systems Object-Oriented Data (OODBMS) in order to avoid the difficulties of the passage of the object model to the relational model seeking efficiency and simplicity.

However, today, developers of high-performance applications combine both relational and object technologies, achieving a base fast, scalable, reliable and post-relational database that runs smoothly with objects as well as with SQL.

In the Final Master Work focus the thematic to the Object Database Pure, Persistence of objects and languages. To this end, we conduct an epistemological study where we analyze the evolution of the databases, the state of the art and future trends of them. Finally, through a Projection Teaching adapt the theme to Programming module taught in the CFGS Multiplatform Application Development .

KEYWORDS

Internet; Databases; Relational Technology; Object technology ; Web; Objects; Object database ; Bases Management Systems Object Oriented Data; Epistemological Study; Projection Teaching; Programming.

1. INTRODUCCIÓN

En el presente documento vamos a desarrollar el Trabajo Final del Máster en Profesorado de Educación Secundaria Obligatoria, Bachillerato, Formación Profesional y Enseñanza de Idiomas, impartido por la Universidad de Jaén durante el curso lectivo 2015/2016.

La temática del Trabajo Final de Máster lleva por título: **Persistencia de objetos: Bases de datos Orientadas a Objetos. Persistencia en Java y C#.** El tema ha sido extraído de la Orden EDU/3138/2011, de 15 de noviembre (actualmente sin vigencia), que contiene los temarios de las especialidades de los Cuerpos de Profesores de Enseñanza Secundaria y Profesores Técnicos de Formación Profesional**.** En concreto nuestro tema pertenece al segundo punto del apartado 30, **Persistencia de Objetos**, del temario de oposiciones de Profesores Técnicos de Formación Profesional en Sistemas y Aplicaciones Informáticas.

Por otro lado, el tema puede ser integrado en el plan de estudios de los Ciclos Formativos de Grado Superior en Desarrollo de Aplicaciones Multiplataforma y en Desarrollo de Aplicaciones Web, en concreto, en el módulo de **Programación** impartido en el primer curso. No obstante, para la proyección didáctica hemos seleccionado el CFGS Desarrollo de Aplicaciones Multiplataforma, puesto que es el ofertado en el centro de educación secundaria seleccionado, **IES Saladillo (Algeciras)**.

El presente documento ha sido dividido en dos partes bien diferenciadas. La primera sección está dedicada al **Estudio Epistemológico** sobre la temática en cuestión, a través de una aproximación a la evolución de las bases de datos, estudio de las bases de datos orientadas a objetos y el estado del arte de las mismas, así como las tendencias futuras de las bases de datos. En la segunda parte, haremos una **Proyección Didáctica** del tema adaptándolo al proceso de Enseñanza-Aprendizaje del ciclo en el que se encuadra. En esta última sección, vamos a llevar a cabo un análisis del contexto interno y externo del centro, estudiar el perfil del alumnado, profundizaremos en los elementos curriculares propios de la temáticas y de la etapa, junto con la aportación de propuestas innovadoras de actividades a llevar a cabo en el aula.

2. ESTUDIO EPISTEMOLÓGICO

2.1 INTRODUCCIÓN

Las aplicaciones tradicionales de las bases de datos consisten en tareas de procesamiento de datos, como la gestión bancaria y de nóminas, con tipos de datos relativamente sencillos, que se adaptan bien al modelo relacional. A medida que los sistemas de bases de datos se fueron aplicando a un rango más amplio de aplicaciones, como el diseño asistido por computadora y los sistemas de información geográfica, las limitaciones impuestas por el modelo relacional se convirtieron en un obstáculo. La solución fue la introducción de bases de datos basadas en objetos, que permiten trabajar con tipos de datos complejos.

Esta sección del Trabajo Final de Máster estará dedicada al estudio de las generaciones de las bases de datos, los conceptos fundamentales de las bases de datos orientadas a objetos, el estado del arte de las mismas, así como las tendencias futuras de las bases de datos.

2.2 EVOLUCIÓN DE LAS BASES DE DATOS

Básicamente, cuando hablamos de base de datos nos referimos a una colección de datos que forma parte de un determinado contexto. Existen diversos tipos de bases de datos, pero nos vamos a dedicar a las bases de datos digitales que son las usadas por los sistemas informáticos.

Con el paso del tiempo, el modo en que las personas organizan sus datos digitalmente ha ido cambiando. El primer tipo de organización fue a través de archivos de acceso secuencial. Este tipo de almacenamiento presentaba el inconveniente de que hasta encontrar la información necesaria necesitaba recorrer línea por línea. Sin embargo, las bases de datos permiten un acceso directo al usar estructuras de organización.

Principalmente, distinguimos tres generaciones de Bases de Datos.

2.2.1 PRIMERA GENERACIÓN

En los años 60, aparecieron las primeras bases de datos que usaban los modelos jerárquicos y de red, los cuales usaban registros formados por campos y proporcionaban una organización lógica de los datos en árboles y grafos.

En el **modelo jerárquico** los datos son almacenados por niveles. Se dispone un nodo padre o raíz que se enlaza con varios hijos. Las relaciones son unidireccionales de hijos a padres. Ejemplo: En una Universidad existe una relación de docentes hacia el departamento al que pertenecen, pero no viceversa.

El **modelo en red** es similar al jerárquico, con la diferencia de que en lugar de tener un padre se pueden tener varios, pudiendo tener relaciones muchos-a-muchos., Este modelo exige mantener dos relaciones físicas: de docentes hacia departamentos, así como de departamentos hacia docentes.

Surgen los primeros productos de bases de datos, los cuales utilizaban lenguajes procedimentales, sin independencia física/lógica y presentaban una flexibilidad muy limitada. Ejemplos: como DBOM, IMS, IDS, total o IDS.

Además, en esta primera época se incorpora a los Sistemas Gestores de Bases de Datos facilidades de comunicación de datos, posibilitando de esta forma que múltiples usuarios pudieran acceder a la base de datos a través de una red de comunicación [4].

2.2.2 SEGUNDA GENERACIÓN

Edgar Codd propuso las **bases de datos relacionales**, en los años 70. El principal elemento de almacenamiento de este modelo son las tablas, permite relacionar una tabla consigo misma, así como la representación de relaciones muchos-a-muchos, sin necesidad de duplicar la relación, y de relaciones entre más de dos tablas.

Los productos relacionales presentan una flexibilidad, eficiencia, seguridad en los entornos transaccionales, mayor independencia física/lógica, y usan lenguajes declarativos. Los SGB Relacionales presentan mayor facilidad de uso y una base teórica más sólida [4].

2.2.3 TERCERA GENERACIÓN

En 1980, surgen las Bases de Datos Distribuidas, se estandariza el lenguaje SQL, se produce la separación entre apariencia lógica e implementación física, y se ponen de manifiesto las **Bases de Datos Orientadas a Objetos**. Los Sistemas Gestores de Bases de Datos se basaban principalmente en modelos de objetos, que pueden ser bien puros o híbrido (relacional + orientación a objetos).

En la década de los 90, la arquitectura cliente/servidor se presenta en dos capas. Aparecen los primeros productos de bases de objeto, siendo los modelos de referencia ISO y ANSI, surgen el consorcio ODMG (estándares de Orientación a Objetos), SQL 92 y SQL 99 . Asimismo, el lenguaje XML presenta un auge, y podemos hablar de los modelos **XML** puros o como una capa sobre el modelo relacional [4].

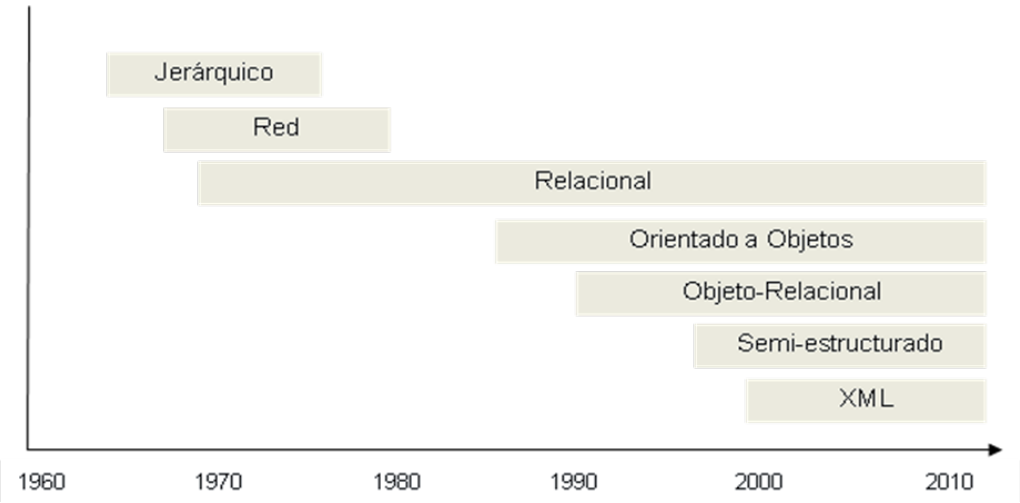


Figura 1: Evolución cronológica de los modelos de bases de datos [10]

Con respecto a los Sistemas Gestores de Bases de Datos podemos mencionar aquellos relacionados con la orientación a objetos, temporalidad, seguridad, paralelismos, bases de datos multidimensionales o tecnología Grid.

A comienzos del siglo XXI, podemos referirnos ya a una arquitectura cliente/servidor en tres capas, modelo objeto-relacional, bases de datos móviles, bases de datos multimedia, base de datos XML, SQL/MM, SQL 2003 y bases de datos grid [4].

2.3 MOTIVACIÓN DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

En este apartado analizaremos los motivos que hicieron que surgieran las Bases de Datos Orientadas a Objetos

En los años 80 y 90, la industria del software experimenta un importante crecimiento. Comienza a cobrar importancia la **Programación Orientada a Objetos**, junto a la **Programación Estructurada**.

Asimismo, con el objetivo de representar una realidad cada vez más compleja en los sistemas de información, junto a los **tipos de datos primitivos** y a las estructuras de datos sencillas, aparecen nuevos niveles de organización más complejos, **tipos de datos abstractos** (pilas, árbol…), clases, interfaces, etc.

Los sistemas de información comienzan a ser más complejos manejando una importante cantidad de conceptos del mundo real, y requiriendo desarrollos rápidos y seguros. La información a ser almacenada comienza también a crecer en complejidad, almacenándose **gráficos**, **vídeo**, **audio**, **diagramas**, **huellas digitales**, etc., además de los tradicionales números y texto.

Los **sistemas relacionales** han tenido éxito en el desarrollo de bases de datos requeridas por aplicaciones de bases de datos tradicionales. Sin embargo, el modelo relacional presenta ciertas carencias a la hora de diseñar e implementar aplicaciones de bases de datos más complejas, como bases de datos para aplicaciones de ingeniería del diseño y fabricación (CAD/CAM/CIM), tratamiento de documentos, sistemas de información geográfica, diseño asistido por computadora, experimentos científicos, telecomunicaciones, multimedia, , etc.

Por este motivo, debido a la creciente complejidad en los requisitos de las nuevas aplicaciones y sistemas software, así como del almacenamiento de tipos de información cada vez más complejos, surge la necesidad de proporcionar otras alternativas que puedan adaptarse a escenarios más complejos y subsanar las carencias de las bases de datos relacionales.

Por tanto, los Sistemas de Gestión de Bases de Objetos surgen por dos razones fundamentales:

1. Necesidad de Persistencia de datos, por necesidades de los Lenguajes de Programación Orientados a Objetos.
2. Limitaciones de las bases de datos relacionales en el manejo de datos complejos.

2.3.1 NECESIDADES DE LOS LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

En primer lugar, comentar que los modelos de datos y estructuras de datos de los lenguajes de programación están desacoplados, hecho que dificulta el acceso a la información de la base de datos desde programas escritos en lenguajes como C++ o Java. Además, las diferencias existentes entre los sistemas de tipos de las bases de datos y de los lenguajes de programación hace más complicado el almacenamiento y la recuperación de los datos. Por lo que, se hace necesario minimizar dichas discrepancias.

Por otro lado, el hecho de tener que expresar el acceso a las bases de datos a través del lenguaje SQL (diferente del lenguaje de programación) también dificulta el trabajo del programador. Por tanto, la solución radica en que ambos sigan el mismo paradigma, es decir, la solución está en los Sistemas de Gestión de Bases de Datos Puras. Dicha aproximación engloba la gestión de los datos y su comportamiento, e incluye los lenguajes ODL y OQL.

Los **lenguajes de programación persistentes** son lenguajes de programación que ya existen y que añaden persistencia además de otras características de las bases de datos, haciendo uso del sistema de tipos nativo del lenguaje de programación. El concepto de **sistemas de bases de datos orientadas a objetos** hace referencia a los SBD que soportan tipos orientados a objetos y permiten el acceso directo a los datos desde los LPOO usando para ello el sistema de tipos nativo del lenguaje [9].

Las Bases de Datos proporcionan a la Orientación de Objetos la Persistencia, lo que supone:

- Independencia de datos/programa.
- Almacenamiento eficiente y gestión de datos en memoria masiva.
- Lenguaje de consulta eficiente y de alto nivel.
- Gestión de transacciones, garantizando el acceso concurrente, seguridad y recuperación ante fallos.
- Control de integridad.

2.3.2 LIMITACIONES DE LAS BASES DE DATOS RELACIONALES

En realidad, las bases de datos relacionales no son apropiadas para aquellas aplicaciones que manejen estructuras de datos complejas. Los motivos son los siguientes:

- Tienen estructuras simples.
- Escasa riqueza semántica.
- No soportan tipos definidos por el usuario.
- No soportan recursividad.
- Falta de procedimientos/disparadores.
- No permite herencia.

Por otro lado, si convertimos objetos y relaciones al modelo relacional puede ocurrir que:

- Un objeto se descomponga en un gran número de tablas, originando errores.
- Se necesite un gran número de joins para su recuperación, empeorando de esta forma el rendimiento [9].

En la siguiente figura podemos observar cómo se realiza el manejo de los datos en Bases de Datos Orientadas a Objetos (BDOO) y Bases de Datos Relacional (BDR).

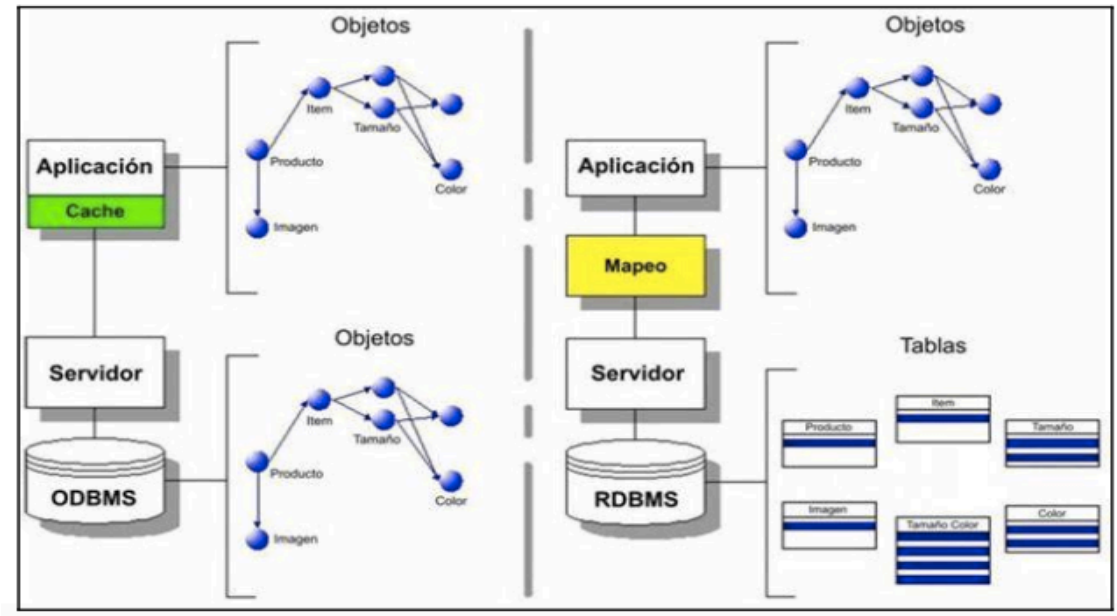


Figura 2: SGBOO y SGBDR [13]

2.4. ESTADO DEL ARTE

Como Estado del Arte analizaremos cuál es el presente de las bases de datos en general y de Bases de Datos Orientadas a Objetos que es el tema que nos ocupa en el Trabajo Final de Máster.

Además, de las necesidades de los Lenguajes de Programación Orientados a Objetos y de las limitaciones que presentan las Bases de Datos Relacionales, las aplicaciones actuales también requieren de necesidades que no pueden ser cubiertas por las bases de datos relacionales y sí por las orientadas a objetos, como son:

- Interconexión e interoperabilidad.
- Soportar objetos complejos y datos multimedia.
- Identificadores únicos.
- Soporte de referencias e interrelaciones.
- Manipulación navegacional y de conjunto de registros.
- Jerarquías de objetos y herencia.

- Integrar los datos con sus procedimientos asociados.
- Modelos extensibles mediante tipos de datos definidos por el usuario.
- Gestionar versiones.
- Facilitar la evolución.
- Transacciones de larga duración.

Por tanto, las Bases de Datos Orientadas a Objetos fueron propuestas con el objetivo de satisfacer las necesidades de ciertas aplicaciones para un modelo más rico y extensible, y de esta forma complementar a las bases de datos relacionarles.

El modelo de datos proporcionado por las Bases de Datos Orientadas a Objetos es equivalente al de los lenguajes de programación orientados a objetos, como por ejemplo, C++ y Java. Esto supone una gran ventaja, y es que al compartir el modelo de datos, las BDOO pueden ser integradas con el software usado para desarrollar aplicaciones de forma directa y prácticamente transparente [3].

Sin embargo, en las aplicaciones basadas en BDR es necesario usar dos lenguajes muy diferentes, uno para la aplicación y SQL para el acceso a la base de datos. Esto conlleva a realizar costosas inversiones entre los formatos usados en cada uno de los dos lenguajes. Además, la tendencia de la Programación Orientada a Objetos también se hizo presente introduciendo la técnica de programación **ORM (Object Relational Mapping)** destinada a convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. Esto crea una BDOO virtual, sobre la base de datos relacional. Sin embargo, las BDOO tienen el inconveniente de la ausencia de un lenguaje de consulta sencillo y rápido como SQL [4].

Los principales vendedores de SGBD Relacionales reconocieron en su día la necesidad de disponer de nuevas características para el modelado de datos complejo. Por tanto, las versiones más recientes de estos sistemas han ido incorporando muchas de las funcionalidades que en un principio se propusieron para las BDOO, dando lugar a las Bases de Datos Objeto-Relacionales (BDOR) [4].

Muchas de estas funcionalidades están incluidas en las últimas versiones del estándar SQL. Por tanto, la funcionalidades que hoy en día aportan la orientación orientada a objetos a las bases de datos son las siguientes [4]:

- Almacenar Grandes Objetos Binarios en tablas relacionales. Estos objetos son conocidos como "**blobs**", del inglés **Binary Large Objects**, y pueden ser imágenes, sonido o vídeo.
- **Herramientas de gestión de documentos**, ofreciendo soporte XML, operadores específicos para buscar texto, etc.
- Proporcionar otras **extensiones** para el soporte de **datos geográficos**.

Asimismo, actualmente, hablamos de **varios niveles** de soporte de la tecnología orientada a objetos en bases de datos [4]:

- **Librerías** para almacenar de forma persistente los objetos. Sin embargo, a veces no incorporan soporte transaccional o multiusuario.

- **Bases de Datos Objeto-Relacionales**, destinadas a aplicaciones que requieren usar algunos tipos de datos complejos en un entorno relacional, y están basadas en extensiones orientadas a objetos de SQL.
- **Bases de Datos Orientadas a Objetos Puras**, las cuales proporcionan una gestión de BD orientadas a objetos a todos los niveles, partiendo de la definición de datos hasta el lenguaje de consulta.

2.4.1 BASES DE DATOS ORIENTADAS A OBJETOS COMERCIALES

A continuación, enumeramos algunas Bases de Datos Orientadas a Objetos disponibles en mercado [3]:

- **Objetivity/DB**: Ofrece soporte para Java C++, Python y otros lenguajes. Ofrece un alto rendimiento y escalabilidad. No es un producto libre aunque ofrecen versiones de prueba durante un periodo determinado.
- **db40**: Base de Datos Open Source para JAVA y .NET. Se distribuye bajo licencia GPL.
- **Intersystems Cache®**: Ofrece soporte para Java, C++, .NET, etc. Manejan grandes volúmenes de información y ejecutan sentencias SQL de forma más rápida que algunas bases de datos relacionales. Todo esto con consumo mínimo de recursos y de hardware.
- **EyeDB**: Sistema Gestor de Bases de Datos Orientadas a Objetos basado en la especificación ODMG el cuál está desarrollado y soportado por la compañía francesa SYSRA. Proporciona un modelo avanzado de objetos (herencia, colecciones, arrays, métodos, triggers, constraints, etc.), un lenguaje de definición de objetos en ODMG ODL, lenguaje de manipulación y consulta de datos basado en ODMG OQL e interfaces de programación para C++ y Java. Se distribuye bajo la licencia GNU (*GNU lesser General Public License*). Se trata de software libre.

2.4.2 USO DE BASES DE DATOS ORIENTADAS A OBJETOS EN LA INDUSTRIA

En este apartado presentamos algunas empresas que actualmente utilizan los servicios de las Bases de Datos Orientadas a Objetos en diferentes ramas de la industria [8]:

- **PRONAUTIC**: Es una empresa española dedicada a la venta de accesorios náuticos y recambio de motor para embarcaciones. Ésta necesitaba una solución con la que automatizar sus procesos de negocio en las áreas de finanzas, pedidos y compras. Tenía el objetivo de mejorar sus sistemas de información llevando cabo un análisis detallado de las líneas más rentables, los mejores vendedores, los precios más óptimos y otros factores claves. Adquirió **Oracle e-Businness Suite Special Edition**.
- **XEROX GLOBAL SERVICES**: Empresa estadounidense que utiliza **SQL Server** para gestionar más de 7 millones de transacciones diarias con una disponibilidad de casi el 100%. Unos de los servicios más importantes de esta empresa es Xerox Office Service,

que lleva a cabo una gestión de activos y mantenimiento preventivo para grandes cantidades de impresoras, copiadoras, dispositivos multifunción y faxes. Al optimizar dicha gestión, Xerox reduce el coste total de impresión a sus clientes.

- **MACBA (Museu d´Art Contemporani de Barcelona)**: Abast Solutions implantó **ERP Oracle eBusiness Suite** en el museo para las distintas áreas funcionales (Finanzas, Compras y Contratación, Gestión de inventario, Proyectos, Gestión de Espacios, Gestión de Recursos Humanos, Gestión Documental, CRM e Indicadores de Gestión), con el objetivo de optimizar los procesos en las diferentes áreas de negocio, permitir obtener información contrastada en tiempo real, disponer de un sistema de información estable, escalable y con proyección de futuro y conseguir un sistema de fácil acceso desde cualquier lugar a través de internet.
- **NASDAQ**: Es el mercado electrónico más importante de EE.UU y continuamente está buscando la forma de dar un mejor servicio a sus miembros. **SQL Server** gestiona más de 100.000 consultas diaria usando Snapshot Isolation para resolver las peticiones en tiempo real sin ralentizar la base de datos.
- **Centro de Regulación Genómica**: Se trata de uno de los centros de investigación biomédica más importantes de Europa, el cual necesitaba unificar su información corporativa, mejorar el acceso de los empleados a recursos informativos y poder realizar análisis e informes detallados. Para conseguir esto se utilizó **Oracle Application Server Portal** y **Oracle E-Businness Suite**.

2.4.3 DATA WAREHOUSE

Sin embargo, en la actualidad, las bases de datos no pueden permitirse ser simples repositorios. Los datos carecen de importancia si no pueden ser analizados y aportan información relevante para la toma de decisiones. Así que, se trabaja con Almacenes de Datos, también conocido como **Arquitectura Data Warehouse**, lo cual que permite trabajar con diferentes estados de una base de datos a través del tiempo y es de ayuda para las posteriores tomas de decisiones [4].

Básicamente, **Data Warehouse** es una base de datos corporativa caracterizada por integrar y depurar información de distintas fuentes, para luego procesarla permitiendo su análisis desde diferentes perspectivas y con una gran velocidad de respuesta..

La ventaja principal de este tipo de bases de datos se encuentra en las estructuras en las que se almacena la información que pueden ser modelos de tablas en estrella, en copo de nieve, cubos relacionales, etc.. Este tipo de persistencia de la información es homogénea y fiable, y permite la consulta y el tratamiento jerarquizado de la misma.

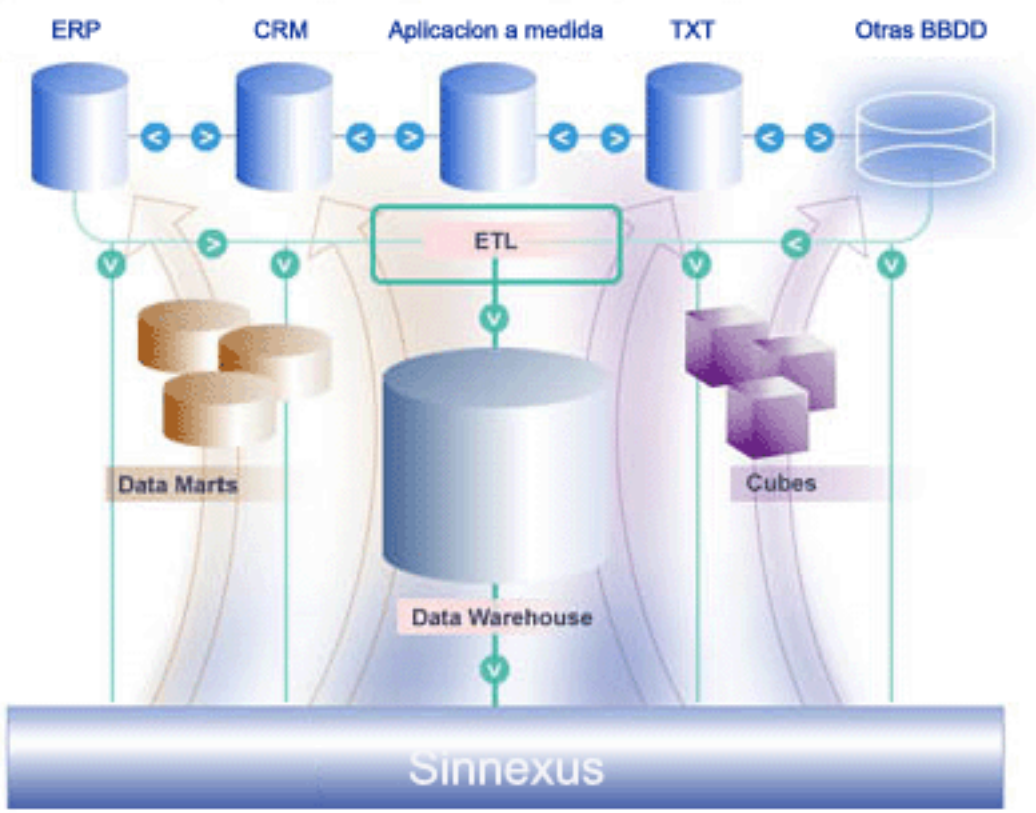


Figura 3: Visión de la Arquitectura Data Warehouse [4]

Algunas de las principales características de un datawarehouse son las siguientes [4]:

- Los datos almacenados se integran en una estructura consistente.
- Los datos son organizados por temáticas a fin de facilitar su acceso y comprensión por los usuarios finales.
- El tiempo está implícito en la información contenida en un datawarehouse. La información almacenada sirve, entre otros, para realizar análisis de tendencias.
- El almacén de información de un datawarehouse puede ser leído, pero no modificado, por lo que una actualización implica la incorporación de los últimos valores que tomaron las variables sin sufrir ningún tipo de acción lo que ya existía.
- Datawarehouse contiene **metadatos**, los cuales permiten saber la procedencia de la información, su periodicidad de refresco, su fiabilidad, forma de cálculo... etc. También permiten simplificar y automatizar la obtención de la información desde los sistemas operacionales a los sistemas informacionales.

En Septiembre de 2013, varias empresas de Ecuador fueron entrevistadas, como EMAC, ETAPA, SRI, entidades bancarias, empresas de desarrollo de software, etc. De esta investigación se dedujo que **ninguna empresa usaba Base de Datos Orientadas a Objetos**

Puras, que todas las empresas usaban Bases de Datos Relacionales y que el 60% había usado alguna vez herramienta de análisis de datos para Data Warehouse [4].

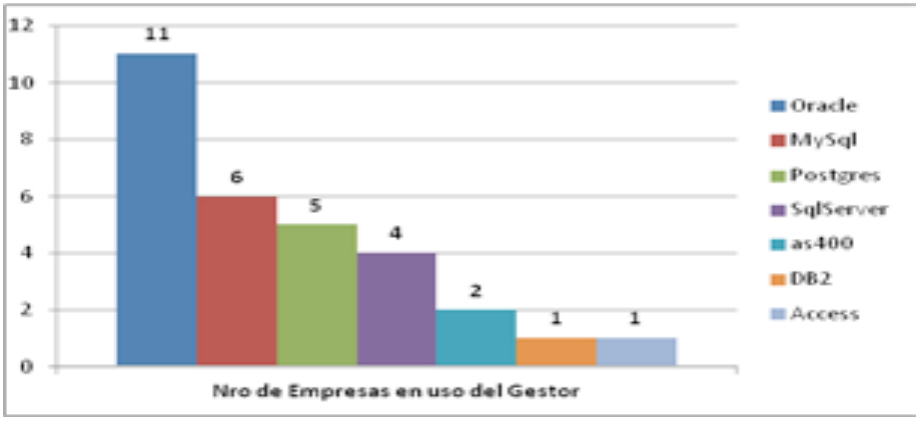


Figura 4: Distribución de Gestores de BD Relacionales usadas por las empresas [4]

La realidad es que en el mercado actual las Bases de Datos Orientadas a Objetos no han logrado encontrar una buena posición, sin embargo, sí que el modelo de Bases de Datos Objeto-Relacional ha ido poco a poco incorporándose en los Sistemas de Gestión de Bases de Datos más utilizados como **Oracle** o **PostgresSQL**.

Gracias a las funcionalidades que proporcionan los sistemas objeto-relacionales se han desarrollado nuevas aplicaciones con tipos de datos complejos como son las **Bases de Datos Geográficas**.

2.4.4 BASES DE DATOS GEOGRÁFICAS

Una Base de Datos Geográfica (BDG) es un conjunto de datos geográficos organizados con el objetivo de analizar y gestionar el territorio dentro de aplicaciones de **Sistemas de Información Geográfica**. En adición, las Bases de Datos Geográficas se usan con el objetivo de implementar servicios geográficos relacionados con las **Infraestructuras de Datos Espaciales**, y su contenido principal en los procesos de producción cartográficos [11].

El punto clave de una Base de Datos Geográfica es el **modelo de datos**, es decir, la formalización conceptual de las entidades geográficas del mundo real. La implementación del modelo debe ser tal que facilite la explotación y optimización del almacenamiento con el objetivo de alcanzar el mejor rendimiento en las consultas.

Por otro lado, la **Información Geográfica** constituye uno de productos más solicitados por los ciudadanos a las administraciones públicas. Así pues, el aumento de esta demanda ha provocado modificaciones en la captura, almacenamiento, tratamiento, mantenimiento y actualización de la información geográfica y espacial.

La captura masiva y selectiva permite clasificar la Información Geográfica por temas. Asimismo, el almacenamiento de gran volumen de Información Gráfica requiere la puesta en marcha de servidores y Bases de Datos Geográficas específicas para el tratamiento espacial de los datos. El mantenimiento y la actualización son dos parámetros demandados por el usuario final y están destinados a mejorar la calidad geométrica, semántica y topológica de la Información Geográfica. Las Bases de Datos Geográficas favorecen interoperabilidad con otras bases de datos y permiten la multirrepresentación a través de procesos de generalización cartográfica.

En la actualidad, El **Instituto Geográfico Nacional** desarrolla diferentes proyectos de I+D+I aplicados a la producción cartográfica, creando y manteniendo Bases de Datos Geográficas que permiten generar cartografía en función de los cambios acontecidos en el mundo real y así romper con el clásico concepto de hoja de serie cartográfica [11]..

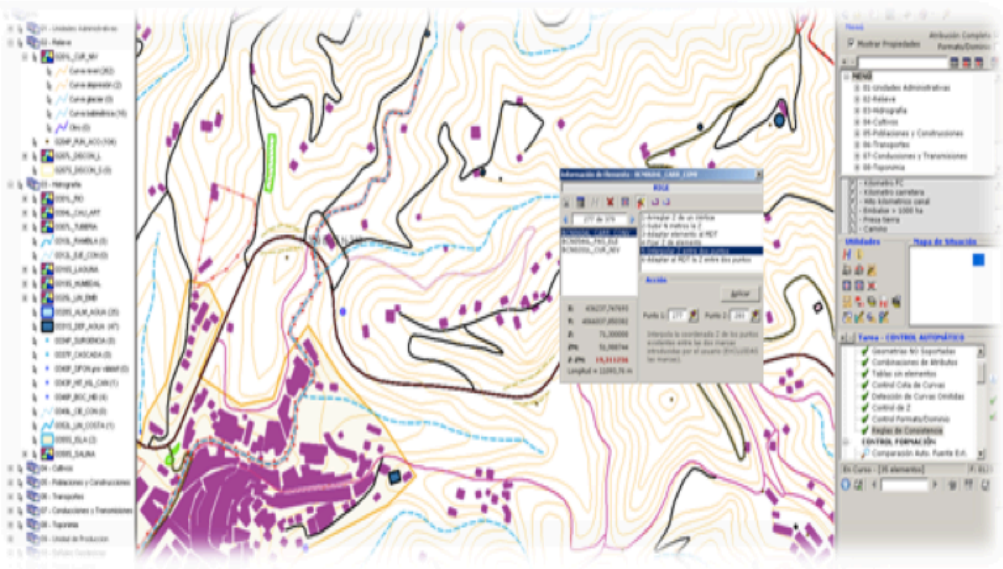


Figura 5: Vista entorno de captura en Base Topográfica Nacional de España [11]

Como ejemplo reciente, podemos mencionar el desarrollo de una aplicación web para dispositivos móviles diseñado con componentes de sistemas de información geográficos, como herramienta de difusión de alertas de amenazas y riesgos naturales para el municipio de La Vega (Granada-Bolivia). Este sistema de información pretende acercar a los habitantes del municipio a la administración municipal, haciéndolos partícipes de la gestión del riesgo, y de esta forma hacer que los dueños de predios y población en general tengan a su disposición la información pertinente de los riesgos [12].

Lo que se pretende es que, una vez identificadas las amenazas y riesgos municipales, la administración municipal o el comité de administración del riesgo, pueda generar alertas tempranas a los ciudadanos que utilicen la herramienta en sus dispositivos móviles, acercando el ciudadano al municipio y haciéndolo partícipe de la dinámica territorial.

2.5 DESARROLLO DE LOS CONCEPTOS

2.5.1 INTRODUCCIÓN A LAS BASES DE DATOS OO

Como ya indicábamos anteriormente los Sistemas Gestores de Bases de Datos Orientados a Objetos (SGBDOO) surgen debido a la falta de capacidad semántica del modelo relacional para soportar aplicaciones complejas, como las de ingeniería, sistemas basados en el conocimiento, tratamiento de documentos, multimedia, gestión de redes, etc. Este tipo de aplicaciones requieren modelar, de forma directa, objetos e interrelaciones complejos almacenar información no estructurada, gestionar diferentes tipos de transacciones, controlar de forma exhaustiva componentes y estructuras, además de manejar versiones y configuraciones.

Asimismo, los SGBDOO rompen la barrera que nos encontramos en los SGBD Relacionales entre los datos que son almacenados en la base de datos y los procesos que se encuentran en las aplicaciones desarrolladas mediante el lenguaje de datos asociado al SGBD, como por ejemplo SQL, embebido en un lenguaje de programación. Los Sistemas Gestores de Bases de Objetos gestionan objetos en los que se encuentran encapsulados los datos y las operaciones que actúan sobre ellos.

En la actualidad, parte del código que hasta no hace mucho se encontraba solo en las aplicaciones, ahora puede almacenarse en la propia Base de Datos. Esta tendencia ya se puso de manifiesto en versiones de productos relacionales que permitían almacenar disparadores (triggers), reglas, procedimientos, etc.

Otra ventaja de los Sistemas Gestores de Bases de Objetos es la reducción de la distancia entre el modelo conceptual de datos expresado en E/R o UML, y el modelo lógico que se diseña para implementar la base de datos. Y es que el paradigma de la orientación a objetos proporciona un único modelo que se encuentra implementado en el SGBO y al que pueden acceder directamente las aplicaciones.

2.5.2 CARACTERÍSTICAS DE LAS BASES DE DATOS OO

Las características de las Bases de Datos Orientadas a Objetos son análogas a los lenguajes de programación orientados a Objetos. Algunas de las características más relevantes de una BDOO son las siguientes [3]:

- Se entienden como un sistema de modelado del mundo real. Cada entidad del mundo será un objeto en la base de datos.
- Los objetos tienen un identificador único que los identifica y los diferencia del resto de objetos del mismo tipo. Esto implica por ejemplo, que cada vez que se quiera modificar un objeto se tenga que recuperar de la base de datos, modificar y almacenar nuevamente. Esta operación es totalmente diferente en los SGBDR.
- Pueden existir objetos con la misma información pero con diferente identidad. Es más, cuando se modifican los valores de los atributos, el objeto sigue siendo el mismo.

- Es posible almacenar objetos complejos en la base de datos sin que por ello haya que realizar operaciones especiales sobre la base de datos.
- El concepto de herencia se mantiene incluso en la base de datos.
- Es el usuario el que modela los objetos de la base de datos y etiqueta los atributos y métodos que son visibles en la interfaz del objeto y cuáles no.
- El SGBDOO se encarga de acceder a los miembros de los objetos sin necesidad de escribir métodos para acceder a ellos.
- Acceso rápido a los datos dado que no es necesario realizar *joins* de tablas.
- Algunos SGBDOO permiten un control de versiones.
- Implantación de conceptos del modelo orientado a objetos como (polimorfismo, sobrecarga, sobrescritura, etc.)

Por tanto, dadas estas características, las Bases de Datos Orientadas a Objetos se hacen más apropiadas cuando tenemos una gran cantidad de tipos de datos diferentes, objetos con comportamientos avanzados o con un gran número de relaciones entre ellos [3].

2.5.3 VENTAJAS/INCONVENIENTES DE BASES DE DATOS OO

Entre las **ventajas** que obtenemos al trabajar con las Bases de Datos Orientadas Objetos se encuentran las siguientes [3]:

- No tener que reensamblar los objetos cada vez que se accede a la base de datos. El resultado de las consultas son objetos, por lo que la velocidad de procesamiento aumenta.
- Cuando cambia un objeto, la forma de actualizarlos en la base de datos es simplemente almacenándolo. Esta acción suele ser una secuencia simple de comandos en el código.
- La reutilización, una de las características de los lenguajes de programación orientados a objetos, se mantiene, con lo que se mejoran los costes de desarrollo.
- El acceso a la información a través de objetos en una aplicación desarrollada como tal es más natural que hacerlo a través de tablas y filas.
- El control de acceso y concurrencia se facilita enormemente dado que se puede bloquear el acceso a ciertos objetos incluso en una jerarquía completa de objetos.
- Estos sistemas funcionan de forma eficiente en entornos cliente/servidor y arquitecturas distribuidas.
- No es necesario redefinir las relaciones entre objetos, puesto que ya existen en el modelo de objetos y la base de datos se encarga de hacer persistente lo ya diseñado.

Por otra parte, las **limitaciones** de las Bases de Datos Orientadas a Objetos serán las siguientes [3]:

- En las bases de datos existe el lenguaje SQL que es un estándar bastante asentado y fuerte. En las bases de datos orientadas a objetos esto no ocurre.
- La estructura de las bases de datos relacionales es simple y fácil de entender, al contrario que la de las bases de datos orientadas a objetos.

- Existen aplicaciones que aunque están escritas con lenguajes orientados a objetos, en ciertas ocasiones es más eficiente almacenar los datos en bases de datos relacionales debido a las consultas que se van a realizar sobre ellas.
- Se reduce la velocidad de acceso debido a que la Base de Datos Orientada a Objetos tiene que tener en cuenta las relaciones de herencia entre clases.
- Aunque se gana en simplicidad en el manejo de la base de datos desde los lenguajes de programación orientados a objetos, cuando se trata de realizar consultas complejas resulta más adecuado una base de datos relacional accedida mediante SQL.

2.5.4 MODELO DE OBJETOS

Para los Sistemas Gestores de Bases de Objetos no ha existido un único modelo como ocurre con el modelo relacional difundido por el Dr. Codd, sino que cada fabricante ha adoptado un modelo diferente. No obstante, todos los autores coinciden en la idea de integrar los dos aspectos de los sistemas de información que tradicionalmente se venían analizando de forma separada: **datos** y **procesos**.

Según el paradigma de la orientación a objetos, un sistema es un conjunto de objetos que se comunican entre sí mediante mensajes. A nivel conceptual, un objeto es una entidad percibida en el sistema, mientras que a nivel de implementación, un objeto se corresponde con un encapsulamiento de un conjunto de operaciones (servicios) y de un estado que recuerda el efecto de los servicios.

El **encapsulamiento** es un concepto abstracto que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa para interactuar con el objeto [1].

Por otro lado, la **ocultación** de la información permite modificar los aspectos privados de un objeto sin que ello afecte al resto de objetos que interactúan con él, siempre que se conserve la misma interfaz [1].

Un objeto se describe por los siguientes elementos:

- **Propiedades**, atributos o estructura del objeto.
- **Servicios**, que proporcionan el comportamiento del objeto.
- **Estado**, que viene determinado por los valores que toman sus atributos.

Todo objeto tiene un identificador de objeto que lo identifica unívocamente, denominado **OID** del inglés Object IDentifier. Además, existen los tipos de objeto que definen la estructura y el comportamiento de éstos que comparten características comunes.

En un Sistema Orientado a Objetos, encontraremos objetos y los valores que pueden tomar los atributos de los objetos. En algunos entornos como Smaltalk no se distingue entre objetos y valores, puesto que se considera que todo es un objeto.

En la orientación a objetos un tipo de objeto puede implementarse de distintas maneras. Una clase es la implementación de un tipo de objeto.

Por otro lado, además de las interrelaciones, existen dos tipos más de asociaciones las cuales ya están incluidas en el modelo E/R:

- **Generalización**: Los tipos y clases se organizan en jerarquías de super/subtipos (super/subclases) que presentan herencia y que corresponden al concepto "es-un", en las que los subtipos o subclases heredan lo servicios o atributos de sus ancestros. Ejemplo: Profesor puede definirse como subtipo de Persona.
- **Agregación**: Permite construir objetos compuestos que corresponde al concepto de "parte-de". Tecla forma parte de ordenador.

A través del envío de mensajes, los objetos interactúan entre sí. Los objetos solicitan la prestación de servicios y entregan los resultados que se obtienen cuando se lleva a cabo un determinado servicio. Por lo tanto, enviar mensajes consistirá en invocar servicios de los otros objetos.

El **polimorfismo** es la capacidad de que un mensaje sea interpretado de maneras diferentes, dependiendo del objeto que lo reciba. Existen dos tipos de polimorfismo fundamentalmente [1]:

- **De subclase**: Cuando un servicio que está definido en una clase se redefine en alguna de sus subclases manteniendo el mismo nombre.
- **De sobrecarga**: Utiliza el mismo nombre para servicios diferentes, no situados en una jerarquía de generalización.

Lo ideal es que la vinculación o binding entre el nombre de un servicio y su implementación se establezca en tiempo de ejecución (vinculación dinámica o tardía). Con este aprovechamos la ventaja que ofrece el polimorfismo. Sin embargo, lo ideal es que la determinación de tipos se haga en tiempo de compilación para evitar errores sintácticos en tiempo de ejecución.

Por último, nos queda hablar del concepto **genericidad** que es la capacidad de definir tipos/clases parametrizables o genéricos. Podemos considerar un tipo genérico como una plantilla que nos permite construir tipos [1].

2.5.5 CARACTERÍSTICAS DE LOS SISTEMAS GESTORES DE BASES DE OBJETOS

Como ya hemos visto, las características de los Sistemas Gestores de Bases de Objetos proceden de la unión entre las bases de datos y la orientación a objetos.

A continuación, analizamos las principales características de un SGBO maximal, tomándolo como modelo de referencia abstracto para que resulte útil para exponer este tipo de sistemas e incluso como base de comparación entre productos.

- **CARACTERÍSTICAS PROPIAS DE LA ORIENTACIÓN A OBJETOS**

Básicamente, los Sistemas Gestores de Bases de Objetos incorporan conceptos básicos del paradigma de la orientación a objetos. Sin embargo, existen otras características

propias de los Sistemas Orientados a Objeto que se incorporan a los Sistemas de Gestión de Bases de Objeto como son [1]:

- **Extensibilidad**: Permite la definición de nuevas clases y modificar las ya existentes de manera dinámica. Esta característica es imprescindible para las aplicaciones CAD/CAM, CASE, etc.
- **Bibliotecas de clases**: Estas bibliotecas permiten la definición de los elementos con un alto nivel de funcionalidad, que se pueden integrar en la base de datos. El inconveniente es que suelen tener una funcionalidad limitada y las bibliotecas de terceros pueden ser incompatibles con el modelo soportado por el SGBO, al no existir un estándar adoptado.

• CARACTERÍSTICAS DE LOS SGBD

En esta sección vamos a exponer las cuáles son las principales características de los Sistemas Gestores de Bases de Datos que deben ser incluidas en los Sistemas Gestores de Objetos [1]:

- **Persistencia**: Todo elemento del sistema independientemente de su tipo debe poder ser persistente por sí mismo. Por un lado, el sistema sólo opera con objetos persistentes y se ofrece algún tipo de operación que permite hacer persistente a los objetos.
- **Modelo de Objetos**: El modelo de objetos de un SGBO es más **r**ealista con respecto al modelo relacional puesto que los objetos representan entidades del mundo real dando una sensación más precisa del problema que un conjunto de tablas. Por otro lado, el modelo de objetos es más flexible que el relacional puesto que posee características como el polimorfismo. Otra diferencia es que el modelo de objetos se basa en el principio de identidad mientras que el relacional se basa en valor.
- **Lenguaje de Datos**: En los SGBD tradicionales se da el problema de mala concordancia entre los lenguajes de datos y los lenguajes anfitriones, debido al problema de diferencias en el paradigma de programación (imperativo en el caso de JAVA y declarativo para SQL) y en el sistema de tipos de datos. Por este motivo, para resolver este problema, la mayor parte de los SGBO han adoptado extensiones de lenguajes de programación orientados a objetos en la descripción y manipulación de datos. Además, la dificultad de realizar consultas “ad-hoc” en los SGBO con este tipo de lenguajes ha llevado a los fabricantes a desarrollar lenguajes “estilo-SQL”
- **Arquitectura a tres niveles**: Dota a las bases de datos de una mayor independencia físico y lógica. En los SGBO las vistas deben ocultar o exponer tanto los atributos como los servicios.
- **Optimizador**: Los lenguajes de los SGBO son navegacionales, por lo que la optimización resulta difícil al ser el programador el que indica los caminos a seguir. A pasar de ello, el rendimiento de un SGBO es superior al de un SGBDR cuando se navega a través de objetos ya cargados en memoria, siempre que se pueda convertir fácilmente los OID de los objetos a posiciones de memoria. El funcionamiento el

optimizador vendrá condicionado en gran medida por las características que posean los lenguajes del SGBO.

- **Seguridad**: El SGBO debe proporcionar todos los mecanismos necesarios para asegurar el control de la confidencialidad, integridad y disponibilidad de la base de objetos.
- **Interfaces para Programas de Aplicación**: Se deben ofrecer interfaces necesarias para que programas escritos en diferentes lenguajes puedan acceder a los objetos de la base de datos.
- **Diccionario de datos**: El SGBO debe gestionar metadatos sobre objetos, clases, jerarquías, colecciones de objetos, etc. con lo que ha de poseer una mayor riqueza semántica que la de los catálogos de los SGBD relacionales.
- **Herramientas para el desarrollo de aplicaciones**: Es fundamental que el SGBO posea diferentes tipos de herramientas para el desarrollo de aplicaciones, que pueden ir desde lenguajes de desarrollo hasta navegadores (browsers) que permiten una mejor manipulación de las bibliotecas de clase.
- **Herramientas para la transferencia de datos**: Los SGBO deben proporcionar herramientas para transferir datos entre diferentes tipos de SGBD, pudiéndose volcar datos de un SGBO a un sistema relacional, o viceversa.
- **Utilidades para el mantenimiento y mejora del rendimiento de las Bases de Datos**: En los SGBO, al igual que en los SGBD Relacionales, deben existir utilidades que permitan la monitorización del uso de recursos con la finalidad de aumentar el rendimiento de los sistemas y proceder a su ajuste.
- **Soporte de datos multimedia**: Los SGBO deben soportar diferentes tipos de datos, además de los formateados, como son: texto, imagen, voz, vídeo, etc. Por ejemplo, en algunos SGBD Relacionales este tipo de datos se almacenan en BLOBs (Binary Large OBjects), o lo que es lo mismo, objetos de almacenamiento masivo. Muchos fabricante denominan a estos tipo de datos objetos compuestos, pero solo se trata de una colección de bits sin estructura interna. Por último, comentar que es vital que los SGBO permitan la creación de tipos abstractos arbitrarios para que se puedan soportar datos multimedia y ofrecer bibliotecas de clase.
- **Herramientas y facilidades de usuario**: El SGBO debe proporcionar diferentes interfaces a los distintos usuarios del sistema, dependiendo de la función que desempeñan estos. Asimismo, las Interfaces Gráficas de Usuario (GUI) que presentan los SGBO siguen el paradigma de la orientación a objetos facilitando de esta forma la interacción del usuario con el sistema.
- **Distribución**: Como ocurre en los SGBD Relacionales, cualquier elemento del SGBO debe poder distribuirse en diferentes emplazamientos de manera transparente al usuario.
- **Control de versiones y configuraciones**: El control de versiones es fundamental en los SGBO debido a su utilización en entornos de diseño y de información de oficinas. También es importante para este tipo de aplicaciones que el SGBO sea capaz de agrupar versiones compatibles de diferentes objetos y tratarlos como una unidad de más alto nivel. Por último, un objeto en un modelo con versiones tiene dos clases

de atributos (versionados y no versionados), por lo que un cambio en cualquiera de los atributos versionados causa la creación de una nueva versión. Se establecen, por tanto, jerarquías de versiones y se hace necesario que el sistema soporte operaciones de ramificación y de fusión.

2.5.6 TIPOS DE SISTEMAS GESTORES DE BASES DE DATOS DE OBJETOS

Según Friedman, tanto para los SGBO como para los SGBDR, se deben tomar en consideración dos aspectos:

- El modelo lógico de datos soportado, ya sea relacional u orientado a objetos.
- El nivel interno o maquinaria que se ofrece, es decir, lo que concierne a la gestión de E/S, concurrencia, índices, transacciones, recuperación, caching, etc.

De acuerdo a estos criterios, los Sistemas Gestores de Bases de Datos pueden clasificarse en [1]:

- **SGBD relacionales puros**: Como eran todos los productos relacionales hasta hace unos años.
- **SGBD relacionales con frontal OO**: Se añade un nivel o capa de Orientación a Objetos sobre un SGBD Relacional. En este caso, la capa de Orientación a Objetos pude actuar con el SGBDR a dos niveles:
  1. **Gestor de datos**, mediante llamadas SQL.
  2. **Gestor de almacenamiento**, mediante llamadas a procedimientos de bajo nivel.
- **SGBO puros**: Podemos citar a O2, Objectivity, ObjectStore, Ontos, Itasca, Versant, Jasmine, Fast Objects, etc.
- **SGBO con SQL**: Se combinan la Orientación a Objetos y el modelo relacional, realizándose la adaptación necesario tanto a nivel de gestor de almacenamiento como de datos del SGBD Relacional.

Por tanto, la evolución de los Sistemas Gestores de Bases de Datos puede simplificarse de la siguiente forma [1]:

- **SGBD Relacionales** que se han ido extendiendo para soportar nuevos tipos de aplicaciones, para lo cual se ha optado por un enfoque Objeto-Relacional.
- **SGBO Puros** que han incorporado un lenguaje estilo SQL que les permite comunicarse con las bases de datos y los programas existentes, para integrarse en los sistemas de información que usan las empresas.

Por último, comentar que existen otros tipos de sistemas que presentan características parecidas a los SGBO e incluso a veces se catalogan como tales. Podemos mencionar a [1]:

- **Lenguajes de Programación Orientados a Objetos Persistentes**: Son diseñados para un número muy reducido de usuarios y pequeñas bases de datos. Sin embargo, los

SGBO se diseñan pensando en un número elevado de usuarios que acceden concurrentemente a grandes bases de datos.

- **Gestores de Objetos**: Se trata de extensiones de sistemas de ficheros o de gestión de memoria virtual, y que poseen un modelo de datos limitado. Ejemplos: Mneme, Camelot, ObServer, LOOM, etc.
- **MetaSGBD**: Son generadores de sistemas de bases de datos que permiten construir un SGBD adaptado a las características de un determinado grupo de aplicaciones. Ejemplos: Génesis o Exodus.

2.5.7 ESTÁNDAR ODMG 3.0

En 1991, Rick Cattel, de Sunsoft propuso a un grupo de expertos que trabajaban en diferentes empresas de SGBO elaborar un estándar basado en las características que presentaban los productos existentes. Fue de esta forma como nació **ODMG**, del inglés ***Object Data Managament Group***, que agrupaba a los principales vendedores de SGBO, como Object Design, Ontos, O2 Technology, Versant, Objectivity, POET Software y Servio Corporation. También contaba con diversos revisores tanto de empresas (Andersen, Hewlett-Packard, EDS, Sybase, Texas Instruments o Persistence), como de Universidades (Maier, Dewitt, Carey, Dittrich, Zdonik, Liskov, King…)

La primera versión tuvo dos revisiones donde se mejoró principalmente el lenguaje de consulta de objetos. La versión 2.0 incorporó nuevos tipos de colección como el tipo diccionario e introdujo la parte de la vinculación con el lenguaje de programación JAVA. La última versión del estándar es la 3.0 en la que se incluye algunas mejoras en cuanto a la vinculación con JAVA, mejoras en el modelo de objetos, así como algunas correcciones y mejoras en las especificaciones del estándar [9].

Los principales objetivos del estándar de bases de datos ODMG 3.0 son los siguientes:

1. Permitir que los clientes de los SGBO puedan escribir aplicaciones portables. Esto supone que el esquema de datos, la vinculación con el lenguaje de programación, el lenguaje de manipulación de datos y el lenguaje de consulta de datos también deberían ser portables.
2. Intentar ser útil permitiendo operatibilidad entre los distintos productos de SGBO, así como para bases de datos heterogéneas distribuidas que se comunican entre sí.

Los principales componentes del ODMG 3.0 son los siguientes:

- Modelo de Objetos.
- Lenguajes de Especificación de Objetos.
- Lenguaje de Consulta de Objetos.
- Vinculación con Lenguajes de Programación.

Estos elementos vamos a tratarlos en las siguientes 4 secciones.

2.5.7.1 MODELO DE OBJETOS

El Modelo de Objetos está basado en el modelo de objetos de OMG (Object Managament Group) y lo extiende, añadiendo algunos componentes como las relaciones, para soportar las necesidades específicas de las bases de datos.

Un modelo de objetos especifica el tipo de semántica que se puede definir explícitamente sobre el SGBO. La semántica de un modelo de objetos determina las características de los objetos, cómo se relacionan y cómo se relacionan e identifican los mismos.

El modelo del estándar ODMG permite recoger más semántica que el modelo relacional, puesto que permite la declaración de relaciones y operaciones.

A continuación, vamos a estudiar los conceptos de tipos, objetos, literales, estado y comportamiento.

- **TIPOS**

Un tipo tiene una especificación externa y una o varias implementaciones. Una especificación externa de un tipo consiste en una descripción abstracta e independiente de la implementación de las operaciones, excepciones y propiedades visibles para el usuario. El modelo de objetos ODMG proporciona los siguientes constructores para soportar una especificación externa:

- **Interfaz**: Describe el comportamiento abstracto de un tipo de objeto.
- **Literal**: Define el estado abstracto de un tipo literal.
- **Clase**: Define el comportamiento abstracto y el estado abstracto de un tipo de objeto.

En el siguiente código, la interfaz ***Profesor*** define el comportamiento abstracto de los objetos ***Profesor***. La clase ***Persona*** define tanto el comportamiento como el estado abstracto de los objetos Persona. La ***Direccion*** define el estado abstracto de literales.

```
interface Profesor { … };

class Persona { … };

struct Direccion {string calle, string localidad}
```

La implementación de un tipo de objeto consiste en una representación y un conjunto de métodos. La representación es una estructura de datos derivada del estado abstracto del tipo mediante una vinculación con un lenguaje. Los métodos son cuerpos de procedimientos que se derivan del comportamiento abstracto del tipo mediante una vinculación con el lenguaje. Para cada operación se define un método que implementa dicho comportamiento. Un método puede leer o modificar la representación del estado de un objeto o invocar operaciones definidas en otros objetos [1].

• OBJETOS

Los objetos son las instancias de los tipos. Un objeto puede ser:

- **Persistente**: Son objetos de la Base de Datos, se sitúan en memoria, su almacenamiento es gestionado por el SGBO y continúan existiendo después de que mueran los procesos que los crearon.
- **Transitorio**: Se sitúan en memoria, son gestionados por el lenguaje de programación en tiempo de ejecución y sólo existen dentro del procedimiento que los creó.

El tiempo de vida de un objeto es independiente del tipo, siendo algunas instancias de los tipos persistentes mientras que otras son transitorias.

Los aspectos más relevantes sobre los objetos son los que mostramos a continuación:

- Creación de un objeto.
- Identificador único no modificable (OID) para distinguir un objeto de otro.
- Nombre del objeto, asignado por el programador o usuario final.
- Tiempo de vida, que determina como se gestiona la memoria y el espacio reservados para un objeto.
- Estructura de un objeto, que puede ser atómica o compuesta por otros objetos. Los objetos compuestos puede ser estructuras (número fijo de elementos que pueden ser de distinto tipo) o colecciones (número variable de elementos que deben ser del mismo tipo)
- La extensión de un tipo es el conjunto de todas las instancias de un tipo.
- Una clave está formada por un atributo o conjunto de estos, los cuales identifican cada objeto de un tipo de forma unívoca.

• LITERALES

Fundamentalmente, existen dos tipos de literales compuestos:

- **Literales de colecciones**: *set, bag, list, array, dictionary*.
- **Literales estructurados**: *date, interval, time* y definidos por el usuario.

Se comportan de la misma forma que los objetos compuestos, salvo con la diferencia de que no tienen identificador y su valor no se puede modificar [1].

• ESTADO: PROPIEDADES DE LOS OBJETOS

Una clase define un conjunto de propiedades, a través de las cuales los usuarios pueden consultar y manipular el estado de la instancia de una clase. En el modelo de objetos ODMG se definen dos tipos de propiedades [1]:

- **Atributos**: Un atributo siempre va asociado a un tipo.

- **Relaciones**: Una relación siempre se establece entre dos tipos que tengan instancias que puedan ser referenciadas mediante identificadores de objeto. Por lo que un literal, al no tener identificador de objeto, no podrá participar en las relaciones.

A continuación, mostramos cómo se define una clase ***Persona*** con sus atributos:

```
class Persona {
    atribute string DNI;
    attribute date fecha_nac;
    attribute string nombre;
}
```

Una declaración de un atributo en una interfaz define únicamente el comportamiento abstracto de sus instancias. En el siguiente ejemplo el atributo ***edad*** se define en la interfaz ***i_Persona***, este atributo no implicaría la definición de su estado sino de su comportamiento, es decir, ***calcular la edad***.

```
interface  i_Persona {
    atribute short edad;
}
```

Como ya apuntábamos anteriormente, las relaciones se definen entre tipos. El modelo de objetos ODMG, sólo soporta las relaciones binarias y las relaciones de herencia (simple y múltiple).

Los tipos de relaciones binarias que son soportadas son: uno-a-uno, uno-a-muchos, o muchos-a-muchos. El concepto de relación en el modelo de objetos es similar al del modelo E/R, y no es un objeto por sí sola (por lo que no tiene identificador de objeto).

En el siguiente ejemplo, mostramos una relación binaria 1:N en ODL, en la que un profesor puede impartir varios cursos, mientras que un curso es impartido por un solo profesor.

```
class PROFESOR
{
    …
    relationship set <CURSO> imparte
    inverse CURSO:: impartido_por;
};
```

```
class CURSO
{
    …
    relationship set <PROFESOR> impartido_por
    inverse PROFESOR:: imparte;
};
```

Por otro lado, el modelo de objetos ODMG soporta dos tipos de relaciones de herencia [1]:

- **Relación ISA**: Implica herencia de comportamiento y se representa con ":". Define la herencia de comportamiento entre tipos de objetos (interfaces o clases). En una relación ISA, los tipos de objeto se clasifican en tipos y subtipos, heredando el subtipo las propiedades y operaciones del supertipo y puede, además, añadir propiedades y operaciones propias. Soporta la herencia múltiple de comportamiento.
- **Relación EXTENDS**: Define la herencia de estado y comportamiento entre tipos de objetos (clases). A través de esta relación se define la herencia simple entre dos clases.

En el siguiente ejemplo, mostramos tres interfaces (***Empleado, Profesor*** y ***Profesor_Practicas***), así como tres clases (***Empleado_Asalariado, Persona*** y ***Persona_Empleada***). Las interfaces ***Profesor*** y ***Profesor_Practicas*** heredan el comportamiento de ***Empleado*** y ***Profesor***, respectivamente definiendo una relación ISA. La clase ***Empleado_Asalariado*** el comportamiento de la interfaz ***Empleado*** (ISA). Por último, la clase ***Persona_Empleada*** hereda el comportamiento de la interfaz ***Empleado*** (ISA) y tanto el estado como el comportamiento de la clase ***Persona*** (extends).

```
interface Empleado { …. };
interface Profesor : Empleado { …. };
interface Profesor_Practicas : Profesor { …. };
class Empleado_Asalariado : Empleado { … };

class Persona
{
    attribute string nombre;
    attribute date fecha_nac;
};
```

```
class Persona_Empleada extends Persona: Empleado
{
    attribute date fecha_contrato;
    attribute currency salario;
};
```

- **COMPORTAMIENTO**

Los tipos además de las propiedades (atributos y relaciones), tienen comportamiento que se especifica con un conjunto de signaturas de las operaciones. Cada signatura define el nombre de la operación, el nombre y tipo de cada uno de los argumentos, los tipos de los valores resultantes y los nombres de las excepciones que las operaciones pueden lanzar.

Una operación solo puede definirse sobre un tipo sencillo. No puede existir una operación sin estar asociada a un tipo o a varios. El nombre de una operación solo tiene que ser único dentro de la definición del tipo, es decir, puede haber operaciones de tipos diferentes con el mismo nombre. En este caso, podemos decir que los nombres de estas operaciones están sobrecargados. Si se invoca a una operación utilizando el nombre sobrecargado se debe seleccionar la operación concreta a ejecutar [1].

A continuación, presentamos un ejemplo donde la clase **Person**a incluye la signatura de una operación que devuelve la edad de la persona.

```
class Persona (extent personas)
{
    attribute string nombre;
    attribute date fecha_nacimiento;
    attribute string sexo;
    short edad();
};
```

2.5.7.2 LENGUAJES DE ESPECIFICACIÓN DE OBJETOS

Los lenguajes de especificación de objetos son independientes de los lenguajes de programación y se usan para definir los esquemas, operaciones y el estado de un SGBO. El principal objetivo de estos lenguajes es facilitar la migración de datos a través de un SGBO, además de facilitar la interoperabilidad entre los distintos productos.

El estándar propone dos lenguajes de especificación de objetos: **Lenguajes de Definición de Objetos (ODL)** y **Lenguaje de Intercambio de Objetos (OIF)**, que veremos a continuación.

- **LENGUAJE DE DEFINICIÓN DE OBJETOS**

El Lenguaje de Definición de Objetos (**ODL**, del inglés ***Object Definition Language***), se usa para definir las especificaciones de tipos de objetos ajustándose al modelo de objetos de ODMG. Dentro de la definición de especificaciones se incluye sus propiedades y operaciones, sin embargo, sólo se define las signaturas, sin incluir la implementación. Por tanto, ODL no está ligado a la sintaxis de ningún lenguaje de programación en particular [1].

Características:

- Soporte de todos los constructores del modelo de objetos de ODMG.
- No es un lenguaje de programación completo, sólo es un lenguaje de definición para la especificación de objetos.
- Independiente del lenguaje de programación.
- Compatible con el Lenguaje de Definición de Interfaz de OMG, es decir, el IDL del inglés Interface Definition Language.
- Extensible a nuevas funcionalidades y optimizaciones físicas.

- **FORMATO DE INTERCAMBIO DE OBJETOS**

El Formato de Intercambio de Objetos (**OIF**, del inglés ***Object Interchange Format***), es un lenguaje de especificación usado para cargar y descargar el estado de los objetos persistentes de un SGBO a un fichero. También se puede usar para intercambiar objetos persistentes entre SGBO, proporcionar documentación y realizar batería de pruebas [1].

Características:

- OIF debe soportar todos los estados conformes al modelo de objetos de ODMG y a las definiciones de esquemas ODL.
- No es un lenguaje de programación completo, sólo un lenguaje de definición para la especificación de objetos persistentes y sus estados.
- Debe diseñarse, siempre que sea posible, siguiendo los estándares relacionados, como STEP o INCITS.
- OIF solo usará las palabras claves de tipo, atributo o identificador de relación proporcionadas en la definición ODL de un esquema de la Base de Objetos.

2.5.7.3 LENGUAJE DE CONSULTA DE OBJETOS

El modelo de datos ODMG define un lenguaje declarativo para consultar y modificar los objetos de la Bases de Objetos, el lenguaje de consulta de objetos, **OQL**, del inglés ***Object Query Language***.

OQL está basado en el lenguaje SQL, con una sintaxis similar pero con una semántica diferente. OQL contempla aspectos de la orientación de objetos, como el identificador del objeto, objetos complejos, operaciones, herencia, polimorfismo, vinculación dinámica y relaciones, así como la navegación a través de las mismas [1].

La sintaxis del lenguaje OQL está preparada para mezclar las consultas con los lenguajes de programación C++, Smaltak y Java. Así pues, una consulta en OQL embebida en uno de estos lenguajes de programación devolverá objetos que coincidan con el sistema de tipos del lenguaje correspondiente [1].

OQL es un lenguaje de consulta fácil de usar, aunque computacionalmente no es completo. Además, OQL es fácilmente optimizable, puesto que es declarativo.

Por otro lado, OQL proporciona primitivas de alto nivel para el tratamiento de todo tipo de colecciones. No proporciona operadores de actualización explícitos, sino que se actualiza a través de operaciones definidas sobre los objetos .

La sintaxis básica del OQL es como en SQL:

```
SELECT ….
FROM …
WHERE …
```

En general, en cada consulta se requiere un punto de entrada a la Base de Objetos, que será un objeto persistente. Las consultas OQL son funciones que siempre se realizarán sobre la extensión de las clases, que será una colección de objetos persistentes y devuelven un objeto del tipo que se deduzca de la consulta.

En el siguiente ejemplo, vamos a seleccionar las distintas fechas de nacimiento de las personas cuyo nombre es “Ada”. La consulta devolverá un literal de tipo conjunto de elementos de tipo fecha sin duplicado (**set<date>**):

```
select distinct p.fecha_nac
from personas p
where p.nombre=’Ada’
```

La siguiente consulta devolverá una estructura que contendrá la fecha de nacimiento y el sexo, es decir, un literal del tipo: **set<struct<date,string>>**:

```
select distinct struct (f:p.fech_nac, s:p.sexo)
from personas p
where p.nombre=’Ada’
```

Como hemos comentado con anterioridad, OQL soporta la definición de consultas navegacionales, permitiendo navegar a través de los objetos. Así para acceder a los objetos y recorrer las relaciones usaremos "." o bien "->"

A continuación, exponemos un ejemplo que muestra la clase **Persona** y una clase hija **Empleado**. Entre la clase Empleado existe una relación reflexiva para indicar que un empleado puede tener varios subordinados.

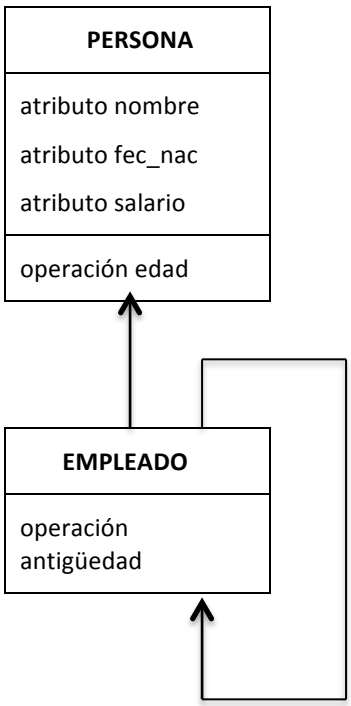


Figura 6: Clase Persona y Clase hija Empleado [1]

Con la siguiente consulta se pretende devolver el nombre de cada empleado y sus subordinados que tengan un salario superior a 1500 euros. Es decir, devolverá un literal del tipo conjunto de elementos de tipo estructura: **set<struct(n:string,smp:bag<empleados>)>**

```
select distinct struct (n:x.nombre, sub:
        (select s
        from s in x.subordinados
        where s.salario>1500)
from x in empleados
```

En esta otra consulta mostramos cómo se pueden incluir consultas anidadas en la cláusula FROM. Esta consulta devolverá, para cada empleado cuya antigüedad sea de 7 años y cuyo nombre sea Antonio, la edad del empleado y salario (con duplicados). Es decir,

devolverá un literal de tipo bolsa (con repetidos) con elementos de tipo estructura: **bag<struct(e:integer,s:integer)>**

```
select struct (e:x.edad, s:x.salario)
from (select y
        from empleados y
        where y.antiguedad=”7” as x
where x.nombre=”Antonio”
```

2.5.7.4 VINCULACIÓN CON LENGUAJES DE PROGRAMACIÓN: C++ Y JAVA

- **VINCULACIÓN CON C++**

Ya desde su primera versión, ODMG define una vinculación entre las implementaciones de ODMG y el lenguaje de programación C++. El estándar define cómo escribir código C++ para manipular objetos persistentes de forma portable. A esto se le conoce como C++OML, o el lenguaje de manipulación de objetos. También se incluye en el estándar una versión de ODL que usa la sintaxis de C++, el mecanismo para invocar OQL y procedimientos para realizar operaciones y transacciones en el Sistema Gestor de Bases de Objetos [1].

- **VINCULACIÓN CON JAVA**

ODMG define además la vinculación entre el modelo de objetos del estándar y el lenguaje de programación JAVA, concretamente para la plataforma JAVA TM 2. Esta vinculación también incluye los mecanismos para invocar sentencias OQL y procedimientos para operaciones y transacciones en el SGBO [1].

2.5.8 METODOLOGÍA DE DISEÑO DE BD BASADA EN MDA

El ciclo de vida en la creación de aplicaciones de Bases de Datos consta de una serie de etapas, en las que se van generando salidas, que son entradas para las siguientes etapas. En nuestro caso, vamos a seguir una aproximación dirigida por modelo para el diseño de bases de objetos. Nos vamos a basar en la **Arquitectura Dirigida por Modelos** (MDA) propuesta por el grupo OMG.

En una metodología basada en MDA, las etapas coinciden con los niveles de la arquitectura:

1. **Modelado independiente de plataforma**: A nivel **PIM**, del inglés ***Platform Independent Model***, se realiza lo que tradicionalmente se conoce como el modelado conceptual de datos a partir de los requisitos del usuario, sin tomar en consideración

la tecnología seleccionada, puesto que se trata de un modelo independiente de la plataforma. Este PIM de datos se representará mediante un diagrama de clases UML, debido a que es el lenguaje de modelado estándar para sistemas de información orientados a objetos [1].

2. **Modelado específico de plataforma**: A nivel **PSM**, del inglés ***Platform Specific Model***, se lleva a cabo lo que tradicionalmente conocemos como el diseño lógico de la Base de Datos. En este nivel, los modelos ya son específicos de la tecnología concreta elegida. Puesto que este apartado está dedicado al diseño de Base de Datos, el modelo PSM elegido es el modelo de objetos ODMG. Para pasar el nivel específico de plataforma (PSM) aplicamos las reglas de transformación que vemos a seguidamente [1].

2.5.8.1 REGLAS DE TRANSFORMACIÓN ENTRE MODELOS

De forma similar a como las metodologías para bases de datos relacionales proponen algunas reglas para la transformación de un esquema conceptual en un esquema lógico. A continuación, vamos a proponer las reglas de transformación para pasar del nivel PIM al PSM utilizando la tecnología de Bases de Objetos:

PIM de datos	PSM de datos
Clase persistente UML	Clase ODL + extensión de la clase
Atributo de la clase	Atributo de la clase ODL
Atributo multivaluado <<MA>>	Atributo de tipo colección (set o list)
Atributo compuesto <<CA>>	Atributo de tipo estructura (struct)
Restricción	Operaciones en el constructor
Método	Signatura de una operación
Relación binaria	Relación binaria ODL
Generalización	Relación extends o IS-A
Agregación	
Miembro-Colección	Atributo en el Miembro
Compuesto-Componente	Atributo en el Compuesto (estructura)

A través de estas reglas de transformación se pretende pasar de un esquema conceptual a un esquema lógico en ODL con la menor pérdida de semántica posible.

- **TRANSFORMACIÓN DE CLASES**

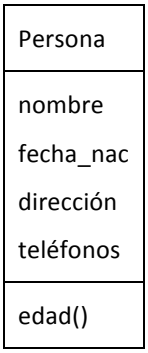
Cada clase persistente UML se traduce a una clase de ODL. Como estamos definiendo un esquema de bases de datos, asumimos que todas las clases son persistentes, por lo que para cada clase ODL habrá que definir explícitamente su extensión.

Cada atributo de una clase UML se traduce en un atributo ODL.

- Un atributo multivaluado <<MA>> se traduce en un tipo colección.
- Un atributo compuesto <<CA>> se convertirá en una estructura ODL.

Las restricciones tienen que ser implementadas mediante operaciones en el constructor de la clase. Los métodos se definirán por medio de la definición del signatura de una operación el ODL [1].

PIM



PSM

```
class Persona (extent personas)
{
        attribute string nombre;
        attribute date fecha_nac;
        attribute struct T_Dirección {unigned short numero, string calle, string
localidad}
        dirección;
        attribute set <string> teléfonos;
        short edad();
}
```

Figura 7: Ejemplo de transformación de una clase UML

- **TRANSFORMACIÓN DE RELACIONES**

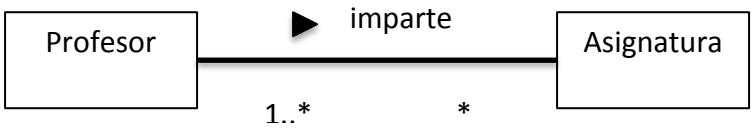
Con respecto a la transformación de relaciones UML a relaciones en ODL se podrán realizar diferentes conversiones dependiendo de las cardinalidades de la relación.

Las relaciones binarias se transforman mediante la definición de relaciones en ODL. El nombre a la relación en ODL se corresponderá con el nombre de la relación.

La cardinalidad máxima de una relación se recogerá en ODMG en la propia definición de la relación. Si es mayor que 1, la relación se definirá mediante un tipo de colección (set, list, bag). Si es 1, esta restricción se implementará en la operación del constructor.

Las relaciones reflexivas se representan en ODL como cualquier relación. El único aspecto importante a tener en cuenta es que deben ser definidas dos veces para poder representar los dos caminos de la relación [1].

PIM



PSM

```
class Profesor (extent profesores)
{
        relationship set <Asignatura> imparte_P
                inverse imparte_A::Asignatura;
}

class Asignatura (extent asignatura)
{
        relationship set <Profesor> imparte_A
                inverse imparte_P::Profesor;

}
```

Figura 8: Ejemplo de transformación de relación binaria

Si las relaciones tienen atributos, existen dos posibilidades diferentes:

- Si la cardinalidad es **N:M**, se debería convertir la relación en una clase nueva con los atributos de la relación así como dos relaciones adicionales.

- Si la cardinalidad es **1:N** se podría incluir el atributo en las clases que participan en la relación con cardinalidad 1.

El modelo de datos del ODMG únicamente soporta relaciones binarias, por lo que las relaciones n-arias, se deberán definir como en el modelo relacional, es decir, creando una nueva clase, con la pérdida de semántica que esto conlleva.

- **TRANSFORMACIÓN DE GENERALIZACIONES**

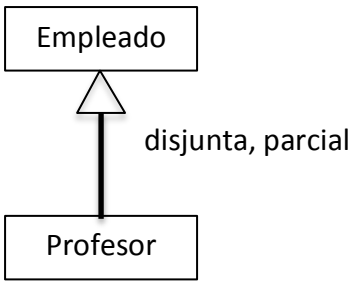
Las generalizaciones a nivel conceptual (PIM) puede ser **disjuntas** o **solapadas**, **totales** o **parciales**. Estas dos clasificaciones son ortogonales entre ellas.

La generalización disjunta y parcial es soportada directamente por el modelo de datos de ODMG por medio de la relación **extend**.

El solapamiento no es soportado por el modelo de datos del ODMG.

En cuanto a la generalización total, el modelo de datos del ODMG no define ninguna notación para soportarla explícitamente. Así, una generalización total se podría definir como una relación IS-A en ODL siendo el supertipo una interfaz. Sin embargo, es importante destacar que una interfaz no se corresponde con una clase abstracta, puesto que una interfaz, al contrario que una clase, no representa el estado. Simplemente, se trata de la mejor forma de representar una generalización total en ODL [1].

PIM



PSM

```
class Empleado (extent empleados)
{
}

class Profesor extends Empleado (extent profesores)
{
}
```

Figura 9: Ejemplo de transformación de generalización

- **TRANSFORMACIÓN DE AGREGACIONES**

Existen diferentes formas de representar el concepto de agregación: agregación como un tipo de relación binaria y la notación de árbol de agregación. Existen algunas diferencias semánticas entre agregaciones (Miembro-Colección) y el árbol de agregación (Compuesto-Componente).

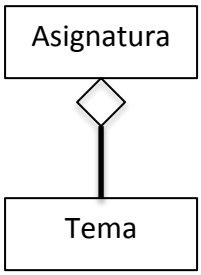
- **Agregaciones Miembro-Colección**:

Representa una colección de objetos, siendo todos ellos de la misma clase y juntos forman otra clase. Un ejemplo clásico es el de la colección de árboles que forman un bosque. Este tipo de agregación se podría traducir como un atributo de tipo colección en el modelo de datos del ODMG.

ODL no proporciona ningún constructor que soporte directamente la agregación. La agregación Miembro-colección se soporta como un atributo de tipo colección, de la misma forma que se soportan los atributos multivaluados.

A continuación, vemos un ejemplo de transformación de una agregación Miembro-Colección. Una asignatura que está compuesta por temas se transformará a ODL, incluyendo en el todo un atributo de tipo conjunto (set) de partes, que en este caso, serán los temas que componen la asignatura [1].

PIM



PSM

```
class Tema (extent temas)
{
}

class Asignatura (extent asignaturas)
{
        attribute set <Tema> compone_A;
}
```

Figura 10: Ejemplo de transformación de una agregación Miembro-Colección

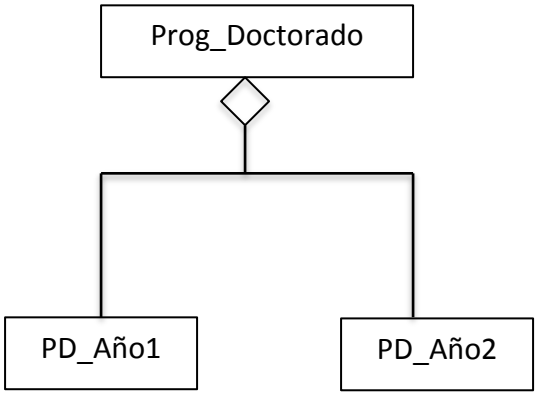
- **Agregaciones Compuesto-Componente**:

Una agregación Compuesto-Componente representa una clase estructurada que está compuesta de, al menos, dos clases diferentes. La diferencia principal con respecto a la agregación Miembro-Colección es que los objetos componentes pertenecen a una clase diferente. Además, los objetos componentes tienen una relación estructural entre ellos. Sin embargo, en una agregación Miembro-Colección no existe una relación estructural entre sus miembros.

La agregación Compuesto-Componente se representará en el modelo de datos de ODMG, definiendo en la clase compuesto un atributo de tipo estructura, con un miembro para cada clase componente. Tanto en el modelo de datos del ODMG, como en la mayoría de bases de objetos, no se soporta el concepto de agregación, y por lo tanto, no existen diferencias entre atributos y agregaciones [1].

En el siguiente ejemplo, podemos ver cómo se transforma en ODL una agregación Compuesto-Componente, en la que el compuesto es un programa de doctorado y los componentes son el primer y el segundo año de doctorado.

PIM



PSM

```
class PD_Año1 (extent pd_año1_s) {          }

class PD_Año2 (extent pd_año2_s) {

}
class P_Doctorado (extent ps_doctorado)
{
        attribute struct consta{PD_Año1 a1, PD_Año2 a2}
}
```

Figura 11: Ejemplo de transformación de una agregación Compuesto-Componente

- **TRANSFORMACIÓN DE COLECCIONES Y TIPOS ESTRUCTURADOS**

Como hemos podido comprobar, los tipos colección y estructurados son conceptos clave en el diseño de bases de objetos, y constituyen una de las principales diferencias con respecto a las bases de datos relacionales. Permiten la definición de atributos multivaluados, de las cardinalidades de las relaciones, agregaciones, etc.

El modelo de datos del ODMG soporta diferentes tipos de colección, por tanto, lo más importante es elegir el tipo más apropiado para cada caso.

Como regla general, se deberían usar los tipos **colección** de la siguiente forma:

- **Tipo Bag**: Si la colección admite duplicados, tiene un número ilimitado de elementos, y el orden no es relevante.
- **Tipo Set**: Si la colección también tiene un número ilimitado de elementos y el orden no es relevante, pero no admite duplicados.
- **Tipo List**: Si el tipo colección tiene un número ilimitado de elementos y el orden es relevante, aunque los elementos estén duplicados o no.
- **Tipo Dictionary**: El tipo colección tiene un número ilimitado de parejas de elementos (clave-valor), sin claves duplicadas.
- **Tipo Array**: Si el tipo colección tiene un número limitado de elementos que ocupan una posición fija e indexable.

En cuanto al Tipo **Struct**, se usa cuando el elemento que se desea representar es un elemento compuesto y cada componente tiene un tipo de datos diferente. Una estructura tiene un número limitado de componentes. Por ejemplo, una dirección es una estructura con cuatro componentes: calle, número, localidad y ciudad.

La principal diferencia entre una estructura y una clase es que la clase representa un tipo de objeto y una estructura un tipo valor, es decir, un literal en la terminología del ODMG. La distinción entre objetos y valores ha sido una de las más importantes discusiones en el paradigma de la orientación a objetos. Smalltalk no distingue entre objetos y valores, por lo que el tipo dirección se tiene que representar como un literal (que no es lo mismo que un valor) o bien como una clase. Sin embargo, C++ soporta estructuras y clases, de forma que la dirección se representaría como una estructura.

2.5.9 LENGUAJES DE PROGRAMACIÓN PERSISTENTES

Los lenguajes de las bases de datos se diferencia de los lenguajes de programación tradicionales en que trabajan directamente con datos que son persistentes, es decir, los datos siguen existiendo una vez que el programa que los creó haya concluido. Las relaciones de las base de datos y las tuplas de las relaciones son ejemplos de datos persistentes. Por el contrario, los únicos datos persistentes con los que los lenguajes de programación tradicionales trabajan directamente son los archivos.

El acceso a las bases de datos es sólo un componente de las aplicaciones del mundo real. Mientras que los lenguajes para el tratamiento de datos como SQL son bastante efectivos en el acceso a los datos, se necesita un lenguaje de programación para implementar otros componentes de las aplicaciones como las interfaces de usuario o la comunicación con otras computadoras. La manera tradicional de realizar las interfaces de las bases de datos con los lenguajes de programación es incorporar SQL dentro del lenguaje de programación.

Los **lenguajes de programación persistentes** son lenguajes de programación extendidos con estructuras para el tratamiento de los datos persistentes. Los lenguajes de programación persistentes pueden distinguirse de los lenguajes con SQL incorporado, al menos, de dos formas:

1. En los lenguajes incorporados, el sistema de tipos del lenguaje anfitrión suele ser diferente del sistema de tipos del lenguaje para el tratamiento de datos. Los programadores son los responsables de las conversiones de tipos entre el lenguaje anfitrión y SQL, lo que presenta varios inconvenientes. Por el contrario, en los lenguajes de programación persistentes, el lenguaje de consultas se halla totalmente integrado con el lenguaje anfitrión y ambos comparten el mismo sistema de tipos. Los objetos se pueden crear y guardar en la base de datos sin ninguna modificación explícita del tipo o del formato. Los cambios de formato necesarios se realizan de manera transparente [2].

2. Los programadores que usan lenguajes de consultas incorporado son responsables de la escritura de código explícito para la búsqueda en la memoria de los datos de la base de datos. Si se realizan actualizaciones, los programadores deben escribir explícitamente código para volver a guardar los datos actualizados en la base de datos. Por el contrario, en los lenguajes de programación persistentes, los programadores pueden trabajar con datos persistentes sin la necesidad de escribir explícitamente código para buscarlos en la memoria o volver a guardarlos en el disco [2].

En esta sección, vamos a describir la forma en que se pueden extender los lenguajes de programación orientados a objetos, como C++ y Java, para hacerlos lenguajes de programación persistentes. Las características de estos lenguajes permiten que los programadores trabajen con los datos directamente desde el lenguaje de programación, sin necesidad de recurrir a los lenguajes de tratamiento de datos como SQL. Por tanto, ofrecen una integración más estrecha de los lenguajes de programación con las bases de datos que, por ejemplo, SQL incorporado.

No obstante, los lenguajes de programación persistentes presentan ciertos **inconvenientes** que hay que tener presentes al decidir si conviene usarlos o no. Puesto que los lenguajes de programación suelen ser potentes, resulta relativamente sencillo cometer errores de programación que dañen las bases de datos. La complejidad de los lenguajes hace

que la optimización automática de alto nivel, como la reducción de E/S de disco, resulte más difícil [2].

En esta sección, describimos varios problemas teóricos que hay que abordar a la hora de añadir persistencia a los lenguajes de programación ya existentes. En primer lugar, abordamos los problemas independientes de los lenguajes y después se tratan problemas que son específicos de los lenguajes C++ y Java.

2.5.9.1 PERSISTENCIA DE LOS OBJETOS

Los lenguajes de programación orientados a objetos ya poseen un concepto de objeto, un sistema de tipos para definir los tipos de los objetos y constructores para crearlos. Sin embargo, esos objetos son **transitorios**, los cuales desaparecen en cuanto finaliza el programa. Si se desea transformar uno de estos lenguajes en un lenguaje para la programación de bases de datos, el primer paso consiste en proporcionar una manera de hacer persistentes a los objetos [2].

Se ha propuesto varios enfoques:

- **Persistencia por clases**: Todos los objetos de la clase son, por tanto, persistentes de manera predeterminada. Todos los objetos de las clases no persistentes son transitorios. Este enfoque no es flexible, dado que suele resultar útil disponer en una misma clase tanto de objetos transitorios como de objetos persistentes.
- **Persistencia por creación**: Los objetos son persistentes o transitorios en función de la forma de crearlos. Varios sistemas de bases de datos orientados a objetos siguen este enfoque.
- **Persistencia por marcas**: Todos los objetos se crean como transitorios, pero si un objeto tiene que persistir más allá de la ejecución del programa, hay que marcarlo como persistente de manera explícita antes de que éste concluya. Este enfoque, a diferencia del anterior, pospone la decisión sobre la persistencia o la transitoriedad hasta después de la creación del objeto.
- **Persistencia por alcance**: Uno o varios objetos se declaran objetos persistentes (objetos raíz) de manera explícita. El resto de objetos serán persistentes si pueden ser alcanzados desde algún objeto raíz mediante una secuencia de referencias. Por tanto, todos los objetos a los que se haga referencia desde los objetos persistentes raíz serán persistentes. Una ventaja es que resulta sencillo hacer que sean persistentes estructuras de datos completas con sólo declarar como persistente su raíz. Sin embargo, el sistema de bases de datos sufre la carga de tener que seguir las cadenas de referencias para detectar los objetos que son persistentes, y eso puede resultar costoso.

2.5.9.2 SISTEMAS PERSISTENTES EN C++

Varias de las características orientadas a objetos del lenguaje C++ ayudan a proporcionar un buen soporte para la persistencia sin modificar el propio lenguaje. Por ejemplo, se puede declarar una clase denominada Persistent_Object (objeto persistente) con

los atributos y los métodos para dar soporte a la persistencia, y cualquier otra clase que deba ser persistente puede hacerse subclase de esta clase y heredará el soporte de la persistencia. El lenguaje C++ permite también redefinir los nombres de las funciones y los operadores estándar (+, -, el operador de desvinculación de punteros →, etc.) en función del tipo de operandos a los que se aplican. Esta posibilidad se denomina sobrecarga y se usa para redefinir los operadores para que se comporten de la manera deseada cuando operan con objetos persistentes.

Proporcionar apoyo a la persistencia mediante las bibliotecas de clases presenta la ventaja de que se realizan cambios mínimos en C++ y resulta relativamente fácil de implementar. Sin embargo, presenta el inconveniente de que los programadores tienen que usar más tiempo en escribir los programas que trabajan con objetos persistentes y no es sencillo especificar las restricciones de integridad del esquema ni ofrecer soporte para las consultas declarativas. Algunas implementaciones persistentes de C++ soportan extensiones de la sintaxis de C++ para facilitar estas tareas [2].

Es necesario abordar los siguientes problemas a la hora de añadir soporte a la persistencia a C++ y a otros lenguajes:

- **Punteros persistentes**: Se debe definir un nuevo tipo de datos para que represente los punteros persistentes.
- **Creación de objetos persistentes**: El operador **new** de C++ se usa para crear objetos persistentes mediante la definición de una versión **sobrecargada** del operador que usa argumentos adicionales especificando que deben crearse en la base de datos. Por tanto, en lugar de **new T()** se usa **new(bd) T()** para crear un objeto persistente.
- **Extensiones de las clases**: La norma ODMG de C++ exige que el nombre de la clase se pase como parámetro adicional a la operación **new**. Esto permite mantener varias extensiones para cada clase, pasando diferentes nombres.
- **Relaciones**: Las relaciones entre las clases se suelen representar almacenando punteros de cada objeto a los objetos con los que está relacionado. Los sistemas persistentes de C++ ofrecen una manera de especificar esas restricciones y de hacer que se cumplan mediante la creación y borrado automático de los punteros.
- **Interfaz iteradora**: Dado que los programas tienen que iterar sobre los miembros de las clases, es necesario una interfaz para iterar sobre los miembros de las extensiones de las clase. La interfaz iteradora también permite especificar selecciones, de modo que sólo hay que capturar los objetos que satisfagan el predicado de selección.
- **Transacciones**: Los sistemas persistentes de C++ ofrecen soporte para comenzar las transacciones y para comprometerlas o provocar su retroceso.
- **Actualizaciones**: Uno de los objetivos de ofrecer soporte a la persistencia a los lenguajes de programación es permitir la persistencia transparente. Es decir, una función que opera sobre un objeto no debe necesitar saber que el objeto es persistente, por lo que se pueden usar las mismas funciones sobre los objetos independientemente de que sean persistentes o no. Sin embargo, surge el problema de que resulta difícil de detectar cuándo se ha actualizado un objeto.

- **Lenguajes de consultas**: Los iteradores ofrecen soporte para consultas de selección sencillas. Para soportar consultas de selección complejas los sistemas persistentes de C++ definen un lenguaje de consultas.

Gran número de sistemas de bases de datos orientados a objetos basados en C++ se desarrollaron a finales del siglo XX. Sin embargo, el mercado para esas bases de datos resultó mucho más pequeño de lo esperado, puesto que la mayor parte de los requisitos de las aplicaciones se cumplen usando SQL mediante interfaces como ODBC o JDBC. En los años 90 el Grupo de Gestión de Datos de Objetos (ODMG) definió las normas para agregar persistencia a C++ y Java. No obstante, el grupo concluyó sus actividades alrededor de 2002 [2].

Aunque los sistemas de bases de datos orientados a objetos no encontraron el éxito comercial que esperaban, la razón de añadir persistencia a los lenguajes de programación sigue siendo válida. Hay aplicaciones con grandes exigencias de rendimiento que se ejecutan en sistemas de bases de datos orientadas a objetos, el uso de SQL impondría una sobrecarga de rendimiento excesiva para muchos de estos sistemas. Con los sistemas de bases de datos relacionales orientados a objetos que proporcionan soporte para los tipos de datos complejos, incluidas las referencias, resulta más sencillo almacenar los objetos de los lenguajes de programación en bases de datos de SQL.

2.5.9.3 SISTEMAS JAVA PERSISTENTES

La demanda de soporte de la persistencia de los datos en los programas de Java se ha incrementado al mismo tiempo que el uso del lenguaje ha ido creciendo. Los primeros intentos de creación de una norma para la persistencia en Java fueron liderados por el consorcio ODMG. Posteriormente, el consorcio concluyó sus esfuerzos, pero transfirió su diseño al proyecto **Objetos de Bases de Datos de Java** (Java Database Objects, JDO), que coordina Sun Microsystems [2].

El modelo JDO para la persistencia de los objetos en los programas de Java es diferente del modelo de soporte de la persistencia en los programas C++. Entre sus características se hallan:

- **Persistencia por alcance**: Los objetos no se crean explícitamente en la base de datos. El registro explícito de un objeto como persistente hace que el objeto sea persistente. Además, cualquier objeto alcanzable desde un objeto persistente pasa a ser persistente.
- **Mejora del código de bytes**: En lugar de declarar en el código de Java que una clase es persistente, se especifican en un archivo de configuración (con extensión .jdo) las clases cuyos objetos pueden hacerse persistentes. Se ejecuta un programa *mejorador* específico de la implementación que lee el archivo de configuración y lleva a cabo dos tareas:

1. Puede crear estructuras en la base de datos para almacenar objetos de esa clase.
2. Modifica el código de bytes (generado al compilar el programa de Java) para que maneje tareas relacionadas con la persistencia.

- **Asignación de bases de datos**: JDO no define la forma en que se almacenan los datos en la base de datos subyacente. Por ejemplo, una situación frecuente es que los objetos se almacenen en una base de datos relacional. El programa mejorador puede crear en la base de datos un esquema adecuado para almacenar los objetos de las clases. La manera exacta en que lo hace depende de la implementación y no está definida por JDO.
- **Extensiones de clase**: Las extensiones de clase se crean y se conservan de manera automática para cada clase declarada como persistente. Todos los objetos que se hacen persistentes se añaden de manera automática a la extensión de clase correspondiente a su clase. Los programas de JDO pueden tener acceso a las extensiones de clase e iterar sobre los miembros seleccionados.
- **Tipo de referencia único**: No hay diferencia de tipos entre referencias a los objetos transitorios y las referencias a los objetos persistentes.

2.5.10 SISTEMAS ORIENTADOS A OBJETOS Y SISTEMAS RELACIONALES ORIENTADOS A OBJETOS

Las bases de datos orientadas a objetos se crean alrededor de los lenguajes de programación persistentes, mientras que las bases de datos relacionales orientadas a objetos son bases de datos orientadas a objetos construidas sobre el modelo relacional.

Las extensiones persistentes de los lenguajes de programación y los sistemas relacionales orientados a objetos se dirigen hacia mercados diferentes.

Los puntos fuertes de los diversos tipos de sistemas de bases de datos pueden resumirse de la siguiente forma [2]:

- **Sistemas relacionales**: Tipos de datos sencillos, lenguajes de consultas potentes y protección elevada.
- **Bases de datos orientadas a objetos basadas en lenguajes de programación persistentes**: Tipos de datos complejos, integración con los lenguajes de programación, elevado rendimiento.
- **Sistemas relacionales orientados a objetos**: Tipos de datos complejos, lenguajes de consultas potentes, protección elevada.

2.6 TENDENCIAS FUTURAS

Según el **International Data Corporation (IDC)**, principal proveedor mundial de inteligencia de mercado, servicios de consultoría y eventos para los mercados de tecnología de la información, telecomunicaciones y tecnología de consumo, nos encontramos ante un

momento ideal para el mundo de las bases de datos, con nuevas iniciativas emergentes y futuras en apoyo de la computación en la nube, gestión de datos en memoria y Big Data [4].

Por un lado, la computación en la nube ofrece las bases de datos como servicios. Se contrata el servicio de una base de datos en la nube, en lugar de mantener un servidor y una base de datos. Como ejemplo de proveedor de servicio podemos mencionar a SQLAzure de Microsoft que configura la base según los requerimientos del cliente y ofrece acceso a través de un navegador web. El mismo proveedor de servicio es el encargado del mantenimiento, configuración de la base de datos y del servidor, garantizando una alta disponibilidad. El usuario mientras tanto se dedica solo y exclusivamente a trabajar con la base de datos.

Por otro lado, el manejo de datos en memoria hace referencia a bases de datos manejadas en la memoria RAM garantizando así la rápida velocidad de ejecución, junto con algún método que garantice la persistencia de la información.

Los **BigData** manejan información obtenida desde diversas fuentes y formatos, como es el caso de páginas web, redes sociales, el análisis del genoma humano, la física de partículas, entre otros. Estos almacenes de datos presentan dificultades que no pueden ser resueltas mediante el uso de sistemas de gestión de bases de datos tradicionales [5].

BigData agrupa las técnicas de almacenamiento, análisis y manejo de inmensos repositorios de datos y centra sus características en tres partes: Volumen, Variedad y Velocidad.

La información generada a partir de la abundancia de sensores, micrófonos, cámaras, etc., en nuestra vida diaria será dentro de poco el segmento más grande de toda la información disponible. En este punto, BigData será una herramienta indispensable para quienes se dedican a la investigación pues gracias a ella será posible descubrir cosas que sin estas herramientas habría tomado años de investigación y recopilación de información. La cuestión que afronta el Big Data es, por tanto, el modo en el que esa ingente cantidad de datos (lo que en muchas ocasiones es nombrado como el data deluge, el diluvio universal de datos) que se almacenan y producen diariamente pueda ser gestionado de tal modo que se convierta en conocimiento; y con el conocimiento, en valor [5].

El concepto de Big data apenas se empieza a conocer en el medio organizacional, por lo que se ignoran aspectos relevantes del tema. Las empresas desconocen qué hacer con el gran volumen de datos que generan y les llega de otras. Uno de los problemas del Big Data es la forma como crece cada día en volumen, velocidad y variedad, debido al desarrollo algorítmico de las nuevas tecnologías.

Actualmente, se estudian las formas de acelerar consultas, algoritmos de compresión, obtener informes que toman menor tiempo en generarse para que sean útiles en la toma de decisiones, almacenar datos ahorrando el máximo espacio y sin afectar al rendimiento.

Existen varias arquitecturas propuestas de Big data, pero Microsoft propone uno que consta de cuatro componentes [6]:

- **Fuentes de Datos (Data Sources)**: Estos datos presentan tres características que definen el Big Data: volumen, velocidad y variedad. Este tipo de datos son independientes del contexto.
- **Transformación de Datos (Data Transformation)**: Consta de cuatro subetapas. Cada una de ellas puede tener su etapa de preprocesamiento específico, creación de metadatos, utilizar diferentes infraestructuras de datos especializados de acuerdo con sus necesidades, tener su propia privacidad, así como sus propias políticas.
- **Infraestructura de Datos (Data Infrastructure)**: Se considera la infraestructura como un paquete de almacenamiento de datos o software de base de datos, servidores, almacenamiento y redes utilizados en apoyo de las funciones de transformación de datos y de almacenamiento de datos según sea necesario.
- **Uso de Datos (Data Usage)**: Este aspecto depende del usuario y sus necesidades particulares, pero estos se pueden presentar en diferentes formatos y bajo ciertas consideraciones de seguridad.

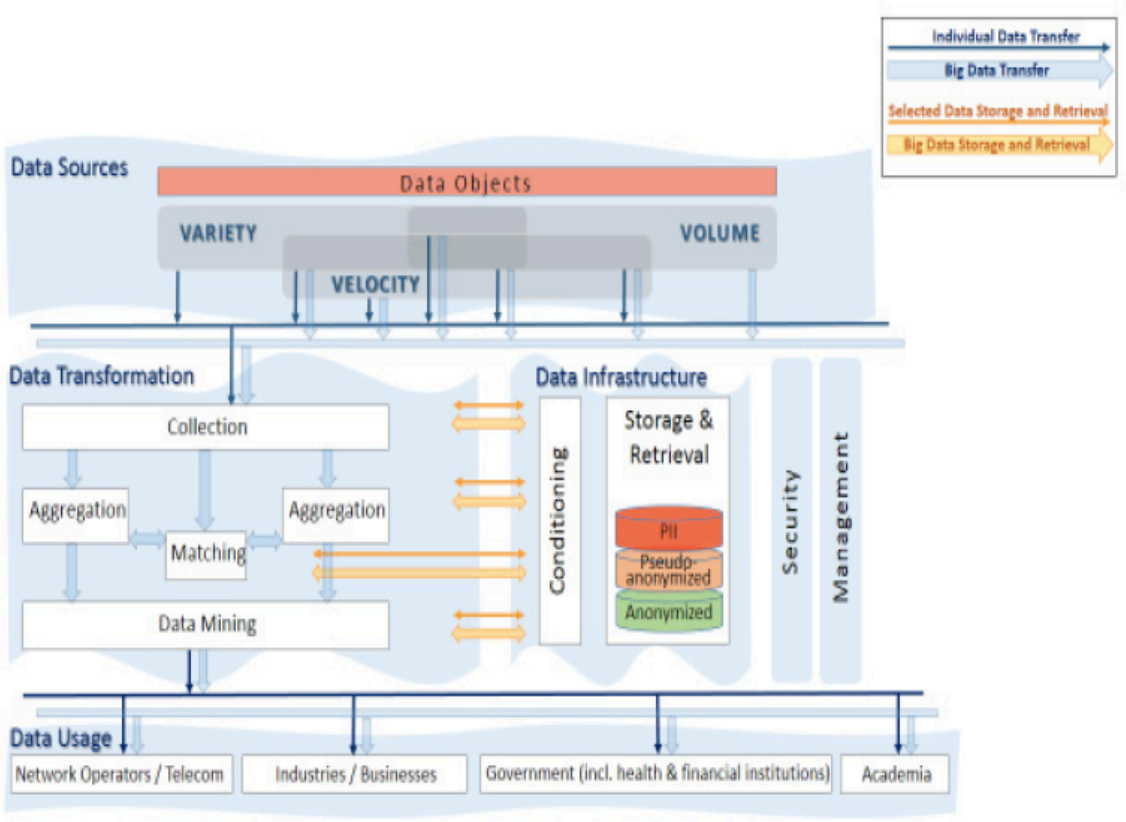


Figura 12: Arquitectura Big Data propuesta por Microsoft

Por otro lado, han surgido nuevos Sistemas Gestores de Bases de Datos NoSQL (depósitos llave-valor, basados en documentos, tabular y orientados a grafos). La mayoría de

estos SGBO son **Document-oriented databases** donde la información es un documento con formato JSON o XML. Ejemplos: MongoDB y CouchDB [5].

Además, podemos mencionar a la base de datos orientada a objetos ligeros NoSQL **Sterling** para .Net 4.0 , Silverlight 4 y 5, y Windows Phone 7 que funciona con sus estructuras de clase existentes. Sterling es compatible con LINQ completo al objeto de consultas sobre claves e índices para una rápida recuperación de información de grandes conjuntos de datos. Sus características son: no intrusividad, ligero, flexible y portátil.

Sterling, aún es un proyecto, por lo que actualmente, no hay tiempo de respuesta garantizado para correcciones de errores y no hay línea de soporte técnico. Una de las ventajas es que el cliente tiene 100% de acceso sin costo alguno, a tomar la fuente, modificarla, arreglarla, o utilizarla. No hay cuota comercial para este proyecto. Se esperan futuras versiones [7].

Sin embargo, es importante destacar que aún existen muchas preguntas abiertas y áreas de investigación activas alrededor de los Sistemas NoSQL. Por ejemplo, se tiene que los almacenes **clave-valor** presentan problemas para realizar consultas complejas, algunas de las bases de datos **orientadas a grafos** no pueden realizar particionamiento, y la mayoría de las implementaciones de bases de datos **documentales** y tipo **BigTable** son incapaces de efectuar uniones o transacciones que abarquen varias filas o documentos [5].

Por ello, no se descarta que en el **futuro**, ambos sistemas, bases de datos relacionales y almacenes de datos NoSQL, se complementen y que la **tendencia** sea la construcción de sistemas híbridos compuestos por múltiples almacenes de datos cada uno de ellos basado en diferentes principios [5].

2.7 JUSTIFICACIÓN DE LA UNIDAD DIDÁCTICA

Una base de datos orientada a objetos es algo muy diferente a una base de datos relacional. Las bases de datos relacionales se han estado usando durante mucho tiempo, pero dada la creciente utilización de los lenguajes orientados a objetos como **Java**, una Base de Datos Orientada a Objetos es un mecanismo muy útil en el desarrollo de proyectos.

Las bases de datos relacionales tienen un estándar fuerte como es el SQL mientras que las Bases de Datos OO pueden ser consultadas de diferentes formas. Por ello, para trabajar la unidad de trabajo propuesta en la proyección didáctica titulada ***"Persistencia de los objetos en bases de datos orientadas a objetos"***, vamos a utilizar **db4o**, la cual es una base de datos con licencia GPL (Licencia Pública General de GNU) y puede utilizarse sin problema para desarrollo, uso interno o asociada a otras herramientas GPL.

Db4o es una verdadera base de objetos nativa de Java y la distribución es un único .zip.

Una de las características que ofrecen muchas Bases de Datos Orientadas a Objetos es que se integran muy bien en el lenguaje de programación. En el caso de db4o se utilizará una ***API*** (Application Program Interface) que ofrece para manejar la base de datos. La

documentación del API viene en formato ***JavaDoc*** y una de las interfaces más importantes es ***com.db4o.ObjectContainer*** [3].

A lo largo de la unidad se verán distintas formas de consultar la BD dependiendo de las características del proyecto:

- **Query By Example (QBE)**: La forma más básica de consultar la base de datos. Es sencilla puesto que funciona presentando un ejemplo y se recuperarán los datos que coincidan con el mismo.
- **Native Queries (NQ)**: Son consultas nativas. Es la interfaz principal de la base de datos y aconsejado por los desarrolladores db4o.
- **SODA (Simple Object Data Access)**: Es la API interna y puede utilizarse para generar consultas dinámicamente. Es mucho más potente y rápida que QBE y NQ, dado que estos dos tipos de consultas deben ser traducidas a SODA para ejecutarse.

3. PROYECCIÓN DIDÁCTICA

3.1 INTRODUCCIÓN

Una de las técnicas que todo docente debe dominar es la de organizar su trabajo de una forma sistemática y científica que se apoye en fundamentos que las Ciencias de la Educación sostienen como buenos para la práctica docente.

Una herramienta que indiscutiblemente puede ayudar al profesor a desarrollar su tarea de una forma más profesional es la Programación Didáctica o de Aula, en la que se concrete un plan de actuación abierto y flexible, se adecue a un determinado contexto, y sea viable en cuanto al tiempo espacio y recursos disponibles, con el objetivo de minimizar la necesidad de improvisación en el aula y atender las características específicas del alumnado a través de la Atención a la Diversidad.

En esta segunda parte del Trabajo Final de Máster diseñaremos una planificación de la labor docente en el aula, en la que se concretan los objetivos de aprendizaje, desarrollados a través de una serie de contenidos, donde se proponen actividades de enseñanza-aprendizaje, las cuales serán posteriormente evaluadas a través de unos criterios de evaluación.

3.2 PRESENTACIÓN DEL MÓDULO Y DE LA UNIDAD DIDÁCTICA

La proyección didáctica que nos ocupa se va a llevar a cabo para el módulo titulado "**Programación**", impartido en el primer curso de los siguientes Ciclos Formativos de Grado Superior:

1. **Desarrollo de Aplicaciones Multiplataforma (DAM)**
2. **Desarrollo de Aplicaciones Web (DAW)**

Los **CFGS DAM** y **DAW** pertenecen a la **Familia Profesional de Informática** y tiene una duración de **2000 horas**, equivalente a un máximo de **5 trimestres** de **formación en el centro educativo**, más la **formación en el centro de trabajo** correspondiente y la realización del **Proyecto Integrado**.

Para el desarrollo del Trabajo Final de Máster hemos considerado el CFGS Desarrollo de Aplicaciones Multiplataforma.

El módulo de **Programación**, está dotado de una carga lectiva de **256 horas** distribuidas en **8 horas semanales** durante aproximadamente **32 semanas**. En cada semana se imparten **4 sesiones** de **2 horas** cada una.

La unidad didáctica que vamos a desarrollar se corresponde con la última unidad del módulo y se titula "***Persistencia de los objetos en bases de datos orientadas a objetos***" con una duración de **12 h**.

3.3 ANÁLISIS DEL ENTORNO DEL CENTRO

Vamos a contextualizar la proyección didáctica para un Instituto de Educación Secundaria Público, en concreto, **IES Saladillo** de Algeciras, y para un nivel educativo específico, por lo que es imprescindible tener presente las características socio-económicas del entorno, el nivel cultural y formativo de la familia, el sistema de comunicaciones, la climatología, el entorno productivo así como las características del centro educativo.

3.3.1 CONTEXTO EXTERNO

- **Características socio-económicas del entorno**:

  El centro se encuentra en el extremo sudeste de la provincia de Cádiz en el límite de la provincia de Málaga. La zona ha experimentado un importante aumento en el número de habitantes durante los últimos 15 años debido a la creciente inmigración. El centro se ubica en un entorno urbano, habitado por población con un nivel socioeconómico bajo, siendo la principal actividad económica el sector servicios.

- **Nivel cultural y formativo de la familia**:

  El nivel cultural de las familias de la ciudad puede considerarse medio, aunque hay una gran bolsa de inmigración que plantean problemas de comunicación y de entendimiento con el resto de la población, siendo esta inmigración desde un alto nivel adquisitivo hasta otro tipo de un nivel muy bajo.

- **Sistema de comunicaciones**:

  Al tratarse de una enseñanza post obligatoria, el alumnado puede proceder de cualquier población colindante. Sin embargo, el transporte público es escaso. Un porcentaje muy mínimo procede de otras poblaciones, donde el único medio de transporte es el coche particular.

- **Climatología**:

  Puesto que la climatología no es adversa, en general, el absentismo escolar es mínimo y el que se produce no es por este motivo.

- **Entorno Empresarial o Productivo**:

  Algeciras es un municipio del Campo de Gibraltar con 118.920 habitantes censados a fecha del 2015. Está ubicado a 21 km del Peñón de Gibraltar, zona donde se han desarrollado numerosas empresas relacionadas con la programación que demandan programadores en lenguajes como Java, C++, C#, PHP o Phyton. Además, a los

trabajadores se les paga en libras, cuyo valor es superior al euro. Por tanto, los técnicos superiores de los ciclos de DAM o DAW pueden encontrar una salida laboral a muy poca distancia, como programador, comercial de aplicaciones informáticos, etc.

3.3.2 CONTEXTO INTERNO

Se ha considerado como referente un centro docente de tipo público, con una oferta educativa LOE-LOMCE, con un buen equipamiento en lo que al área se refiere y situado en un municipio de la provincia de Cádiz. Se imparte los siguientes cursos:

- Educación Secundaria Obligatoria.
- Bachillerato de Ciencias y Tecnologías.
- Bachillerato de Humanidades y Ciencias Sociales.
- Formación Profesional Básica de Informática y Comunicaciones.
- CFGM Sistemas Microinformáticos y Redes.
- CFGS Desarrollo de Aplicaciones Multiplataforma.

3.4. CARACTERÍSTICAS DEL ALUMNADO

En el Ciclo Superior de Administración de Sistemas Informáticos el alumnado presenta una edad avanzada, entre 19 y 40 años. En cuanto al nivel educativo aproximadamente un **60% del alumnado procede del CFGM Sistemas Microinformáticos y Redes**, otro **30% de Bachillerato** y un **10% proceden de la Universidad o mundo laboral**. A excepción de los alumnos procedentes de Bachillerato, el resto ha debido hacer una prueba de acceso.

Como ya comentábamos anteriormente, el Ciclo Formativo en cuestión, al tratarse de un tipo de enseñanza postobligatoria y al no ser ofertado en otros centros públicos docentes de la ciudad y alrededores, tiene una parte del alumnado procedente de diversas zonas próximas a Algeciras, como Los Barrios o San Roque, dando lugar a una importante diversidad cultural y social.

Este tipo de alumnado presenta una actitud muy responsable y comprometida.

Además, el ver cercano el momento de la finalización de sus estudios y la realización del módulo de “Formación en Centros de Trabajo” motiva con creces a los alumnos, pues para algunos es su primer contacto con el mundo empresarial, y para todos es su ocasión de poner en práctica los conocimientos adquiridos, en un entorno “real” y con el respaldo del centro formativo.

3.5 MARCO LEGISLATIVO

A continuación, situamos nuestra proyección didáctica dentro del siguiente marco legislativo y educativo:

Atendiendo al **Derecho a la Educación**, destacamos:

- **Constitución Española de 1978**, en la que a través del ***Artículo 27***, atribuye a todos los españoles el derecho a la educación. Garantiza las libertades de enseñanza, de cátedra y de creación de Centros, así como el derecho a recibir formación religiosa y moral de acuerdo con las propias convicciones.
- **Estatuto de Autonomía para Andalucía**, aprobado por la Ley Orgánica 2/2007, de 19 de marzo, de reforma del Estatuto de Autonomía para Andalucía, establece, en su ***Artículo 52***, las competencias correspondientes a la Comunicad Autónoma en materia de enseñanza no universitaria. El ***Artículo 10*** garantiza el acceso de todos los andaluces a una educación permanente y de calidad que les permita su realización personal y social.

Atendiendo al **Sistema Educativo** destacamos la siguiente normativa:

- **Real Decreto 1128/2003**, de 5 de septiembre, se regula el Catálogo Nacional de Cualificaciones Profesionales. La finalidad de este catálogo es posibilitar la integración de las ofertas de formación profesional, adecuándolas a las características y demandas del sistema productivo, promover la formación a lo largo de la vida y facilitar la movilidad de los trabajadores, así como la unidad del mercado de trabajo. Y todo ello, garantizando los niveles básicos de calidad derivados de la permanente observación y análisis del sistema productivo, así como de las demandas de la sociedad.
- **Ley Orgánica de la Educación 2/2006**, de 3 de mayo de 2006. En su **Artículo 1**, regula el derecho a una educación de calidad, equitativa, flexible, orientativa, motivadora y concebida como un aprendizaje permanente a lo largo de toda la vida.
- **Ley Orgánica para la Mejora de la Calidad Educativa 8/2013**, de 9 de diciembre. Es una ley con carácter de ley orgánica que modifica la Ley Orgánica 2/2006, de 3 de mayo, de Educación (LOE), seis artículos y una disposición adicional de la Ley Orgánica 8/1985, de 3 de julio, que establece el Derecho a la Educación (LODE).
- **Ley de Educación de Andalucía (LEA) 17/2007, de 10 de diciembre.** En el **Artículo 5**, se establece que es necesario abordar los temas transversales para ciclo formativo, aunque este nivel de enseñanza sea postobligatorio. En su **Artículo 68**, establece que la formación profesional comprende el conjunto de acciones formativas que capacitan para el desempeño cualificado de las diversas profesiones, el acceso al empleo y la participación activa en la vida social, cultural y económica. El **Artículo 69**, regula que todos los ciclos formativos de formación profesional inicial incluirán en su currículo además de los módulos asociados a competencias profesionales, la formación relativa a prevención de riesgos laborales, tecnologías de la información y la comunicación, fomento de la cultura emprendedora, creación y gestión de empresas, y autoempleo y conocimiento del mercado de trabajo y de las relaciones laborales.

- **Real Decreto 450/2010**, de 16 de abril, por el que se establece el Título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma y se fijan sus enseñanzas mínimas.
- **Orden de 16 de junio de 2011**, por la que se desarrolla el currículo correspondiente a los títulos de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma.

3.6 UNIDAD DIDÁCTICA: PERSISTENCIA DE LOS OBJETOS EN BASES DE DATOS ORIENTADAS A OBJETOS

3.6.1 OBJETIVOS

Los objetivos educativos expresan el nivel de desarrollo que se espera alcancen los alumnos y alumnas como consecuencia de la intervención educativa y responden a la cuestión **¿para qué enseñar?**.

La meta educativa no debe ser que los alumnos aprendan meros conocimientos, sino que sean capaces de manejarse con ellos. Toda intervención educativa persigue el desarrollo integral del individuo, por lo que el principal objetivo de la educación es el desarrollo de capacidades. Son el referente indispensable para la evaluación del grado de los diferentes tipos de capacidades adquiridos por los alumnos. Según el grado de concreción se suele hablar de los siguientes tipos de Objetivos:

- **Objetivos Generales de Formación Profesional**.
- **Objetivos Generales de Ciclo Formativo**.
- **Resultados de Aprendizaje**.
- **Objetivos Didácticos**.

3.6.1.1 OBJETIVOS GENERALES DE FORMACIÓN PROFESIONAL

Se encuentran establecidos en el **Artículo 40 de la LOE** y consiste en preparar a los alumnos y alumnas para la actividad en un campo profesional y facilitar su adaptación a las modificaciones laborales, así como contribuir a su desarrollo personal, al ejercicio de la ciudadanía democrática y al aprendizaje permanente.

3.6.1.2 OBJETIVOS GENERALES DE CICLO FORMATIVO

Establecen las **capacidades** que se espera que hayan adquirido los alumnos como consecuencia del proceso de enseñanza al final de cada ciclo formativo. Son, en definitiva, los elementos curriculares que concretan los fines educativos para la FP según la familia profesional. Estos objetivos se refieren a capacidades globales que se trabajarán desde todos los elementos del currículo. Los elementos generales del Ciclo Formativo de Grado Superior Desarrollo de Aplicaciones Multiplataforma, se concretan en:

- **Real Decreto 450/2010**, de 16 de abril, por el que se establece el Título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma y se fijan sus enseñanzas mínimas.
- **Orden de 16 de junio de 2011**, por la que se desarrolla el currículo correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma.

Los Objetivos Generales del Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma aparecen reflejados en el **Artículo 3** de la Orden de 16 de junio de 2011.

La formación del módulo de **Programación** contribuye a alcanzar los objetivos generales del ciclo formativo que se relacionan a continuación:

e) Seleccionar y emplear lenguajes, herramientas y librerías, interpretando las especificaciones para desarrollar aplicaciones multiplataforma con acceso a bases de datos.

j) Seleccionar y emplear técnicas, lenguajes y entornos de desarrollo, evaluando sus posibilidades, para desarrollar aplicaciones en teléfonos, PDA y otros dispositivos móviles.

q) Seleccionar y emplear lenguajes y herramientas, atendiendo a los requerimientos, para desarrollar componentes personalizados en sistemas ERP-CRM.

w) Identificar los cambios tecnológicos, organizativos, económicos y laborales en su actividad, analizando sus implicaciones en el ámbito de trabajo, para mantener el espíritu de innovación.

3.6.1.3 RESULTADOS DE APRENDIZAJE

Indican las capacidades que los alumnos deben haber adquirido en cada materia al finalizar el ciclo correspondiente. Son objetivos que adoptan el mismo estilo de formulación que los objetivos generales, pero añadiendo una referencia explícita a los contenidos. Los objetivos del módulo Programación se concretan en términos de Resultados de Aprendizaje en la Orden 16 de junio de 2011, de conformidad a lo dispuesto en el RD 450/2010.

A continuación, presentamos los Resultados de Aprendizaje del módulo de **Programación**:

1. Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado.
2. Escribe y prueba programas sencillos, reconociendo y aplicando los fundamentos de la programación orientada a objetos.
3. Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.
4. Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.
5. Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.
6. Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.

7. Desarrolla programas aplicando características avanzadas de los lenguajes orientados a objetos y del entorno de programación.
8. **Utiliza bases de datos orientadas a objetos, analizando sus características y aplicando técnicas para mantener la persistencia de la información.**
9. Gestiona información almacenada en bases de datos relacionales manteniendo la integridad y consistencia de los datos.

Nuestra unidad didáctica está relacionada con el **Resultado de Aprendizaje 8**.

3.6.1.4 OBJETIVOS DIDÁCTICOS

Se formulan al elaborar las unidades didácticas de las programaciones y hacen referencia a los contenidos seleccionados para cada materia.

Del **Resultado de Aprendizaje 8**, **"Utiliza bases de datos orientadas a objetos, analizando sus características y aplicando técnicas para mantener la persistencia de la información",** extraemos los siguientes objetivos didácticos:

- Identificar las características de las bases de datos orientadas a objetos.
- Analizar su aplicación en el desarrollo de aplicaciones mediante lenguajes orientados a objetos.
- Instalar sistemas gestores de bases de datos orientados a objetos.
- Clasificar y analizar los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.
- Crear bases de datos y las estructuras necesarias para el almacenamiento de objetos.
- Programar aplicaciones que almacenan objetos en las bases de datos creadas.
- Realizar programas para recuperar, actualizar y eliminar objetos de las bases de datos.
- Realizar programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.

3.6.2 CONTRIBUCIÓN A LAS COMPETENCIAS PROFESIONALES

Considerando que en las enseñanzas propias de los ciclos formativos se ha de tener en cuenta las necesidades del entorno productivo, justificamos la necesidad de recoger el nivel de contribución del módulo del perfil profesional que el sistema productivo necesita.

3.6.3 CONTRIBUCIÓN A LA COMPETENCIA GENERAL

Basándonos en el **Artículo 4** del Real Decreto 450/2010, de 16 de abril, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma y se

fijan sus enseñanzas mínimas, podemos establecer la correspondencia del módulo profesional de Programación con la siguiente competencia general:

- Desarrollar, implantar, documentar y mantener aplicaciones informáticas multiplataforma, utilizando tecnologías y entornos de desarrollo específicos, garantizando el acceso a los datos de forma segura y cumpliendo los criterios de usabilidad y calidad exigidas en los estándares establecidos.

3.6.4 CONTRIBUCIÓN A LAS COMPETENCIAS PROFESIONALES, PERSONALES Y SOCIALES

Según lo dispuesto en la Orden de 16 de junio de 2011, la formación del módulo de **Programación** contribuye a alcanzar las competencias profesionales, personales y sociales del título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma que se relacionan a continuación:

a) Configurar y explotar sistemas informáticos, adaptando la configuración lógica del sistema según las necesidades de uso y los criterios establecidos.

e) Desarrollar aplicaciones multiplataforma con acceso a bases de datos utilizando lenguajes, librerías y herramientas adecuados a las especificaciones.

f) Desarrollar aplicaciones implementando un sistema completo de formularios e informes que permitan gestionar de forma integral la información almacenada.

i) Participar en el desarrollo de juegos y aplicaciones en el ámbito del entretenimiento y la educación empleando técnicas, motores y entornos de desarrollo específicos.

j) Desarrollar aplicaciones para teléfonos, PDA y otros dispositivos móviles empleando técnicas y entornos de desarrollo específicos.

t) Establecer vías eficaces de relación profesional y comunicación con sus superiores, compañeros y subordinados, respetando la autonomía y competencias de las distintas personas.

w) Mantener el espíritu de innovación y actualización en el ámbito de su trabajo para adaptarse a los cambios tecnológicos y organizativos de su entorno profesional.

En concreto, nuestra **unidad didáctica** contribuye a alcanzar directamente las competencia **e)**, **f)**, **t)** y **w).**

3.6.5 LÍNEAS DE ACTUACIÓN

Las líneas de actuación en el proceso de enseñanza-aprendizaje que permiten alcanzar los objetivos del módulo y de la unidad didáctica versarán sobre:

- La interpretación y aplicación de los principios de la programación orientada a objetos.

- La evaluación, selección y utilización de herramientas y lenguajes de programación orientadas a objetos.
- La utilización de las características específicas de lenguajes y entornos de programación en el desarrollo de aplicaciones informáticas.
- La identificación de las funcionalidades aportadas por los sistemas gestores de bases de datos y su incorporación a los programas desarrollados.
- La documentación de los programas desarrollados.

3.6.6 CONTENIDOS DEL MÓDULO

Los contenidos responden a la cuestión **¿qué enseñar?**, y pueden ser definidos como el conjunto de saberes, hechos, conceptos, habilidades, actitudes, en torno a los cuales se organizan las actividades desarrolladas en el aula.

La selección de contenidos en una programación y unidad didáctica requiere de una cuidadosa reflexión, puesto que su elección está mediatizada por los objetivos que se pretenden conseguir en el curso al que va dirigida. Además, los contenidos deben cumplir una serie de requisitos que pueden resumirse en **significado**, **funcionalidad** y **adecuación del alumno/a**.

3.6.6.1 SECUENCIACIÓN Y TEMPORALIZACIÓN DE UNIDADES DIDÁCTICAS

Como indicábamos en el apartado 3.2, El módulo de **Programación**, está dotado de una carga lectiva de **256 horas** distribuidas en **8 horas semanales** durante aproximadamente **32 semanas**.

A continuación, se presenta la secuencia de unidades de trabajo en que se ha estructurado el módulo profesional, su relación con los bloques de contenido, así como la duración estimada de cada una de las unidades. La última parte del Trabajo Final de Máster está dedicado a la ***Unidad Didáctica 11: Persistencia de los objetos en bases de datos orientadas a objetos.***

Partiendo del análisis de la Orden de 16 de junio de 2011, del entorno productivo de la zona, de los conocimientos previos de los alumnos, así como de la capacidad de éstos para poder asimilar nuevos conocimientos hemos considerado 11 unidades de trabajo más una de presentación.

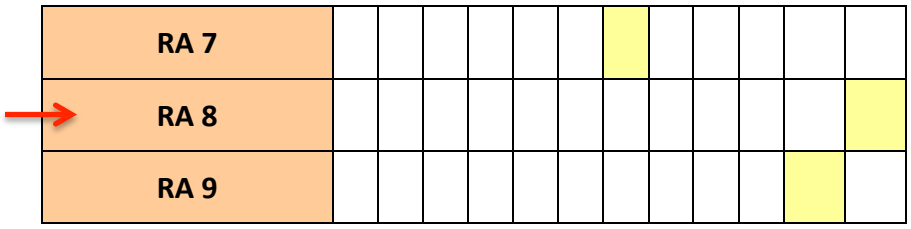
Unidades Didácticas Secuenciadas	Duración
UT 0: Presentación del módulo de Programación.	**2 h**
UT 1: Elementos de un programa informático.	**22 h**

UT 2: Programación Orientada a Objetos. Objetos.	**20 h**
UT 3: Estructuras básicas de control.	**26 h**
UT 4: Programación Orientada a Objetos. Clases.	**28 h**
UT 5: Estructuras de Almacenamiento.	**34 h**
UT 6: Mecanismos de abstracción: clases, paquetes, subclases e interfaces.	**32 h**
UT 7: Clases genéricas y control de excepciones.	**24 h**
UT 8: Lectura y escritura de información	**22 h**
UT 9: Interfaces gráficas de usuario.	**18 h**
UT 10: Bases de Datos Relacionales.	**16 h**
UT 11: Persistencia de los objetos en bases de datos orientadas a objetos.	**12 h**
TOTAL	**256 h**

3.6.6.2 JUSTIFICACIÓN DE UNIDADES DIDÁCTICAS

Las unidades didácticas tienen como fin último la consecución de los Resultados de Aprendizaje en que se han expresado los objetivos generales del módulo. La relación que existirá entre unas y otras será la siguiente:

RA \ UT	0	1	2	3	4	5	6	7	8	9	10	11
RA 1												
RA 2												
RA 3												
RA 4												
RA 5												
RA 6												



Como ya vimos con anterioridad, el resultado de aprendizaje que se ha utilizado para expresar los objetivos y criterios de evaluación de nuestra unidad didáctica es el número 8.

3.6.6.3 CONTENIDOS DE LA UNIDAD DIDÁCTICA

A continuación, llevamos a cabo el detalle de los contenidos de la **Unidad de Trabajo 11: *Persistencia de los objetos en bases de datos orientadas a objetos***, relacionados con los objetivos didácticos y los criterios de evaluación:

OBJETIVOS DIDÁCTICOS	CONTENIDOS	CRITERIOS DE EVALUACIÓN
a) Identificar las características de las bases de datos orientadas a objetos.	**1. Bases de Datos Orientados a Objetos.** **1.1 Características.** **1.2 Análisis de su aplicación**	a) Se han identificado las características de las bases de datos orientadas a objetos.
b) Analizar su aplicación en el desarrollo de aplicaciones mediante lenguajes orientados a objetos.		b) Se ha analizado su aplicación en el desarrollo de aplicaciones mediante lenguajes orientados a objetos.
c) Instalar sistemas gestores de bases de datos orientados a objetos.	**2. Instalación del gestor de bases de datos.** **2.1 Métodos para la gestión de información.**	c) Se han instalado sistemas gestores de bases de datos orientados a objetos.
d) Clasificar y analizar los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.		d) Se han clasificado y analizado los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.
e) Crear bases de datos y las estructuras necesaria para el almacenamiento de objetos.	**3. Creación de bases de datos.** **3.1 Creación estructuras para almacenar objetos.** **3.2 Programación de aplicaciones para almacenar objetos en BD.**	e) Se han creado bases de datos y las estructuras necesaria para el almacenamiento de objetos.
f) Programar aplicaciones que almacenen objetos en las bases de datos creadas.		f) Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.

g) Realizar programas para recuperar, actualizar y eliminar objetos de las bases de datos.	**4. Mecanismos de consulta.** **5. El lenguaje de consultas, sintaxis, expresiones, operadores.** **6. Recuperación, modificación y borrado de información.**	g) Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.
h) Realizar programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.	**7. Tipos de datos objeto.** **7.1 Atributos** **7.2 Métodos** **8. Tipos de datos colección**	h) Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.

3.6.7 METODOLOGÍA

La Metodología didáctica responde a la cuestión **¿cómo enseñar?** y hace referencia al conjunto de decisiones que deben ser tomadas a fin de orientar el desarrollo en el aula de los procesos de enseñanza y aprendizaje. Las opciones metodológicas estarán orientadas al aprendizaje **significativo**, **constructivista** y **funcional** de los diferentes contenidos.

Los aspectos metodológicos que se pretenden aplicar en el módulo Programación descansan en la idea de que el alumno/a se considere parte activa de la actividad docente, de manera que se pretende involucrarlo en el proceso de asimilación de nuevos conceptos y adquisición de capacidades, no como un mero contenedor de éstas, sino como un productor directo de estos conocimiento y habilidades en sí mismo.

El docente también debe orientar el trabajo escolar de sus alumnos, proporcionando las indicaciones necesarias para que los alumnos puedan resolver los problemas que el estudio les plantea. A lo largo de las unidades didácticas que desarrollamos en el módulo, fomentaremos los hábitos de tenacidad constancia, laboriosidad, etc. Un aspecto importante de esta función orientadora es decidir qué actitudes se debe conseguir en los estudiantes, cuáles deben modificarse y cómo reforzar las positivas.

3.6.7.1 TIPOS DE ACTIVIDADES ENSEÑANZA-APRENDIZAJE

Las actividades de enseñanza-aprendizaje son la manera activa y ordenada de llevar a cabo las propuestas metodológicas o experiencias de aprendizaje. Las actividades hacen referencia a las tareas realizadas por los alumnos y alumnas con la finalidad de adquirir determinados aprendizajes. Por tanto, en toda actividad se maneja cierta información, procedente de unas determinadas fuentes, mediante unos procedimientos concretos y en relación a unas determinadas metas.

A lo largo de las diferentes unidades didácticas del módulo de Programación llevamos a cabo tres tipos de actividades:

1. **Actividades Complementarias y Extraescolares**: Se realizarán fuera del aula y estarán relacionadas con los objetivos y contenidos establecidos en las unidades didácticas. Como ejemplos, podemos citar:
    - Visita al Centro de Proceso de Datos de Diputación de Cádiz (EPICSA)
    - Visita a feria de nuevas tecnologías de Informática y Comunicación en Málaga, de forma que los alumnos y alumnas puedan observar, desde un primer plano, los últimos productos que aparecen en el mercado.

2. **Actividades con el Departamento**:
    - Las actividades extraescolares ya mencionadas.
    - Biblioteca del Departamento de Informática.
    - Plataforma Educativa Moodle.

3. **Actividades con los Alumnos**: De todas las tipologías aportadas por los distintos autores, la propuesta de Tyler y Wheeler es la más apropiada, puesto que materializa claramente los principios psicopedagógicos y didácticos derivados de la teoría constructivista del aprendizaje. Los tipos de actividades que vamos a llevar a cabo en cada unidad didáctica son los siguientes:
    a) Actividades **Presentación-Motivación**, que son dirigidas por el profesor para que el alumnado descubra progresivamente la estructura de los contenidos. Este tipo de actividades las abordaremos al inicio de cada unidad de trabajo.
    b) Actividades de **evaluación de los conocimientos previos**, para obtener información acerca de lo que saben y qué procedimientos, destrezas y habilidades han desarrollado los alumnos sobre un tema concreto. Este tipo de actividad se realizará al inicio de curso a través de un cuestionario inicial y al iniciar cada unidad a través de una tormenta de ideas.
    c) Actividades de **desarrollo** de los contenidos, los cuales permiten al alumno/a la adquisición de nuevos contenidos. Se realizarán algunos supuestos prácticos de la relación de ejercicios propuestos para la unidad de trabajo y relacionados con los contenidos explicados en la sesión. El docente ayudará a resolver los primeros.
    d) Actividades de **consolidación**, con las que los alumnos contrastan las nuevas ideas con las previas y aplican los nuevos aprendizajes. Continuando con el supuesto anterior en que están resolviendo los ejercicios de una relación, una vez que el profesor ayuda a resolver algunas actividades, dejará al alumno que resuelva el resto y, de esta forma, consolide sus conocimientos.
    e) Actividades de **síntesis-resumen**, que permiten a los alumnos establecer relación entre los distintos contenidos aprendidos, así como la contrastación con aquellos conocimientos que ya tenían. Al finalizar una unidad de trabajo, se elaborará un resumen de los nuevos contenidos aprendidos de manera individual, el profesor elaborará junto con los alumnos un mapa conceptual en pizarra.

f) Actividades de **recuperación** y **ampliación**, las cuales se afrontarán de forma síncrona. Una vez resueltas las actividades propuestas en la relación de ejercicios para cada unidad de trabajo, el docente entregará actividades de refuerzo a aquellos alumnos que no hayan alcanzado los conocimientos trabajados, y el resto de alumnos continuarán construyendo conocimientos a través de las actividades de ampliación.

g) Actividades de **evaluación**, que pretenden proporcionar una justificación acerca de lo aprendido por los alumnos/as. Por tanto, se debe tomar en consideración lo establecido en los criterios de evaluación propuestos. Al finalizar cada unidad de trabajo, se realizará una prueba específica de carácter teórico-práctico que permitan evaluar a los alumnos a nivel conceptual y procedimental. Este tipo de actividades no sólo evaluará el trabajo del alumnado, sino que también le permite conocer al docente si su metodología empleada durante su enseñanza es la correcta, pudiéndola modificar a tiempo.

3.6.7.2 ESTILO DE METODOLOGÍA

En esta sección, desarrollaremos cuál sería el estilo de metodología seguida en una determinada sesión.

Si se tratara de la **primera sesión** de una unidad de trabajo, se comenzaría introduciendo los contenidos de la misma. Además, se inculcará a los alumnos la importancia de los conocimientos a adquirir en la unidad, con respecto a la demanda de los mismos en el sistema productivo y laboral. También, indagaremos sobre los conocimientos previos de los alumnos a través de un tormenta de ideas para saber desde dónde partir.

Para el caso de una **sesión intermedia**, el docente resuelve en primer lugar las dudas relacionadas con la sesión anterior. Seguidamente, se avanza con los nuevos contenidos de la unidad de trabajo y se realizan aquellas actividades de la relación de ejercicios vinculadas a los nuevos conocimientos adquiridos en la sesión, de los cuales los primeros ejercicios serán resueltos por el docente, a través del equipo y el proyector, salvo situaciones excepcionales en las que internet no funcione correctamente y la solución sea demasiado extensa que se entregará la documentación en papel.

Una vez realizadas las actividades de desarrollo, los alumnos y alumnas consolidarán sus conocimientos continuando con los ejercicios de la relación. Tas concluir el tiempo estimado para la realización de cada ejercicio, un alumno lo resolverá a través de la herramienta de monitorización ***iTALC*** y del proyector, y se estudiarán posibles soluciones alternativas por parte del resto de alumnos y del docente.

Finalmente, si se tratara de una de las **sesiones finales**, el docente, con la ayuda de los alumnos, elaborará un esquema conceptual y gráfico desde el equipo y proyectado en pantalla blanca, sobre los nuevos conocimientos adquiridos, de cara al cuestionario de evaluación de la unidad de trabajo que se realizará en la siguiente sesión. Durante la realización del mismo, el docente resolverá dudas del alumnado. Una vez concluida esta actividad, el docente hará entrega a los alumnos/as actividades de refuerzo/ampliación

según corresponda. El docente estimará el tiempo necesario para la realización de las mismas.

3.6.7.3 AGRUPAMIENTOS

Las investigaciones en Psicología Educativa en los últimos años han puesto de manifiesto que la interacción entre los alumnos constituye un factor muy importante, puesto que favorece el desarrollo de la socialización, tiene efectos positivos en el desarrollo intelectual e incrementa la motivación del alumnado.

No existe un tipo de agrupamiento ideal, sino que estará en función de los objetivos que se pretendan conseguir con la actividad de enseñanza-aprendizaje.

Seguidamente, estudiaremos el tipo de agrupamientos que se utilizarán en nuestra unidad, así como el papel del profesor como dinamizador del grupo-clase:

- **Trabajo individual**: El trabajo individual es el que posibilita un mayor grado de personalización de la enseñanza, adaptándose al ritmo y posibilidades de cada alumno. Resulta muy eficaz para afianzar conceptos y al profesor le permite realizar un seguimiento más minucioso del proceso de aprendizaje de cada alumno, permitiéndole comprobar el nivel de comprensión alcanzado y detectar dónde se encuentran las dificultades. Las tareas individuales en la unidad consistirán en:
    - Realizar las actividades propuestas en cada unidad de trabajo, tanto de consolidación como de refuerzo o ampliación.
    - Relacionar las actividades de evaluación de conocimientos previos y los recientemente aprendidos en una determinada unidad de trabajo.
    - Relacionar y memorizar datos y conceptos.
    - La reflexión personal.
    - Preparar trabajos en la fase individual de recogida de datos.
    - Preparar una explicación oral a los compañeros sobre actividades, trabajos, prácticas…
    - Autocorrección de ejercicios.

- **Trabajo en parejas:** En este tipo de agrupamiento se potencian al máximo las posibilidades de comunicar, compartir y realizar trabajos simultáneamente, contando en todo momento con la participación activa de todos los miembros del grupo. En la medida de lo posible, realizaremos agrupamientos por parejas a alumnos que difieran en sexo, cultura, nivel económico, nivel de aprendizaje… El trabajo en parejas resulta muy eficaz para la consecución de los siguientes objetivos:
    - Favorecer las destrezas y actitudes cooperativas, así como la participación activa en tareas comunitarias.
    - Favorecer la individualización y personalización de la enseñanza, así como la adaptación al ritmo, interés, capacidades y estilos de aprendizaje, de

manera que los alumnos conectan, dentro de una concepción de aprendizaje significativo, los nuevos conceptos que han adquirido.
    - Permitir el aprendizaje de procesos metodológicos. El trabajo en grupo reducido es idóneo para el trabajo de investigación activa, en el que el alumno implementa numerosas estrategias de aprendizaje: elaborar un plan de trabajo, buscar y sistematizar información, formular hipótesis, etc.
    - Aclarar informaciones que se hayan impartido previamente en el aula.
    - Enriquecer al grupo con aportaciones diferenciadas, consiguiendo así un mayor rendimiento en logros de tiempo y medios didácticos.
    - Desarrollar la autonomía y responsabilidad del alumnado.

- **Trabajo en grupo-clase:** El papel del profesor en este tipo de agrupamiento es dinamizar el grupo, con el fin de que surjan nuevas ideas, al mismo tiempo que hace de hilo conductor de las aportaciones sobre diferentes temas. Se considera que el grupo del módulo de Programación está integrado por un máximo de 20 alumnos. Básicamente, el objetivo de las actividades presentadas en el grupo-clase consiste en aportar gran cantidad de información de manera uniforme mediante mensajes orales y/o visuales. En nuestra unidad, utilizaremos este tipo de agrupamiento en las siguientes situaciones:
    - Presentar asuntos o temas de interés general: nuevas salidas profesionales, formativas, etc.
    - Determinar y regular normas de convivencia.
    - Realización de debates sobre temas de interés: lenguajes de programación, nuevas aplicaciones informáticas en el mercado, estado laboral actual…
    - Explicaciones colectivas: exposición de trabajos, prácticas, ejercicios…
    - Conclusiones de trabajos o prácticas realizadas en pequeños grupos.
    - Proyecciones audiovisuales.

3.6.7.4 ORGANIZACIÓN DEL ESPACIO

Organizar el espacio hace referencia a la estructuración del lugar físico donde tendrán lugar las actividades de enseñanza-aprendizaje, en nuestro, caso el **aula de ordenadores**. Su organización nos permite analizar las posibilidades de configuración del espacio del aula (mobiliario, equipos informáticos, medios audiovisuales…). Programar el espacio nos permitirá evitar errores y aprovechar los recursos ofrecidos por el centro.

La organización del aula ha de ser flexible, de modo que podamos disponer del mobiliario de diferentes formas, dependiendo de la naturaleza de la actividad de enseñanza-aprendizaje a desarrollar y de los objetivos planteados.

La Orden 16 de junio de 2011, establece que el ciclo formativo de Desarrollo en Aplicaciones Multiplataforma requerirá para la implantación de las enseñanzas, de un **aula de informática de gestión** y **de un aula polivalente**.

Aula de Informática de 1º CFGS DAM organizada en 5 filas con pupitres agrupados frente a la pizarra. Hay un total de 20 equipos. La mesa del profesor se encuentra al fondo de la clase frente a los puestos de trabajo de los alumnos y el cañón proyector en el techo del aula.

Figura 13: Aula polivalente 1º CFGS Desarrollo de Aplicaciones Multiplataforma

El Departamento de Informática cuenta con un aula taller con mesas de trabajo donde el alumnado llevará a cabo el montaje de sistemas informáticos, así como el mantenimiento a nivel de hardware y software de los mismos. Se dispone además de equipos informáticos con conexión a internet para consultas.

Figura 14: Taller de Montaje y Mantenimiento de Equipos

3.6.7.5 MATERIALES Y RECURSOS DIDÁCTICOS

Los **materiales curriculares** que ayudarán al docente a adoptar decisiones referentes al proceso de enseñanza-aprendizaje y evaluar su acción docente serán entre otros: ***Proyecto Educativo, Programaciones de aula, Contenidos de las áreas, Adaptaciones curriculares, Unidades didácticas, Orientaciones para la evaluación…***

Los recursos didácticos utilizados por el profesor y los alumnos para desarrollar el proceso de enseñanza-aprendizaje están constituidos por diversos equipos que ayudarán al docente a presentar y desarrollar los contenidos, y a los alumnos a adquirir los conocimientos y destrezas necesarias.

Los recursos didácticos se clasifican en:

- **Recursos comunes**: Cañón retroproyector, pizarra blanca y rotuladores.
- **Recursos hardware**: Un PC para cada alumno y otro para el profesor. En cada puesto informático está instalado el sistema operativo Windows 10 y Ubuntu 14.04.
- **Recursos Software**: En la medida de los posible se apuesta por el software libre (OpenOffice, Ubuntu, db4o). Sin embargo, también se hace uso de software propietario (Windows) debido a que las empresas de la zona usan, sobre todo, software propietario. También, se hace uso del programa **iTALC 2.0.0** para llevar a cabo la monitorización del aula, ayudando al profesor a controlar el trabajo del alumnado a través de herramientas que le permiten: bloqueo de equipos, control remoto, envío de mensajes, ejecución remota de aplicaciones, apagado de equipos…
- **Tecnologías de la Información y Comunicación (TIC)**: Destacamos la plataforma **MOODLE**, a través de la cual, el docente proporciona apuntes, avisos, correcciones…, y los alumnos pueden hacer entrega de las prácticas o exámenes.
- **Recursos de Internet**: Haremos uso del MSDN (Microsoft Developer Network), la biblioteca en línea para desarrolladores de Microsoft. Se trata de una base con documentación variada acerca de las distintas aplicaciones de desarrollo que tiene Microsoft. MSDN dispone de una especie de enciclopedia con todas las funciones, clases, objetos y rutinas que se puedan utilizar en sus programas de desarrollo de aplicaciones para que sirva de explicación a la comunidad de programadores. En particular, nosotros utilizaremos MSDN para resolver problemas con software de Microsoft (sistemas operativos, aplicaciones, etc). También haremos uso del boletín de noticias con las últimas novedades del mundo.
- **Recursos de información**: No nos remitimos al uso de un único libro, sino que el profesor es el encargado de proporcionar el material didáctico y recomendar el uso de algunos libros, manuales y determinadas páginas de internet que complementen los conocimientos adquiridos en clase.
- **Documentación interna de las herramientas Software**: Tanto el sistema de ayuda como los manuales incluidos en el Software explican las materias a abordar adjuntando ejemplos prácticos.
- **Recursos de ilustración audiovisual**: Para la explicación de contenidos se hace uso del cañón proyector, uso de esquemas, diagramas, tablas cronológicas, presentación informáticas, etc.

Además, los alumnos deberán disponer de un cuaderno donde vayan recogiendo ordenadamente su información de las clases.

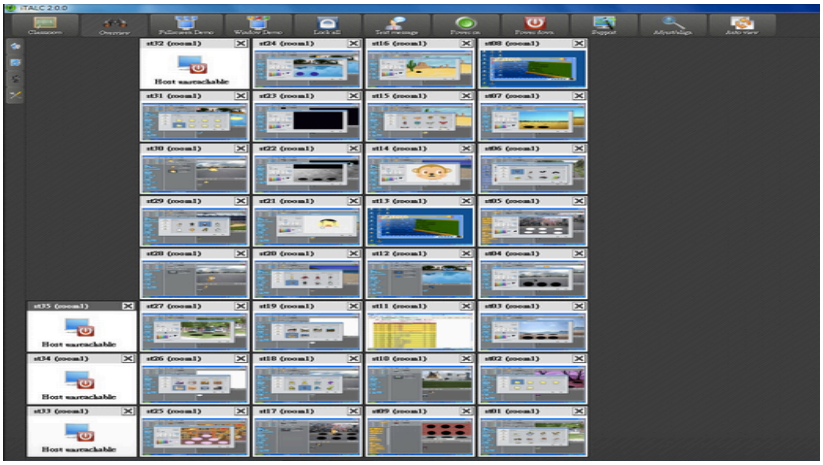


Figura 15: Programa de Monitorización iTALC 2.0.0

Figura 16: Plataforma Moodle IES Saladillo

Figura 17: Microsoft MSDN

3.6.7.6 PREPARACIÓN PARA LA EVALUACIÓN FINAL

Los alumnos/as que no hayan superado positivamente alguna unidad de trabajo o deseen elevar su calificación podrán hacerlo en la evaluación **Final**.

Durante este período de evaluación vamos a utilizar el mismo horario lectivo que ha utilizado durante el curso.

Desde el 1 hasta el 22 de junio se impartirá clases de refuerzo donde se hará un repaso general de todas las unidades de trabajo que se han impartido durante el curso y se resolverán dudas puntuales del alumno. Además, por cada unidad de trabajo, se irán asignando supuestos prácticos para que sean resueltos por los alumnos/as que tengan que recuperar dicha unidad. Estos supuestos serán posteriormente evaluados.

3.6.7.7 DESARROLLO DE LAS SESIONES

Al tratarse de una unidad didáctica de 12 horas y las sesiones tienen una duración de 2h, vamos a utilizar para la unidad de trabajo un total de 6 sesiones de 120 minutos cada una. A continuación, presentamos en la siguiente tabla las actividades que se van a realizar a lo largo de las sesiones, junto a la metodología llevada a cabo y los recursos utilizados.

SESIÓN 1ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A1 Presentación-Motivación Unidad Didáctica El docente presentará las unidad de trabajo, los capítulos de los que consta, así como los objetivos de aprendizaje de la misma.	**10´**	Grupo Clase .	Aula Informática	Proyector Pizarra digital Índice del tema
A2 Conocimientos Previos Evaluación de los conocimientos previos a través de una tormenta de ideas. El docente propondrá actividades que creen situaciones en las que el alumno/a reconozca sus limitaciones en sus saberes sobre las bases de datos orientadas a objetos.	**5'**	Grupo Clase	Aula Informática	Proyector Índice del tema
A3 Desarrollo Contenidos **Exposición sobre las características de**	**45'**	Grupo Clase	Aula Informática	Proyector Pizarra

las Bases de Datos Orientadas a Objetos El docente analizará las diferencias de los modelos de BD tradicionales frente a los modelo de BD 0rientadas a 0bjetos. A continuación, mostrará las características de las BDOO y los SGBD que existen: • SGBD puros. • SGBD híbridos u objeto-relacionales Se combina la resolución de supuestos prácticos. **A4 Desarrollo Contenidos** **Exposición referente al modelo de datos orientado a objetos** El docente mostrará las características de las relaciones en el modelo orientado a objetos. Posteriormente, mostrará cuáles son los mecanismos para crear los diferentes tipos de relaciones (uno-a-muchos, muchos-a-muchos). Finalmente, indicará cuáles son los mecanismos que garantizan la integridad en las relaciones. Se combina la resolución de supuestos prácticos.	**60'**			digital Apuntes Equipo equipado con un IDE Moodle iTALC

SESIÓN 2ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A5 Síntesis-Resumen Al inicio de la sesión, se hará un resumen de los contenidos aprendidos la sesión anterior y se resolverán dudas.	**5´**	Grupo Clase .	Aula Informática	Proyector Pizarra digital
A6 Desarrollo de Contenidos **Exposición sobre el lenguaje UML para el diseño de esquemas conceptuales de BD Orientadas a Objetos** El profesor indicará los pasos a seguir para el diseño de esquemas conceptuales de BD Orientadas a Objetos haciendo uso del lenguaje UML. Mostrará supuestos prácticos de uso de UML en el diseño de BD Orientadas a Objetos que, posteriormente, reproducirá el alumnado. Se combina la resolución de supuestos prácticos. **A7 Desarrollo de Contenidos** **Exposición y demostración relativas a la base de datos orientada a objetos db40 nativa de Java y de licencia GPL** El profesor profundizará en los componentes básicos de una BD Orientada a Objetos: objetos, literales, clases, interfaces, propiedades y transacciones. Instalaremos el Gestor de Bases de Datos y trabajaremos con una verdadera base de datos orientada a objetos.	**55’** **60’**	Grupo Clase	Aula Informática	Proyector Pizarra digital Apuntes Equipo equipado con un IDE Moodle iTALC Gestor

SESIÓN 3ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A8 Síntesis-Resumen Al inicio de la sesión, se hará un resumen de los contenidos aprendidos la sesión anterior y se resolverán dudas.	**5´**	Grupo Clase .	Aula Informática	Proyector Pizarra digital
A9 Desarrollo de Contenidos **Exposición y demostración referentes a cómo realizar aplicaciones y bases de datos** Realizaremos pequeñas aplicaciones con Java y trabajaremos sobre las operaciones básicas con bases de datos (crear/acceder, almacenar, recuperar, actualizar y borrar). Para manejar la base de datos utilizaremos API (Application Program Interface)	**55'**	Grupo Clase	Aula Informática	Proyector Pizarra digital Apuntes Equipo equipado con un IDE Moodle iTALC
A10 Ejercicios de Consolidación Con este tipo de actividades, los alumnos/as contrastan las nuevas ideas aprendidas en esta sesión con las previas y aplican nuevos aprendizajes.	**60'**	Individual o Parejas	Aula Informática	Apuntes Equipo equipado con un IDE Moodle iTALC

SESIÓN 4ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A11 Síntesis-Resumen Al inicio de la sesión, se hará un resumen de los contenidos aprendidos la sesión anterior y se resolverán dudas.	**5´**	Grupo Clase .	Aula Informática	Proyector Pizarra digital
A12 Desarrollo de Contenidos **Exposición y demostración referentes a los distintos sistemas de recuperación de datos en db40** El docente mostrará las posibilidades de consultar la base de datos mediante: • Query By Example (QBE) • Native Queries (NQ) • SODA (Simple Object Data Access) Se combina la resolución de supuestos prácticos.	**55'**	Grupo Clase	Aula Informática	Proyector Pizarra digital Apuntes Equipo equipado con un IDE Moodle iTALC
A13 Ejercicios de Consolidación Con este tipo de actividades, los alumnos/as contrastan las nuevas ideas aprendidas en esta sesión con las previas y aplican nuevos aprendizajes.	**60´**	Individual o Parejas	Aula Informática	Apuntes Equipo equipado con un IDE Moodle iTALC

SESIÓN 5ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A14 Síntesis-Resumen Al finalizar la unidad didáctica, se elaborará un resumen de los nuevos contenidos aprendidos de manera individual, el profesor elaborará junto con los alumnos un mapa conceptual global de toda la unidad en pizarra, se resolverán dudas, etc.	**10´**	Grupo Clase .	Aula Informática	Proyector Pizarra digital
A15 Ejercicios de Refuerzo El objetivo de estas actividades es atender a los alumnos/as que no han conseguido los aprendizajes previstos. Se realizarán actividades del mismo estilo a las de desarrollo y consolidación.	**50'**	Individual	Aula Informática	Apuntes Equipo equipado con un IDE Moodle iTALC
A16 Ejercicios de Ampliación Permitirán construir nuevos conocimientos a los alumnos/as que hayan realizado de forma satisfactoria las actividades establecidas.	**50´**	Individual o Parejas	Aula Informática	Apuntes Equipo equipado con un IDE Moodle iTALC
A17 Secuencia AICLE en Inglés Una vez que nos hemos asegurado que el alumnado ha estudiado la materia, vamos a reforzar los contenidos estudiados en la unidad al mismo tiempo que aprendemos un segundo idioma (inglés), haciendo uso de las 5 destrezas: Reading, Writing, Speaking, Listening and Interaction.	**60**	Grupo Clase Parejas Individual	Aula Informática	Ejercicios AICLE Apuntes Equipo equipado con un IDE Moodle iTALC

SESIÓN 6ª				
Actividades Enseñanza-Aprendizaje	**Duración**	**Agrupamientos**	**Espacio**	**Recursos**
A18 Evaluación El alumnado deberá presentar todas las actividades propuestas haciendo uso de capturas de pantalla, valorándose la claridad de ideas, el orden, posibles ampliaciones y la presentación. Además, deberán realizar un ejercicio con cuestiones de respuestas breve y supuestos prácticos.	**120´**	Individual .	Aula Informática	Examen Equipo equipado con un IDE Moodle iTALC

3.6.8 EVALUACIÓN

La evaluación forma parte del proceso de enseñanza y aprendizaje y supone un recurso metodológico imprescindible por su valor como elemento motivador para el alumnado y para el propio profesorado.

La evaluación determinará el grado de consecución de los objetivos e intenciones del proyecto educativo, abarcando tanto el alumnado como a los distintos componentes del currículo.

Procederemos al análisis de la Evaluación del proceso de enseñanza desde el punto de vista de los siguientes tres aspectos:

1. **Criterios de Evaluación. ¿Qué evaluar?**
2. **Procedimientos e instrumentos de evaluación ¿cómo evaluar?**
3. **Momentos de evaluación ¿Cuándo evaluar?**

3.6.8.1 CRITERIOS DE EVALUACIÓN

Para la unidad didáctica que vamos a desarrollar ***"Persistencia de los objetos en bases de datos orientadas a objetos",*** vamos a utilizar los criterios de evaluación expuestos en el Resultado de Aprendizaje 8:

a) Se han identificado las características de las bases de datos orientadas a objetos.
b) Se ha analizado su aplicación en el desarrollo de aplicaciones mediante lenguajes orientados a objetos.
c) Se han instalado sistemas gestores de bases de datos orientados a objetos.

d) Se han clasificado y analizado los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.
e) Se han creado bases de datos y las estructuras necesarias para el almacenamiento de objetos.
f) Se han programado aplicaciones que almacenen objetos en las bases de datos creadas.
g) Se han realizado programas para recuperar, actualizar y eliminar objetos de las bases de datos.
h) Se han realizado programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.

3.6.8.2 PROCEDIMIENTOS E INSTRUMENTOS DE EVALUACIÓN

La función prioritaria de los procedimientos e instrumentos de evaluación es la recogida de datos del proceso educativo. La adecuada selección, utilización y revisión de estos procedimientos e instrumentos permitirá que esta recogida de información resulte del todo rigurosa, sistemática y controlada, con lo que los resultados finales de la evaluación serán fiables, válidos y, por tanto, útiles para la mejora de los procesos de enseñanza y aprendizaje.

En la siguiente tabla, mostramos los procedimientos e instrumentos de recogida de información para la evaluación del aprendizaje que utilizaremos, así como la ponderación que tendrán sobre la calificación final del alumno en la unidad didáctica:

El seguimiento y análisis de las producciones de los alumnos, a nivel individual o de grupo, a través de cuadernos de trabajo, presentaciones, entrevistas, proyectos, trabajos monográficos, cuestionarios, etc.	**40%**
La aplicación de pruebas específicas, prácticas, orales o escritas, abiertas o cerradas, para la evaluación de los contenidos.	**50%**
Los debates y presentaciones que permitirán observar y obtener información sobre aspectos actitudinales, de integración y actuación social.	**10%**

A partir de estos procedimientos e instrumentos evaluaremos nuestra unidad de trabajo obteniendo una calificación numérica comprendida entre 1 y 10 puntos (con dos decimales). La superación positiva de la unidad de trabajo es eliminatorio.

El módulo profesional también se calificará mediante un entero comprendido entre 1 y 10 puntos (sin decimales), que se obtendrá a partir de la media aritmética de las calificaciones correspondientes a las 11 unidades didácticas que engloba. Se considerará superado el módulo cuando dicha calificación sea de 5 puntos o superior.

3.6.8.3 CRITERIOS DE CALIFICACIÓN

Para esta unidad de trabajo 11, al igual que para el resto, los criterios de calificación utilizados son los siguientes:

- **Examen 50%**: Se trata de una prueba teórico-práctica.
  - Cada falta de ortografía resta **0.1** puntos, pudiendo restar como máximo **1** punto en el examen.
  - El resultado del examen debe ser como mínimo un **4** para poder hacer media con las notas de Prácticas y Actitud.

- **Prácticas 40%**:
  - Entrega del **cuaderno** de clase y organización de los apuntes (**5%**).
  - Actividades **prácticas** (**25%**) y entrega de **trabajos** (**5%**).
  - Secuencia **AICLE** (**5%**)
  - Se debe obtener como mínimo un **4** en esta parte para hacer media con las notas de Examen y Actitud.

- **Actitud 10%**: Para la calificación de la actitud seguimos los siguientes criterios:
  - **Asistencia** (**4%**): El alumno parte de un **4**. Por cada falta de asistencia sin justificar se resta **0.5** puntos. La impuntualidad resta **0.2** puntos.
  - **Comportamiento** (**3%**): El alumno parte de un **3**. Si hay una falta de respeto leve, disrupciones, no cuida el material…, se resta **1** punto.
  - **Participación** (**3%)**: Se parte de **0**. El dinamismo en clase, pizarra, ayuda prestada a compañeros…, suma **0.5** puntos.

El alumno/a superará positivamente la unidad de trabajo si la calificación total es un 5 o superior.

A continuación, presentamos dos tablas que el docente utilizará para la evaluación continua de la unidad didáctica. Donde:

- **Frecuencia**: Siempre (S); Con Frecuencia (F); A veces (AV); Raramente (R)
- **Autonomía**: Sin Ayuda (SA); Con indicaciones (CI); Con algo de ayuda (CA)
- **Cualidad**: Excelente (E); Muy Bien (MB), Bien (B), Regular (R)

Criterios Evaluación	Frecuencia	Autonomía	Cualidad
Identifica las características de las BDOO.			
Analiza su aplicación en el desarrollo de aplicaciones mediante LPOO.			

Instala sistemas gestores BDOO.				
Clasifica y analiza los distintos métodos soportados por los sistemas gestores para la gestión de la información almacenada.				
Crea bases de datos y estructuras necesarias para el almacenamiento de objetos.				
Programa aplicaciones que almacenen objetos en las bases de datos creadas.				
Realiza programas para recuperar, actualizar y eliminar objetos de BD.				
Realiza programas para almacenar y gestionar tipos de datos estructurados, compuestos y relacionados.				
Proceso de Aprendizaje		**Frecuencia**	**Autonomía**	**Cualidad**
Cuaderno	Trabajo individual			
	Correcciones de errores			
Trabajos o Proyectos	Puntualidad en la entrega			
	Expresión escrita			
	Busca y analiza información de diferentes fuentes			
	Claridad de contenidos y síntesis			
Trabajo en grupo	Realiza la parte que le corresponde del trabajo			
	Comparte su aprendizaje			
	Es crítico y acepta las críticas			

3.6.8.4 MECANISMOS DE RECUPERACIÓN

En el supuesto de que la calificación de la unidad de trabajo sea inferior a 5, el alumno no podrá superar la unidad de trabajo, por lo que se procederá a establecer **mecanismos de recuperación** en función de las causas que hayan motivado dicha calificación. Estos

mecanismos consistirán en una serie de actividades de refuerzo que permitan al alumno asentar los contenidos de la unidad. Básicamente, distinguimos tres tipos de mecanismos:

- De **apoyo**: Seguimiento continuo de las actividades de refuerzo y entrevista con el alumno/a para indicarle los aspectos que debe cuidar.
- De **trabajo**: Trabajar más los aspectos evaluados negativamente realizando actividades de repaso.
- De **control**: Realizando una prueba específica.

Los alumnos/as tendrán dos oportunidades de poder recuperar la unidad, una final del trimestre correspondiente y otra segunda, en la evaluación final.

3.6.8.5 CARACTERÍSTICAS DE LA EVALUACIÓN FINAL

Todos los alumnos deben tener una evaluación final pero pueden eliminar materia a través de las evaluaciones parciales. Por tanto, podrán realizar la evaluación final de nuestro módulo aquellos alumnos/as que no hayan alcanzado las capacidades mínimas o aquéllos/as que aún habiéndolas alcanzado desee elevar la calificación de alguna unidad didáctica.

Para estos alumnos que acudan a la evaluación final se propondrán actividades de recuperación personalizadas para la consecución de las capacidades mínimas anteriormente indicadas. Las actividades de recuperación se fijarán a partir del informe que se le hará a cada alumno que no haya superado el módulo en las evaluaciones parciales.

La realización de las actividades de recuperación se adecuará, dentro de las horas lectivas programadas en el horario del docente, a las capacidades terminales no adquiridas por el alumno. Dichas actividades deberán ser realizadas por el alumno/a a lo largo del periodo establecido a tal efecto en la comunidad autónoma de Andalucía, y serán evaluadas mediante la aplicación de pruebas específicas escritas (entre el 1 y 22 de junio).

Para la evaluación de la unidad de trabajo, se considerará que los ejercicios prácticos tienen un peso del **40%** del total y la prueba específica un **50%**. El **10%** restante evalúa la actitud que el alumno/a mostró durante la impartición de la unidad de trabajo a recuperar. La actitud no se puede recuperar en la evaluación final.

Se considera que la unidad de trabajo está superada si se obtiene una calificación **mayor o igual que 5**.

3.6.9 ATENCIÓN AL ALUMNADO CON NECESIDADES ESPECÍFICAS DE APOYO EDUCATIVO (AANEAE)

Se atribuye el término de Necesidades Educativas Especiales (NEE) a aquellos alumnos/as que presentan mayores dificultades que el resto de su mismo grupo y nivel educativo a la hora de alcanzar, por vía ordinaria, los objetivos y contenidos curriculares propuestos. Se engloba dentro del término de NEE a tres tipos de colectivos:

- Alumnos con incorporación tardía y con dificultades de integración.

- Alumnos superdotados intelectualmente.
- Alumnos con necesidades educativas que presentan discapacidades físicas, psíquicas, sensoriales o graves trastornos de personalidad o conducta.

En nuestro módulo y en concreto en la unidad didáctica que nos ocupa, se va a contemplar la atención a la diversidad según el **ritmo de aprendizaje**:

- Habrá alumnos/as que presenten un ritmo más lento de aprendizaje, por lo que se plantearán actividades adaptadas a su ritmo y con el adecuado nivel de dificultad, insistiendo en los contenidos mínimos de la unidad didáctica, para conseguir los objetivos planteados en la misma.
- También se tendrá en consideración la existencia de alumnos/as con un ritmo más acelerado, para los cuales se planteará un número adicional de actividades y supuestos prácticos que les permita desarrollar su capacidad de investigación y razonamiento.

En adición, se va a contemplar la existencia de alumnos/as que presenten **Necesidades Educativas Especiales**. En el grupo al cual dirigimos la proyección didáctica nos centramos en una alumna que presenta una **hipoacusia no severa con pérdida de la sensibilidad de 30 dB**. Por tanto, para facilitar su integración debemos tener presente los siguientes aspectos:

En la organización del aula:

- Presentar toda la información posible en soporte visual.
- Informar con claridad, de forma regular y sistemática, acerca de las actividades que ha de realizar.
- Fomentar en clase el apoyo entre iguales, de manera que se establezcan alumnos colaboradores que ayuden a la alumna.
- Reservar un puesto en primera fila, cuidando la iluminación y la visión.

En la metodología didáctica:

- Se procurará hablar siempre de frente, buscando la mayor iluminación posible y mejor ángulo de visión.
- Subir a la plataforma MOODLE la programación de la asignatura.
- Utilizar los tablones y plataforma para los avisos por escritos dirigidos a la clase.

En las ayudas técnicas:

- Es necesario conocer las ayudas técnicas que utiliza la alumna y mostrar nuestra colaboración.

En la evaluación:

- Proporcionar información previa del examen y sus requisitos, así como criterios de valoración.
- Comunicar expresamente y por escrito las observaciones que se hagan oralmente.
- Para resolver dudas durante un examen escrito, explicaremos el contenido con una estructura lingüística más sencilla.
- Permitir el uso de ayudas técnicas.

En caso de ser necesario, solicitaremos asesoramiento del Departamento de Orientación.

3.6.10 INTERDISCIPLINARIEDAD

Interdisciplinariedad se define como el conjunto de disciplinas conexas entre sí y con relaciones bien definidas, a fin de que sus actividades no se produzcan de forma aislada, dispersa y fraccionada, y de esta forma no entrar en contradicciones, no repetir contenidos y no crear inseguridad.

Distinguimos entre **interdisciplinariedad horizontal** (si los módulos pertenecen al mismo curso) e **interdisciplinariedad vertical** (si los módulos pertenecen a diferentes cursos)

La unidad de trabajo que nos ocupa ***"Persistencia de los objetos en bases de datos orientadas a objetos"***, presenta una interdisciplinariedad horizontal con el módulo de **Bases de Datos** (1º CFGS DAM) y una interdisciplinariedad vertical con el módulo de **Acceso a datos** (2º CFGS DAM).

Por lo tanto, para el desarrollo de los contenidos de esta unidad didáctica sería muy conveniente coordinarse con el profesor del módulo de "Bases de Datos", y así poder aprovechar los contenidos adquiridos en dicho módulo y aplicarlos en el diseño y creación de bases de datos orientadas a objetos. Del mismo modo, las capacidades adquiridas en la unidad servirán de referente para partir de una base en los contenidos impartidos en el módulo de "Acceso a datos" de 2º curso.

3.6.11 TEMAS TRANSVERSALES

Los temas transversales son un conjunto de contenidos de enseñanza esencialmente actitudinales que recogen aspectos que han alcanzado especial relevancia en el desarrollo de la sociedad durante los últimos años en relación con los valores morales, la igualdad de oportunidades, la salud, el medio ambiente y el consumo.

Pese a que la enseñanza para la que estamos programando es postobligatoria, es necesario abordarlos tal y como así se establece en los objetivos del **Artículo 5 de la Ley de Educación de Andalucía**.

Hemos considerado apropiado abordar los temas transversales de forma globalizada en lugar de incluirlos de forma aislada en un boque de contenidos, posibilitando así la formación científico-técnica junto al ético y moral.

Los temas transversales que podrán ser aplicados al módulo de ***"Programación",*** y en concreto, a la ***Unidad Didáctica 11 "Persistencia de los objetos en bases de datos"*** son los siguientes:

- **Educación Moral-Cívica**: Se expondrá la importancia que tiene el respeto hacia las normas de utilización del software (licencias). También incentivaremos el uso de software libre, demos, etc. Por otro lado, afrontaremos conflictos provocados por las limitaciones tecnológicas siempre presentes en un entorno tecnológico tan dinámico y en continua evolución como es el sector informático. Y, por último, se van a desarrollar habilidades de relación social e interpersonal, potenciando las actitudes comunicativas, de negociación y de trabajo en grupo, siempre desde el respeto.
- **Educación para el Consumidor**: Se investigará sobre las novedades en el campo de las tecnologías y comunicaciones que hay en el merado y, posteriormente, se harán trabajos y/o exposiciones y/o debates sobre las investigaciones realizadas.
- **Educación Ambiental**: Se proporcionará el material didáctico a través de la plataforma Moodle, correo electrónico, almacenamiento en la nube, etc., para proporcionar material didáctico y para la entrega/corrección de prácticas.
- **Seguridad Laboral**: Para abordar este tema se propondrán mediadas organizativas que faciliten una carga de trabajo adecuada, incentivaremos que los elementos del puesto de trabajo (sillas, mesas, equipos, etc.) deben de tener las condiciones ergonómicas adecuadas.
- **Bilingüismo**: Puesto que estamos programando para un centro y un ciclo formativo bilingüe en el idioma de inglés, vamos a proponer en esta unidad didáctica una batería de ejercicios (**Secuencia AICLE**) y la colaboración del intérprete de idiomas del centro, con el objetivo que nuestros alumnos/as puedan practicar las cinco destrezas del idioma (Reading, Writing, Speaking, Listening and Interaction), al mismo tiempo que refuerzan los conocimientos de informática adquiridos en la unidad.

3.7. ACTIVIDADES DE LA UNIDAD DIDÁCTICA

3.7.1 ACTIVIDADES DE DESARROLLO

1. ***(Ejercicio Resuelto)*** **Crea la siguiente clase que será la utilizada para realizar operaciones básicas con la base de datos. Nuestra clase va a ser la siguiente.**

```
public class alumno{
     private String nombre;
     private int edad;
     private double nota;
     public alumno(){
            this.nombre = null;
            edad = 0;
            nota = 0;
     }
     public alumno(String n,int e) {
            this.nombre = n;
            this.edad = e;
            this.nota = -1; //nota no establecida
     }
     public alumno(String nom,int e, double not) {
            this.nombre = nom;
            this.edad = e;
            this.nota = not;
     }
     public void setNombre(String n) {
            this.nombre = n;
     }
     public String getNombre() {
            return this.nombre;
     }
     public void setNota(double n) {
            this.nota=n;
     }
     public double getNota() {
            return this.nota;
     }
     public void setEdad(int e) {
            this.edad=e;
     }
     public int getEdad() {
            return this.edad;
     }
     public String toString() {

               if (this.nota != -1)
                      return this.nombre+"("+this.edad+") Nota:"+this.nota;
               return this.nombre+"("+this.edad+")";
        }
    }
```

La clase almacena los datos de los alumnos y tiene los métodos mínimos para poder trabajar con ella. A continuación, vamos a ver y comprobar las operaciones básicas que se pueden realizar sobre esta base de datos.

1.1 Crear/acceder a la base de datos:

La estructura básica de trabajo con la base de datos es la siguiente:

Apertura de la BD (conexión) ➔ Realizar operaciones ➔ Cerrar la BD (desconexión)

```
ObjectContainer bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),"alumnos.
db4o");
try {
//Realizar operaciones o
//llamadas a métodos
}
finally {
bd.close();
}
```

1.2 Almacenar objetos:

Para almacenar objetos en la base de datos basta con llamar al método ***store()*** del objeto bd creado de tipo ***ObjectContainer***. Un ejemplo de método que almacena tres objetos Alumno es el siguiente:

```
public static void almacenarAlumnos(ObjectContainer bd) {
    alumno a1 = new alumno("Juan Gámez",23,8.75);
    bd.store(a1);
    System.out.println(a1.getNombre()+" Almacenado");
    alumno a2 = new alumno("Emilio Anaya",24,6.25);
    bd.store(a2);
    System.out.println(a2.getNombre()+" Almacenado");
    alumno a3 = new alumno("Ángeles Blanco",26,7);
    bd.store(a3);
    System.out.println(a3.getNombre()+" Almacenado");
}
```

1.3 Recuperar objetos de la base de datos:

Para mostrar el resultado de las consultas realizadas a la base de datos usamos el método ***mostrarResultado()*** el cual toma como parámetro un ***ObjectSet(conjunto de objetos)*** y va recorriéndolo mostrando uno a uno los datos recuperados.

```
public static void mostrarResultado(ObjectSet  res){
    System.out.println("Recuperados "+res.size()+" Objetos");
    while(res.hasNext()) {
       System.out.println(res.next());
    }
}
```

Si queremos recuperar todos los alumnos de la base de datos, se le pasa un dato vacío (datos de tipo *String* a *null* y campos numéricos a 0) que le indica a la base de datos que deberá recuperar todos los objetos.

```
public static void muestraAlumnos(ObjectContainer bd) {
    alumno a = new alumno(null, 0, 0);
    ObjectSet res = bd.queryByExample(a);
    mostrarResultado(res);
}
```

Recuperar un alumno en concreto cuya edad sea 26.

```
public static void muestraAlumnos26(ObjectContainer bd) {
    alumno a = new alumno(null, 26, 0);
    ObjectSet res = bd.queryByExample(a);
    mostrarResultado(res);
}
```

1.4 Actualizar objetos en la base de datos:

Para actualizar un objeto basta con modificarlo e invocar al método ***store()*** para que se actualice en la base de datos.

```
public static void actualizarNotaAlumno(ObjectContainer bd,String nombre,double nota)
{
    ObjectSet res = bd.queryByExample(new alumno(nombre,0,0));
    alumno a = (alumno)res.next();
    a.setNota(nota);
    bd.store(a);
    muestraAlumnos(bd);
}
```

```
actualizarNotaAlumno(bd,"Emilio Anaya",9.5);
```

1.5 Borrar objetos de la base de datos:

Borrar un alumno de la base de datos.

```
public static void borrarAlumnoporNombre(ObjectContainer bd,String nombre) {
    ObjectSet res = bd.queryByExample(new alumno(nombre,0,0));
    alumno a = (alumno)res.next();
    bd.delete(a);
    muestraAlumnos(bd);
}
```

```
borrarAlumnoporNombre(bd,"Juan Gámez");
```

2. ***(Ejercicio Resuelto)*** **Usando la librería API SODA, realiza las siguientes consultas, probando el código que se proporciona.**

2.1 Genera un listado de todos los objetos alumno de la base de datos:

```
public static void consultaSODAAlumnos(ObjectContainer bd) {
    Query query=bd.query();
    query.constrain(alumno.class);
    ObjectSet result=query.execute();
    mostrarResultado(result);
}
```

2.2 Obtener todos los alumnos que se llamen "Emilio Anaya".

```
public static void consultaSODAEmilio(ObjectContainer bd) {
    Query query=bd.query();
    query.constrain(alumno.class);
    query.descend("nombre").constrain("Emilio Anaya");
    ObjectSet result=query.execute();
    mostrarResultado(result);
}
```

2.3 Obtener aquellos alumnos que tengan 23 años.

```
public static void consultaSODAAlumVtres(ObjectContainer bd) {
    Query query=bd.query();
    query.constrain(alumno.class);
    query.descend("edad").constrain(23);
    ObjectSet result=query.execute();
    mostrarResultado(result);
}
```

2.4 Mostrar los alumnos que no tengan 23 años.

```
Query query=bd.query();
query.constrain(alumno.class);
query.descend("edad").constrain(23).not();
ObjectSet result=query.execute();
mostrarResultado(result);
```

2.5 Obtener aquellos alumnos cuya nota sea menor de 7 o edad mayor de 25 años.

```
import java.io.*;
import com.db4o.*;
import com.db4o.query.*;
public class testSODA  {
public static void main(String[] args) {
    new File("alumnos.db4o").delete();
    ObjectContainer bd = Db4oEmbedded.openFile(Db4oEmbedded
    .newConfiguration(), "alumnos.db4o");
    try {
          almacenarAlumnos(bd);
```

```
            consultaSODA(bd);
        } catch(Exception e){
            e.printStackTrace();
        }finally {
            bd.close();
        }
    }

    public static void mostrarResultado(ObjectSet  res){
        System.out.println("Recuperados "+res.size()+" Objetos");
        while(res.hasNext()) {
           System.out.println(res.next());
        }
    }

    public static void almacenarAlumnos(ObjectContainer bd) {
        alumno a1 = new alumno("Juan Gámez",23,8.75);
        bd.store(a1);
        System.out.println(a1.getNombre()+" Almacenado");
        alumno a2 = new alumno("Emilio Anaya",24,6.25);
        bd.store(a2);
        System.out.println(a2.getNombre()+" Almacenado");
        alumno a3 = new alumno("Angeles Blanco",26,7);
        bd.store(a3);
        System.out.println(a3.getNombre()+" Almacenado");
    }

    public static void consultaSODA(ObjectContainer bd) {
        Query query=bd.query();
        query.constrain(alumno.class);
        Constraint constr=query.descend("nota").constrain(7).smaller();
        query.descend("edad").constrain(25).greater().or(constr);
        ObjectSet result=query.execute();
        mostrarResultado(result);
    }

    }
```

3. **Imaginemos que para una Asignatura determinada se van a realizar una serie de proyectos. Cada proyecto está asignado a un alumno en concreto, por lo tanto, la clase Proyecto incluirá un objeto de la clase Alumno. La clase Proyecto estaría implementada de la siguiente forma:**

```
public class proyecto{
    private String descripcion;
    private alumno al;

    public proyecto(String descripcion) {
        this.descripcion=descripcion;
        this.al=null;
    }

    public alumno getAlumno() {
        return al;
    }

    public void setAlumno(alumno a) {
        this.al = a;
    }
```

```
        public String getDescripcion() {
            return descripcion;
        }

        public String toString(){
               return descripcion + "-> "+ al;
        }
    }
```

Crea una consulta QBE que muestre los proyectos de aquellos alumnos que tengan 23 años.

4. **Imaginemos que tenemos una serie de profesores que se encargan de llevar los proyectos de una serie de alumnos. Lo lógico es implementar la serie de proyectos atendidos por un profesor en un array de objetos de tipo proyecto. Completa la clase profesor que proporcionamos a continuación, con los siguientes métodos, codifícala en un método y comprueba que funciona:**
   - ***void setNombre(String)***
   - ***string getNombre()***
   - ***boolean tieneProyecto(String)***. Devuelve *True* si el profesor lleva un proyecto cuya descripción de proyecto coincide con la pasada como parámetro. *False* en caso contrario.
   - ***boolean tieneAlumno(String)***. Devuelve *True* si el profesor lleva a un alumno cuyo nombre coincide con el pasado como parámetro. *False* en caso contrario.

Mostramos la clase Profesor:

```
public class profesor{
     private proyecto listaproyectos[];
     private String nombre;
     profesor(proyecto[] l,String n){
             this.nombre = n;
             this.listaproyectos=l;
     }
     public String getNombre(){
             return nombre;
     }
     public proyecto[] getProyectos(){
             return this.listaproyectos;
     }
     public int getNumProyectos(){
             return this.listaproyectos.length;
     }
     public String toString(){
             String str = new String();
             for (proyecto l : listaproyectos){
                     str=str+ " *** " +l.toString();
             }
             return nombre + " [[[ "+ str+ " ]]] ";
     }
}
```

5. ***(Ejercicio Resuelto)*** **Implementar un método en el que se creen 5 alumnos, cada uno con su proyecto asignado, algunos de ellos asignados a un profesor y el resto a otro profesor.**

```
public static void almacenarProfesores(ObjectContainer bd) {
    proyecto p1 = new proyecto("Robot con microcontroladores");
    alumno a1 = new alumno("Juan Gámez",23,8.75);
    p1.setAlumno(a1);
    proyecto p2 = new proyecto("Webmin sobre Ubuntu");
    alumno a2 = new alumno("Emilio Anaya",24,6.25);
    p2.setAlumno(a2);
    proyecto p3 = new proyecto("Joomla y Drupal");
    alumno a3 = new alumno("Ángeles Blanco",26,7);
    p3.setAlumno(a3);
    proyecto p4 = new proyecto("Doctor Profiler");
    alumno a4 = new alumno("Luis Alberto García",29,9.5);
    p4.setAlumno(a4);
    proyecto p5 = new proyecto("Gambas II Linux");
    alumno a5 = new alumno("Robert Sinnock",24,9);
    p5.setAlumno(a5);
    profesor pr1 = new profesor(new proyecto[]{p1,p2,p3},"Juan Carlos")
    bd.store(pr1);
    System.out.println(pr1.toString()+" Almacenado");
    profesor pr2 = new profesor(new proyecto[]{p4,p5},"Emma");
    bd.store(pr2);
    System.out.println(pr2.toString()+" Almacenado");
}

new proyecto[]{p1,p2,p3}
```

6. ***(Ejercicio Resuelto)*** **Utilizando el método del ejercicio 5, recupera aquellos alumnos que estén realizando un proyecto con Joomla y Drupal (Consulta sobre un objeto proyecto de un array).**

```
public static void muestraProfesorDrupal(ObjectContainer bd){
    proyecto p = new proyecto("Joomla y Drupal");
    profesor pr = new profesor(new proyecto[]{p},null);
    ObjectSet res = bd.queryByExample(pr);
    mostrarResultado(res);
}
```

7. ***(Ejercicio Resuelto)*** **Utilizando el método del ejercicio 5, recupera aquellos profesores que tengan un alumno con 29 años (Consulta por un objeto alumno dentro del objeto proyecto).**

```
public static void muestraProfesorAlumno29(ObjectContainer bd){
    alumno a = new alumno(null, 29, 0);
    proyecto p = new proyecto(null);
    p.setAlumno(a);
    profesor pr = new profesor(new proyecto[]{p},null);
    ObjectSet res = bd.queryByExample(pr);
    mostrarResultado(res);
}
```

8. ***(Ejercicio Resuelto)*** **Realiza un método con el siguiente prototipo:**

   public static void consultaSODA2 (ObjectContainer bd)

   El método deberá hacer lo siguiente:

   - **Insertar un nuevo alumno con los siguientes datos:**
     - **Nombre: "Juan Serrano"**
     - **Edad: 29**
     - **Nota: 9.75**
   - **Mostrar los alumnos de la base de datos ordenados por nombre ascendentemente.**

```
public static void consultaSODA2(ObjectContainer bd) {
    alumno a=new alumno("Juan Serrano",29,9.75);
    bd.store(a);
    Query query=bd.query();
    query.constrain(alumno.class);
    query.descend("nombre").orderAscending();
    ObjectSet result=query.execute();
    mostrarResultado(result);
    bd.delete(a);
}
```

9. ***(Ejercicio Resuelto)*** **Realiza un método con el siguiente prototipo:**

   public static void borraPorEdad (ObjectContainer bd, int edad)

   Este método deberá borrar de la base de datos aquellos alumnos que coincidan con la edad indicada en el segundo parámetro. El método además deberá de mostrar la información de los alumnos eliminados.

```
public static void borraPorEdad(ObjectContainer bd, int edad) {
    Query query=bd.query();
    query.constrain(alumno.class);
    query.descend("edad").constrain(edad);
    ObjectSet result=query.execute();
    while(result.hasNext()) {
          alumno a= (alumno)result.next();
          System.out.println("Borrado: "+a);
          bd.delete(a);
    }
}
```

3.7.2 ACTIVIDADES DE CONSOLIDACIÓN

1. **Realiza una consulta a la base de datos de alumnos vista en los Ejercicios de Desarrollo del tema y selecciona aquellos alumnos que no se llamen "Fernando Gil" y ordénalos ascendentemente.**

2. **Realiza una consulta a la base de datos de alumnos vista en los Ejercicios de Desarrollo del tema y selecciona aquellos alumnos que no se llamen "Andrés Rosique" o "Juan Gámez".**

3. **Se propone al alumno que cree un juego de preguntas. Para ello, se creará una clase pregunta con una pregunta a realizar, cuatro posibles respuestas y una única respuesta verdadera. El alumno deberá crear un fichero de texto con una serie de preguntas. Cada pregunta ocupará 5 líneas del fichero de texto. La forma de guardar los datos en el fichero será la siguiente:**
   - ***Pregunta***
   - ***Respuesta 1***
   - ***Respuesta 2***
   - ***Respuesta 3***
   - ***Respuesta 4***
   - ***Solución***

   A continuación, mostramos un ejemplo:

   ¿Cuál es el inventor del lenguaje Java?

   Steve Jobs

   James Goeling

   Bill Gates

   Andrew Tanembaum

   2

   El programa deberá leer el fichero, crear los objetos de tipo pregunta y almacenarlos en la base de datos. También, deberá de tener un método que realice 10 preguntas al usuario y muestre un sumario a la conclusión con las preguntas acertadas y erradas.

4. **Modifica el programa anterior para que al usuario se le hagan preguntas de forma aleatoria y sin repetición y que el número de preguntas a realizar pueda ser establecido previamente antes de iniciarse la batería de preguntas.**

5. **Modifica el programa anterior creando una clase categoría en la que se puedan clasificar las preguntas. Cada categoría tendrá una serie de preguntas almacenadas en**

un array. El usuario, cuando ejecute el programa, deberá de elegir la categoría de la que quiere ser preguntada.

6. Añádele una interfaz gráfica al programa anterior. La interfaz deberá contemplar todas las características del diseño propuesto. Mejora la interfaz añadiendo el campo *jugador*. Cuando un jugador ingresa en el programa debe introducir su nombre y posteriormente jugar. Crea una opción en el programa en la que se muestre el jugador que ostenta el record del juego.

3.7.3 ACTIVIDADES DE REFUERZO

1. Realiza una consulta a la base de datos de alumnos vista en los Ejercicios de Desarrollo del tema y selecciona aquellos alumnos cuya nota esté comprendida entre 7 y 9.

2. Crea un programa como el generado en el ejercicio 3 de los Ejercicios de Consolidación que permita generar las preguntas del archivo de datos. El programa deberá de tener una interfaz gráfica. El diseño de la interfaz lo realizará el alumno atendiendo las necesidades del programa.

3.7.4 ACTIVIDADES DE AMPLIACIÓN

1. Necesitamos crear una pequeña aplicación que gestione el parque de ordenadores de una pequeña empresa. Cada ordenador tendrá las siguientes características:
   - Marca
   - Modelo
   - Procesador
   - TipoMemoria
   - CantidadMemoria
   - Ubicación
   - Número Serie
   - Número interno (código suministrado por la empresa)

   El programa deberá poder Insertar, Modificar y Eliminar equipos. Cuando se inserta un equipo el sistema deberá de dotarle de un código interno. Se deberá poder realizar consultas como:

   - Listado de equipos de una ubicación determinada.
   - Cuántos equipos hay por ubicación.
   - Tipo de memoria y cantidad de los equipos.

   El diseño de la interfaz lo realizará el alumno atendiendo las necesidades del programa.

4. BIBLIOGRAFÍA Y REFERENCIAS WEBS

Bibliografía de Departamento y Aula

[1] Mario G Piattini, Esperanza Marcos, Coral Calero, Belén Vela (2006). ***"Tecnología y diseño de bases de datos"***. Editorial Ra-Ma.

[2] Silberschatz, Korth, Sudarshan (2006) ***"Fundamentos de Bases de Datos. Quinta Edición"***. Editorial McGrawHill.

[3] Juan Carlos Moreno Pérez (2014). ***"Programación"***. Editorial Ra-Ma.

Referencias Webs

[4] Olga Zalamea (2013). ***"Evolución de las Bases de Datos"***. Revista Galileo. Número 23. Disponible en: http://ucuenca.edu.ec/ojs/index.php/galileo/article/view/157/155. Recuperado en Junio de 2016.

[5] Jorge Mario Londoño, Sonia Jaramillo Valbuena (2014). ***"Sistemas para almacenar grandes volúmenes de datos. Big data stores".*** Revista Gerencia Tecnológica Informática. Volumen 13. Número 37. Disponible en: http://revistas.uis.edu.co/index.php/revistagti/article/view/4689. Recuperado en Junio de 2016.

[6] Juan José Camargo Vega, Jonathan Felipe Camargo Ortega, Luis Joyanes Aguilar (2015). ***"Arquitectura tecnológica para Big Data"***. Revista científica. Volumen 1. Número 21. Disponible en: http://revistas.udistrital.edu.co/ojs/index.php/revcie/article/view/8451/10835. Recuperado en Junio de 2016.

[7] Jeremy Likness. ***"Base de Datos Orientada a Objetos NoSQL para .NET, Silverlight 4 y 5 y Windows Phone 7"***. Disponible en: https://sterling.codeplex.com/. Recuperado en Junio de 2016.

[8] Juan Francisco González Reyes (2011). ***"Bases de datos orientadas a objetos en la industria".*** Disponible en: http://es.slideshare.net/ziscko89/ejemplos-industria-bdoojuanfcoglz07230471ejemplos-de-uso-de-bases-de-datos-orientadas-a-objetos-en-la-industria. Recuperado en Junio de 2016.

[9] CATTEL, R. G et al. (2000). "***The Object Database Standard: ODMG 3.0"***. Morgan Kaufmann. Disponible en: https://books.google.es/books?id=M9fejgEACAAJ&dq=cattell+the+object+database+standard+odmg+3.0&hl=es&sa=X&ved=0ahUKEwisnJz04rbNAhVECsAKHVioCi4Q6AEIHjAA. Recuperado en Junio de 2016.

[10] María José Aramburu Cabo, Ismael Sanz Blasco (2013). ***"Bases de datos avanzadas"***. Disponible en: http://repositori.uji.es/xmlui/bitstream/handle/10234/48034/s73.pdf. Recuperado en Junio de 2016

[11] Instituto Geográfico Nacional. ***"Cartografía y Bases Geográficas"***. Disponible en: https://www.ign.es/ign/layoutIn/actividadesBDGintro.do. Recuperado en Junio de 2016.

[12] Jesús Efrain Revelo Burbano (2015). ***"Sistema de Información Geográfica para dispositivos móviles como herramienta de difusión de alertas de amenazas y riesgos naturales"*** .Trabajo Fin de Grado en la especialidad en Geomática por la Universidad Militar Nueva Granada. Disponible en: http://repository.unimilitar.edu.co/bitstream/10654/6572/1/trabajo de grado Jesus Revelo.pdf. Recuperado en Junio de 2016.

[13] **"Estado del arte de las bases de datos orientadas a objetos embebidas"**. Disponible en: https://repository.eafit.edu.co/bitstream/handle/10784/1591/7.%20marcoTeorico.pdf?sequence=7&isAllowed=y. Recuperado en Junio de 2016.

Bibliografía Legal

[14] **Orden EDU/3138/2011**, de 15 de noviembre. BOE Número 278, de 18/11/2011

[15] **Real Decreto 1128/2003**, de 5 de septiembre, BOE Número 223, de 17/09/2003

[16] **Ley Orgánica de la Educación 2/2006**, de 3 de mayo. BOE Número 106, de 04/05/2006

[17] **Ley Orgánica para la Mejora de la Calidad Educativa 8/2013**, de 9 de diciembre. BOE Número 295, de 10/12/ 2013

[18] Ley de Educación de Andalucía (LEA) 17/2007, de 10 de diciembre. BOJA Número 256 de 26/12/2007

[19] **Real Decreto 450/2010**, de 16 de abril. BOE Número 123 de 20/05/2010

[20] **Orden de 16 de junio de 2011**. BOJA Número 144 de 25/07/2011