DISEÑO Y DESARROLLO DE SISTEMAS DE INFORMACIÓN

RESUMEN DE TEMAS 1, 2 Y 3 (PRIMER PARCIAL)

Este resumen se basa en apuntes de clase y es conveniente apoyarlos con las diapositivas.

Índice

Tema 1: Introducción a los Sistemas de Información

Definición y panorama general

Un **sistema** es un conjunto de elementos o partes coordinadas que siguen una norma y que, relacionándose entre sí, consiguen un objetivo. Los sistemas de información empresarial existen debido al gran volumen de información que estos manejan.

Un **dato** es una representación simbólica de un atributo o variable, es decir, una pieza independiente que carece de contexto, mientras que la **información** es el dato o conjunto de datos en un contexto.

Un **entorno informático** está compuesto de: hardware, software, datos y personas.

Sistemas de Información empresarial

Un **sistema de información empresarial** es un conjunto de elementos interrelacionados que tienen como fin apoyar las actividades de una empresa mediante unos recursos:

- Físicos: personal y material/maquinaria (infraestructuras y energía).
- Conceptuales: dinero e información.

Los sistemas de información empresarial se dividen verticalmente en distintos **niveles gerenciales** (de arriba hacia abajo): planificación estratégica, control gerencial y control operativo; y horizontalmente en **áreas funcionales**: finanzas, RR.HH., servicios de información, producción y marketing.

El **servicio de información** forma parte de todas las áreas funcionales de una empresa, por lo que actualmente no es un área independiente.

El **entorno** afecta a la planificación **planificación estratégica**, mientras que el **interior** afecta al **control operativo**. La planificación estratégica utiliza **información resumida**, mientras que el control operativo usa **información detallada**.

Existen distintos sistemas de información empresarial en función de la estructura anterior:

- EIS (Sistemas de información ejecutiva): Sólo muestra información de manera sencilla.
- DSS (Sistemas de soporte a la decisión): Ayudan al control gerencial intermedio a tomar decisiones. Estos sistemas son predictivos y no realizan ninguna operación final. Combina estados anteriores con la situación actual para efectuar las recomendaciones y estima las consecuencias de una determinada decisión.
- KBS (Sistemas expertos - IA): Intentan codificar el comportamiento humano para luego reproducirlo.
- MIS (Sistemas de información gerencial): Dan información sobre áreas concretas de la empresa del pasado y del presente. Los resultados de información de un área repercuten en otras.
- TPS (Sistemas de procesamiento transaccional): Secuencias que deben hacerse todas seguidamente, o no se hacen.
- ERP (Planificador de recursos empresarial): Software central que incorporan módulos para investigar/manejar cierta información.
    - CRM (Gestor de relaciones con el cliente): se encuentran dentro de los ERP y realizan un seguimiento del cliente. Introducen dentro de un S.I. lo que ya hacen los vendedores.

Tema 2: Desarrollo de Sistemas de Información

Modelos de ciclo de vida

Un modelo es un conjunto de herramientas para la representación, manipulación, etc. de los datos.

Hay varios modelos de ciclos de vida, y entre ellos difieren en qué cantidad de trabajo se realiza en cada momento.

El **modelo en cascada** ("ciclo de vida clásico") no es el único, ni tampoco el mejor, en muchas situaciones. Este es un modelo de compartimentos/estancos, es decir, se trabaja en sólo un compartimento y para ir al anterior o el siguiente es necesario parar y trabajar en el que corresponde.

En el **modelo en espiral** no se conocen los requisitos al principio. Se van conociendo en el desarrollo y se va expandiendo.

Análisis de requerimientos

Los **requisitos de datos** recogen la necesidad de información del sistema.

Los **requisitos funcionales** recogen las funciones concretas del sistema.

Las **restricciones semánticas** son limitaciones a los requisitos funcionales en función de una combinación de datos. Pone en relación un requisito funcional con uno o varios requisitos de datos para restringir su funcionamiento.

En cualquier resolución de problemas tenemos que hablar sobre el problema y luego resolverlo. La **planificación** y el **análisis** describen el problema pero no la solución. Usan como herramienta el lenguaje natural; esto es esencial para llegar a un acuerdo con el cliente del sistema. Aquí se recaban datos sobre el problema. Una vez hecho todo el análisis se pasa al diseño para que trabajen en la solución.

Diseño

El **diseño** analiza la solución y la **implementación** realiza la solución. En el diseño no se requiere la participación del cliente.

En el ciclo de vida clásico el diseño se divide en dos partes: diseño **preliminar** y diseño **detallado**. En el primero se habla sobre la **estructura** de la solución, no sobre su contenido. Se analiza el sistema desde la perspectiva del problema estudiando la transformación del sistema. Haciendo una división vertical podemos dividir entre diseño **conceptual** y diseño **lógico/funcional**.

Por tanto nos quedan cuatro tipos de diseño: preliminar conceptual, preliminar funcional, detallado conceptual y detallado funcional. Para cada uno se usan herramientas distintas.

Para el almacenamiento de información tenemos los SGDB.

- Modelo **conceptual**: tiene como objetivos comprender la estructura, semántica, relaciones y restricciones de la BD, además de describir el contenido de ésta. Representación de los conceptos del problema → modelo conceptual → **modelo E/R**. Es característico de este modelo la expresividad, sencillez, minimalidad, representación gráfica y formalidad.
    - Enfoque centralizado: todo el sistema está diseñado centralmente.
    - Vistas de usuario/aplicación: diseño según cómo se ve afectada cada una de las partes.

El modelo conceptual no puede representar datos, y para ello tenemos el siguiente.

- Modelo **lógico**: puede representar los datos → **modelo relacional**
- Modelo **físico**: un diseño lógico-relacional no tiene porqué ser correcto. Descripción de la BD a nivel interno→ **SQL.**

El **modelado de flujo de datos** es una herramienta gráfica que permite representar procesos que cogen información, la transforman y la almacenan y/o la muestran como salida → Modelo de flujo de datos Pero la estructura de la solución no es la solución.

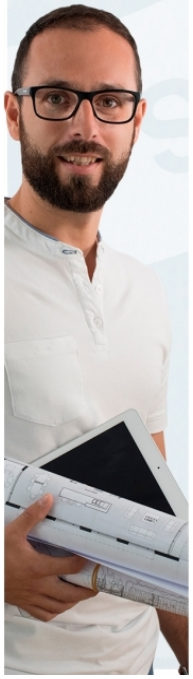
El **modelado de flujo de control** describe la secuencia de pasos para describir la solución → Operaciones de datos

Implementación

En la implementación plasmamos el diseño en un lenguaje (utilizar un SGDB). Tenemos por un lado la funcionalidad a nivel de BD (lenguaje de programación para DB) y por otro la funcionalidad del sistema en sí (lenguaje de desarrollo de aplicaciones).

Arquitecturas de sistemas de información

- **Arquitectura centralizada:** servidor que ejecuta la aplicación – terminales para consulta.
- **Cliente servidor:** tanto el servidor como el cliente tienen responsabilidades y carga. Surge el problema de tener una arquitectura para cada cliente.
- **Arquitectura por niveles:** el cliente nunca se conecta con los datos directamente. Hay un servidor intermedio en la capa de aplicaciones que sirve a los clientes, aislando la cada de datos en el servidor primario.
- **Arquitectura de servicios:** intenta encapsular toda transmisión de datos en estándares (flujos XML protegidos). Los proveedores de servicios tienen que describir sus servicios en un lenguaje estándar publicitados por los brokers, que publicitan los servicios en distintos proveedores.



Tema 3: Modelos de datos

Al principio se utilizaban ficheros para gestionar los datos, pero surgieron los modelos de datos cuando el volumen de datos a manejar empezó a ser enorme. Los tres primeros son el modelo jerárquico, el modelo en red y el modelo relacional. Un modelo tiene una serie de elementos con respecto a los datos y a las operaciones.

El modelo de fichero plano sólo almacena registros. Este modelo es muy simple, aunque dado el creciente volumen de información los sistemas gestores almacenan la información en ficheros de texto plano, pues es imposible manejar y procesar toda la información.

El **modelo de datos jerárquico** es una variación que permite representar la información de manera estructurada en una estructura de árbol, de modo que los valores se relacionan entre sí de forma 1:N de manera descendente y a nivel físico. Cuando las relaciones entre datos era jerárquica este modelo funcionaba muy bien, pero cada uno de los niveles sólo podía tener un padre, luego es imposible tener herencia múltiple.

El **modelo de datos en red** plantea que el modelo jerárquico, que daba problemas, tenga relaciones N:N, obteniendo así un árbol que admite múltiples padres, lo cual permite reducir las redundancias. Desaparece la herencia entre los campos ya que los objetos están relacionados entre sí directamente, y la integridad entre conceptos se mantiene, reduciendo así las redundancias.

El modelo de datos más extendido es el **modelo relacional**, planteado por Ted Codd. Este modelo se fundamenta fuertemente en la lógica proposicional. Se basa en el concepto de "relación", que formalmente se puede ver como un par de conjuntos (R,r), donde R=esquema (lo horizontal) y r=instancia (lo vertical).
El **esquema** es un conjunto de valores donde cada atributo (información del mundo) tiene un dominio.
La **instancia** es una aplicación de un esquema a un conjunto finito de datos (contenido de la tabla o parte de ella en un momento dado). Es un conjunto de todas la combinaciones posibles de los valores del dominio; realmente se almacenan las combinaciones que son posibles y se cumplen. El modelo relacional cumple la **primera forma normal**: para cada atributo, cada tupla sólo puede tener un valor atómico, es decir, en cada fila, una columna tiene un valor concreto del dominio, no varios valores separados por comas, pues en la consulta sería complicado de interpretar.

Una **relación** es lo que visualmente se entiende por una tabla. Una **base de datos relacional** es un conjunto finito de relaciones donde las filas son las tuplas y las columnas los atributos, pero sólo con esto tendríamos relaciones aisladas. Una **instancia de la BD** es una combinación de las relaciones posibles, pero no todas las instancias son posibles, y para mantener la corrección semántica tenemos dos restricciones de integridad, es decir, reglas que mantienen correcta la información que almacena. Por esto, hablamos de **esquema de base de datos**, que es una colección de esquemas de relación junto con las restricciones de integridad (restricciones que han de respetarse para que la BD sea válida).

- **Regla de integridad de entidad:** los atributos que forman parte de una clave no pueden estar vacíos ni repetirse.
- **Regla de integridad referencial:** la clave externa de una tabla de referencia siempre debe aludir a una fila válida de la tabla a la que se haga referencia. El valor de la clave externa debe ser igual a un valor del domino activo de la clave primaria, o nulo.

Para mantener la semántica deben respetarse las restricciones de integridad:

- **Restricciones específicas**, que son aquellas que se establecen sobre los datos y son propias de cada base de datos concreta.
- **Restricciones genéricas**, que se aplican a los atributos en función de su papel en la BD. Son meta-reglas, normas genéricas que generan reglas.

Bases de datos transaccionales

Una transacción es algo que debe realizarse por completo o no se realiza, porque para conservar la semántica la operación debe completarse entera. Se le exigen una serie de condiciones: ACID (Atomicidad, Consistencia, Aislamiento (*isolation*), Durabilidad). Garantiza la consistencia y estabilidad de las operaciones (invariante de representación). Cualquier transacción ACID tiene que garantizar que, ejecutada secuencialmente, quede en estado consistente antes y después.

Un conjunto de claves candidatas de atributos en una relación verifican unicidad y minimalidad. Una clave es mínima si ninguna de sus partes es clave por sí misma.

La clave externa establece que puede haber relaciones entre tablas distintas.

Los valores de los atributos son atómicos (no se permiten conjuntos de valores dentro de una tupla porque no sabemos si interpretarlos conjuntivamente o disyuntivamente).

Para mantener la integridad del sistema tenemos:

- **Integridad de entidad:** los atributos que forman parte de una clave son no nulos y el valor del conjunto de ellos no está repetido en procesos de inserción y actualización.
- **Integridad referencial:** al insertar el atributo hay que comprobar que el valor de la CE es nulo o concuerda con un valor de la CP. Si se actualiza la CE, comprobar las condiciones de la CE. Si se actualiza la CP se debe actualizar la CE. Si se borra la CP, se borra en cadena o se da valor nulo a la clave CE.

SQL es el lenguaje predominante para consultas. Está basado en álgebra relacional y cálculo relacional. Se divide en el DDL (gestiona la parte vertical, que es la estructura de una relación) y el DML (gestiona el contenido de una relación).

Modelo orientado a objetos

Surgió a partir de la programación orientada a objetos. La ventaja que planteaba era la ausencia de relaciones, donde las entidades pertenecientes al mismo tipo están agrupadas juntas. Tienen una estructura y comportamientos específicos. Se pierde la bidimensionalidad de las estructuras anteriores.

El primer problema que encontramos es que se presenta bastante limitado. Al fin y al cabo los objetos activos tienen que estar disponibles en memoria, pero cuando se acaba el programa dichos objetos tienen que acabar almacenados en un fichero, volviendo así al problema inicial, que es cómo almacenar los datos en el disco duro, por lo que hay un compromiso de persistencia. Al iniciar otra vez un programa los datos serán los mismos, pero no los objetos, pues habrán cambiado y serán otros ya que no se encuentran en la misma posición de memoria.

Un objeto tiene comportamiento y datos, encapsulando los datos y las operaciones que se pueden realizar sobre ellos: TDA. Ligado al concepto de la abstracción de datos tenemos el invariante de representación, que es una condición que se tiene que cumplir en todo momento.

Los lenguajes de consulta son eficientes y de alto nivel, e independientes de la máquina física.

Las bases de datos relacionales plantean un problema dada la primera forma normal, y es que dentro de una celda no pueden entrar objetos. Tienen además poca riqueza semántica pues son interpretaciones de la propia semántica. No soporta los tipos definidos por el usuario (sólo los dominio), es decir, *number*, *char*, etc. Hoy en día sí que soportan recursividad, pero esto no era posible antes. Hay una falta de

procedimientos y disparadores, pues es un modelo de datos, no funcional (*select* no es funcional). Además no admite herencia. Por todo esto se considera que las bases de datos relacionales no son apropiadas para manejar estructuras de datos complejas.

Si se convierten objetos al modelo relacional deja de ser un modelo orientado a objetos.

Características obligatorias:

- Persistencia: los datos tienen que ser exactamente los mismos en todo momento.
- Gestión de concurrencia: si dos usuarios intenta acceder al mismo dato concurrentemente no pasa nada en consultas.
- Recuperación ante fallos: antes de escribir algo escribe que va a hacerlo, lo cual permite ante un fallo o caída del sistema recuperarse al último estado guardado.
- Tiene que tener un lenguaje para inserción y recuperación de datos.

El problema de la orientación a objetos es que los lenguajes de consulta se vuelven complejos. Muchas veces hay que buscar la solución más simple y puede no ser necesario montar una base de datos orientada a objetos.

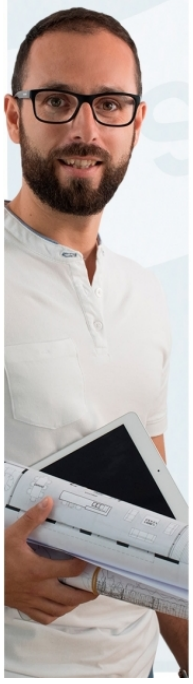
Modelo de datos objeto/relacional

Surgió por la necesidad de tener lo mejor de cada modelo. Es un sistema de gestión de BD, similar a una base de datos relacional, ya que intenta incorporar características del modelo orientado a objetos. El contenido de un objeto en el modelo objeto/relacional se representa en una tabla y cada objeto se almacena en una tabla distinta. Un objeto se puede relacionar con varios. El objeto de una subclase puede heredar extendiendo el relacional. Por todo esto, el modelo es más cercano al modelo relacional que al orientado a objetos.

Sistemas Gestores de Bases de Datos de 3º Generación

Los SGBD de 3ª Generación proporcionan gestión de objetos y reglas más ricas. Permite herencia y funciones (tiene un lenguaje de programación propio). La identificación de los objetos la proporciona el SGDB-3G si el objeto no tiene una clave propia. Los disparadores son fundamentales para este sistema. Pero tienen una serie de problemas:

1. En el modelo relacional la escalabilidad es complicada: horizontalmente (añadir columnas) implica un rediseño, y verticalmente, si se llega al límite de almacenamiento, no hay posibilidad de solucionar nada.
2. Si de por sí el sistema de tablas es complejo, en el objeto/relacional con las meta-tablas, que permiten relaciones entre objetos, el meta-esquema crece desproporcionadamente y genera problemas.
3. A veces se vuelve tan lenta que hay que normalizar la BD. Hay que dividirlo en dos o más tablas que estén más o menos relacionadas de modo que luego se puedan unir. Se suele resolver de forma funcional, comprobando que los datos que fueron separados cumplan ciertas condiciones.
4. Los *binary large objects* pueden contener hasta 32GB en un mismo atributo. La desestructuración de los datos complican la forma de la consulta. Desde esta perspectiva el modelo relacional permite la jerarquización, con lo cual las consultas se complican en la forma de expresarlas y en la ejecución, que también resulta más complicada.
5. En el modelo relacional las transacciones tienen subtransacciones internas que una vez terminadas se descartan, luego aumenta demasiado y de manera innecesaria el nivel de ejecuciones.



NoSQL

Por los problemas anteriores surge NoSQL (*not only SQL*, no es sólo SQL). En el modelo para BD NoSQL la información se almacena conforme se obtiene, sin estructurar. Los lenguajes NoSQL (está compuesto de varios lenguajes/soluciones) no usan SQL como lenguaje principal, pero sí accesorio.

Para realizar las consultas la información se encuentra sin estructurar, por tanto no existe la posibilidad de hacer un join (reunión natural), y para solucionar esto se utilizan 'clusters', estructuras híbridas para hacer reuniones naturales de manera rápida. Al introducir datos por tantos lados no garantizan completamente ACID (problemas de consistencia principalmente). Plante una gran ventaja: al no haber estructuras se escala bien horizontalmente.

En oposición de ACID aparece otro esquema para intentar garantizar la situación de los datos: BASE

- **Ba**sic availability: toda pregunta tiene respuesta en todo momento; pero no se tiene garantía de si la respuesta es correcta o no correcta, o si los datos son consistentes, o no. Esto ocurre porque el sistema es bastante mutable.
- **S**oft-state: hay que asumir que el estado del sistema cambia constantemente debido a la consistencia eventual.
- **E**ventual consistency: se puede garantizar la consistencia localmente, cuando se para la entrada de datos, hasta el momento de la comprobación en un punto concreto, pero lo que ocurre después (entrada de datos nueva) no se puede saber.

El modelo relacional es tan sumamente estático que cambiar cosas es complicado. NoSQL no tiene estructura de los datos y proporciona más:

- Flexibilidad: al no estar estructurados los datos no consulta sobre subconjuntos.
- Escalabilidad: proporciona la ventaja de introducir datos en otros nodos, haciendo cada uno su trabajo, con la posibilidad de luego reunirlos.
- Alto rendimiento: están optimizados para modelos de datos específicos y son capaces de almacenar muy rápidamente. Al dividir los datos responden también con gran velocidad.
- Alta funcionalidad: todas las soluciones NoSQL incorporan bibliotecas de acceso bastante completas, de modo que las aplicaciones pueden obtener datos muy variados.

Teorema de Brewer

"Es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes garantías: consistencia, disponibilidad y tolerancia a particiones."

- Consistencia: todos los nodos ven los mismos datos al mismo tiempo.
- Disponibilidad: garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no. En un sistema distribuidos tienen que estar disponibles todos los nodos o copias de estos.
- Tolerancia a particiones: el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales en alguna de las partes en la que se divide el sistema. Existe posibilidad de dividirlo.

Tipos de soluciones NoSQL

- Clave-Valor: tabla hash donde cada valor almacena un dato con cualquier complejidad. No existen tablas con identificadores (claves primarias), sino simplemente identificadores para cada cosa.
- Tabular o columnar: cada entrada puede tener al lado un subconjunto de columnas, de forma que para describir algo se puede introducir partes de cada tabla (conjuntos de columnas), algo parecido a herencia múltiple. Esta idea está asociada al tipo clave-valor. Proporciona ciertas ventajas: buena

gestión del tamaño (crece horizontalmente); permite cargar estructuras masivas; orientadas al flujo de datos; muy disponible; al estar en distintos nodos cada uno puede consultar lo que necesite de manera independiente.

- Documentales: almacenan documentos en estándares documentales como JSON, XML, BSON, etc., que contienen colecciones de clave-valor. En el documento la clave es la etiqueta. El objeto está autocontenido, luego no hace falta información para describirlo externamente. Los documentos están escritos en lenguaje natural. Al no haber estructuras son muy cercanas al programador, tienen una capacidad de desarrollo rápido y se orientan fácilmente a la web.
- Grafos: representan relaciones entre conceptos. Se puede representar un dominio en forma de grafo, de modo que si se intentan estudiar relaciones entre valores (no entre atributos), estas bases de datos son bastante potentes.

En esta comparativa, según la estructura de datos necesaria o de la estructuralidad de los datos, se puede optar por distintas soluciones.

El problema de NoSQL es su complejidad en la instalación, consultas y modelos de datos usados. Es un sistema con falta de madurez; no obstante conviene para manejar grandes volúmenes de datos y datos sociales. Aunque es bastante rápido, las operaciones son escasas, a diferencia del modelo relacional, pero al no garantizar ACID hay que pensar que no va a ser siempre la mejor opción.

NewSQL intenta aprovechar las mejores características con nuevas arquitecturas, motores SQL optimizados y protección transparente.