

Universidad Nacional del Nordeste
Facultad de Ciencias Exactas y Naturales y
Agrimensura

Monografía de Adscripción:
Sistemas de Bases de Datos Multiplataforma para Aplicaciones
Distribuidas



Alumna Iriana Nadia Strycek - LU: 39.753
Prof. Director: Mgter. David Luis La Red Martinez
Licenciatura en Sistemas de Información
Corrientes-Argentina
2010

Índice general

| | |
|--|----------|
| 1. Introducción | 1 |
| 2. Sistemas de Bases de Datos | 5 |
| 2.1. Definición | 5 |
| 2.2. Componentes de una Base de Datos | 6 |
| 2.3. Tipos de Usuarios en Base de Datos | 6 |
| 2.4. Conceptos Básicos de Base de datos | 7 |
| 2.5. Niveles de Abstracción en Base de datos | 8 |
| 2.6. Subsistema de un DBMS: | 10 |
| 2.7. DBMS | 11 |
| 2.7.1. Características y Objetos: | 11 |
| 2.8. Modelo Relacional | 30 |
| 2.8.1. NORMALIZACIÓN: | 31 |
| 2.9. SGBD | 34 |
| 2.9.1. ORACLE | 34 |
| 2.9.2. SQL SERVER | 35 |
| 2.9.3. DB2 Universal Database | 37 |
| 2.9.4. INFORMIX | 38 |
| 2.9.5. MySQL y PostgreSQL: | 39 |

| | |
|--|-----------|
| 3. Aplicaciones Distribuidas | 43 |
| 3.1. Tipos de Aplicaciones Distribuidas | 46 |
| 3.2. Arquitectura de las Aplicaciones Distribuidas | 53 |
| 3.2.1. La Capa de Servidor | 55 |
| 3.2.2. Servicios de Base de Datos | 57 |
| 3.2.3. Otros Servicios | 60 |
| 3.2.4. La capa de Negocios | 61 |
| 3.2.5. La capa de Presentación | 77 |
| 4. DB2 Universal Database | 83 |
| 4.1. Características | 84 |
| 5. Conclusiones | 91 |
| Bibliografía | 93 |
| Índice alfabético | 95 |

Capítulo 1

Introducción

Las bases son cualquier conjunto de datos organizados para su almacenamiento en la memoria de un ordenador, diseñado para facilitar su mantenimiento y acceso de una manera estándar. Los datos suelen aparecer en forma de texto, números o gráficos.

Otra definición más completa de bases de datos afirma que es un “conjunto exhaustivo, no redundante, de datos estructurados, organizados independientemente de su utilización y su implementación en máquina, accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predecible en el tiempo, donde la información se encuentra almacenada en una memoria auxiliar que permite el acceso directo a un conjunto de programas que manipulan esos datos”.

Multiplataforma es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multi-

plataforma podría ejecutarse en Windows en un procesador x86, en GNU/Linux en un procesador x86, y en Mac OS X en un x86, sin ningún tipo de problemas.

El advenimiento de Internet ha provocado una tendencia inevitable al desarrollo de aplicaciones distribuidas. Mediante una arquitectura de procesamiento distribuido se pueden dispersar los procesadores, los datos y otros elementos de conforman una aplicación. Esta dispersión ofrece un sistema mas sensible a las necesidades de los usuarios, capaz de ofrecer tiempos de respuesta mejores y minimizar los costes de comunicación.

Debido al crecimiento de la complejidad y diversificación del Web, las aplicaciones Web distribuidas han toman mayor importancia. Muchos de los sitios para negocios en línea o comercio electrónico son aplicaciones Web distribuidas de gran escala. En general, estas aplicaciones Web son sistemas complejos, basados en una variedad de componentes de hardware y software, protocolos, lenguajes, interfaces, y estándares.

En los últimos tiempos se ha escuchado hablar sobre una nueva base de datos, llamada “universal”, que puede almacenar y hacer búsquedas no solamente de datos alfanuméricos sino también de imágenes, audio, video y otros objetos.

Esta ventaja de las bases de datos universales abre un sin número de oportunidades que permiten mejorar tanto los servicios como las aplicaciones.

La réplica de datos es la tecnología clave para aprovechar todo el poder de los ambientes ya que permite enviar los datos a cualquier sitio para cubrir todos los requerimientos de la empresa, desde oficinas centrales a sucursales, usuarios móviles proveedores, clientes y socios de negocios.

DB2 Universal Data Base incluye todo lo necesario para implementar una solución de replicación de datos en cualquier tipo de ambientes distribuidos o heterogéneos.

Capítulo 2

Introducción a los Sistemas de Bases de Datos

2.1. Definición

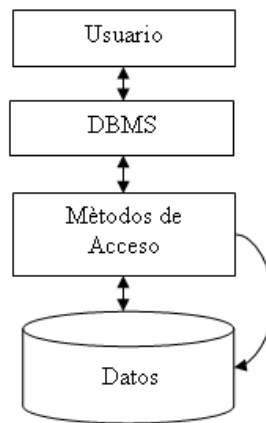
Un Sistema de Bases de Datos:

- Es un sistema que almacena datos que están relacionados.
- Es un repositorio en donde guardamos información integrada que podemos almacenar y recuperar.
- Un conjunto de información almacenada en memoria auxiliar que permite acceso directo y un conjunto de programas que manipulan esos datos.

2.2. Componentes de una Base de Datos

Los principales componentes de una *Base de Datos* son:

- Hardware: constituido por dispositivo de almacenamiento.
- Software: que es el DBMS o Sistema Administrador de Base de Datos.
- Datos: los cuales están almacenados de acuerdo a la estructura externa y van a ser procesados para convertirse en información.



Estructura de Base de Datos

2.3. Tipos de Usuarios en Base de Datos

Los tipos de usuarios son:

- Usuario Final: es la persona que utiliza los datos, esta persona ve datos convertidos en información.

- Desarrollador de Aplicaciones: es la persona que desarrolla los sistemas que interactúan con la Base de Datos.
- DBA: es la persona que asegura integridad, consistencia, redundancia, seguridad, este es el Administrador de Base de Datos quien se encarga de realizar el mantenimiento diario o periódico de los datos.

Las personas que tienen acceso al DBMS se clasifican de la siguiente manera:

USUARIOS INGENUOS: son aquellos que interactúan con el sistema por medio de aplicaciones permanentes.

USUARIOS SOFISTICADOS: son aquellos con la capacidad de acceder a la información por medios de lenguajes de consulta.

PROGRAMADORES DE APLICACIÓN: son aquellos con un amplio dominio del DML capaces de generar nuevos módulos o utilerías capaces de manejar nuevos datos en el sistema.

USUARIOS ESPECIALIZADOS: son aquellos que desarrollan módulos que no se refieren precisamente al manejo de los datos, si no a aplicaciones avanzadas como sistemas expertos, reconocimientos de imágenes, procesamiento de audio y demás.

2.4. Conceptos Básicos de Base de datos

Archivo: son conjuntos de registros.

Registros: son conjuntos de campos.

Campos: es la mínima unidad de referencia.

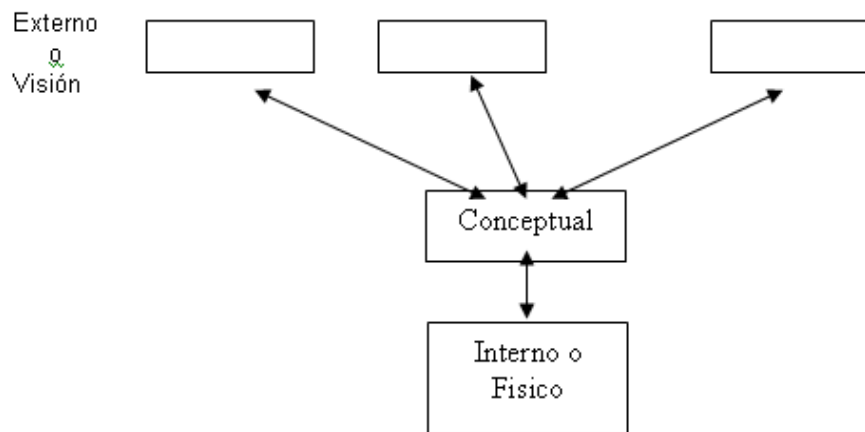
Archivo Clientes

| | | | |
|------------|----|--------|------------|
| Juan Pérez | 20 | \$500 | ← registro |
| Ana Alban | 28 | \$1000 | |
| José Mora | 30 | \$1500 | |

↑
campo

Componentes de una Base de Datos

2.5. Niveles de Abstracción en Base de datos



Niveles de Abstraccion en Bases de Datos

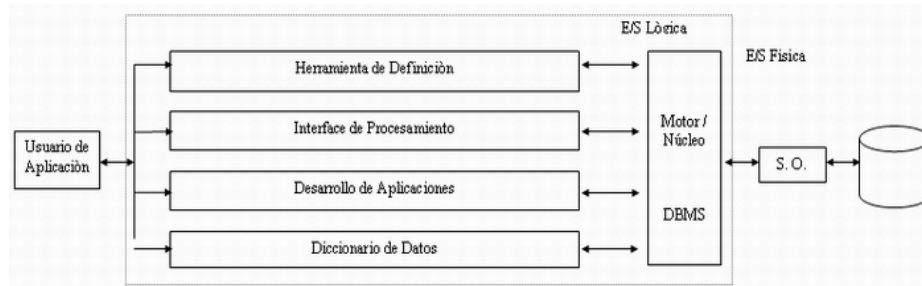
Externo: esa es la visión del usuario final, se ve como se manejan los datos ya convertidos en información. Es aquel en el que se presenta al usuario final y que puede utilizar combinaciones o relaciones entre los datos que conforman a la base de datos global. Puede definirse como la forma en el que el usuario aprecia la información y sus relaciones.

Conceptual: se ve como esta estructurada la Base Datos. Es aquel en el que se definen las estructuras lógicas de almacenamiento y las relaciones que se darán entre ellas. Ejemplos comunes de este nivel son el diseño de los registros y las ligas que permitirán la conexión entre registros de un mismo archivo, de archivos distintos incluso, de ligas hacia archivos.

Interno: se ve como se almacena los datos físicamente. Es aquel en el que se determinan las características de almacenamiento en el medio secundario. Los diseñadores de este nivel poseen un amplio dominio de cuestiones técnicas y de manejo de hardware. Muchas veces se opta por mantener el nivel físico proporcionado por el sistema operativo para facilitar y agilizar el desarrollo.

DBMS (Data Managment System (Sistema Administrador de Base de Datos)): Los Sistemas Gestores de Bases de Datos son un tipo de software muy específico, dedicado a servir de interfaz entre las bases de datos y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. En otros textos que tratan este tema, o temas relacionados, se mencionan los términos SGBD y DBMS, siendo ambos equivalentes, y acrónimos, respectivamente, de Sistema Gestor de Bases de Datos y Data Base Management System, su expresión inglesa.

2.6. Subsistema de un DBMS:



Subsistema de un DBMS

Motor o Núcleo DBMS: recibe los requerimientos lógicos de E/S y los convierte en operaciones de lectura y escritura.

Lógicos: son cualquier tipo de consulta requerimiento de lectura con ingreso de datos (requerimiento de estructura) es ayudado por el Sistema Operativo para convertir estos requerimientos lógicos en físicos que actúan sobre dispositivos de almacenamiento.

Herramientas de definición: permite definir y modificar la estructura de la Base de Datos, en este nivel definimos lo que se conoce como “Esquema ” que es la definición total de Base de Datos, es que definimos la estructura de la tabla, los tipos de campos, las restricciones para los campos.

Subesquema: manejo de vistas de datos, de niveles externos.

Esquema: manejo de niveles conceptuales.

Interface de Procesamiento: provee de las facilidades de actualización, despliegue y visualización de datos.

Desarrollo de Aplicaciones: permite generar una aplicación por ej.: generadores de formas, pantalla, código, herramientas case, etc.

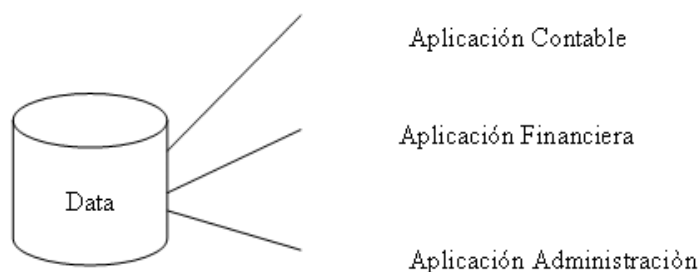
Diccionario de Datos: este es el componente al subsistema con el que interactúan directamente el DBA, le proporciona niveles de consulta y reportes útiles para su trabajo de administración. Es la descripción de la estructura de Base de Datos y relaciones entre datos, y programas.

2.7. DBMS

2.7.1. Características y Objetos:

Independencia de Datos: el DBMS provee una independencia de datos vs. las aplicaciones.

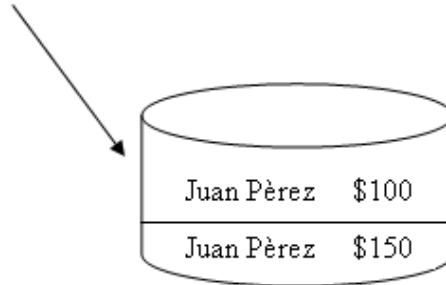
El cambio en datos no implica cambio en programas y viceversa (menor coste de mantenimiento).



Se cambia las aplicaciones y no se afecta la Data

Independencia de Datos

Minimizar Redundancia (Datos repetidos): desperdicio de Espacio de Almacenamiento.



Minimizar Redundancia

Independencia de datos es proteger nuestro programa de aplicaciones frente a las modificaciones en la estructura de datos y viceversa, ya sea en forma física ò lógica:

Independencia Física: es protección a los programas de aplicación debido a cambios en la estructura de archivos, con cambios en las características de los campos. Ej: cambio de clave primaria a secundaria.

Independencia Lógica: protección a los programas de aplicación cuando se modifica el esquema.

Redundancia, datos repetidos y distribuidos en cualquier parte. El efecto que ocasiona la redundancia es tener inconsistencia de datos y desperdicio de espacio de almacenamiento.

Esta se presenta cuando se repiten innecesariamente datos en los archivos que conforman la base de datos.

Inconsistencia de Datos: dato que esta en lugar con un valor y se encuentra en otro lugar con otro valor. Ej: se actualiza el archivo cliente pero no se actualiza el archivo de transacciones. Ocurre cuando existe información contradictoria o incongruente en la base de datos.

Integridad de Datos: es el conjunto de seguridades que son utilizadas para mantener los datos correctos. Se utilizan cuando no existe a través de todo el sistema procedimientos uniformes de validación para los datos.

Fuente de Error: estas fuentes de error se origina si el programa de entrada de datos no esta validado. Ej: fallas de hardware, actualizaciones incompletas, defectos del software, inserción de datos no validos, errores humanos.

Una técnica que usa el DBMS de una entrada de datos no válida es la validación.

Validación: es proteger los datos, validar los datos en la entrada de datos.

Existen tipos de validaciones:

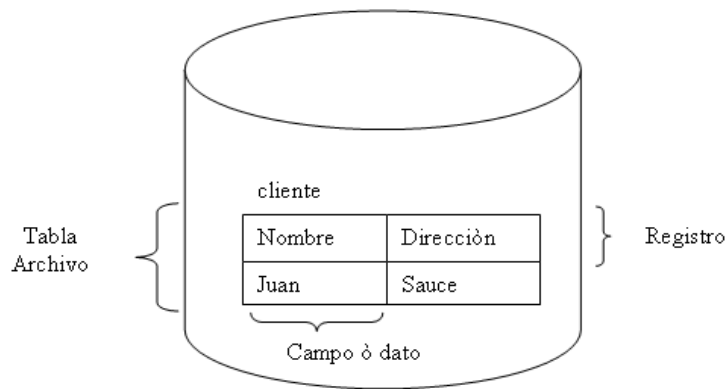
- Tipo de Dato: es si se define un campo como carácter o char y no se pueden ingresar números enteros.
- Valor de Dato: si se define un valor entero se puede especificar un rango y no se puede pasar de ese valor.
- Valores Claves / No Nulos: asegura registros únicos y cuyos valores no sean nulos.

Integridad Referencial: asegura al DBMS que no exista registros hijos sin sus registros padres correspondientes.

Control de Concurrencia o Simultaneidad: Se da en ambiente multi-usuario, tratando de acceder aun objeto de datos al mismo tiempo. Ocurre cuando el sistema es multiusuario y no se establecen los controles adecuados para sincronizar los procesos que afectan a la base de datos. Comúnmente se

refiere a la poca o nula efectividad de los procedimientos de bloqueo.

Granularidad: es el tamaño de las unidades aseguradas. Ej: la granularidad puede proteger un campo, un registro, un archivo, etc.



Control de Concurrency

Dead-lock(bloqueo): es la técnica que evita errores de concurrencia, se da cuando se desarrolla una espera circular entre dos transacciones y cada una de estas solicita una actualización sobre el mismo archivo, no permite a otros usuarios el recurso hasta que termine el proceso, se da la espera circular.

Recuperación de Datos:

Recuperar los datos frente a las fuentes de error mencionadas anteriormente. La restauración de la Base de Datos a su estado normal es responsabilidad del DBA, quien es el responsable de implantar procedimientos de detección de error y recuperación.

El DBA es quien tiene el control centralizado de la base de datos. Se persigue con esto reducir el número de personas que tengan acceso a los detalles técnicos y de diseño para la operación del DBMS.

Las soluciones principales de un DBA son:

DEFINICION DEL ESQUEMA: Crea el esquema original de la base de datos y genera el diccionario de datos por medio de proposiciones en DDL.

DEFINICION DE ESTRUCTURAS DE ALMACENAMIENTO Y METODOS DE ACCESO: Se encarga de generar a seleccionar estructuras para el medio secundario y definir los métodos de acceso a la información, esto ultimo por medio de proposiciones en DML.

MODIFICACION DE ESQUEMA Y ORGANIZACIÓN: Es una actividad poco frecuente que consiste en rediseñar el esquema de la base de datos. Esto se haría necesario ante la modificación abrupta de las condiciones originales que dieron pie al diseño del esquema primario. Las proposiciones para llevar a cabo esta tarea se realizan en DDL.

CONCESION DE AUTORIZACIONES DE ACCESO: Se encarga de registrar a los usuarios para permitir su acceso al DBMS. Asigna a cada uno de ellos una serie de atributos que le permiten gozar de privilegios como el acceso a determinadas áreas de aplicación, de los datos o del uso de recursos en el sistema.

ESPECIFICACION DE LAS LIMITANTES DE INTEGRIDAD: Crea una serie de tablas donde se especifica el conjunto de restricciones que serán aplicables durante los procesos de actualización.

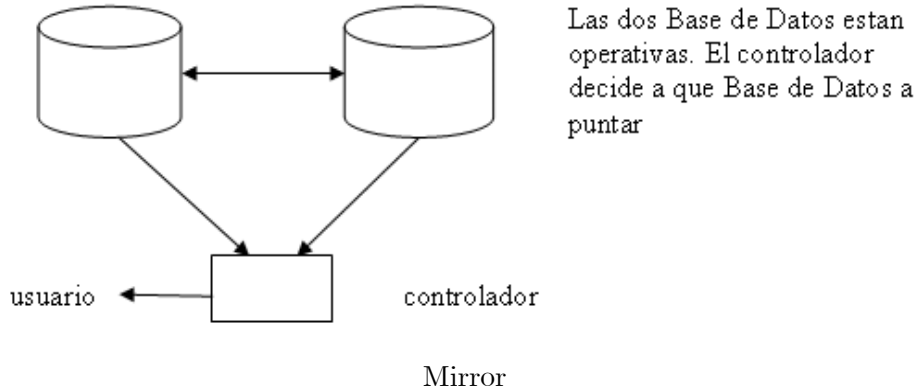
Para recuperar:

Backup (respaldo): disco duro, cinta.

Backup caliente: Base de Datos está operativa.

Backup frio: Base de Datos no está operativa.

Mirror o Espejo



Archivos de Logs: (registro de transacción): Son las transacciones diarias que se registran en la Base de Datos. Cuando ocurre un problema se acude a los archivos de logs se hace un REVERSO y también se puede recuperar la ultima transacción que se hizo.

Seguridad de los Datos: Se presentan cuando no es posible establecer claves de acceso y resguardo en forma uniforme para todo el sistema, facilitando así el acceso a intrusos.

La seguridad de los datos se puede definir en los siguientes aspectos:

- Objeto a asegurar: el primer objeto a asegurar son los objetos, programas y finalmente al esquema.
- Codificación de Claves: el DBMS provee la seguridad de los Login (usuario y password).
- Control de Acceso: se especifican seguridades contra accesos indicados

orientado a personas no autorizada.

Control y Administración de Recursos: El DBMS debe proveer al DBA de todos los mecanismos para control y administración de recursos.

Establecimiento de Relaciones entre Datos: El DBMS debe proveer los recursos para el establecimiento de relaciones entre los datos, cuales son las relaciones: $1 \rightarrow 1$, $1 \rightarrow n$, $n \rightarrow n$

Ciclo de vida de las operaciones de Base de datos:

Etapas:

- Planificación del Proyecto.
- Definición del Sistema.
- Recolección y Análisis de los Requisitos.
- Diseño de la Base de Datos.
- Selección del SGDB / DBMS.
- Diseño de la Aplicación.
- Prototipo.
- Implementación.
- Conversión y Carga de Datos.
- Prueba.
- Mantenimiento.

Estas etapas no son estrictamente secuenciales de hecho hay que repetir algunas de las etapas varias veces haciendo lo que se conoce como “Ciclos de Re-alimentación” por Ej: los problemas que se encuentran en la etapa de Diseño de la Base de Datos pueden requerir una recolección de requisitos adicional y su posterior análisis.

Planificación del Proyecto: Esta etapa con lleva la planificación de como se puede llevar acabo las etapas de ciclo de vida de la manera más eficiente, hay tres componentes principales:

- El trabajo que se va a realizar.
- Los recurso para llevarlo acabo.
- El dinero para pagar todo ello.

Definición del Sistema: En esta etapa se especifica el ámbito y los índices de la aplicación de la Base de Datos así como con que otros sistemas interactúan. También hay que determinar quienes son los usuarios y las áreas de la aplicación.

Recolección y Análisis de los Requisitos: En esta etapa se recoge y analiza los requerimientos de los usuarios y de las áreas de aplicación.

Esta información se la puede recoger de varias formas:

- Entrevistando el personal de la empresa concretamente aquellos que son considerando expertos en la área que se de.
- Observando el funcionamiento de la empresa.

- Examinando documentos sobre todo aquellos que se utilizan para recoger o visualizar la información.
- Utilizando cuestionario para recoger información de grandes grupos de usuarios.
- Utilizando la experiencia adquirida en el Diseño de Sistemas similares.

Esta etapa tiene como resultado en conjunto de documentos con las especificaciones de requisitos de los usuarios en donde se describen las operaciones que se realizan en la empresa desde distintos puntos de vista.

Los requisitos de desarrollo involucran el software y hardware necesario para la implementación, los recursos humanos necesarios (tanto internos como externos), la formación al personal.

Diseño de Base de datos: En esta etapa se crea un esquema conceptual de la base de datos. Se desarrollan las especificaciones hasta el punto en que puede comenzar la implementación. Durante esta etapa se crean modelos detallados de las vistas de usuario y sobre todo las relaciones entre cada elemento del sistema, documentando los derechos de uso y manipulación de los diferentes grupos de usuarios.

Si parte de la información necesaria para crear algún elemento establecido ya se encuentra implementado en otro sistema de almacenamiento hay que documentar que relación existirá entre uno y otro y detallar los sistemas que eviten la duplicidad o incoherencia de los datos.

El diseño consta, como se vio anteriormente, de tres fases: el diseño global o conceptual, el diseño lógico y el modelo físico.

La primera fase consiste en la producción de un esquema conceptual que es independiente de todas las consideraciones físicas. Este modelo se refina después en un esquema lógico eliminando las construcciones que no se puede representar en el modelo de Base de Datos escogido (relacional, orientado a objeto, etc).

En la tercera fase el esquema lógico que traduce un esquema físico para el sistema gestor de Base de Datos escogido.

La fase de diseño físico considera las estructuras de Almacenamiento y los métodos de acceso necesarios para proporcionar un acceso eficiente a la Base de Datos en memoria secundaria.

Selección del SGBD / DBMS: Si no se dispone de un Sistema Gestor de Base de Datos o que si se encuentra obsoleto se debe escoger un SGBD que sea adecuado para el sistema de información, esta elección se debe hacer en cualquier momento antes del diseño lógico.

Diseño de aplicación: En esta etapa se diseñan los programas de aplicación que usaran y aplicarán la Base de Datos, en esta etapa el diseño de la Base de Datos es paralelo, en la mayor parte de los casos no se puede finalizar el diseño de las aplicaciones hasta que se ha terminado el diseño de Base de Datos.

Por otra lado la Base de Datos exige para dar soporte a las aplicaciones una retroalimentación desde el diseño de las aplicaciones al diseño de la Base de Datos. En esta etapa hay que asegurarse de que toda la funcionalidad especificada en los requisitos de usuarios se encuentra en el diseño de la aplicación.

Prototipo: Esta etapa es opcional, es para construir prototipos de la

aplicación que permiten a los diseñadores y al usuario probar el sistema, un prototipo es un modelo de trabajo de las aplicaciones del sistema. El prototipo no tiene toda la funcionalidad del sistema final pero es suficiente para que los usuarios puedan usar el sistema e identificar que aspectos que están bien, y los cuales no son adecuados. Además de poder sugerir mejorando o incluyendo nuevos elementos.

Implementación: En esta etapa se crean las definiciones de la Base de Datos a nivel conceptual externo o interno, así como los programas de aplicación la implementación de la Base de Datos se realiza mediante las sentencias SQL, estas sentencias se encargan de crear el sistema de la base, los ficheros donde se almacenarán los datos y las vistas de los usuarios.

Los programas de aplicación se implementan utilizando lenguaje de tercera y cuarta generación, partes de estas aplicaciones son transacciones de la Base de Datos que se implementan también mediante lenguaje SQL. La sentencia de este lenguaje se pueden embeber en un lenguaje de programación anfitrión como Visual Basic, Java, etc. También se implementan en esta etapa todos los controles de seguridad e integridad. Una vez totalmente detallado el modelo conceptual se comienza con la implementación física del modelo de datos, a medida que se va avanzando en el modelo el administrador del sistema va asegurando la corrección del modelo y el validado la utilidad del mismo.

Conversión y Carga de Datos: Esta etapa es necesaria cuando se esta reemplazando un sistema antiguo por uno nuevo. Los datos se cargan desde el sistema viejo al nuevo directamente o si es necesario se convierte al formato que requiera el nuevo SGBD y luego se carga esta etapa se la suele llamar "Migración".

Prueba: En esta etapa se prueba y válida el sistema con los requisitos especificados por los usuarios. Para ello se debe diseñar una materia de test con datos reales que se deben llevar acabo de manera metódica y rigurosa. Si la fase de prueba se lleva correctamente descubrirá los errores en los programas de aplicación y en la estructura de la Base de Datos.

Mantenimiento: Una vez que el sistema esta completamente probado o implementado se pone en marcha. El sistema esta ahora en la fase de mantenimiento en la que se lleva acabo las siguientes tareas:

- monitoreo de las prestaciones del sistema y mantenimiento,
- y actualización del sistema.

En esta última etapa todos los usuarios del sistema acceden a la base de datos y deben asegurarse el correcto funcionamiento de la misma, que sus derechos son los adecuados, teniendo a su disposición cuanta información necesiten. También deberán asegurarse que el acceso a los datos es cómodo, práctico, seguro y que se han eliminado, en la medida de lo posible, las posibilidades de error.

El administrador se asegura que todos los derechos y todas las restricciones han sido implementadas correctamente y que se ha seguido en manual de estilo en la totalidad de la implementación

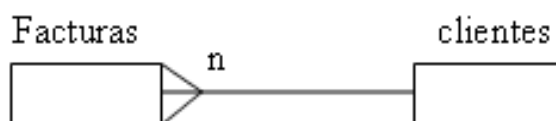
Modelo Entidad - Relación:

Modelaje: es el proceso mediante el cual podemos identificar las propiedades dinámicas ò estáticas de un dominio de aplicación con mira a su transformación en un diseño interpretable en un sistema computarizado. Es el plasmar los

requerimientos de los usuarios en un programa para poder implementarlo.

Entidad: es el objeto sobre el cual se requiere mantener ò almacenar información.

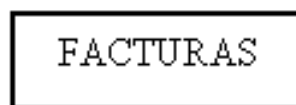
Relación: es la asociación significativa y estable entre dos entidades.



Relacion

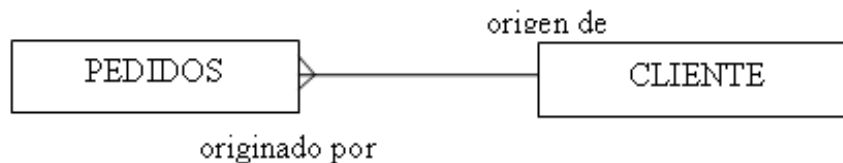
Atributo: son las propiedades que describen y califican una entidad. Ej: Entidad cliente (nombre, apellido, dirección, edad, sexo).

Las entidades se las representa mediante cajas que se colocan el nombre de la entidad con letras mayúsculas. Ej:



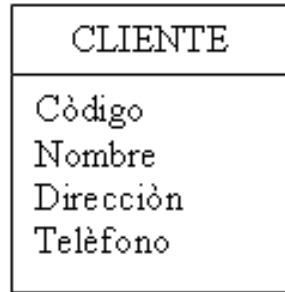
Entidad

Las relaciones se representan con líneas que conectan las cajas de las entidades. Ej:



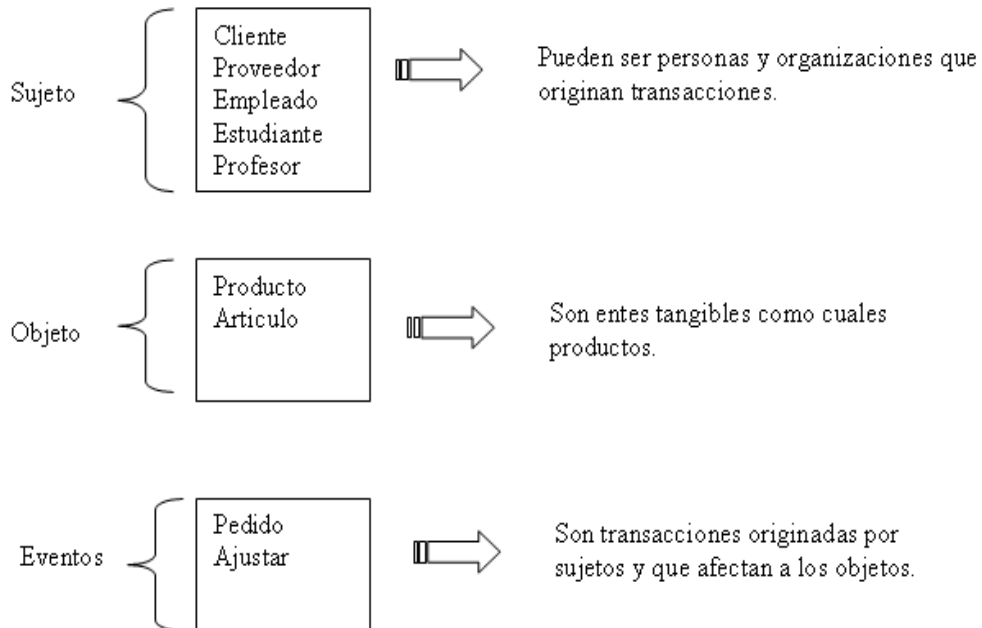
Relaciones

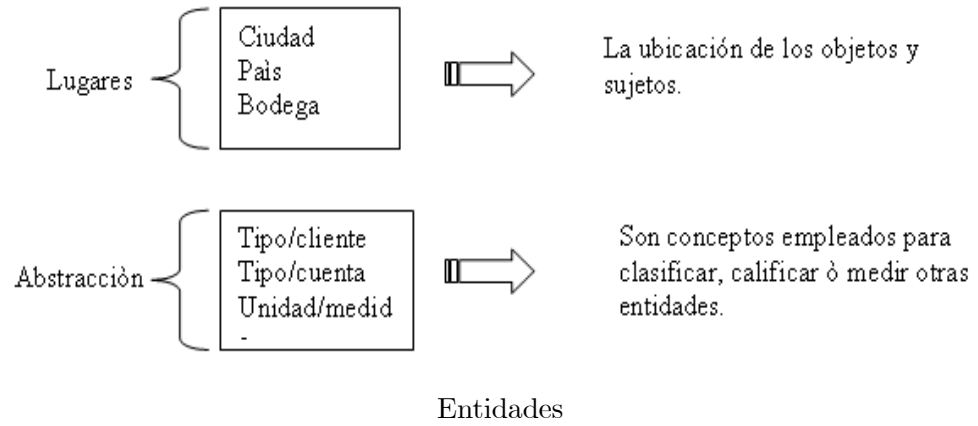
Los atributos se incluyen dentro de las cajas de las entidades y se escriben con minúsculas. Ej:



Atributos

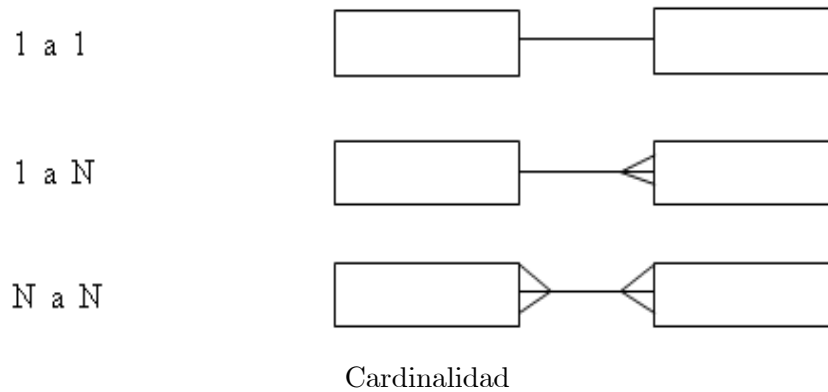
Entidades: se puede considerar entidades a los sujetos, objetos, a los eventos, a los lugares y a las abstracciones.



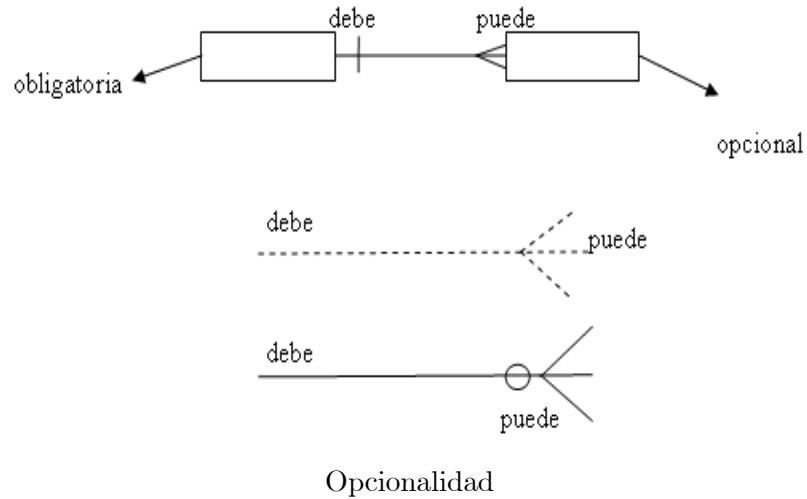


Relaciones: las relaciones tiene tres propiedades o características:

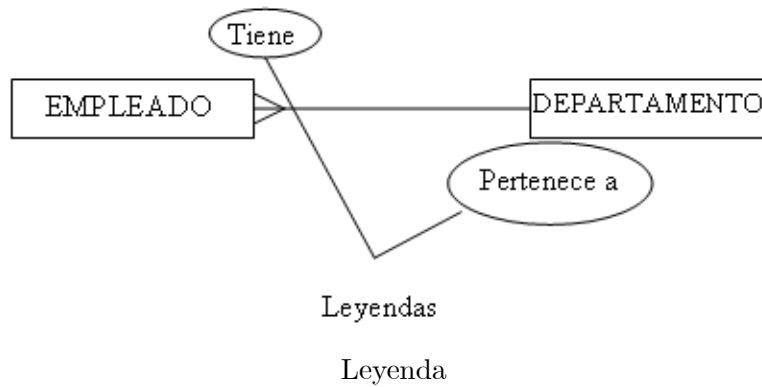
- *Grado o Cardinalidad:* que se clasifica en:



- *Opcionalidad:* es la participación obligatoria u opcional en la entidad de la relación.



- *Leyenda:* es una expresión que escribe el rol de cada entidad en la relación.



Como se lee el Grado o Cardinalidad:

- *Uno a muchos:* una instancia de la entidad A se relaciona con una o más instancias de la entidad B.



Uno a Muchos

- *Muchos a muchos*: una instancia de la entidad A se relaciona con una o más instancias de la entidad B y una instancia de la entidad B se relaciona con una o más instancias de le entidad B.



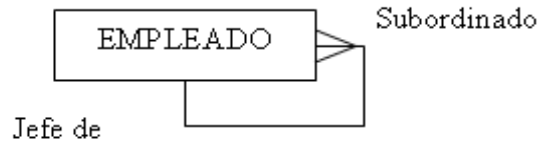
Muchos a muchos

- *Uno a uno*: una instancia de la entidad A se relaciona con uno y sólo una instancia de la entidad B.



Uno a uno

Relación Recursiva: Una instancia de una entidad se asocia con instancia de sí misma, es opcional en los dos extremos, es decir, no hay el carácter de obligatorio. Ej:



Un Empleado puede ser jefe de uno o más empleados y un Empleado puede ser subordinado de un y sólo un Empleado

Relacion Recursiva

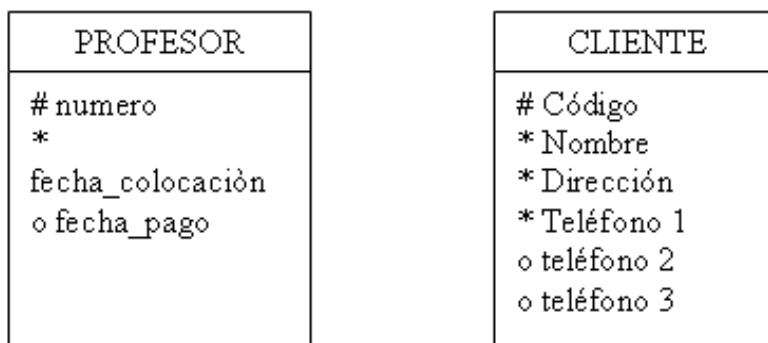
Atributo: Los atributos son empleados para identificar, describir, calificar o expresar el estado de una entidad.

Todo entidad posee un atributo o combinación de atributos que se denomina “clave primaria” y que emplea para diferenciar cada instancia de los demás.

Adicionalmente los atributos pueden ser obligatorios u opcionales:

- A los atributos que forman parte de la clave primaria se los identifica anteponiéndoles el signo de numero (#).
- A los atributos obligatorios se les antepone el asterisco (*).
- A los atributos opcionales se les antepone un circulo (o).

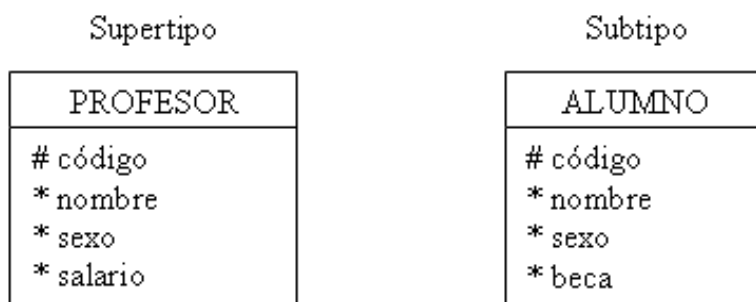
Ejemplo:



Atributos Obligatorios u Opcionales

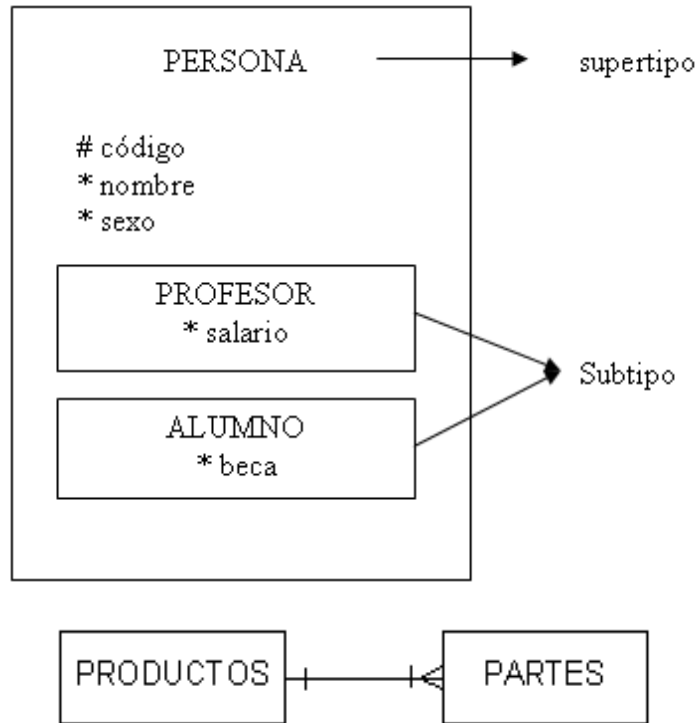
En un diagrama entidad-relaci3n tambi3n se puede agrupar las entidades en supertipo y en subtipo:

- Los supertipo agrupan a dos o m3s entidades subtipo.
- Los subtipo heredan los atributos de las entidades supertipo.



Supertipo-Subtipo

- Cada subtipo puede tener relaciones propias independientes del supertipo.
- Los subtipos se representan como cajas dibujadas dentro de la caja del supertipo.



2.8. Modelo Relacional

| Modelo Relacional | Programador | Campo |
|-------------------|-------------|---------|
| Relación | Archivo | Tabla |
| <u>Tupla</u> | Registro | Fila |
| Atributo | Campo | Columna |

Modelo Relacional

El conjunto de una base de datos es el conjunto de tabla relacional.

La tabla: Es un conjunto de restricciones.

2.8.1. NORMALIZACIÓN:

El proceso que revisa que la tabla este bien estructurada se llama normalización.

La normalización esta basada en el concepto de formas normales cada forma normal tiene un conjunto de reglas que deben ser verificada (1NF, 2NF, 3NF).

Estas formas normales son anidadas, es decir que para que una relación este en 3FN debe haber pasado por 2FN y esta por la 1FN.

Conceptos usados en la normalización:

- *Dependencia Funcional:* es la relación que existe entre dos atributos.

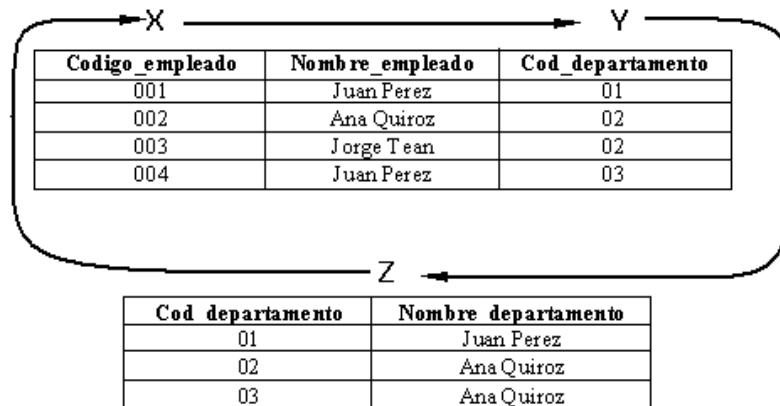
Ejemplo: Dado un valor de X existe un valor de Y entonces Y es funcionalmente dependiente de X.

- *Claves o llaves:* Es el atributo que le da la diferencia a cada tabla este atributo hace que no tengamos tuplas o filas repetidas.

| <u>Cod_cliente</u> | <u>Nombre_cliente</u> |
|--------------------|-----------------------|
| 001 | Juan Perez |
| 002 | Ana Quiroz |
| 003 | Ana Quiroz |
| 004 | Juan Perez |
| 005 | José Lopez |
| | |

Dependencia transitoria: Es la dependencia que esta encadenada.

$X \rightarrow Y \rightarrow Z$ = Dado un valor de "X" existe un valor de "Y" y dado un valor de "Y" existe un valor de "Z" entonces se dice que "z" es transitivamente dependiente de "X".



Dependencia Transitoria

Primera Forma Normal (1FN): Las celdas o campos deben tener valo-

res singulares.

Las entradas de cualquier columna o atributo deben ser de la misma clase. Cada columna debe tener un nombre único.

Dos filas o tuplas no pueden ser iguales.

| ID | Deporte | Valor |
|-----|--------------|-------|
| 100 | Ski | 200 |
| 150 | Natación | 50 |
| 175 | <u>Squas</u> | 50 |
| 200 | Natación | 50 |

Al realizar operaciones sobre la tabla se pueden presentar problemas, estos problemas son llamadas anomalías, estas anomalías pueden ser de inserción, actualización, eliminación, etc.

Segunda Forma Normal (2FN): Todo atributo no clave depende de un atributo clave “Eliminar dependencias parciales a la clave Primaria de una Tabla”.

Tercera Forma Normal (3FN): Una relación esta en 3FN si y solo si esta en 2FN y tiene dependencias transitivas, es decir, dependencia encadenada.

2.9. Características de SGBD Más Populares

2.9.1. ORACLE

Junto con SQL Server, lidera el mercado NT. Puede funcionar en una gran cantidad de sistemas operativos y diversidad de Hardware.

Prácticamente tenemos a todas las familias de UNIX, MVS, VM, Siemens ICL, y Novell Netware.

Además el funcionamiento está optimizado para ajustarse a las peculiaridades de cada sistema operativo.

La idea de Oracle de potenciar los grupos de trabajo distribuido se necesita un acceso a los datos de dicho grupo.

En lo referente a Internet las aplicaciones web pueden acceder a los datos almacenados en la bases de datos de Oracle así como presentar documentos HTML generados dinámicamente a partir de un modelo de una consulta.

Oracle soporta paralelismo dentro de una consulta lo que proporciona un incremento notable en su ejecución.

También soporta Procesos de Transacciones On-Line y Data WareHousing.

Atendiendo a las características de manejabilidad, escalabilidad, rendimiento y soporte entre plataformas, se tiene una arquitectura de servidor con ejecución multihilo y rendimiento de multiprocesadores simétricos (SMP).

Las bases de datos pueden crecer hasta límites que en la práctica pueden considerarse inalcanzables.

Las características más relevantes en lo referente a la escalabilidad residen en las tablas particionadas.

Los Protocolos de Red Soportados por Oracle son los siguientes: Net 8, TCP/IP, IPX/SPX, Pipes con nombre, DECNET, DCE, NDS y LU 6.2 (AP-PC).

Oracle es básicamente una herramienta cliente/servidor para la gestión de Bases de Datos.

Para desarrollar en Oracle se utiliza PL/SQL, un lenguaje de 5^a generación, bastante potente para tratar y gestionar la base de datos, también por norma general se suele utilizar SQL al crear un formulario.

2.9.2. SQL SERVER

Características:

Escalabilidad: Se adapta a las necesidades de la empresa, soportando desde unos pocos usuarios a varios miles.

Potencia: *Microsoft SQL Server* es la mejor base de datos para Windows NT Server.

Gestión: Con una completa interfaz gráfica que reduce la complejidad innecesaria de las tareas de administración y gestión de la base de datos.

Orientada al desarrollo Visual Basic, Visual C++, Visual J++, Visual Interdev, Microfocus Cobol y muchas otras herramientas son compatibles con Microsoft SQL Server.

Diseñada desde su inicio para trabajar en entornos Internet e Intranet, Microsoft SQL Server es capaz de integrar los nuevos desarrollos para estos entornos específicos con los desarrollos heredados de aplicaciones “tradicionales”. Es más, cada aplicación que se desarrolle para ser empleada en entornos de red local puede ser utilizada de forma transparente -en parte o en su totalidad- desde entornos Internet, Intranet y Extranet.

Plataforma de desarrollo fácil y abierta: integrada con las mejores tecnologías de Internet como ActiveX, ADC y Microsoft Transacción Server y con las mejores herramientas de gestión y desarrollo para Internet como Front-Page97, Microsoft Office97 y Visual Interdev.

Diseñada para Internet: Es el único gestor de base de datos que contiene de forma integrada la posibilidad de generar contenido HTML de forma automática.

- La Base de Soluciones Integradas: La Integración total con BacOffice permite resolver toda las necesidades de infraestructura de la empresa con un sólo paquete.
- Potente y Escalable: Microsoft SQL Server es la única base de datos cuyo rendimiento sobre Internet está publicado.

Otras:

- Datos distribuidos y replicación.
- Data Warehousing y amplio soporte de datos.
- Integración Internet y correo electrónico.

- Gestión y administración centralizada de bases de datos. Disponibilidad, fiabilidad y tolerancia a fallos.

- Mejoras en programabilidad y lenguaje.

- Seguridad.

2.9.3. DB2 Universal Database

IBM DB2 UDB es el primer sistema de gestión de bases de datos relacionales multimedia preparado para Internet que es lo suficientemente potente como para satisfacer las demandas de las grandes empresas y flexible para operar en pequeñas y medianas organizaciones.

Las principales características de DB2 para Linux son su funcionalidad web, su soporte Java y la inclusión de DB2 UDB Web Control Center.

DB2 UDB Web Control Center permite a los administradores gestionar la base de datos a través de la web.

DB2 Universal Database para Linux permite a los desarrolladores trabajar con JDBC y SQLJ lo que proporciona mayor rendimiento en las aplicaciones de bases de datos Java.

Los servidores DB2 soportan cualquier comunicación basadas en los protocolos APPC, IPX/SPX, NetBios, TCP/IP, Pipes con nombres.

2.9.4. INFORMIX

La presencia de Informix se deja sentir en una cantidad de plataformas, concretamente Windows NT y UNIX.

Completo conjunto de herramientas gráficas.

Existe la posibilidad de gestionar múltiples bases de datos remotas.

Avanzada escalabilidad en las bases de datos y un alto rendimiento en cualquier entorno.

Una de las características que se encuentra implementada para WinNT son:

- la implementación de procesadores virtuales.
- manipulación directa de acceso a disco.
- uso de la multitarea prevenible propia del sistema.
- implementación como si fuera un servicio más.
- integración con el registro del sistema.
- utilización del servicio de eventos NT para alertas.

Internet: Informix incluye de forma integrada el servidor Netscape Fax-Trap, para soporte y conectividad de aplicaciones Web e Intranet.

Como características esenciales tenemos:

- Servicios escalables Web/Inet de alto rendimiento.

- Entorno de desarrollo basado en Java/JavaScript seguridad basada en SSL con autenticación.
- Encriptación en integridad de mensajes.
- Control de accesos.
- Soporte de certificados del lado del cliente.
- Asistentes para la publicación en Internet.

La seguridad e integridad de los datos es uno de los objetivos de Informix.

Algunas Diferencias con ORACLE:

1. Oracle siempre ha sido considerada una base de datos para uso más general que Informix.
2. Informix dispone de infinidad de Datablades.
3. En Oracle, se tiene que definir los usuarios dentro la base de datos (gestión interna de usuarios). En cambio, Informix utiliza los mismos usuarios de Linux (los que creas con adduser), simplificando la administración.

2.9.5. MySQL y PostgreSQL:

MySQL:

Su principal objetivo de diseño fue la VELOCIDAD. Se sacrificaron algunas características esenciales en sistemas más “serios” con este fin.

Otra característica importante es que consume MUY POCOS RECURSOS, tanto de CPU como de memoria.

Licencia GPL a partir de la versión 3.23.19.

Ventajas:

- Mayor rendimiento.
- Mayor velocidad tanto al conectar con el servidor como al servir selects y demás.
- Mejores utilidades de administración (backup, recuperación de errores, etc.).
- Aunque se cuelgue, no suele perder información ni corromper los datos.
- Mejor integración con PHP.
- No hay límites en el tamaño de los registros.
- Mejor control de acceso, en el sentido de qué usuarios tienen acceso a qué tablas y con qué permisos.
- MySQL se comporta mejor que Postgres a la hora de modificar o añadir campos a una tabla “en caliente”.

Inconvenientes:

- No soporta transacciones, “roll-backs” ni subselects.
- No considera las claves ajenas. Ignora la integridad referencial, dejándola en manos del programador de la aplicación.

PostgreSQL:

Postgres intenta ser un sistema de bases de datos de mayor nivel que MySQL, a la altura de Oracle, Sybase o Interbase. Licencia BSD.

Ventajas:

- Por su arquitectura de diseño, escala muy bien al aumentar el número de CPUs y la cantidad de RAM.
- Soporta transacciones y desde la versión 7.0, claves ajenas (con comprobaciones de integridad referencial).
- Tiene mejor soporte para triggers y procedimientos en el servidor.
- Soporta un subconjunto de SQL92 MAYOR que el que soporta MySQL. Además, tiene ciertas características orientadas a objetos.

Inconvenientes:

- Consume BASTANTES más recursos y carga más el sistema.
- Límite del tamaño de cada fila de las tablas a 8k!!! (hasta la versión 7.1).
- Es de 2 a 3 veces más lenta que MySQL.
- Menos funciones en PHP.

Capítulo 3

Aplicaciones Distribuidas

Una aplicación distribuida es aquella cuyo objetivo final se alcanza mediante la ejecución de diversos procesos independientes que por lo general se ejecutan en equipos diferentes y que de una forma u otra se pasan datos entre ellos mediante protocolos de comunicaciones bien establecidos.

Formas arquitectónicas (o de diseño) que adquieren las aplicaciones distribuidas: Cada una de ellas plantea ciertas ventajas e inconvenientes que será necesario tener en cuenta a la hora de crear el código para los diferentes componentes de la aplicación.

Características de las aplicaciones distribuidas: Independientemente de su arquitectura, todas las aplicaciones distribuidas comparten ciertas características que las diferencian notablemente de las aplicaciones centralizadas y de las aplicaciones de escritorio. Conocer esas características y evaluar su impacto en las aplicaciones es primordial para escoger el diseño más adecuado de las

mismas. No es cierto que una aplicación se pueda diseñar de cualquier manera.

A continuación se ofrece una breve enumeración de los aspectos primordiales que han de evaluarse a la hora de diseñar una aplicación distribuida:

Concurrencia: De igual forma que en las aplicaciones centralizadas, las aplicaciones distribuidas serán utilizadas por cierto número de usuarios concurrentemente. Aspectos como las transacciones, los bloqueos de recursos o el uso de la CPU de los equipos a los que acceden muchos usuarios son determinantes a la hora de diseñar una arquitectura con la máxima eficacia.

Topología de la red: A pesar de que al día de hoy los anchos de banda cada vez son más amplios, el tráfico de red puede ser un aspecto importante que condicione el tiempo de respuesta de la aplicación. En muchos casos también será necesario tener en cuenta el tipo de red (LAN o WAN), o si la aplicación será o no accesible a través de Internet. La forma de distribuir los procesos de la aplicación tendrá que tomar en consideración el tipo de red que soportará el tráfico de datos.

Ubicación de la lógica: Dado que en una aplicación distribuida intervienen varios procesos, será necesario decidir en cuál de los posibles procesos físicos se sitúa cada componente lógico de la aplicación. Mientras que algunos procesos, como la presentación de datos o la recuperación de los mismos, tienen un sitio natural, otros, como la validación o la navegación, pueden ocupar diversos lugares dentro del diagrama que conforma la estructura de la aplicación. En muchas ocasiones la ubicación de los componentes lógicos impacta sobre el rendimiento, sobre la reutilización del código o sobre la facilidad de programación.

Homogeneidad de las plataformas: En una aplicación distribuida los sistemas operativos involucrados o los lenguajes de desarrollo utilizados pueden ser un factor a tener en cuenta a la hora de decidir algunos aspectos importantes, como por ejemplo el modo de pasar datos entre procesos. La utilización de estándares puede ser muy útil a la hora de crear aplicaciones distribuidas que permanezcan abiertas a diversos sistemas heterogéneos, pero si las plataformas son similares es posible alcanzar mejor rendimiento sacrificando interoperabilidad.

Seguridad: Una aplicación distribuida mantiene procesos que de una forma u otra están a la escucha en una red, lo que aumenta la vulnerabilidad de la aplicación. Será necesario establecer políticas de seguridad que impidan el acceso no autorizado a los procesos. Pedir al usuario un nombre y una contraseña al iniciar el programa es probable que no sea suficiente.

No existe una solución única para todos los problemas. A pesar de que en la actualidad se priman ciertas arquitecturas sobre otras, la realidad es que la decisión final dependerá de todos los factores anteriormente citados, y de otros que a veces no se tienen mucho en cuenta pero que también poseen su importancia, como el coste total de la aplicación o la complejidad de la solución respecto al problema planteado. Ni es recomendable subestimar las necesidades de una aplicación compleja, ni tampoco conviene sobredimensionar la estructura de una aplicación sencilla que puede resolverse con medios más asequibles.

Con las aplicaciones distribuidas el mero análisis funcional de las características de una aplicación ya no es suficiente y es necesario también considerar la distribución y estructura de los procesos involucrados. La tarea del archi-

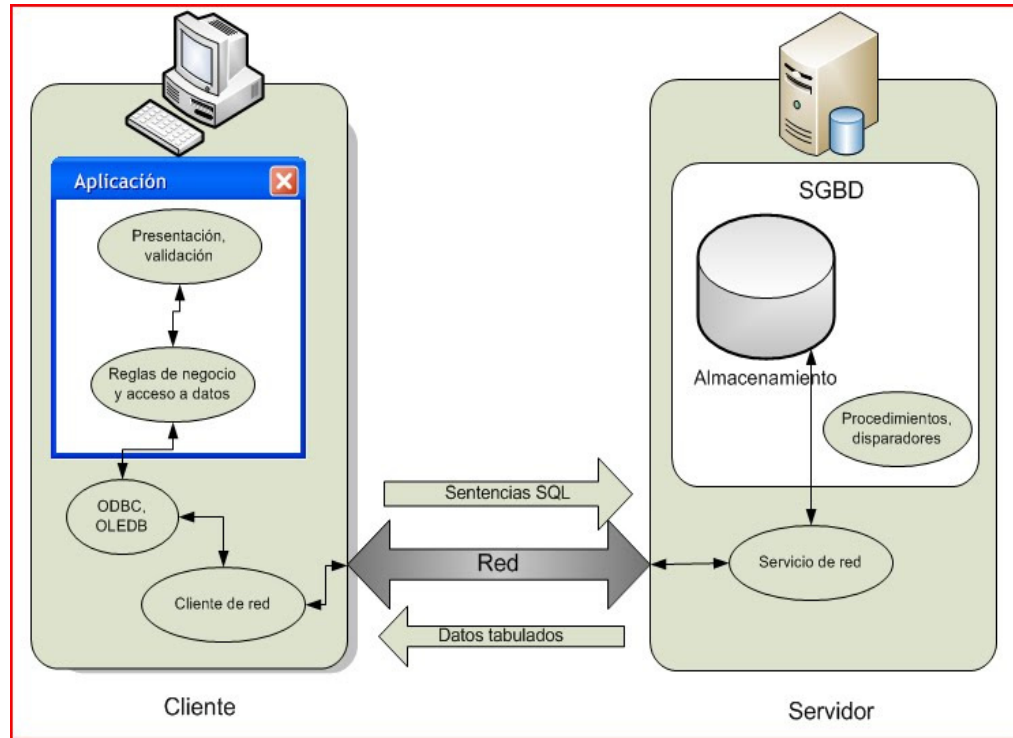
tecto de aplicaciones consistirá precisamente en tomar la decisión de diseño más adecuada para resolver el problema, ajustándose lo mejor posible a los requerimientos y tomando en cuenta todos los factores implicados.

3.1. Tipos de Aplicaciones Distribuidas

Se tratarán tan sólo dos tipos de aplicaciones distribuidas: las **cliente-servidor**, y las **aplicaciones en n-capas**.

Aplicaciones Cliente-Servidor

En las aplicaciones cliente-servidor que se llamarán “tradicionales” sólo se encuentran dos procesos principales. Uno de ellos se encarga fundamentalmente de proporcionar los datos que se le solicitan y de procesar los datos que se le envían. Se llama servidor tanto al proceso que realiza estas funciones como al equipo en el que dicho proceso está alojado. El otro proceso, al que se llama cliente, se ejecuta en el equipo del usuario que maneja la aplicación, y sus funciones principales son solicitar datos al servidor, presentarlos al usuario para que este realice cierto trabajo con ellos y enviar los cambios al servidor para su reproceso si es necesario.



Esquema de una aplicación cliente-servidor tradicional

En las aplicaciones cliente-servidor el proceso servidor es por lo general un **sistema gestor de bases de datos** (SGBD, o DBMS por sus iniciales en inglés, Database Management System). Los SGBD, como Microsoft SQL Server u Oracle, mantienen en el equipo servidor un **servicio de red** que recoge las peticiones que le llegan en forma de sentencias SQL y las transmite al verdadero proceso gestor de la base de datos, que se encarga de seleccionar los registros indicados por la petición o bien de realizar las operaciones de actualización de los registros. Si la solicitud genera registros, el resultado de la consulta se transforma en datos tabulados que se envían de vuelta por la red al equipo que los ha solicitado.

Los SGBD además cumplen otras muchas funciones, como la de mantener

la integridad de los datos, proporcionar seguridad, aislar al resto de la aplicación de la localización de los datos, etc. Este es el punto en que entra la distribución de procesos de una arquitectura distribuida. Los SGBD permiten incluir buena parte de la lógica de negocio de la aplicación en forma de *procedimientos almacenados*, *disparadores*, *reglas intrínsecas* de la base de datos, etc. Por ejemplo, es posible realizar transacciones que cubran la actualización de los registros de varias tablas manteniendo la atomicidad de la operación (es decir, que todas las actualizaciones se realicen o no se realice ninguna), mediante el uso de procedimientos almacenados.

También es muy habitual disponer en disparadores, que se ejecutan asociados a sentencias de actualización, ciertas operaciones de cálculo automático o de actualización en cascada. Estas operaciones, además, suelen ser de alto rendimiento ya que los SGBD ejecutan estas operaciones de forma muy optimizada y a bajo nivel. En suma, el SGBD no es solamente un pasivo almacén de datos, sino que constituye una pieza importante en los procesos de la aplicación distribuida.

En el otro extremo de la red se encuentran las estaciones de trabajo que alojan las aplicaciones cliente. Para que las estaciones de trabajo puedan realizar peticiones al servidor es necesario que cuenten con el **cliente de red** adecuado al SGBD y que esté correctamente configurado. Los clientes de red son proporcionados por el fabricante del SGBD y son específicos para cada SGBD. El cliente de red “sabe” cómo conectarse al servicio de red del SGBD y transforma las peticiones del cliente en peticiones comprensibles por el SGBD. Esto implica que si el SGBD cambia por cualquier motivo será necesario distribuir el nuevo cliente de red entre todas las estaciones de trabajo que accedan a él.

El trabajo del cliente consiste principalmente en ofrecer al usuario un interfaz que le permita solicitar datos, visualizarlos en pantalla, trabajar con ellos y enviar las posibles actualizaciones al servidor. El programa cliente transforma la petición del usuario en sentencias SQL que se envían al servidor. Si este devuelve datos, el cliente recoge los datos tabulados y los muestra en un formato accesible al usuario (en una rejilla de datos, en un gráfico, etc.). El interfaz de usuario del programa permite al usuario manipular dichos datos, y el programa convierte esa manipulación otra vez en sentencias SQL que se envían al servidor y que provocarán la actualización de los registros.

A pesar de que, técnicamente, cualquier aplicación distribuida puede considerarse una aplicación cliente-servidor, se ha reservado este término para designar la arquitectura interna de ciertas aplicaciones en las cuales los componentes lógicos de los procesos se distribuyen de la forma física más sencilla posible. La aplicación cliente incluye dentro de sí misma todos los componentes de validación, presentación y manipulación de los datos, y también incluye la conectividad con el cliente de red que permite la comunicación con el SGBD. El servidor, por su lado, puede actuar como mero almacén de datos o incorporar reglas de negocio en forma de procedimientos almacenados o disparadores.

Las *aplicaciones cliente-servidor tradicionales* son la forma más básica de *aplicación distribuida*. Dada la simplicidad de su arquitectura, el coste de implementación suele ser más bajo que el de una aplicación distribuida en n-capas. También presenta cierta ventaja en lo que a rendimiento se refiere, ya que al existir menos capas de software entre el cliente y el servidor los datos pasan de un lado al otro más rápidamente. Pero aquí acaban sus ventajas.

El primer escollo que se afronta en una aplicación de este estilo es que

la conectividad con el servidor se realiza desde cada una de las estaciones de trabajo. Esto genera dos problemas: en primer lugar, cada una de las estaciones de trabajo concurrentes está consumiendo recursos del servidor en forma de conexiones abiertas. En segundo lugar, la conectividad entre las estaciones de trabajo y el servidor es específica para cada SGBD, lo que implica un mayor esfuerzo a la hora de configurar los sistemas y dificulta mucho la posibilidad de que una aplicación sea capaz de trabajar con diferentes SGBD, ya que un cambio probablemente requerirá una recodificación de la aplicación cliente o de alguna de sus partes, con el consiguiente problema de redistribución.

Además, la reutilización del código es mínima. Al incluir dentro de sí misma las reglas de presentación, actualización o validación de datos, una segunda aplicación que requiera acceder a las mismas tablas o utilizar las mismas validaciones no podrá beneficiarse del código ya escrito, a no ser que esté escrito en la misma plataforma y sea posible duplicarlo, método que complica el mantenimiento de las aplicaciones.

Un factor adicional en contra de este modelo es que requiere distribuir mucho software entre los equipos cliente. En el caso de una aplicación de poco volumen esto quizás no sea relevante, pero cuando las aplicaciones alcanzan cierto volumen distribuir y mantener gran cantidad de software entre los equipos clientes puede ser no sólo costoso sino inviable. Una aplicación tipo ERP, por ejemplo, es un caso en el que necesariamente se ha de abandonar la arquitectura cliente-servidor tradicional.

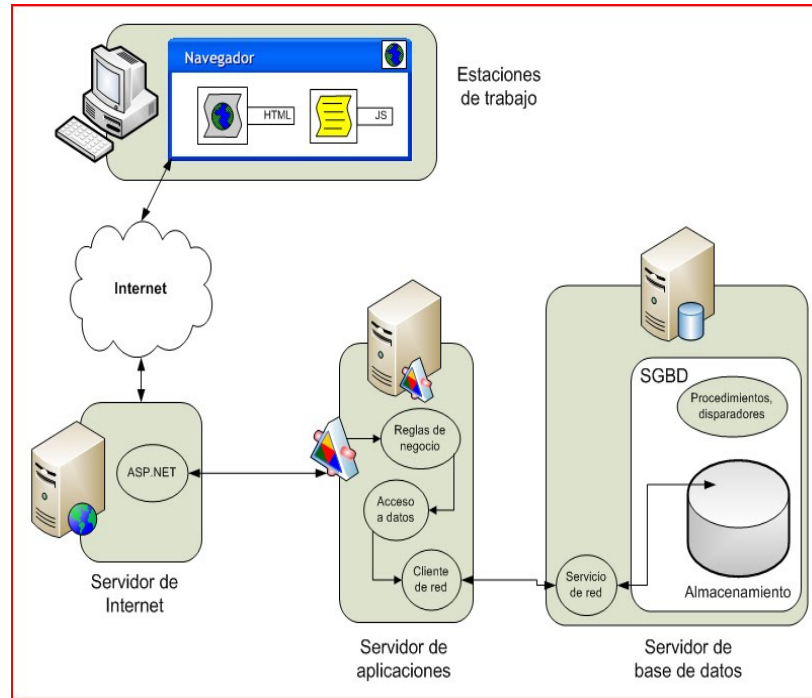
Por último, este modelo arquitectónico es prácticamente inviable cuando nuestra aplicación cliente ha de acceder al servidor mediante Internet. Los protocolos de red que utilizan los servicios y clientes de red de un SGBD

suelen utilizar puertos que habitualmente están cerrados a los cortafuegos y, a pesar de que es posible cambiar esos puertos, abrir al acceso público el servicio de red de un SGBD supone crear una vulnerabilidad importante.

En suma, las aplicaciones cliente-servidor tradicionales deben escogerse sólo en casos muy específicos en los que el rendimiento o la simplicidad son factores determinantes. Cuando necesitamos compartir funcionalidad, mejorar la distribución o sencillamente facilitar el mantenimiento, es necesario optar por un modelo mucho más abierto y más distribuido. Y, por supuesto, Internet nos aboca completamente a una arquitectura distribuida en n-capas.

Aplicaciones en n-capas

En una aplicación distribuida en n-capas los diferentes procesos están distribuidos en diferentes capas no sólo lógicas, sino también físicas. Los procesos se ejecutan en diferentes equipos, que pueden incluso residir en plataformas o sistemas operativos completamente distintos. Cada equipo posee una configuración distinta y está optimizado para realizar el papel que le ha sido asignado dentro de la estructura de la aplicación, de modo que tanto los recursos como la eficiencia global del sistema se optimicen.



Esquema de una aplicación distribuida en n-capas

Un ejemplo posible sería una aplicación de comercio electrónico en Internet. En primer lugar hallaríamos el servidor que contiene los datos, cuyo SGBD puede incluir ciertos procedimientos almacenados o disparadores que sean globales a la lógica de la base de datos. En segundo lugar se hallaría un equipo que fuera capaz de contener ciertos componentes que realicen determinadas reglas de negocio de la aplicación, como la recuperación de datos o la comprobación de seguridad. Un tercer equipo podría comunicarse con este y ofrecer servicios de generación dinámica de páginas web, como sería el caso de un servidor IIS que alojase una aplicación web mediante ASP.NET. En el último escalón estaría el navegador, que podría ejecutar ciertas validaciones de usuario mediante código javascript en las propias páginas.

3.2. Arquitectura de las Aplicaciones Distribuidas

En una aplicación distribuida en **n-capas** los diferentes elementos que integran la aplicación se agrupan de forma lógica según la funcionalidad que reciben o suministran al o desde el resto de los elementos. Así, algunos elementos se limitarán a recibir peticiones de datos mientras que otros interactuarán con el usuario y su función será principalmente la de solicitar a otros elementos la información que el usuario precisa.

Atendiendo al papel que los distintos elementos juegan dentro de la aplicación, distinguimos con claridad tres grupos lógicos en los que podemos agrupar elementos según su funcionalidad:

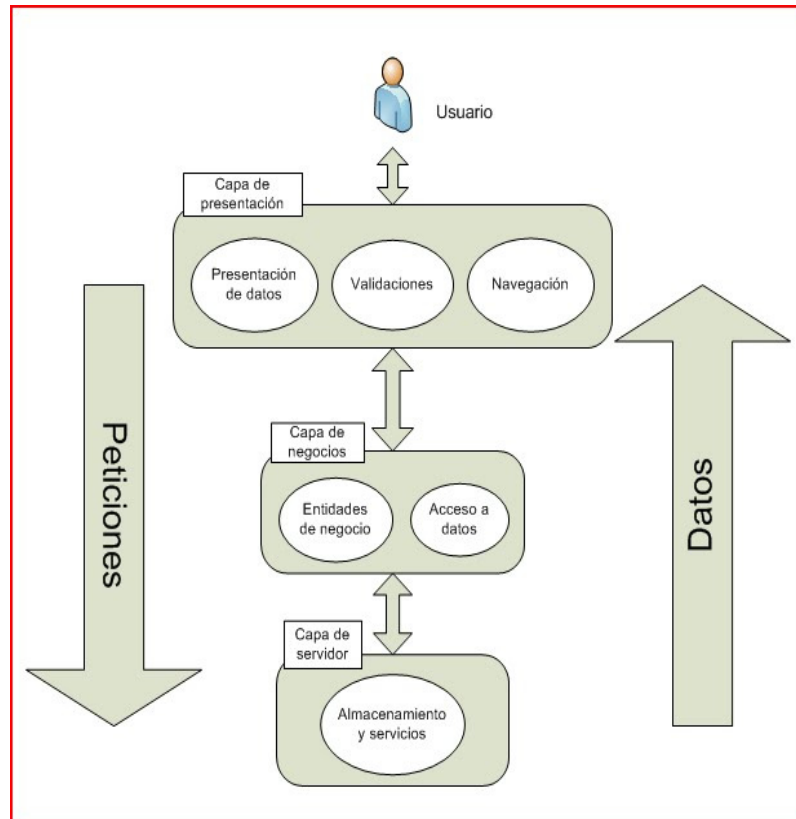
La capa de servidor incluye aquellos elementos que se encargan de recibir las peticiones de datos o de acceso a servicios básicos del sistema y de suministrar a otros elementos la información solicitada.

La capa de negocios encapsula las reglas de acceso a datos y la gestión de procesos internos de la aplicación.

La capa de presentación se encarga de la lógica necesaria para interactuar con el usuario de la aplicación.

Una vez agrupada la funcionalidad en capas lógicas es fácil relacionar unas con otras. El usuario interactuará con la capa de presentación, solicitando datos o desencadenando acciones. Las solicitudes serán atendidas por la capa de negocios, que se encargará de su gestión o de la traducción necesaria para que la capa de servidor realice la tarea solicitada. Si la capa de servidor debe proporcionar datos estos se devolverán a la capa de negocios, la cual los

gestionará o transmitirá a la capa de presentación.



Esquema lógico de las capas en una aplicación distribuida

Es importante notar que el esquema que se muestra **es un esquema lógico, no físico**. El modo de distribuir físicamente las capas (en diferentes ejecutables o DLL, o en diferentes equipos) se corresponderá con el esquema lógico en todo o en parte, pero no necesariamente existirá una correspondencia exacta entre la distribución lógica de los elementos y su distribución física. La capa de negocios podría residir en diferentes máquinas, por ejemplo, o las entidades de negocio y la capa de servidor podrían formar parte de la misma DLL.

Adicionalmente, en las aplicaciones distribuidas existen otras funcionalidades, como la seguridad o el registro de sucesos, que no son exclusivas de un grupo lógico ya que deben aparecer en todos los elementos de la aplicación.

3.2.1. La Capa de Servidor

Servicios

En la capa de servidor se encontrarán los procesos de la aplicación que se encargan de recibir las peticiones de las capas superiores y, si es necesario, devolver los datos solicitados. Esta función será desempeñada por un **servicio**.

Los servicios son procesos que se ejecutan en los equipos servidores y que se mantienen a la escucha esperando que los procesos cliente les soliciten funcionalidad o datos. Por lo general los servicios residen en equipos dedicados cuya configuración y características físicas están especialmente diseñados para realizar esta función.

Para poder realizar estas funciones los servicios han de poseer ciertas características que les diferencian claramente de una aplicación de escritorio:

Ejecución desatendida: Un servicio normalmente se mantiene ejecutándose en la memoria del equipo que lo aloja. A pesar de que algunas tecnologías, como DCOM o Remoting, permiten que el objeto que responderá a la petición no esté cargado en memoria hasta que alguien realiza la petición, en la infraestructura del servicio siempre ha de haber un elemento que se mantenga en ejecución, listo para responder a las peticiones de los clientes. La ejecución del servicio, por lo tanto, no precisa de la intervención de ningún usuario.

Conectividad: Un servicio suele estar en un equipo dedicado, mientras que los equipos cliente acceden a él a través de la red. Esto implica que debe existir algún mecanismo que permita que una petición remota active la respuesta del servicio. La infraestructura de red del servicio debe estar configurada para enlazarse con algún protocolo de red, por lo general TCP/IP.

Concurrencia: Dado que múltiples equipos accederán al servicio simultáneamente, es necesario que los servicios implementen algún mecanismo para gestionar la concurrencia de acceso. Existen dos formas básicas de gestionar la concurrencia:

a) Acceso simultáneo: En el acceso simultáneo los servicios crean hilos de proceso paralelos que gestionan las peticiones simultáneamente. Cada hilo de proceso posee sus propias estructuras de datos independientes entre sí, de modo que los datos generados por una petición sean inaccesibles a otra petición. En este caso es posible encontrarse con un problema si las peticiones simultáneas son muy abundantes, ya que eso creará una multitud de hilos de ejecución simultáneos que tal vez sature la capacidad de ejecución del servidor. Sin embargo, si la capacidad de ejecución del servidor es suficiente, los tiempos de respuesta son mejores.

b) Acceso serializado: En este caso las peticiones se atienden una a una, en un único hilo de ejecución. El servicio dispondrá de algún sistema para almacenar en “colas” las peticiones que vayan llegando, y las ejecutará una tras otra. En este caso es difícil que se produzca una saturación de los recursos del servidor, pero los tiempos de respuesta son peores si las peticiones simultáneas son muy abundantes.

Seguridad: El servicio deberá asegurarse de que la llamada está autorizada

y será necesario implementar algún mecanismo que recoja las credenciales del usuario y autorice la ejecución.

3.2.2. Servicios de Base de Datos

Los servicios de base de datos son los más frecuentes en las aplicaciones distribuidas. Los SGBD como SQL Server u Oracle disponen de toda la infraestructura de servicios necesarios para que los equipos cliente les realicen peticiones y para responder a ellas. Se ejecutan como un servicio de forma totalmente desatendida, se enlazan al protocolo de red para escuchar peticiones de otros equipos, gestionan la concurrencia de las llamadas e incorporan mecanismos de seguridad propios o integrables con el directorio activo.

Una de las características más importantes de los SGBD es que permiten **crear reglas de negocio**. Estas reglas pueden invocarse remotamente desde los clientes y se escriben en lenguajes propios del SGBD (Transact-SQL en el caso de SQL Server, por ejemplo). Los SGBD compilan y ejecutan de la forma más óptima posible estas reglas, de modo que su ejecución **siempre es de alto rendimiento**. Podríamos dividir estas reglas en tres tipos según su modo de ejecución:

Procedimientos almacenados: Los procedimientos almacenados se ejecutan como consecuencia de una llamada directa de un cliente. El lenguaje SQL posee instrucciones que permiten invocar la ejecución de procedimientos almacenados en el servidor.

Disparadores: Los disparadores son procedimientos que se ejecutan como consecuencia indirecta de una sentencia SQL efectuada por el cliente. Por lo

general están asociados a sentencias de tipo INSERT, UPDATE o DELETE y disponen de medios para acceder a los registros a los cuales afecta la sentencia.

Procedimientos programados: Es frecuente que los SGBD dispongan de algún sistema para programar el lanzamiento de procesos según un plan programado por calendario, de forma única o periódica. En este caso la intervención del cliente no es necesaria y el propio SGBD activa el proceso cuando llega el momento, de forma totalmente desatendida.

Sea cual sea su modo de ejecución, **las reglas de negocio situadas en el servidor SGBD son muy recomendables en cualquier aplicación distribuida.** No solamente su ejecución es muy rápida, sino que además está centralizada y cualquier cliente que acceda al servicio podrá beneficiarse de ellas. El coste de mantenimiento es mínimo y su modificación interna (es decir, si sólo se modifica la lógica del proceso y no sus parámetros de entrada o salida) no requerirá modificaciones en las aplicaciones cliente. Además su ejecución es transaccional y permite el bloqueo exclusivo, lo cual facilita la atomicidad y la gestión de la concurrencia en accesos simultáneos.

Un claro ejemplo de proceso que sería conveniente incluir entre las reglas de negocio de una aplicación sería la modificación del inventario de un almacén. El cliente realizaría una petición para la creación de una factura a un procedimiento almacenado, el cual descontaría del inventario los artículos que componen la factura. El proceso sería transaccional, de modo que si no fuera posible realizar la actualización del inventario la factura no se crearía y el usuario recibiría un error. Otro caso sería la auditoría de acceso a las tablas. Es posible crear disparadores asociados a los procesos de modificación de modo que cada vez que un usuario modifica un registro la operación quede

registrada en una tabla de auditoría.

Los inconvenientes que plantea la colocación de reglas de negocio en el servidor tienen relación con el tipo de aplicación que se desea construir. Por ejemplo, en una aplicación comercial es posible que no deseemos que se puedan modificar esas reglas, de modo que situar la lógica en procedimientos almacenados (accesibles, por ejemplo, al administrador del sistema), sea exponer una vulnerabilidad a la coherencia de la aplicación. También puede suceder que no se desee exponer las reglas de negocio de la aplicación a la curiosidad de terceros, de modo que situarlas en procedimientos del servidor no es recomendable. Desde luego, es posible restringir el acceso al código fuente de estas reglas de negocio, pero eso implica una complicación adicional en la seguridad y en el mantenimiento que tal vez en una aplicación comercial sea contraproducente.

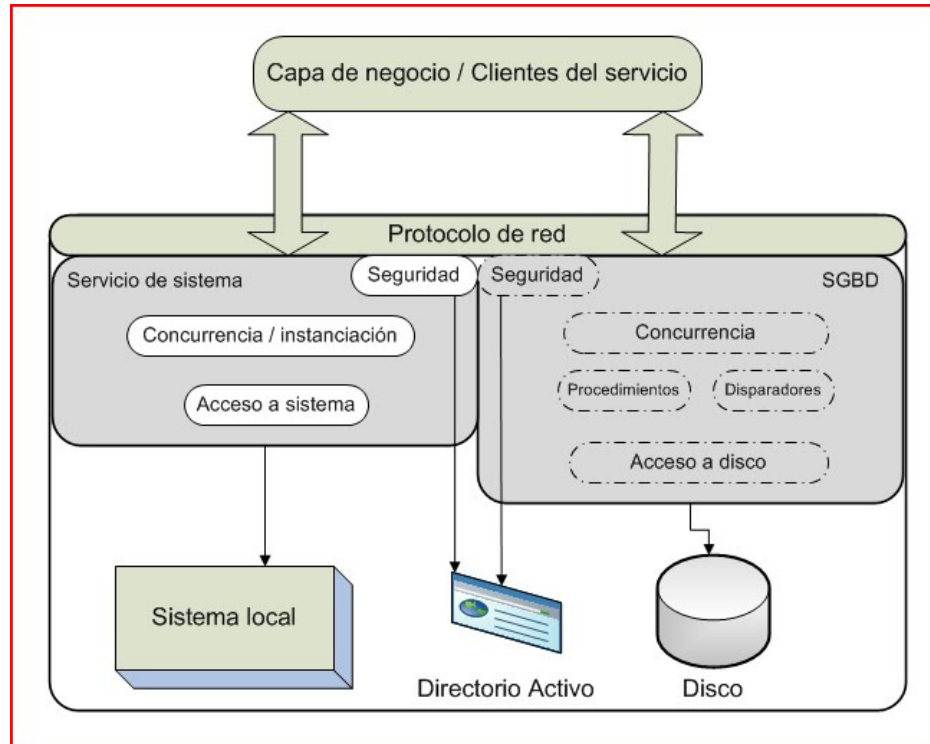
Otro inconveniente, aunque este cada vez es menos relevante, hace referencia a la dificultad de programar estas reglas de negocio en los lenguajes propios de los SGBD. Por lo general los lenguajes propios de los SGBD no son tan potentes ni tan flexibles como los lenguajes de programación “externos”. Sin embargo, SGBDs como SQL Server ya permiten escribir procedimientos almacenados en lenguajes convencionales, de modo que este inconveniente cada vez tiene menos peso.

En general es recomendable el uso de reglas de negocio en el SGBD cuando la aplicación sea corporativa o cuando su administración pueda centralizarse. La eficiencia en la ejecución, la facilidad en el mantenimiento y la encapsulación respecto al resto de la aplicación convierten a los SGBD en una muy buena opción a la hora de situar reglas de negocio de la aplicación.

3.2.3. Otros Servicios

Aunque los servicios de datos son los más frecuentemente utilizados en las aplicaciones distribuidas, en ocasiones requerimos otro tipo de servicios que un SGBD no puede suministrar. Por ejemplo, podría ser necesario extraer información de sistema de un equipo o acceder remotamente a un periférico local.

Los sistemas actuales son cada vez más seguros, de modo que el acceso directo a través de la red a los servicios locales no sólo no es recomendable, sino que en según qué casos puede incluso llegar a ser tan complicado que represente un problema. En este caso el planteamiento requiere la creación de un servicio que permita el acceso remoto a los recursos locales. Estos servicios deben cumplir con las mismas reglas que los servicios que acceden a datos, pero esta vez es probable que no dispongamos de la infraestructura que los SGBD ya incorporan, así que será necesario implementar esas características en nuestro código. Por lo que se refiere a las reglas de negocio, en este tipo de servicios no suelen ser abundantes. Los servicios de sistemas por lo general ofrecen una **fachada** exterior a ciertas características del sistema, y su código se limita a encapsular el acceso a las mismas. Si es necesario establecer alguna regla de negocio el lugar adecuado para situarla será la capa de negocios, de la que trataremos en la siguiente sección.



Esquema de la capa de servidor

Mientras que en el SGBD los servicios de infraestructura están incluidos, en los servicios propios se deberá implementar los módulos de infraestructura.

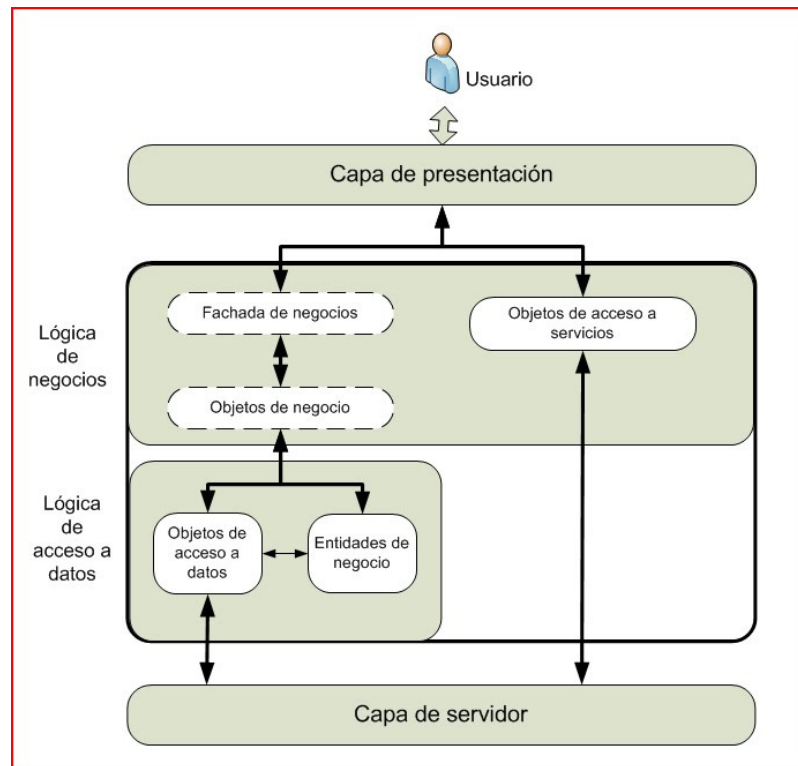
3.2.4. La capa de Negocios

División de la capa de negocios

La capa de negocios representa el grueso de la lógica de funcionamiento de la aplicación distribuida. En esta capa se sitúan las normas de acceso a datos, la lógica de tratamiento de los mismos, y en general cualquier elemento de la aplicación que pueda reutilizarse. El objetivo de la creación de esta

capa “intermedia” es aislar la capa de presentación de la capa de servidor, de forma que las estructuras de datos subyacentes y la lógica que las utilizan sean independientes de la capa de presentación. De esta forma, también, el mantenimiento de las normas de negocio será más sencillo y, sobre todo, será **reutilizable** desde cualquier capa de presentación, sea del tipo que sea.

A pesar de que se suele utilizar el nombre de capa de negocios para referenciar a todos los elementos que componen esta capa intermedia de software, por lo general la capa de negocios suele dividirse en dos tipos de elementos, atendiendo a la función que desempeñan en la capa.



Estructura típica de una capa de negocios.

Los elementos opcionales aparecen con líneas discontinuas.

Lógica de acceso a datos

La lógica de acceso a datos incluye los elementos necesarios para que la aplicación se conecte a orígenes de datos y recupere estructuras de datos que serán utilizadas por el resto de la aplicación. En una aplicación distribuida, **los únicos elementos que se conectan a la base de datos son los objetos de acceso a datos**, y el resto de elementos de la aplicación se limitan a enlazar con estos objetos para solicitar datos y enviar órdenes a los orígenes de datos.

Los motivos para encapsular todo el acceso a datos en la lógica de acceso a datos son múltiples. En primer lugar, **no será necesario distribuir la información de conexión por todo el sistema**, ya que el único punto desde el que se efectuará el acceso directo a los orígenes de datos será el equipo en el que resida físicamente la lógica de acceso a datos. Tampoco será necesario distribuir el software cliente del SGBD por diferentes máquinas, lo que facilita el mantenimiento y la instalación de la aplicación.

Además, encapsular la lógica de acceso a datos permite que la aplicación sea **agnóstica respecto al origen de datos**, es decir, puede realizar sus tareas sin tener la necesidad de saber en qué SGBD concreto residen los datos, ni en qué punto de la red se halla el servidor, lo que facilita la configuración del sistema. Este sistema posibilita la utilización de varios SGBD en una aplicación o facilita la migración de un SGBD a otro. También permite que la aplicación **ignore la estructura real de los orígenes de datos**, ya que es la propia lógica de acceso a datos la que expondrá las estructuras con las que trabajará la aplicación, acomodándolas a las necesidades de la misma.

Otro factor importante es la **reutilización**. La separación de esta lógica

permite reutilizar los componentes de acceso a datos en diversas aplicaciones sin necesidad de copiar el código y manteniendo la coherencia en el comportamiento del acceso a datos en todas ellas.

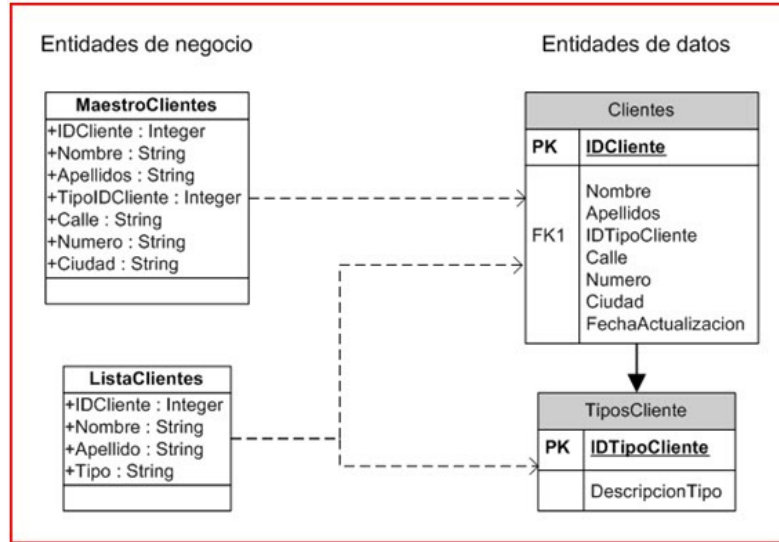
A pesar de que otros elementos, como los que componen la lógica de negocios, son opcionales, los elementos de lógica de acceso a datos deben estar presentes en toda arquitectura distribuida que se diseñe, debido a las ventajas que aportan.

A continuación desglosaremos la lógica de acceso a datos en sus dos componentes principales: **las entidades de negocio** y los **objetos de acceso a datos**.

Entidades de negocio

Las entidades de negocio son estructuras de datos que la aplicación maneja y que representan a las entidades de datos definidas en los orígenes de datos. Una entidad de negocio tendrá elementos que se correspondan, en todo o en parte, con los elementos de la entidad de datos a la que representan.

Por lo general las entidades de negocio **no poseen métodos sino propiedades**, ya que su finalidad es la de describir la entidad de negocio a la que representan. Podemos encontrar dos tipos de entidad de negocio según la utilización que se haga de ellas en la aplicación.



Entidades de mantenimiento: Las entidades de mantenimiento se utilizan para leer, insertar, actualizar o eliminar registros del origen de datos. Cada una de sus propiedades reflejará un campo de la tabla subyacente, a menos que la lógica de negocio imponga que dicho campo no ha de ser manejable o visible desde las aplicaciones. Por ejemplo, una tabla “Clientes” del origen de datos podría tener su réplica en la entidad de negocios Maestro-Clientes, que tendrá una propiedad por cada campo de la tabla. No obstante, en la tabla puede existir un campo FechaActualizacion que se maneje internamente desde el propio SGBD y que no sea visible desde la aplicación, en cuyo caso la entidad de negocio no debe exponerlo.

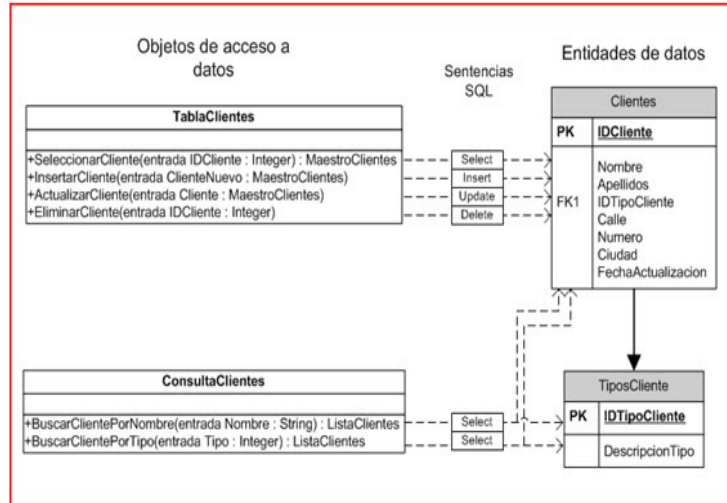
Entidades de lista: Las entidades de lista se utilizan para recuperar estructuras de datos obtenidas como consecuencia de una consulta. Por lo general no necesitarán todos los datos de la entidad subyacente, y también es frecuente que incluyan campos pertenecientes a entidades secundarias. Ambos

casos suponen que estas entidades no son adecuadas para realizar operaciones de inserción o actualización, de modo que se utilizan como entidades de sólo lectura. Por ejemplo, una entidad ListaClientes puede incluir algunos campos de la tabla “Clientes” y algún otro campo de una entidad secundaria, como quizás la descripción del tipo de cliente.

Objetos de acceso a datos

Los objetos de acceso a datos son los intermediarios entre la aplicación y los orígenes de datos. Estos objetos **y ninguno más en la aplicación** son los encargados de conectarse con los orígenes de datos y enviarles sentencias SQL, órdenes de ejecución de procesos o cualquier otra operación que implique acceso a los datos de la aplicación. Cualquier configuración de acceso a los orígenes de datos o cualquier cambio en la forma de acceder a los mismos afectará a estos objetos y a ningún elemento más de la aplicación.

Los objetos de acceso a datos utilizarán en su comunicación con los orígenes de datos la forma nativa de comunicación que dichos orígenes de datos exijan; por ejemplo, utilizarán sentencias SQL si el origen de datos es un SGBD o acceso a disco si el origen de datos es un fichero de texto. Sin embargo, los objetos de acceso a datos exponen estas operaciones al resto de elementos de la aplicación mediante **métodos de clases**. El código de los objetos de datos será el encargado de traducir a instrucciones nativas del origen de datos las llamadas que realicen el resto de elementos de la aplicación. Los métodos expuestos por los objetos de datos han de incluir los argumentos que precisen las llamadas a los orígenes de datos.



Objetos de acceso a datos y su relación con las entidades de datos.

En este caso el objeto **ConsultaClientes** incorpora un método de lista por cada tipo de consulta disponible. También se asume que los argumentos y los retornos de los métodos utilizan las entidades de negocio directamente, aunque no siempre ha de ser así.

Un objeto de acceso a datos encapsula **el acceso a una sola entidad de datos del origen**, ya sea una tabla o un fichero individual. Si la lógica de la aplicación precisa que varias entidades de datos sean tratadas como una única entidad, debe implementarse un objeto de complejidad mayor en la lógica de negocios que será el encargado de encapsular la lógica de los objetos de acceso a datos subyacentes.

Un objeto de datos puede exponer tres tipos de métodos de acceso a datos:

Métodos CRUD (Create, Read, Update, Delete): Los métodos CRUD se corresponden con las instrucciones necesarias para realizar un mantenimiento

de una tabla, es decir, seleccionar, insertar, eliminar o actualizar un único registro o un conjunto de registros pertenecientes a la misma entidad de datos del origen. Para ello utilizan las entidades de negocio de tipo mantenimiento, independientemente de si esas entidades son utilizadas o no por las capas superiores de la aplicación. Por ejemplo, la tabla “Clientes” dispondrá de un objeto de acceso a datos de tipo `TablaClientes` que expondrá métodos como `SeleccionarCliente`, `InsertarCliente`, `ActualizarCliente` y `EliminarCliente`. Estos métodos poseerán argumentos que se utilizarán en las sentencias SQL que el objeto de acceso a datos lanzará contra el SGBD en el que resida la tabla `Clientes`.

Métodos de lista: Un objeto de acceso a datos puede recuperar datos de sólo lectura para realizar consultas sobre la entidad de datos. Normalmente un objeto de acceso a datos que utilice métodos de lista recuperará entidades de negocio de tipo lista, independientemente de si la arquitectura permite o no que los objetos de capas superiores utilicen también esas entidades-lista. Así, en el ejemplo anterior, se dispondría de un objeto de tipo `ConsultaClientes` que expondrá métodos como `BuscarClientes`. Dependiendo del enfoque que se le de al objeto, puede existir un solo método que permita cualquier consulta o un método por cada tipo de consulta, pero es frecuente que las estructuras que recuperan las consultas sean siempre las mismas, independientemente de los parámetros de la misma.

Métodos de procedimiento: Se llama métodos de procedimiento a las llamadas directas a procedimientos almacenados en el servidor que no devuelven estructuras de datos ni están asociados a una entidad de datos concreta. Por ejemplo, un procedimiento podría cambiar automáticamente los precios de un determinado conjunto de productos, los que pertenecen a una determinada

categoría o fabricante. Esta operación implicaría quizás cambios en más de una tabla, y se trataría de una regla de negocio no asociada a una entidad concreta de datos. Los objetos de este tipo no utilizan ninguna estructura de datos, de modo que no se relacionan con ninguna entidad de negocio.

Cuando la lógica de la aplicación implica que varias entidades de datos han de manejarse conjuntamente es necesario crear objetos de complejidad mayor. Los objetos de datos han de ser simples y directos en su funcionamiento y han de afectar a una sola entidad de datos. Asimismo, han de manejar entidades de negocio también simples.

Lógica de negocios

Cuando las aplicaciones adquieren cierto volumen o las entidades implicadas tienen cierta complejidad, la lógica de acceso a datos por sí sola no es suficiente para encapsular convenientemente el acceso a las entidades de datos. En estos casos será necesario añadir objetos más complejos que a su vez encapsulen los objetos de acceso a datos y los expongan de forma más sencilla a las capas superiores, facilitando su manejo.

Además, en las aplicaciones distribuidas con cierto tamaño es frecuente encontrar reglas de negocio que no tienen nada que ver con el acceso a datos, sino que constituyen mecanismos aparte que de una forma o de otra es deseable extraer de la capa de presentación para su reutilización o para que su mantenimiento sea sencillo.

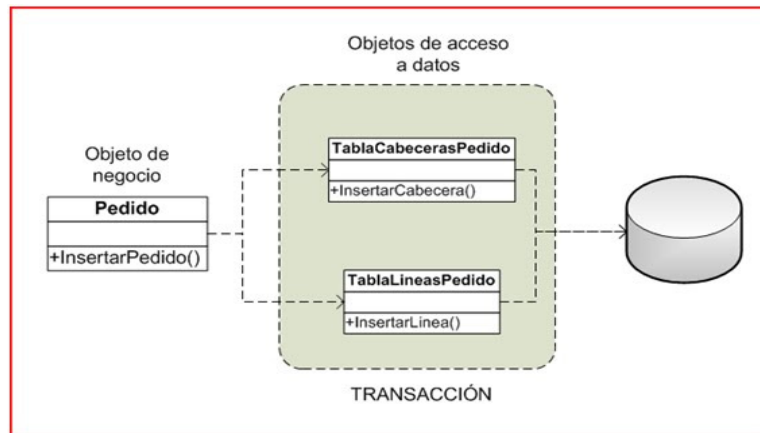
Estas necesidades implican a menudo la creación de una capa adicional de lógica que se llama **lógica de negocios**. Los elementos de la lógica de negocios ya no se conectan a los orígenes de datos ni representan a las entidades de

datos subyacentes, sino que **utilizan los objetos de acceso a datos y las entidades de negocio**, siendo pues una especie de “cliente” de la lógica de acceso a datos.

A continuación se detallarán los objetos más frecuentes que podemos encontrar entre la lógica de negocios.

1) Objetos de negocio

Los objetos de negocio son una abstracción de un conjunto de entidades de datos relacionadas entre sí. El objetivo de estos objetos es encapsular el acceso a varios objetos de acceso a datos en un único objeto, de modo que las capas superiores no hayan de realizar accesos a múltiples objetos para realizar una operación.



Relación de un objeto de negocio con sus objetos de acceso a datos.

Se han omitido los parámetros de las llamadas.

Otro motivo para la encapsulación en objetos de negocio hace referencia al **comportamiento transaccional**. Si una operación requiere realizar actualizaciones en varias entidades de negocio, cada una de ellas “gobernada”

por un único objeto de datos, dicha operación ha de ser transaccional. Para que la transacción sea transparente a los objetos de capas superiores conviene realizar dicha transacción desde un único punto centralizado, no solamente por coherencia lógica sino por facilidad en la codificación de la transacción.

Los objetos de negocio **no son obligatorios, pero sí recomendables**. Una aplicación sencilla quizás no requiera objetos de negocio, e incluso sería contraproducente implementarlos si el número de entidades fuera pequeño o la complejidad de la lógica no lo requiere, ya que el esfuerzo en implementar los objetos de negocio probablemente no compensará las ventajas.

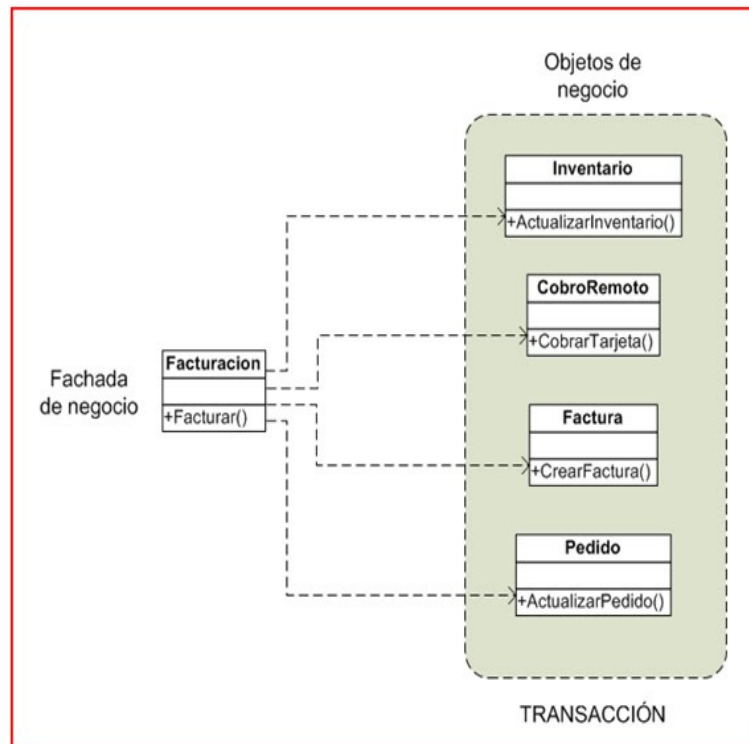
Sin embargo, en una aplicación de cierta complejidad o cuyos requisitos pueden variar a lo largo del tiempo es recomendable crear objetos de negocio que encapsulen los objetos de acceso a datos subyacentes, **aun cuando un objeto de negocio sólo represente a un único objeto de acceso a datos**. La creación de objetos de negocio permitirá abordar con más flexibilidad un nuevo requisito de negocios o una nueva relación entre objetos de acceso a datos que no apareció en el análisis inicial.

2) Fachadas de negocio

Las fachadas de negocio sólo aparecen en grandes aplicaciones con muchos objetos de acceso a datos y muchos objetos de negocio. En este tipo de aplicaciones será necesario encapsular algunos conjuntos de objetos de negocio en un objeto más complejo que encapsule ciertas operaciones a fin de que la programación de la capa de presentación sea más fácil y más coherente.

Asimismo, un buen motivo para crear fachadas de negocio es la compartimentación de los objetos de negocio en unidades funcionales con permisos

específicos de uso. Por ejemplo, en un gran desarrollo corporativo se puede desear que los objetos de negocio relacionados con las nóminas permitan tanto la consulta como la actualización. Sin embargo, pueden existir dos aplicaciones en la compañía que utilicen dichos objetos, y mientras que una de ellas permite a sus usuarios la actualización, la otra sólo permite la consulta. En este escenario sería prudente encapsular los objetos de negocio en dos fachadas de negocio. En la primera de ellas se dispondría de métodos de consulta y actualización de nóminas, mientras que en la segunda fachada de negocio sólo estarían definidas las consultas. De esta forma será imposible que la aplicación no autorizada pueda siquiera accidentalmente implementar un método de actualización de nóminas, ya que las aplicaciones no acceden directamente a los objetos de negocio sino a la fachada que se ha creado específicamente para la aplicación y que no posee métodos para actualizar nóminas.



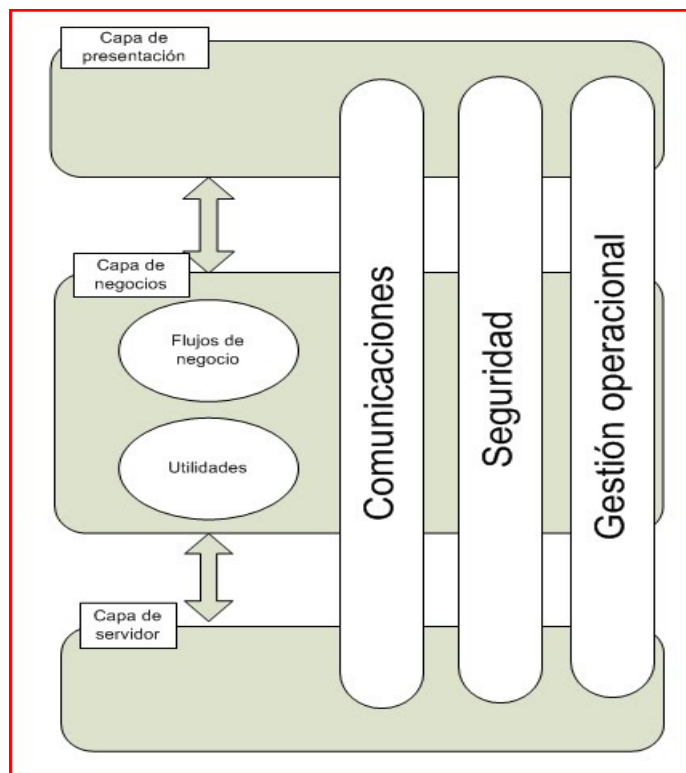
Esquema de una fachada de negocio.

La aplicación de facturación no tiene acceso directo a los objetos de negocio pero la operación “Facturar” incluye los pasos necesarios de forma transparente a la aplicación. Una operación añadida (como podría ser una auditoría) no afectaría a la capa de presentación y sólo requeriría cambiar el comportamiento de la fachada.

Las fachadas de negocio **también han de observar un comportamiento transaccional**, de modo que los objetos de negocio han de poder incluir sus propias transacciones dentro de una transacción más amplia que pueda estar generada por una fachada de negocios.

3) Otros objetos de negocio

En la capa de negocios se hallan presentes con frecuencia otros objetos cuya finalidad no es el acceso a datos, sino cubrir otras funcionalidades de la aplicación que de esta forma quedan encapsuladas para que sea sencillo mantenerlas o reutilizarlas. La funcionalidad de estos objetos puede ser muy diversa; en esta sección se comentarán los casos más frecuentes.



Otros objetos integrantes de la capa de negocio

Uno de los casos son los **flujos de negocio**. Los flujos de negocio son reglas que describen los diversos estados por los que ha de atravesar una determinada entidad de negocio. Por ejemplo, un pedido cuantioso podría requerir la confirmación de un jefe de compras, la reserva de espacio en el almacén, etc. Durante el proceso, el pedido atravesará varios estados y entre un cambio de

estado y otro podría pasar cierto tiempo, quizás incluso días.

A pesar de que la definición de los diferentes pasos de un flujo de negocio puede estar alojada en una base de datos, el propio mecanismo del flujo suele ser parte de la lógica de la aplicación. Esto hace que la dinámica de los flujos de negocio o bien sea rígida, sí en la base de datos sólo se encuentra la definición de los estados, o bien nos obligue a compilar la aplicación cuando cambiamos algún flujo de información.

También es frecuente en las aplicaciones distribuidas el empleo de **objetos de utilidades**. En estos objetos suelen incorporarse determinadas funciones que no requieren el uso de instancias de clase y que proporcionan funcionalidad reutilizable por distintos objetos. Por ejemplo, una función que reciba una variable de cadena como argumento y extraiga de ella un determinado tipo de información, como un código postal, podría ser una función que se utilice desde varios objetos de negocio. Por lo general este tipo de funciones no requieren que las clases en las que residen mantengan el estado, y además son susceptibles de ser llamadas desde cualquier instancia de objeto, de modo que lo recomendable es disponerlas en métodos estáticos.

Por último podemos mencionar lo que llamaremos **servicios transversales**. Los servicios transversales proporcionan funcionalidades necesarias en todas las capas de la aplicación y su implementación no se limita a una sola capa sino que su ámbito de actuación se despliega en todas las capas de la solución. En realidad no forman parte de la capa de negocios sino que se hallan presentes de una u otra forma en todas las capas de la aplicación.

Un ejemplo de un servicio transversal es el de la **seguridad**. La implementación de la seguridad por lo general afectará de forma transversal a toda

la aplicación, ya que será necesario mantener contextos de seguridad en todas las capas, y también será necesario que el tratamiento de la seguridad sea coherente entre ellas o por lo menos existan mecanismos de adaptación si los sistemas de seguridad cambian en las fronteras físicas entre capas. La implementación de un sistema de seguridad integral para la solución deberá tener en cuenta todas las capas implicadas y probablemente estará ligada en mayor o menor medida al despliegue físico de las capas lógicas.

Otro servicio transversal que erróneamente se deja a veces en un segundo término es el de las **infraestructuras de comunicación**. Los objetos de este tipo se encargan de establecer las comunicaciones entre las diferentes capas físicas en las que residen las capas lógicas de la aplicación. A pesar de que en una situación ideal la comunicación entre distintos sistemas físicos no debería influir en el modo de diseñar la lógica de una aplicación, en la mayoría de las ocasiones las necesidades de despliegue físico de las capas condicionan el despliegue lógico de las mismas, y por ende su diseño.

Por último cabe mencionar entre los objetos frecuentes en las aplicaciones de cierto volumen a los objetos de gestión operacional. Su uso es variado y depende en gran medida de las necesidades de gestión de la propia aplicación. Por ejemplo, es frecuente encontrar en las aplicaciones algunos objetos encargados de gestionar el registro de sucesos o la auditoría.

Existen otros servicios transversales como la gestión de clústeres o el balanceo de carga pero en la mayor parte de las ocasiones estos servicios están proporcionados por el propio sistema operativo y no suele ser necesario programarlos, de modo que no los incluiremos en nuestra descripción de la arquitectura.

3.2.5. La capa de Presentación

La capa de presentación la constituye el software con el que el usuario interactúa para operar con la aplicación. Es probablemente la parte más trabajosa de la misma, ya que es muy frecuente que aplicaciones cuyas reglas de negocio sean relativamente sencillas tengan en cambio un interfaz de usuario complejo y vistoso que le proporcione al usuario una experiencia de manejo fácil y agradable. Además, mientras que en la creación de reglas de negocio normalmente sólo interviene un tipo de programación, preferentemente basada en lenguajes, en la preparación del interfaz de usuario suelen mezclarse varias disciplinas, como el diseño o la usabilidad.

Una error frecuente en la creación de los interfaces de usuario consiste en olvidar que **las reglas de negocio no se hallan en el interfaz**, sino en los objetos subyacentes que residen en las capas inferiores de la solución. La capa de presentación no es más que un sistema de presentación y manejo de datos que se obtienen y se actualizan con los objetos de negocio comunes para todas las aplicaciones que los usan. Si se olvida este aspecto se puede caer en la tentación de colocar reglas de negocio en el interfaz de usuario, imposibilitando la reutilización de las mismas y complicando la distribución y despliegue de la aplicación. Por lo tanto, una regla de oro a observar en toda aplicación distribuida es que la capa de presentación ha de ser **completamente independiente de las reglas de negocio**, y su función se limitará a la presentación y manejo de los datos de la aplicación, que obtendrá mediante el uso de los objetos de la capa de negocios comentados en la sección anterior.

Esto convierte a la capa de presentación en una mera fachada de los procesos que son gestionados por la capa de negocios. Las capas de presentación

suelen ser “delgadas”, es decir, contienen pocas líneas de código, ya que su función principal está cubierta por las características de los elementos “visuales” que las componen. Una tendencia creciente es la **separación entre diseño y código**, ya existente, por ejemplo, en las aplicaciones web dinámicas.

Distribución física de las capas lógicas

Una vez descartada la capa de presentación como ubicación posible para la lógica de negocio, al arquitecto que diseña una aplicación distribuida se le presenta el dilema de la distribución física de cada una de las capas lógicas. Por un lado existe la posibilidad de colocar las reglas de negocio enteramente en el servidor, mediante procedimientos almacenados, disparadores y otras funciones propias del SGBD. Por el otro, la colocación de las reglas de negocio en objetos creados mediante lenguajes independientes de la base de datos también supone ciertos problemas, como la dependencia que naturalmente se creará entre los diversos módulos físicos en los que residan las clases.

A pesar de que no existe una solución única para este problema, en esta sección propondremos dos escenarios posibles muy distintos entre sí a modo de ejemplo y estudiaremos los factores que nos harán decidirnos por una solución u otra. Probablemente una solución deberá incluir elementos de ambos escenarios, así que deberá ser el arquitecto el que juzgue en cada caso la manera óptima de implementar la solución.

Escenario 1: Aplicación corporativa

En una aplicación corporativa de gran tamaño los datos residen en un SGBD de cuyo mantenimiento se encarga un administrador de bases de datos. Se dispone de un gran número de tablas con muchos registros y la seguridad

es un factor importante, ya que no todos los departamentos de la empresa pueden acceder a todos los datos.

Por otro lado, en la corporación existen diferentes aplicaciones que manejan los datos, y hay diferentes equipos de desarrollo que acceden a los mismos. Permitir que cada uno de los equipos de desarrollo establezca sus propias reglas de negocio de la empresa no es asumible, así que la gestión de los datos necesariamente ha de centralizarse.

En este escenario la mayor parte de las reglas de negocio que se ocupan del tratamiento de los datos residirán en **procedimientos almacenados o en reglas del propio SGBD**. Esto permitirá restringir el acceso a los datos a determinados departamentos, así como unificar el tratamiento de los mismos. Además, es muy probable que se requiera acceso de alto rendimiento a los datos desde delegaciones o sedes remotas, lo que conduce a la utilización de métodos óptimos para la consulta de datos.

El mantenimiento de las reglas de negocio será tarea de los administradores de la base de datos. Los desarrolladores deberán exponer sus necesidades a los encargados de crear y modificar los procedimientos almacenados, de forma que el acceso directo a las tablas quedará restringido a los que poseen los permisos necesarios para ello. Las aplicaciones sólo tendrán permisos para acceder a los procedimientos almacenados que utilicen.

Los objetos de la capa de negocio serán “delgados”, ya que poseerán pocas reglas de negocio. Sin embargo es más que probable que necesitemos utilizar **fachadas de negocio** situadas en diferentes módulos, ya que existirán diversas capas de presentación (aplicaciones Windows, aplicaciones Web, acceso mediante PDA, etc.) y la reutilización de las funcionalidades será un factor

fundamental.

Escenario 2: Aplicación comercial

Una empresa que desarrolla y vende una aplicación de gestión comercial se encuentra con un escenario completamente distinto al planteado en la sección anterior. Para comenzar, la aplicación se venderá a clientes en cuya infraestructura de servidores la empresa no puede interferir. El proceso de instalación habrá de ser complejo, ya que no sólo deberá instalar la aplicación en las máquinas del cliente sino que además deberá ejecutar los scripts necesarios para la creación de la base de datos necesaria. En muchos casos se tratará incluso de bases de datos locales (por ejemplo, en Access o en SQL Server Express).

En este escenario la colocación de reglas de negocio en el servidor no es aconsejable. En primer lugar, una actualización de las reglas implica la modificación de los procedimientos almacenados, lo que conlleva una dificultad añadida a la distribución de actualizaciones. En segundo lugar, la empresa será naturalmente celosa de sus reglas de negocio, y no deseará comprometerlas situándolas en procedimientos almacenados a los que el usuario final tendrá acceso. Por otro lado, mantener las reglas de negocio en los componentes (DLL) permite una mayor protección de la propiedad intelectual de la aplicación.

Por otro lado, estas aplicaciones tienen un ámbito de ejecución local o, en el más optimista de los casos, restringido a una LAN. La reutilización de módulos no es probable, así que la división en componentes separados físicamente no será esencial, salvo en lo tocante a la optimización de los recursos consumidos por la aplicación en el equipo local. Los componentes serán este caso más

“gruesos” y, a pesar de que la división lógica seguirá existiendo, es muy posible que no exista una fachada de negocios como tal y que el interfaz de usuario realice llamadas directas a los objetos de negocio o incluso a los objetos de acceso a datos.

Factores que influyen en la distribución física

Para finalizar, enumeraremos los factores que un arquitecto de soluciones debe tener en cuenta a la hora de diseñar la distribución física de los diferentes elementos lógicos de la aplicación:

a. Infraestructura de comunicaciones: Las diversas capas de la aplicación deberán comunicarse entre sí, traspasando datos de un lado a otro y estableciendo referencias entre ellas. Si la distribución física de los equipos en los que se ejecutará la aplicación es simple no será necesario distribuir los componentes en varias capas físicas separadas entre sí, lo que proporcionará a la aplicación más velocidad y más facilidad de instalación y configuración. Si la accesibilidad de los equipos es muy diversa la distribución es un factor importante, y ha de intentarse que las capas de presentación sean lo más delgadas posible. En escenarios muy distribuidos lo más recomendable es una arquitectura orientada a servicios (SOA).

b. Dependencias: Cuando dos capas se comunican entre sí es forzoso establecer entre ellas una cierta dependencia. Es necesario prever estas dependencias para minimizar las modificaciones que requiera el cambio en una de las capas. Distribuir los componentes mejora la escalabilidad y la reutilización pero complica la relación de dependencias entre los mismos.

c. Impacto en las modificaciones posteriores: Es un axioma que cualquier

aplicación requiere mantenimiento. El arquitecto ha de contemplar el impacto de una actualización en la aplicación. Si no es factible mantener los procedimientos almacenados o si se complica la distribución de una actualización quizás no sea aconsejable su uso.

d. Reutilización: En muchas ocasiones será necesario reutilizar reglas de negocio en varias aplicaciones. En estos casos subdividir la aplicación en diversos componentes físicos que cumplan funciones muy específicas será muy útil y proporcionará mucha flexibilidad a la hora de crear nuevas aplicaciones que reutilicen las mismas reglas de negocio.

e. Rendimiento: La colocación de las reglas en el servidor o en componentes compilados puede influir en el rendimiento general de la aplicación si las reglas utilizan grandes conjuntos de registros o cálculos complicados. Utilizar métodos de alto rendimiento puede representar un factor importante en la escalabilidad de la aplicación.

f. Dificultad de implantación: A pesar de las facilidades que ofrecen entornos avanzados como Visual Studio, una aplicación muy distribuida implica una implantación más compleja. Será necesario poner en marcha servicios, establecer referencias remotas y quizás atravesar fronteras de seguridad como dominios o firewalls. Si el tiempo es escaso o el equipo no posee los conocimientos suficientes la distribución de componentes puede ser un factor de complejidad añadida.

Capítulo 4

DB2 Universal Database

La necesidad de mejorar la manera de acceder y manejar los datos ha evolucionado con el transcurso del tiempo hasta llegar a la generación de los sistemas de administración de bases de datos relacionales (RDBMS).

En los últimos tiempos ha surgido una nueva base de datos llamada “Universal”, la cuál es capaz de almacenar y hacer búsquedas no solamente de datos alfanuméricos sino también de imágenes, audio, video y otros objetos.

Esta ventaja de las bases de datos universales abre un gran número de oportunidades que permiten mejorar tanto los servicios como las aplicaciones.

Se puede definir una Base de Datos como una serie de datos organizados y relacionados entre sí, y un conjunto de programas que permitan a los usuarios acceder y modificar esos datos.

Mientras que un archivo normalmente contiene datos acerca de un tipo de entidad (ej.: personal, órdenes, clientes, ventas), una base de datos contiene

datos acerca de muchos tipos de entidades e información acerca de cómo las entidades están lógicamente relacionadas entre sí.

DB2 UDB Universal Database es una Base de Datos Universal.

Es completamente escalable, veloz y confiable.

Corre en modo nativo en casi todas las plataformas como ser: Windows Vista, NT, Sun Solaris, HP-UX, AIX U, OS/2 entre otros.

DB2 es un software de base de datos relacional.

Es completamente multimedia, disponible para su uso en la Web, muy bueno para satisfacer las demandas de las grandes corporaciones y bastante flexible para servir a los medianos y pequeños negocios.

DB2 UDB es un sistema manejador de base de datos relacional fuertemente escalable.

Es suficientemente flexible para atender estructuras e inestructuras manejadoras de datos necesarias para usuarios simples de grandes empresas.

Es conveniente para una gama amplia de aplicaciones de los clientes, quienes pueden desplegar una variedad de plataformas de hardware y software desde dispositivos manuales a los sistemas multiprocesador paralelos masivos.

4.1. Características Generales del DB2 UDB:

DB2 UDB es el producto principal de la estrategia de Data Management de IBM.

DB2 UDB es un sistema para administración de Bases de Datos Relacionales (RDBMS).

Es multiplataforma: especialmente diseñada para ambientes distribuidos, permitiendo que los usuarios locales compartan información con los recursos centrales.

Es el sistema de gestión de datos que entrega una plataforma de base de datos flexible y rentable para construir un sistema robusto para aplicaciones de gestión.

DB2 UDB libera los recursos con amplio apoyo al open source (fuente abierta) y plataformas de desarrollo populares como J2EE y Microsoft .NET.

Integridad: El DB2 UDB incluye características de Integridad, asegurando la protección de los datos aún en caso de que los sistemas sufran un colapso, y de

Seguridad: permitiendo realizar respaldos en línea con distintos grados de granularidad, sin que esto afecte la disponibilidad de acceso a los datos por parte de los usuarios.

Múltiples Usos: Provee la capacidad de hacer frente a múltiples necesidades, desde Procesamiento Transaccional de Misión Crítica (OLTP), hasta análisis exhaustivo de los datos para el soporte a la toma de decisiones (OLAP).

Escalabilidad: Sus características distintivas de Escalabilidad le permiten almacenar información en un amplio rango de equipos, desde un PC portátil hasta un complejo ambiente de mainframes procesando en paralelo.

Universalidad: DB2 UDB es, además, la única base de datos realmente

universal; es multiplataforma (16 plataformas - de las cuales 10 no son de IBM), brinda soporte a un amplio rango de clientes, soporta el acceso de los datos desde Internet y permite almacenar todo tipo de datos:

- Texto, Audio, Imágenes y Video (AIV Extender).
- Documentos XML (XML Extender).

DB2 cumple con estos requerimientos y otros más:

- Soporta el paradigma de network-computing utilizando Java y JDBC. Permite el acceso a DB2 Universal Data Base desde una amplia variedad de plataformas de clientes, utilizando JDBC vía Java applets desde cualquier Web browser. Adicionalmente, se puede desarrollar una aplicación en Java y acceder a DB2 utilizando directamente JDBC. También permite codificar “stored procederes” en Java.
- Brinda excelente nivel de seguridad. Brinda servicios de autenticación y autorización que pueden ser fácilmente integrados a servicios de redes y sistemas operativos.
- Net.Data brinda acceso a sus datos corporativos desde la Web. Ofreciendo un alto nivel de performance y escalabilidad en el acceso de datos residentes en DB2 Universal Data Base, incorporando multimedia.

Aplicaciones Existentes:

Las aplicaciones de negocios más utilizadas son:

- On -Line Transaction Processing (OLTP).

- Data Warehousing.
- Decision Support.
- Data Mining.

Conectividad: Las herramientas de conectividad permiten acceder a los datos más allá de donde ellos se encuentren. El slogan cualquier cliente, a cualquier servidor, en cualquier red está completamente sustentado por la funcionalidad que sus herramientas ofrecen.

DB2 permite acceder a los datos de DB2 en mainframe o AS/400, desde Windows Vista, NT, Windows 95/98, OS/2 o cualquiera de los Unix soportados.

Además, el producto Datajoiner posibilita acceder de forma única y transparente a los datos residentes en Oracle, Sybase, Informix, Microsoft SQL Server, IMS, VSAM y otros.

Data Warehousing: DB2 UDB provee la infraestructura necesaria para soportar el proceso de toma de decisiones en cualquier tamaño y tipo de organización. Es el producto dirigido a resolver la problemática a nivel departamental (Data Marts), ya que un único producto provee la capacidad para acceder a datos en Oracle, Sybase, Informix, Microsoft SQL Server, VSAM o IMS, además de la familia DB2. Permite de forma totalmente gráfica acceder, transformar y distribuir los datos automáticamente y sin programar una línea de código.

Data Mining: DB2 UDB posibilita el análisis orientado al descubrimiento de información escondida en los datos, realizando modelización predictiva,

segmentación de la base de datos, análisis de vínculos, o detección de desviaciones. Incluye las siguientes técnicas: clustering (segmentación), clasificación, predicción, descubrimiento asociativo, descubrimiento secuencial de patrones y secuencias temporales. Todas las técnicas mencionadas permiten realizar segmentación de clientes, detección de fraudes, retención de clientes, ventas cruzadas, etc.

Permite agilizar el tiempo de respuestas de esta consulta.

Recuperación utilizando accesos de sólo índices.

Predicados correlacionados.

Tablas de resumen.

Tablas replicadas.

Uniones hash.

DB2 utiliza una combinación de seguridad externa y control interno de acceso a proteger datos. Proporciona un juego de datos de acceso de las interfaces para los diferentes tipos de usuarios y aplicaciones. Guarda sus datos contra la pérdida, acceso desautorizado, o entradas inválidas. Se puede realizar la administración de la DB2 desde cualquier puesto de trabajo.

La tecnología de replicación heterogénea (heterogeneous replication) en SQL Server permite la publicación automática de los datos en otros sistemas que no sean SQL Server, entre los que se incluyen DB2.

La mayoría de los que utilizan equipos IBM utilizan DB2 porque es confiable y tiene un muy buen soporte técnico.

El DB2 se basa en dos ejes que lo hacen fuerte en su rendimiento: utiliza un sistema multiprocesador (SMP) simétrico y un sistema de procesador paralelo masivo.

El DB2 distribuye y recuerda la ubicación de cada pista donde se encuentra la información. En el contexto de una larga base de datos, este sistema de partición hace que la administración sea mucho más fácil de manejar que una base de datos de la misma medida no particionada.

La base de datos se puede programar para tener una exacta cantidad de particiones que contienen la información del usuario, índice, clave de transacción y archivos de configuración. De esta forma, los administradores definen grupos de nodos, que son una serie de particiones de la base, lo que posteriormente facilita cualquier búsqueda.

Por Otro Lado:

El DB2 - IBM es la tercera base de datos que más se vende, de acuerdo con los VARs recientemente encuestados en el número de junio de 1996 de la revista VAR Business Magazine. El Microsoft SQL Server se anotó un 38 %, Oracle, 21 %, IBM, 10 %, Informix, 9 %, y Sybase un 8 %.

En sistemas grandes la base más usada es DB2 ya que corre en diferentes plataformas operativas, pero en realidad, en la mayoría de los casos la decisión para optar por un software de estas características es corporativa.

Se tiene que ver con las aplicaciones que se tienen desarrolladas y las que se van a implementar.

Influye en la elección el hardware utilizado.

DB2 Universal Database es un sistema de gestión de base de datos relacionales completamente habilitado para la web que se puede escalar, desde procesadores simples hasta multiprocesadores simétricos y agrupamientos paralelos masivos.

Su uso ha mejorado de forma que los usuarios novatos puedan realizar tareas de administración de base de datos.

Mediante el se puede influir en todos los aspectos relativos a la información de la empresa, más allá de simples filas y columnas de datos alfanuméricos, incluyendo información en formato XML, imágenes, video en modalidad continua y otros formatos ricos en los medios.

Capítulo 5

Conclusiones

En consecuencia, la base de datos multiplataforma, es como una unidad virtual, cuyas partes se almacenan físicamente en varias bases de datos “reales” distintas, ubicadas en diferentes lugares.

Algunos beneficios de las Bases de Datos Multiplataforma son, en primer lugar los costes de comunicación; si las bases de datos están muy dispersas y las aplicaciones hacen amplio uso de los datos puede resultar más económico dividir la aplicación y realizarla localmente. El segundo aspecto es que cuesta menos crear un sistema de pequeños ordenadores con la misma potencia que un único ordenador.

Las aplicaciones distribuidas ofrecen la solución más optimizada para grandes sistemas que requieren alta concurrencia o máxima reutilización de código. Los procesos se ejecutan en máquinas dedicadas que se configuran de la manera más adecuada para ofrecer los servicios que requiere cada parte de la aplicación. Ciertamente, crear una aplicación distribuida en varias capas requiere

cierto sobresfuerzo en términos de diseño y conlleva una cierta pérdida de rendimiento frente a las aplicaciones cliente-servidor tradicionales, pero su implantación soluciona tantos problemas que su uso es imprescindible en sistemas muy complejos.

Gracias a su alcance global y bajo costo, Internet puede ser una solución de negocios muy poderosa. Si se quiere realmente aprovechar las oportunidades de negocios utilizando la Web, se debe considerar la mejor manera de llegar a un mercado que está en espera las 24 horas de los 7 días de la semana.

La base de datos que se seleccione debe estar preparada para brindar acceso rápido y alta disponibilidad a información que requiere actualización constante. DB2 Universal Data Base provee la capacidad de hacer backups en línea, evitando interrupciones en la disponibilidad del sistema. Permite realizar operaciones comerciales garantizando un alto nivel de seguridad y confiabilidad.

DB2 responde a las exigencias del e-business de hoy. Detrás del e-business está siempre una base de datos.

Bibliografía

- [1] Elmasri R.Ñavathe B. *Fundamentos de Sistemas de Bases de Datos*. Addison-Wesley., 2002.
- [2] Date C. J. *Introducción a los Sistemas de Bases de Datos*. Prentice-Hall, 2001.
- [3] Kort H Silberschatz A. *Fundamentos de Base de Datos*. Mc. Graw-Hill, 1993.

Índice alfabético

atributo, 28

conurrencia, 44

DB2, 3, 84

DBA, 14

DBMS, 6, 9, 47

DML, 7

Internet, 2

Java, 86

modelo entidad - relación, 22

multiplataforma, 1

OLAP, 85

OLTP, 85

Oracle, 34

RDBMS, 83

relaciones, 25

seguridad, 45

SGBD, 47

SQL Server, 35

