UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ZUMPANGO

INGENIERÍA EN COMPUTACIÓN

BASE DE DATOS AVANZADAS

APUNTES

POR

MTE-MI ROSA ERENDIRA REYES LUNA

20 DE OCTUBRE DE 2017

Índice

Índice....2
Presentación....5
Introducción....5
Propósito....6
Competencias Genéricas....6
Requisitos....6
Unidad de Competencia I....7
Identificar y establecer los elementos de un Sistema de Base de Datos Distribuidos....7
Fundamentos de Bases de Datos distribuidas....8
Arquitectura de los Sistemas de Bases de Datos Distribuidas....9
Características de las Bases de Datos Distribuidas....9
Tipos de BDD: Homogéneas y Heterogéneas. 10
Ventajas de una BDD: 11
Desventajas de una BDD: 11
Fragmentación....11
Tipos de Fragmentación....12
Replicación 13
Transacciones distribuidas....13
Protocolos de compromiso (2FC) 15
Consultas distribuidas....18
Diseño de una Base de Datos Distribuida:....19
Resumen....20
Unidad de Competencia II 22
Conocer la arquitectura de una Data Warehouse....22
Fundamentos de la arquitectura de la Datawarehose....23
Tipo de datos....24

Datos de Negocio 26
Metadatos 27
Tipos de metadatos de la Data Warehouse 28
Arquitectura de datos conceptual 30
Uso de SQL para el acceso a los datos. 34
Resumen 35
Unidad de Competencia III 36
Identificar e implementar una Base de Datos XML 36
Fundamentos de Bases de Datos distribuidas 37
Elementos de XML, Atributos XML. 38
Documentos XML 41
Estructura de un documento 42
Componentes de un documento XML 43
Datos Semiestructurados 47
Definiciones de tipo documento (DTD) 49
Esquema XML 52
Lenguajes de Consulta para XML. 54
Unidad de Competencia IV 59
Identificar e implementar un Sistema de Base de Datos Orientado a Objetos 59
Bases de Datos Orientadas a Objetos 60
Modelo Orientado a Objetos 61
Clase objetos. 61
Herencia 63
Herencia múltiple 64
Identidad de los objetos 64
Sistemas Gestores de Bases de Datos Orientadas a Objetos 65
Características obligatorias: las reglas de Oro 65

Características opcionales 66
Opciones abiertas 66
Object Store 66
Lenguajes de creación y manipulación de objetos 67
¿Qué es una BDOO? 68
Conclusión 71
Tablas e Ilustraciones 73
Referencias 75

Presentación

El documento presenta una compilación de notas retomadas de fuentes arbitradas con la finalidad de fortalecer a los estudiantes del Programa Educativo de Ingeniería en Computación al tratamiento de información, ya que ésta es el activo de mayor impacto en las empresas, instituciones educativas y oficinas de gobierno. Para alcanzar el objetivo trazado se requiere exponer

En función a las necesidades actuales de las empresas, se propone un programa integral en la formación del ingeniero en computación (Universidad Autónoma del Estado de México, 2010); de acuerdo con el plan de estudios se abordan cuatro unidades de competencia para componer la Unidad de Aprendizaje Bases de Datos:

1. Bases teóricas y terminología usada en el diseño e implementación de una base de datos distribuida.
2. Fundamentos para generar consultas avanzadas utilizando el lenguaje SQL.
3. Aspectos de Bases de Datos XML
4. Definición de una Base de Datos Orientada a Objetos

Introducción

Los sistemas informáticos requieren de una materia prima sin errores y oportuna en el momento que se necesita, así que las bases de datos en la actualidad son el auge en las empresas, ya que a través de ellas se administra la información de forma definida y específica detonando el creciente desarrollo de sistemas expertos en la toma de decisiones.

Los mayores cambios se atribuyen principalmente a dos causas, que se dieron desde las décadas de los setenta:

1. El desarrollo de los microprocesadores, que permitieron reducir en tamaño y costo a las computadoras y aumentar en gran medida las capacidades de los mismos y su acceso a más personas.
2. El desarrollo de las redes de área local y de las comunicaciones que permitieron conectar computadoras con posibilidad de transferencia de datos a alta velocidad.

Es en este contexto que aparece el concepto de "Sistemas Distribuidos" que se ha popularizado tanto en la actualidad y que tiene como ámbito de estudio las redes como, por ejemplo: Internet, redes de teléfonos móviles, redes corporativas, redes de empresas, etc. Este documento presenta una panorámica de los aspectos relevantes que están involucrados en los Sistemas Distribuidos".

Propósito

Estudiar los conceptos fundamentales de Bases de Datos Distribuidas, los almacenes de datos e identificar las ventajas del uso del sistema de datos orientados a objetos y XML (Universidad Autónoma del Estado de México, 2010).

Competencias Genéricas

Diseñar, implementar y accesar a distintos tipos de Sistemas de Bases de Datos (Universidad Autónoma del Estado de México, 2010).

Requisitos

Para cursar la unidad de aprendizaje de Bases de datos, el estudiante debe haber cursado la unidad de aprendizaje de Fundamentos de Base de Datos, ya que van seriadas.

Unidad de Competencia I

Identificar y establecer los elementos de un Sistema de Base de Datos Distribuidos

Fundamentos de Bases de Datos distribuidas

Temario

- Características de la Base de Datos Distribuida
- Fragmentación
- Replicación
- Transacciones Distribuidas
- Protocolos de Compromiso (2FC)
- Consultas Distribuidas
- Diseño de una Base de Datos Distribuida

Objetivos

- Identificar claramente las ventajas y desventajas del uso de Sistemas de Base de Datos Distribuidos y los Sistemas de Base de Datos Centralizados.
- Manejar adecuadamente los conceptos de Base de Datos distribuidas.
- Identificar la manera en que funciona la distribución en determinados Sistemas Gestores de Bases de Datos.

Glosario

- Base de Datos: Una base de datos es un conjunto de datos almacenados en memoria externa que están organizados mediante una estructura de datos. (Marqués, 2011)
- Registro: Un registro es una colección de valores de campo que proporcionan información sobre una entidad. (Gómez Fuentes, 2013)
- Dato: Un dato es una representación simbólica de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades. (Marqués, 2011)
- Red: Es la interconexión de varios dispositivos y proporciona un medio para el intercambio de información entre ellos. (Stallings, 2004)

Arquitectura de los Sistemas de Bases de Datos Distribuidas

La arquitectura de un sistema de bases de datos está influenciada en gran medida por el sistema informático subyacente en el que se ejecuta, en particular por aspectos de la arquitectura de la computadora como la conexión en red, el paralelismo y la distribución (Siberschatz, Korth, & Sudarshan, 2013):

- ✓ La conexión en red de varias computadoras permite que algunas tareas se ejecuten en un sistema servidor y que otras ejecuten en los sistemas clientes. Esta división de trabajo ha conducido al desarrollo de sistemas de bases de datos cliente-servidor.
- ✓ El procesamiento paralelo dentro de una computadora permite acelerar las transacciones unas respuestas más rápidas, así como la capacidad de ejecutar más transacciones por segundo. Las consultas pueden procesarse de manera que se explote el paralelismo ofrecido por el sistema informático subyacente. La necesidad del procesamiento paralelo de consultas ha conducido al desarrollo de los sistemas de base de datos paralelos.
- ✓ La distribución de datos a través de las distintas sedes o departamentos de una organización permite que estos datos residan donde han sido generados, pero continuar siendo accesibles desde otros lugares o departamentos diferentes. El hecho de guardar copias de la base de datos en diferentes sitios permite que puedan continuar las operaciones sobre la base de datos, aunque algún sitio sea afectado por cualquier desastre natural. Se han desarrollado lo sistemas de base de datos para manejar datos distribuidos geográfica o administrativamente a lo largo de múltiples sistemas de bases de datos.

Características de las Bases de Datos Distribuidas

Las Bases de datos distribuidas son un grupo de datos que pertenecen a un sistema, pero a su vez está repartido entre ordenadores de una misma red, ya sea a nivel local o cada uno en una diferente localización geográfica, cada sitio en la red es autónomo en sus capacidades de procesamiento y es capaz de realizar operaciones locales y en cada uno de estos ordenadores debe estar ejecutándose una aplicación a nivel global

que permita la consulta de todos los datos como si se tratase de uno solo. (Siberschatz, Korth, & Sudarshan, 2013)

Centralizado	Distribuido
Control centralizado: un solo DBA	Control jerárquico: DBA global y DBA local
Independencia de Datos: Organización de los datos es transparente para el programador	Transparencia en la Distribución: Localización de los datos es un aspecto adicional de independencia de datos
Reducción de redundancia: Una sola copia de datos que se comparta	Replicación de Datos: Copias múltiples de datos que incrementa la localidad y la disponibilidad de datos
Estructuras físicas complejas para accesos eficientes	No hay estructuras intersitios. Uso de optimización global para reducir transferencia de datos
Seguridad	Problemas de seguridad intrínsecos

Tabla 1 Comparación de una Base de Datos Centralizada y una Base de Datos Distribuida

No olvidemos que una base de datos distribuida debe cumplir las condiciones de una Red Computacional. Una red de comunicación que nos proporcione las capacidades para que un proceso ejecutándose en un sitio de la red envíe y reciba mensajes de otro proceso ejecutándose en un sitio distinto.

Tipos de BDD: Homogéneas y Heterogéneas.

En las bases de datos distribuidas homogéneas todos los sitios tienen idénticamente el mismo software de sistema gestor de base de datos, y son conscientes de que los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. En estos sistemas estos sitios renuncian a la parte de su autonomía.

Las bases de datos distribuidas heterogéneas son sitios diferentes que pueden utilizar diferentes esquemas y diferente software de gestión de bases de datos, a diferencia en estas no son conscientes de la existencia de los demás sitios y esto nos causa que

haya facilidades limitadas para la cooperación en el procesamiento de las transacciones de información en la BD. (ELMASRI, 2001)

Ventajas de una BDD:

Organizativas:

- ✓ Adaptación a la organización responde a cambios.
- ✓ Almacenar los datos donde son generados en mayor parte locales.

Económicas:

- ✓ Costos de comunicación y de creación de pequeños sistemas

Técnicas:

- ✓ Flexibilidad, acceso desde distintos lugares y por distintas personas a la vez
- ✓ Fiabilidad/disponibilidad, en un determinado momento / intervalo. Varios sitios, duplicaciones, evitan fallos
- ✓ Modularidad

Desventajas de una BDD:

- ✓ Dificultad de diseño, fases adicionales
- ✓ Poca madurez de los productos comerciales, orientados a replicación
- ✓ Funciones de administración compleja, sincronización y coordinación
- ✓ Dificultad de cambio, inexistencia de metodologías
- ✓ Personal especializado

Fragmentación

En la fragmentación, el sistema divide la relación en varios fragmentos y guarda cada fragmento en un sitio diferente. (Siberschatz, Korth, & Sudarshan, 2013).

Existen dos tipos de esquema de fragmentación de las relaciones: fragmentación horizontal y fragmentación vertical.

La fragmentación horizontal divide la relación asignando cada tupla de la relación en uno o más fragmentos. La fragmentación vertical divide la relación descomponiendo el esquema de la relación.

Tipos de Fragmentación

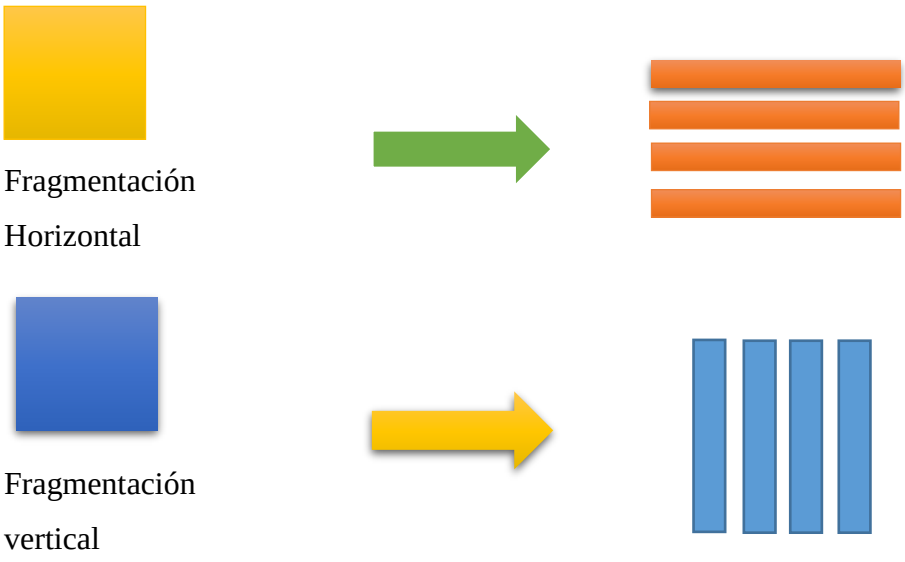

Ilustración 1 Tipos de Fragmentación

La fragmentación horizontal suele utilizarse para conservar las tuplas en los sitios en que más se utilizan, para minimizar la transferencia de datos (Siberschatz, Korth, & Sudarshan, 2013).

.

Se utiliza un predicado Pi para construir fragmentos ri, donde r es una relación:

$$ri = \sigma Pi\ (r)$$

Se reconstruye la relación r tomando la unión de todos los fragmentos; es decir,

$$r = r1 \cup r2 \cup \cdots \cup rn$$

La fragmentación vertical implica la definición de varios subconjuntos de atributos R1, R2, . . ., Rn del esquema R de modo que $R = R1 \cup R2 \cup \cdots \cup Rn$

Cada fragmento ri de r se define mediante:

$$ri = \Pi Ri\ (r)$$

La fragmentación debe hacerse de modo que se pueda reconstruir la relación r a partir de los fragmentos tomando la reunión natural

$$r = r1\ r2\ r3 \cdots rn$$

Replicación

La replicación es la copia sincronizada entre dos servidores de bases de datos de forma que cualquiera de los dos puede entregar los mismos resultados a sus clientes (Camps Paré, y otros, 2005).

Disponibilidad	Paralelismo incrementado.	Sobrecarga incrementada durante la actualización.
• Si alguno de los sitios que contiene la relación r falla, el sistema puede seguir procesando las consultas que impliquen a r en otro sitio distinto, pese al fallo del sitio original.	• En caso de que la mayoría de los accesos a la relación r sólo resulten en la lectura de la relación, varios sitios pueden procesar en paralelo las lecturas que impliquen a r, y esto minimiza el movimiento de los datos entre los sitios.	• El sistema debe asegurar que todas las réplicas de la relación r sean consistentes; en caso contrario pueden producirse cómputos erróneos.

Ilustración 2 Características de las réplicas. (Siberschatz, Korth, & Sudarshan, 2013)

Transacciones distribuidas

Cuando existe una transición distribuida es que una transacción puede invocar operaciones de objetos ubicados en diferentes servidores (Bermúdez, 2014). En el siguiente diagrama se muestra como es una transacción de forma distribuida:

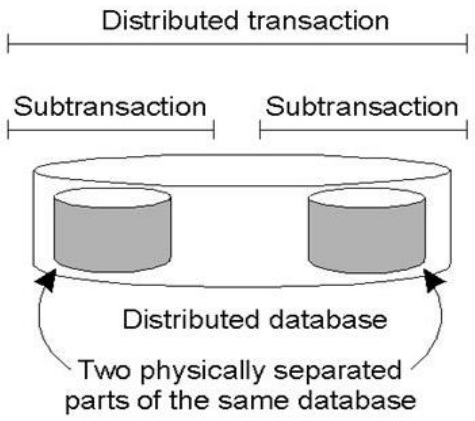

Ilustración 3 Transacción Distribuida. (Peter & Coronel, 2004)

La imagen nos muestra como una transacción distribuida contiene dos substracciones y cada una de ellas están separadas físicamente en la misma base de datos.

Existen dos tipos de transacciones que se deben considerar las cuales son:

- ✓ Transacciones locales: Son las transacciones que tiene acceso a los datos y los actualizan solo en una base de datos local.
- ✓ Transacciones globales: son las que tiene acceso a los datos y los actualizan en varias bases de datos locales.

Se pueden asegurar las propiedades ACID (Atomicity, Consistency, Isolation and Durability) de las transacciones locales. Sin embargo, para las transacciones globales, esta tarea resulta mucho más complicada, dado que puede que participen en la ejecución varios sitios (Siberschatz, Korth, & Sudarshan, 2013). El fallo de alguno de estos sitios, o el de un enlace de comunicaciones que conecte esos sitios, puede dar lugar a cálculos erróneos.

Cada sitio tiene su propio gestor local de transacciones. Los diferentes gestores de transacciones colaboran para ejecutar las transacciones globales. Para comprender el modo en que se pueden implementar estos gestores, se debe de tomar en cuenta que cada sitio tenga estos dos subsistemas:

- ✓ El gestor de transacciones administra la ejecución de las transacciones (o subtransacciones) que tienen acceso a los datos almacenados en un sitio local.

- ✓ El coordinador de transacciones coordina la ejecución de las diferentes transacciones (tanto locales como globales) iniciadas en ese sitio.

Protocolos de compromiso (2FC)

El protocolo commit de dos fases, también conocido como protocolo de compromiso (2FC) es un proceso de consenso distribuido que permite a todos los sitio de un sistema distribuido ponerse de acuerdo para hacer commit a una transacción. Típicamente es usado en Bases de datos distribuidas (Siberschatz, Korth, & Sudarshan, 2013). El objetivo de este protocolo es que todos los sitios realicen la transacción o la aborten.

Consideremos T como una transacción iniciada en el sitio S_k, en que el coordinador es C_i.

Fase 1:

En esta fase el coordinador C_i prepara la transacción T enviando a todos los sitios S_k donde se ejecutará T, un mensaje con las siguientes características <preparar T> y lo guarda en el registro histórico. Al recibir este mensaje S_k determina si desea hacer su parte de T, si su respuesta es negativa añade un registro <no T> al registro histórico, y responde a C_i con un mensaje de abortar T. Pero si la respuesta es sí, añade un registro <T preparada> al registro histórico y envía un mensaje de T preparada a C_i.

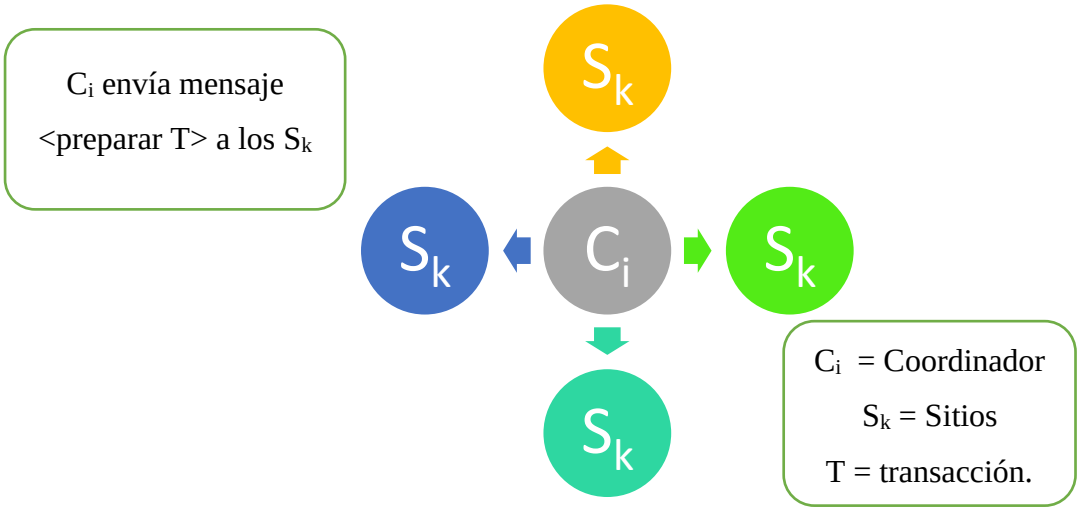

Ilustración 4. Primera fase del protocolo de compromiso (2FC).

Fase 2:

En esta siguiente fase cada S_k responde al C_i con un mensaje, si alguno de estos S_k ha enviado abortar T, el C_i abortara la transacción T, pero si todos los mensajes de los Si son positivos, ósea, envían T preparada, el C_i determina que T puede comprometerse. En función del resultado, se añadirá al registro histórico un registro <T comprometida> o <T abortada> según sea el caso y Ci enviara a todos los S_k un mensaje abortando T o comprometiendo T, y los S_k lo guardaran en el registro histórico.

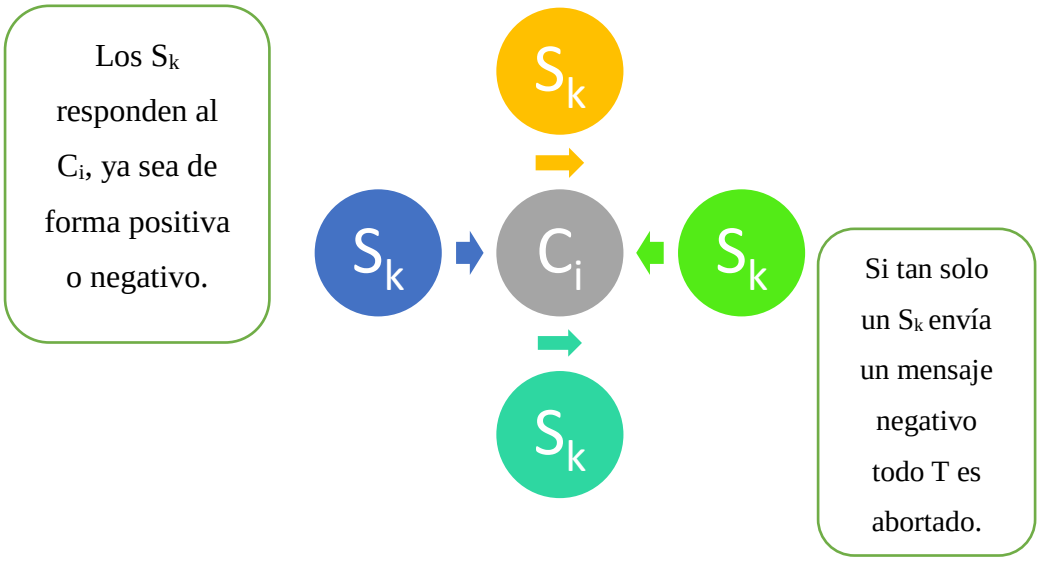

Ilustración 5 Segunda fase del protocolo de compromiso (2FC).

Fallo de un sitio participante:

Si el coordinador C_i detecta que un sitio S_k ha fallado realiza lo siguiente.

- ✓ 1° Si el S_k falla antes de enviar su mensaje T preparada, el C_i da por entendido que respondió abortar T.
- ✓ 2° Si el S_k falla después de enviar T preparada, el C_i ejecuta el resto del protocolo ignorando el fallo del sitio.

Si el S_k se recupera del fallo debe revisar su registro histórico para determinar el destino de la transacción que estaba en ejecución. Para ellos existen varios casos que se deben tomar en cuenta para T:

1. El registro histórico contiene una <T comprometida> el S_k ejecuta rehacer (T).
2. El registro histórico contiene una <T abortada> el S_k ejecuta deshacer (T).
3. El registro histórico contiene una <T preparada> el S_k consulta a Ci, para determinar el destino de T. Si C_i está activo notifica a S_k si T se comprometió o aborto.
4. El registro histórico no contiene ningún registro de (abortada, comprometida o preparada) con respecto a T. Por lo tanto, se sabe que como S_k fallo antes de enviar respuesta preparar T a C_i, C_i aborta T y se continua con el algoritmo entonces esa S_k debe de ejecutar rehacer (T).

Fallo del coordinador:

En caso de que falle el C_i los S_k deberán decidir el destino de la T. Pero en algunos casos los S_k no pueden decidir decir si comprometer o abortar T, y deberán esperar la recuperación del C_i.

1. Si un S_k contiene un registro <T comprometida> en el registro histórico, entonces T debe comprometerse.
2. Su un S_k contiene un registro con <T abortada> en el registro histórico, entonces T debe abortarse.
3. Si ningún S_k contiene un registro <T comprometida> o <T preparada> en su registro histórico, entonces quiere decir que el C_i decidió abortar T para no comprometer T.
4. Si no se da ninguno de los casos anteriores, y todos los S_k tienen un registro de <T preparada> en su registro histórico, dado que el coordinador fallo, no se sabe qué decisión tomo con respecto a T. en este caso los S_k deben averiguar qué decisión tomo el C_i, o esperar hasta que se recupere el mismo. Pero esto no es muy favorable, ya que, T puede seguir utilizando recursos del sistema y retrasar nuevas transacciones, porque se pone en estado de bloqueo.

Consultas distribuidas

Permite que los usuarios consulten zonas remotas, utilizando el mejor camino hacia ese lugar y los mejores recursos que puedan realizar correctamente la consulta (Teran, 2014).

Las consultas distribuidas tienen como objetivo convertir transacciones de usuario en instrucciones para manipulación de datos. No obstante, el orden en que se realizan las transacciones afecta grandemente la velocidad de respuesta del sistema. Así, el procesamiento de consultas presenta un problema de optimización en el cual se determina el orden en el cual se hace la menor cantidad de operaciones. En las bases de datos distribuidas se tiene que considerar los siguientes aspectos:

- ✓ El costo de transmisión de datos en la red.
- ✓ Repetición y fragmentación.
- ✓ Procesamiento de intersección simple.

Además, en las consultas distribuidas el origen de los datos puede estar almacenado en el mismo equipo o en equipos distintos y también detienen el acceso a datos de varios orígenes de datos homogéneos.

En SQL la sentencia join permite combinar registros de dos o más tablas en una base de datos relacional. En el Lenguaje de Consultas Estructurado (SQL), hay tres tipo de *JOIN*: interno, externo, y cruzado.

En el siguiente esquema se mostrarán los pasos para el procesamiento de consultas.

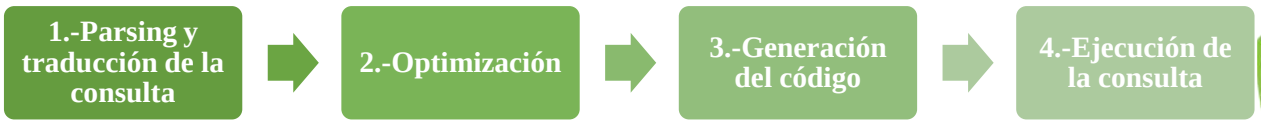

Ilustración 6 Pasos para el procesamiento de consultas.

Diseño de una Base de Datos Distribuida:

Un sistema de base de datos distribuida (SBDD) es entonces el resultado de la integración de una base de datos distribuida con un sistema para su manejo.

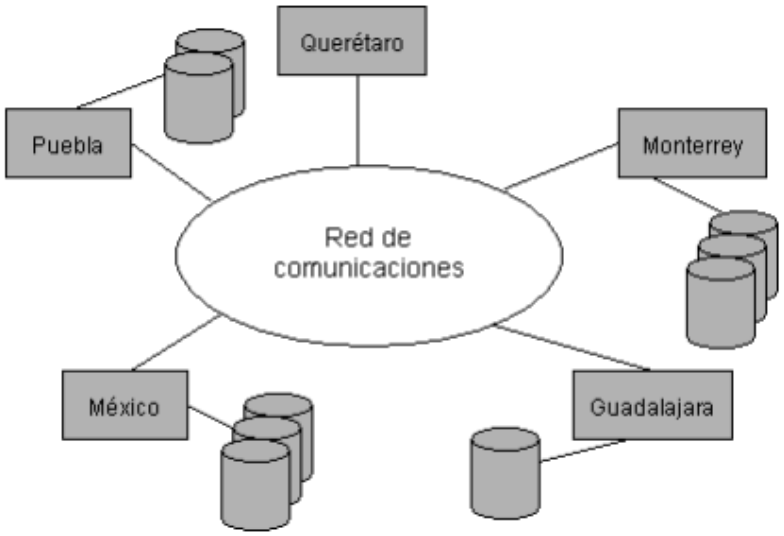

Ilustración 7 Ejemplo de una BDD. (Peter & Coronel, 2004)

Algo muy importante en el diseño de una BDD es la arquitectura y existen tres tipos:

Arquitecturas de memoria compartida: Consiste en diversos procesadores los cuales acceden una misma memoria y una misma unidad de almacenamiento.

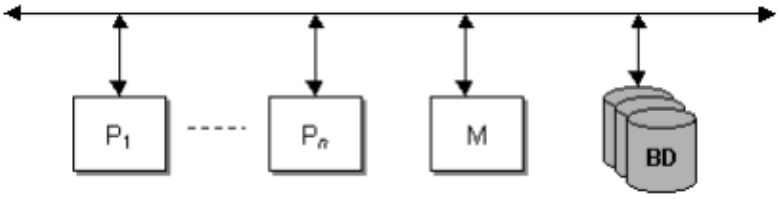

Ilustración 8 Memoria Compartida. (Peter & Coronel, 2004)

Arquitectura de disco compartido: Consiste en diversos procesadores cada uno de ellos con su memoria local, pero compartiendo una misma unidad de almacenamiento.

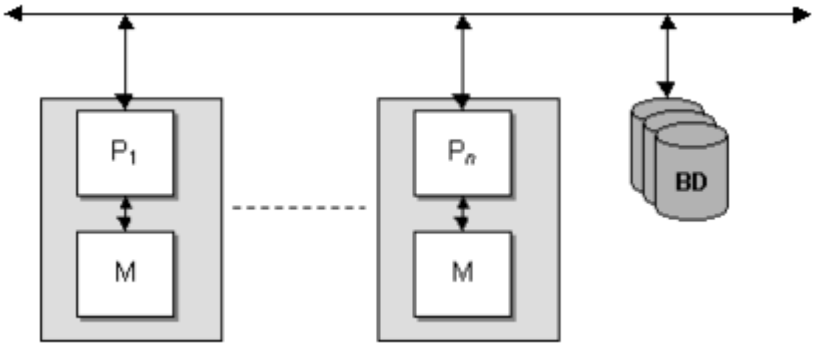

Ilustración 9 . Disco Compartido. (Peter & Coronel, 2004)

Arquitecturas de nada compartido: Consiste en diversos procesadores cada uno con su propia memoria y su propia unidad de almacenamiento.

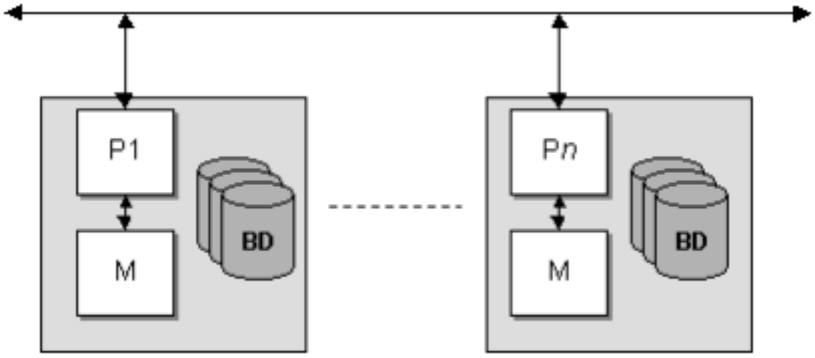

Ilustración 10 Almacenamiento con memoria propia. (Peter & Coronel, 2004)

Resumen

Con todo lo mencionado podemos concluir que es importante saber cómo se maneja la información en el mundo actual, ya que esto definirá el control de información en distintas empresas, como también cabe mencionar que el proceso distribuido de información nos da muchas posibilidades de crecimiento global, ya que éste nos da ventajas en las cuales nos hacen un mejor funcionamiento y mejor disponibilidad de la información; comparado a un sistema centralizado que solo nos ofrece tráfico en la red, poca disponibilidad y sobre todo redundancia, además también nos pudimos dar cuenta que al manejar bases de datos distribuidas se nos hace mucho más fácil poder acceder a nuestra información de una manera inmediata y segura, así como

también si falla el sistema en una localización específica, este no afecta a los demás sistemas y pueden seguir funcionando de manera correcta lo cual en las bases de datos centralizadas era un gran problema. Todos estos avances tecnológicos son importantes ya que hacen más fácil el acceso de información sin perdidas y sin redundancia.

Unidad de Competencia II

Conocer la arquitectura de una Data Warehouse

Fundamentos de la arquitectura de la Datawarehose

Temario

- Tipos de datos
- Datos del negocio
- Metadatos
- Arquitectura de datos conceptual (una sola capa, dos capas y tres capas)
- Uso de SQL para acceso a los datos

Objetivos

- Elaboración de consultas avanzadas en un sistema de administración de base de datos

Glosario

- KDD (Knowledge Discovery in Database): Se refiere al proceso no trivial de descubrir conocimiento e información potencialmente útil dentro de los datos contenidos en algún repositorio de información.
- ETL (Extract Transform Load): Es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos, limpiarlos y cargarlos en otra base de datos, data mart o data warehouse.
- Metodología: Conjunto de métodos que se siguen en una investigación científica, un estudio o una exposición doctrinal.
- OLAP (Online Analytical Processing): Es una solución utilizada en el campo de la llamada inteligencia de negocios cuyo objetivo es agilizar la consulta de grandes cantidades de datos.

Introducción a Data Warehouse

Un almacén de datos (Data Warehouse, DW) es una colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza. Se trata, sobre todo, de un historial completo de la organización, más allá de la información transaccional y operacional, almacenado en una base de datos diseñada para favorecer el análisis y la divulgación eficiente de datos.

Tipo de datos

Para comprender los datos con los que se van a trabajar hay que realizar un proceso de extracción del conocimiento (KDD), este comienza con la recopilación e integración de la información a partir de unos datos iniciales de que se dispone. Las primeras fases del KDD son muy importantes porque determinan que las fases sucesivas sean capaces de extraer el conocimiento válido y útil a partir de la información original. Generalmente, la información que se quiere investigar sobre un cierto dominio de la organización se encuentra en bases de datos y otras fuentes muy diversas, tanto internas como externas.

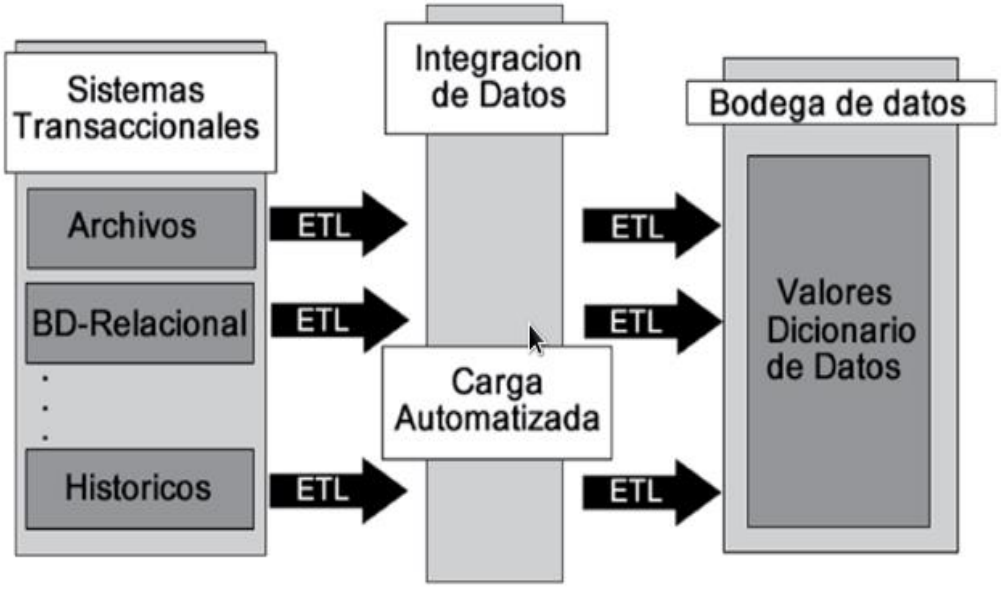

Ilustración 11 Almacén de Datos

En general la información se encuentra ordenada en almacenes de datos y su análisis posterior será mucho más sencillo si las fuentes de origen están unificadas, son

accesibles y están desconectadas del trabajo transaccional. Aparte de la información interna de la organización, los almacenes de datos pueden recoger información externa como demografías, páginas amarillas, psicográficas, uso de internet, información de otras organizaciones y base de datos externas compradas a otras compañías. La disponibilidad de grandes volúmenes de información en esta fase nos lleva a la necesidad de usar técnicas de muestreo para la selección de datos (López, 2008).

Como podemos observar en el diagrama anterior se puede tomar información desde cualquier fuente que esté relacionada con lo que queremos obtener aun y cuando esta contenga información de hace años, pues al final si esta información contiene datos relevantes será candidata a un análisis ETL para que pueda integrarse con los demás datos y con ello integrarse a nuestra bodega de datos.

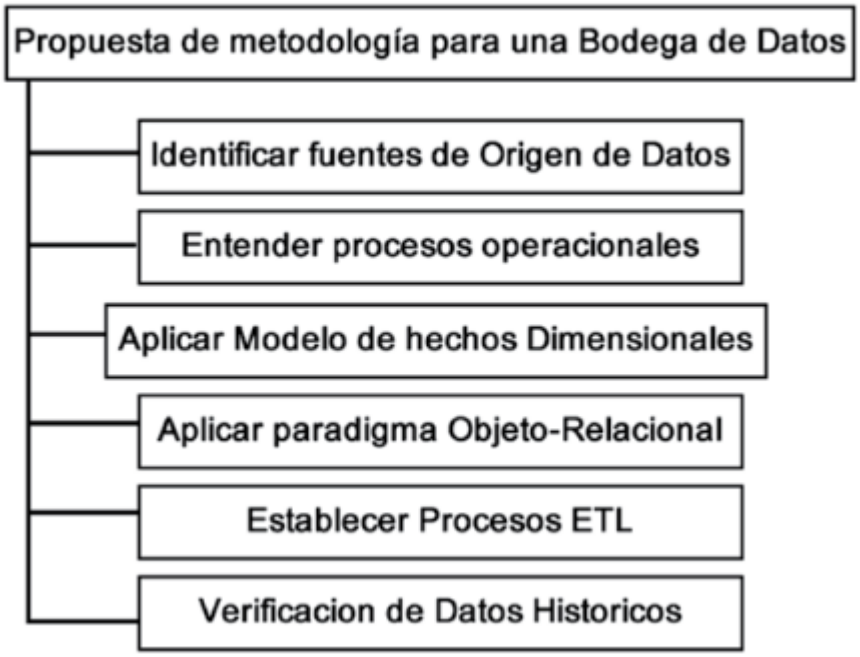

Ilustración 12 Metodología para la creación de un almacén de datos

Como se puede observar dentro de la propuesta para una metodología para una bodega de datos, identificar la fuente de origen de datos es el primer paso y el más importante pues sin buenas fuentes la bodega de datos será redundante y sin sentido (Sarniento, 2011).

Datos de Negocio

Para comprender de mejor manera los requerimientos que puede tener una empresa se debe de realizar un análisis e investigación de su entorno, este proceso permitirá adquirir los requerimientos del negocio y su indicador clave de rendimiento, además se deben de definir las fuentes de información a utilizar con el fin de comprender el ámbito del negocio para una correcta implementación de un esquema dimensional. Previamente se tienen que conocer los problemas que afectan a la organización, por lo que es importante definir lo siguiente:

- ✓ La necesidad actual.
- ✓ Requerimientos del negocio.
- ✓ Indicadores clave de rendimiento (KPI)
- ✓ Fuentes de información

Una vez definido los puntos clave se requiere realizar un análisis de datos OLAP, este tendrá su arquitectura de acuerdo con las necesidades del sistema y tendrá la función de trabajar de manera óptima con los datos, así como con los metadatos (Fuentes, 2010) .

De igual manera existen otras técnicas que ayudan a la resolución de problemas particulares de la organización, basándose en los datos que éstos poseen. Estas técnicas son:

- ✓ Razonamiento estadístico.
- ✓ Visualización.
- ✓ Procesamiento paralelo.
- ✓ Aprendizaje automático
- ✓ Apoyo en la toma de decisiones

Igualmente, la automatización en algoritmos de minería de datos permitirá detectar fácilmente patrones en los datos, razón por la cual estas técnicas son mucho más eficientes que el análisis dirigido a la verificación, cuando se intenta explorar datos procedentes de repositorios de gran tamaño y complejidad elevada.

Metadatos

Los metadatos, en sí, no suponen algo completamente nuevo dentro del mundo bibliotecario. Según Howe (1993), el término fue acuñado por Jack Myers en la década de los 60 para describir conjuntos de datos. La primera acepción que se le dio (y actualmente la más extendida) fue la de dato sobre el dato, ya que proporcionaban la información mínima necesaria para identificar un recurso.

Ilustración 13 Esquema de definición de datos

Un metadato describe los atributos de un recurso, teniendo en cuenta que el recurso puede consistir en un objeto bibliográfico, registros e inventarios archivísticos, objetos geoespaciales, recursos visuales y de museos o implementaciones de software. Aunque puedan presentar diferentes niveles de especificidad o estructura, el objetivo principal es el mismo: describir, identificar y definir un recurso para recuperar, filtrar, informar sobre condiciones de uso, autentificación y evaluación, preservación e interoperabilidad.

En términos fáciles y simples se pueden ver a los metadatos como los datos de los datos, como ejemplo podemos visualizar un archivo de audio con extensión mp3, desde la extensión tenemos metadatos, además, el título, el artista, la foto del álbum y demás propiedades de ese archivo son los metadatos, puesto que es información

(datos) que en conjunto forman ese archivo de audio, son datos que forman otro dato más grande.

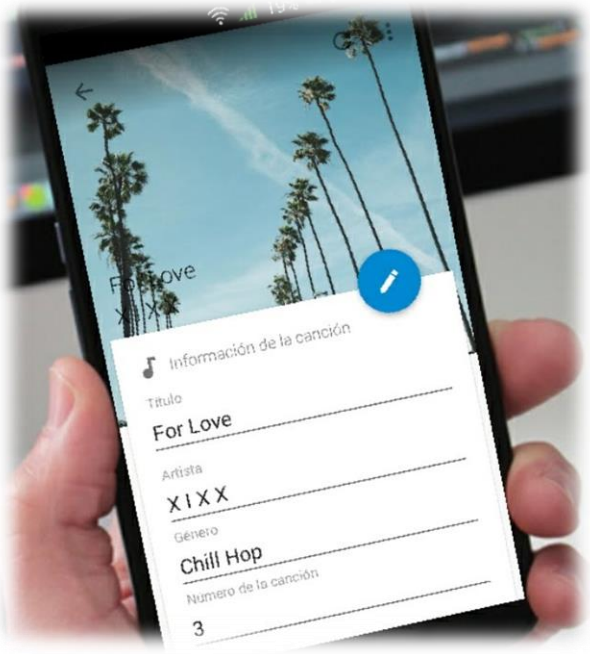

Ilustración 14 Metadatos de un archivo Mp3

Tipos de metadatos de la Data Warehouse

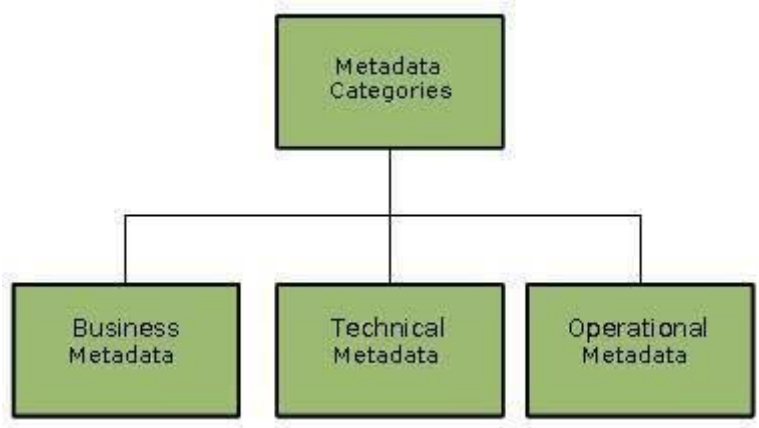

Ilustración 15 Tipos de Metadatos

Metadatos operacionales. Los datos para el depósito de datos provienen de varios sistemas operativos de la empresa. Estos sistemas de origen contienen diferentes estructuras de datos. Los elementos de datos seleccionados para el depósito de datos tienen varias longitudes de campo y tipos de datos. Al seleccionar los datos de los

sistemas de origen para el almacén de datos, divide los registros, combina partes de registros de diferentes archivos de origen y trata con múltiples esquemas de codificación y longitudes de campo. Cuando entregue información a los usuarios finales, debe poder vincularlos a los conjuntos de datos fuente originales. Los metadatos operativos contienen toda esta información sobre las fuentes de datos operacionales.

Metadatos de extracción y transformación. Contienen datos sobre la extracción de datos de los sistemas de origen, a saber, las frecuencias de extracción, los métodos de extracción y las reglas comerciales para la extracción de datos. Además, esta categoría de metadatos contiene información sobre todas las transformaciones de datos que tienen lugar en el área de almacenamiento de datos.

Metadatos del usuario final. Son el mapa de navegación del almacén de datos. Permite a los usuarios finales encontrar información del almacén de datos. Los metadatos del usuario final permiten a los usuarios finales usar su propia terminología comercial y buscar información de la manera en que normalmente piensan en el negocio.

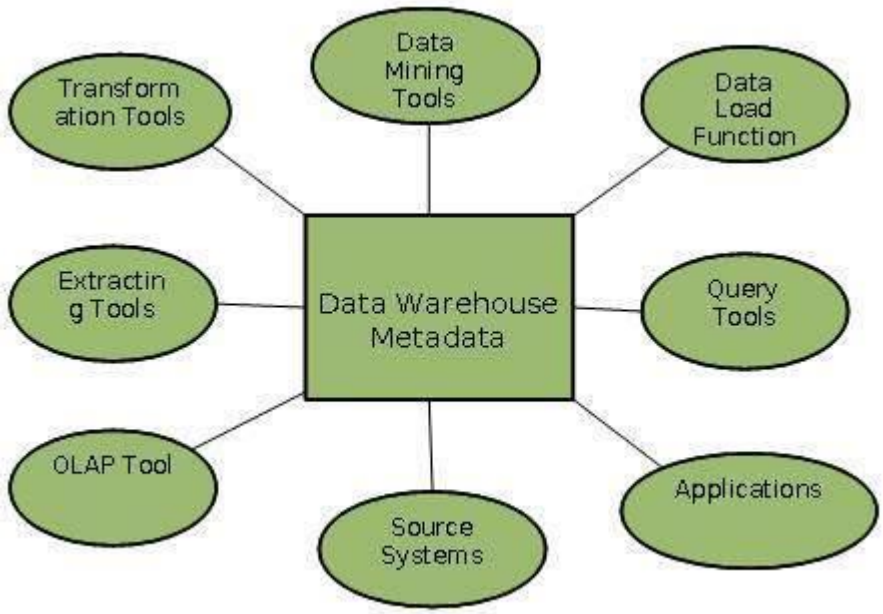

Ilustración 16 Metadatos de Usuario Final

Arquitectura de datos conceptual

El termino de data warehouse hace referencia a la arquitectura de los datos, debido a que tiene una relación especifica con el almacenamiento físico de los datos. tomando en cuenta las fuentes consultadas define la arquitectura de una bodega de datos con cuatro componentes:

- ✓ Los sistemas fuente, donde se gestiona la información relevante de la operación de la organización.
- ✓ El área intermedia (ostaging area), en la cual se hace la integración, unificación y limpieza de los datos que vienen de los diferentes sistemas fuente.
- ✓ El área de almacenamiento, conformada por dos elementos: el repositorio y los metadatos.
- ✓ El área de acceso a los datos a través de diferentes herramientas de consulta, tales como publicación en la web, generadores de reportes dinámicos y predefinidos, herramientas de miner²a de datos y OLAP (Frade & Castillo, 2007)

La arquitectura de una data warehouse la podemos clasificar en tres capas las cuales cumplen funciones específicas. Además, su estructura es diferente para cada una de las capas aumentando su complejidad dependiendo la capa implementada como se muestra en la ilustración.

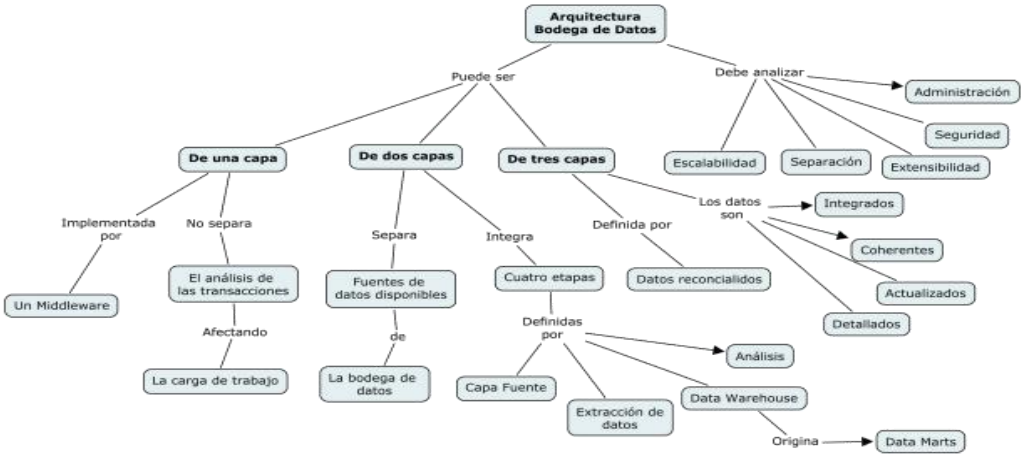

Ilustración 17 Arquitectura de un Data Warehouse

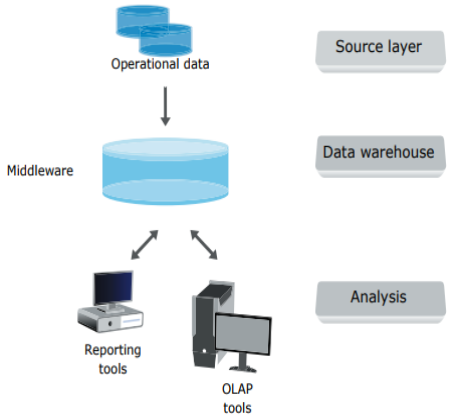

Ilustración 18 Estructura de Data Warehouse de una Capa

- ✓ La estructura general de una Data Warehouse de una capa es la menos usada actualmente en la práctica y tiene como función minimizar la cantidad de datos almacenados omitiendo los datos redundantes. La debilidad de esta arquitectura radica en su incapacidad para cumplir con el requisito de la separación entre el proceso de análisis y transaccional. Las consultas de análisis son enviadas a los datos operacionales una vez el middleware los interpreta.
- ✓ En la segunda capa la complejidad estructural aumenta incluyendo etapas diferentes la cuales se mencionarán más adelante. Se denomina arquitectura de dos capas a la separación física entre las fuentes de datos disponibles y lo dispuesto en la bodega de datos. actualmente la segunda capa muestra 4 etapas las cuales se muestran en la ilustración.

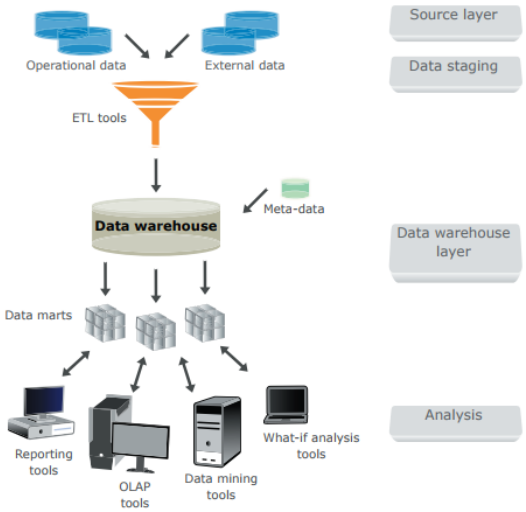

Ilustración 19 Segunda capa de una estructura Data Warehouse

Finalmente, en la tercera capa se agrega algunos elementos los cuales se muestran en la ilustración. En esta capa se materializan los datos operativos obtenidos después de la integración y la limpieza de datos desde el origen. Como resultado, los datos son integrados, coherentes, actuales y detallados. La ilustración muestra una bodega de datos que no es poblada de sus fuentes de forma directa, pero si a partir de datos reconciliados.

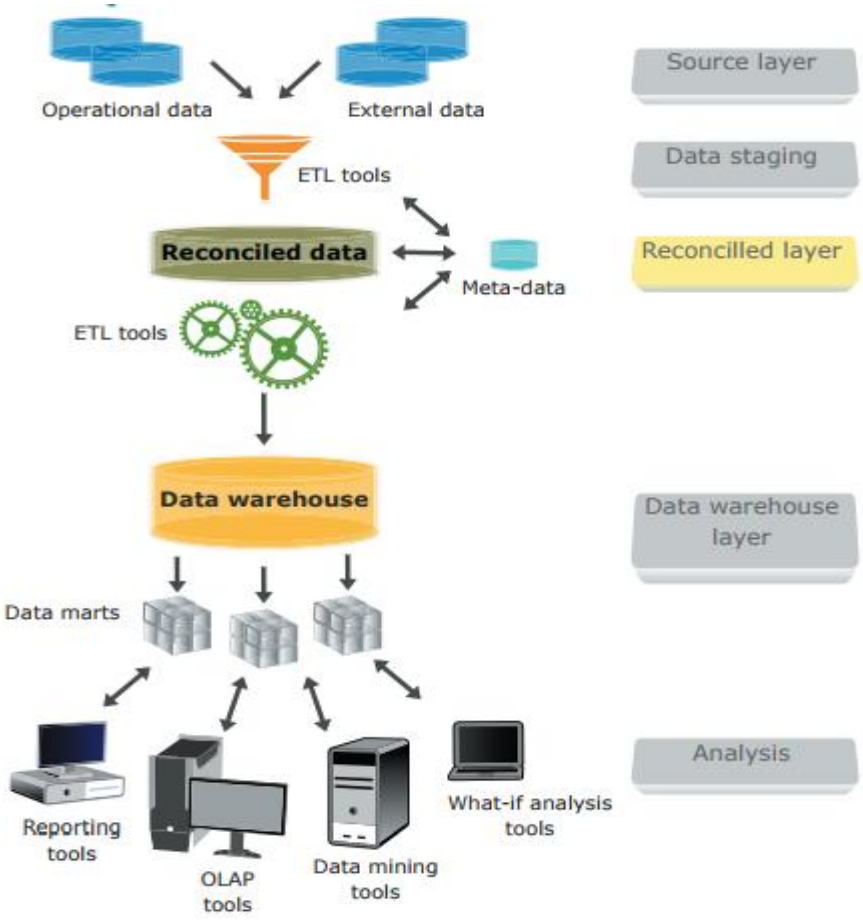

Ilustración 20 Tercera capa de una estructura Data Warehouse

Cada una de las capas tiene algunas etapas comunes entre si las cuales se definen de la siguiente forma:

- ✓ Capa fuente: Un sistema de almacenamiento de datos utiliza fuentes heterogéneas de datos.
- ✓ Extracción de Datos: Los datos almacenados en las fuentes deben ser extraídos y limpiados para remover inconsistencias y llenar espacios vacíos, integrando fuentes de datos heterogéneos a partir de esquemas comunes.

- ✓ Capa de Data Warehouse: La información es almacenada en un repositorio centralizado denominado data warehouse o bodega de datos.
- ✓ Análisis: En esta capa, la integración de datos es eficiente y de acceso flexible para generar informes, analizar la información de forma dinámica y simular escenarios hipotéticos de negocio.

En la implementación de estas capas podemos definir un término muy recurrente el cual es data marts las cuales se deriva de la certeza que cualquier usuario tiene necesidades de información limitada, y aunque típicamente existen requerimientos para análisis funcionales cruzados, el tamaño de los requerimientos es reducido materialmente si limitamos el tamaño del warehouse en sí mismo (Raquel Abella, 1999).

Los data marts se definen en dos tipos: dependientes e independientes como se muestra en las imágenes

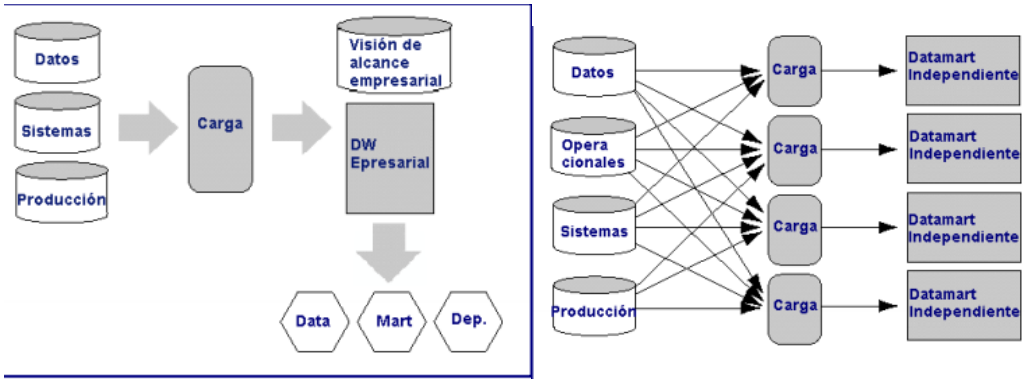

Ilustración 21 Data marts

Para logra definir la arquitectura en general de una data warehouse se analizan los siguientes términos los cuales son definidos de la siguiente forma según el servicio nacional de aprendizaje:

- ✓ Separación: Los procesos de análisis y transacciones deberían ser guardados en lo posible de forma independiente.
- ✓ Escalabilidad: Las arquitecturas de Hardware y Software deben ser fáciles de mejorar conforme al volumen de los datos, definidas para fácil administración y manejo de procesos.
- ✓ Extensibilidad: La arquitectura debe estar en capacidad de recibir nuevas aplicaciones y tecnologías sin rediseñar el sistema

- ✓ Seguridad: Monitorear los accesos es una tarea esencial previa al almacenamiento de los datos en una data warehouse.
- ✓ Administrable: La administración de los datos en la bodega de datos no debe ser compleja. (Silva, 2012)

Uso de SQL para el acceso a los datos.

Como se mencionó anteriormente los data warehouse se pueden representar como cubos en los cuales se almacenan los datos en SQL esto se representa con dos conjuntos uno que represente la agregación y el otro el grupo de datos:

Equivalencia en consultas SQL

Conjunto A – Consultas de agregación

Conjunto B – Consultas con grouping sets

A1. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY GROUPING SETS** ((a,b)) B1. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY** a, b

A2. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY GROUPING SETS** ((a,b),a) B2. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY** a, b **UNION SELECT** a, null, **SUM**(c) **FROM** tab1 **GROUP BY** a

A3. **SELECT** a,b, **SUM**(c) **FROM** tab1 **GROUP BY GROUPING SETS** (a,b) B3. **SELECT** a, null, **SUM**(c) **FROM** tab1 **GROUP BY** a **UNION SELECT** null, b, **SUM**(c) **FROM** tab1 **GROUP BY** b

A4. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY GROUPING SETS** ((a,b),a,b,())

B4. **SELECT** a, b, **SUM**(c) **FROM** tab1 **GROUP BY** a, b **UNION SELECT** a, null, **SUM**(c) **FROM** tab1 **GROUP BY** a, null **UNION SELECT** null, b, **SUM**(c) **FROM** tab1 **GROUP BY** null, b **UNION SELECT** null, null, **SUM**(c) **FROM** tab1

A partir de las fuentes consultadas establecemos que existen dos tipos de representaciones de data warehouse las cuales son implementadas de la siguiente forma en SQL (ARZALUZ, 2015):

- ✓ CUBE: crea un subtotal de todas las posibles combinaciones de los conjuntos de columnas incluidos en sus argumentos
- ✓ GROUP BY CUBE(a,b,c) es equivalente a
- ✓ GROUP BY GROUPING SETS ((a,b,c),(a,b),(b,c),(a,c),(a),(b),(c),())
- ✓ ROLLUP: calcula la agregación en niveles jerárquicos de la dimensión (ARZALUZ, 2015)
- ✓ ROLLUP (a, b, c) es equivalente a
- ✓ GROUPING SETS ((a,b,c),(a,b),(a),())

Resumen

Un Data Warehouse es una base de datos diseñado para permitir las actividades de inteligencia empresarial: existe para ayudar a los usuarios a comprender y mejorar el rendimiento de su organización. Está diseñado para consulta y análisis en lugar de para el procesamiento de transacciones, y por lo general contiene datos históricos derivados de datos de transacciones, pero puede incluir datos de otras fuentes. Los Data Warehoue separan la carga de trabajo del análisis de la carga de trabajo de la transacción y permiten que una organización consolide datos de varias fuentes. Esto ayuda en:

1. Mantenimiento de registros históricos
2. Analizar los datos para obtener una mejor comprensión del negocio y para mejorar el negocio.

Unidad de Competencia III

Identificar e implementar una Base de Datos XML

Fundamentos de Bases de Datos distribuidas

Temario

- Elementos XML, Atributos XML
- Documentos XML
- Datos Semiestructurados
- Definiciones de tipo de documento (DTD)
- Lenguajes de Consulta para XML

Objetivos

- Identificar los conceptos básicos del enfoque XML
- Utilizar el Xquery para el acceso a una Base de Datos XML
- Implementar un Sistema de Bases de XML

Glosario

- XML: Es un subconjunto de SGML (Estándar Generalised Mark-up Language), simplificado y adaptado a Internet.
- DTD: (document type definition) es una descripción de estructura y sintaxis de un documento XML o SGML.
- Grafo: Resultan ser extremadamente útiles para analizar problemas muy diversos.
- Dato: Cifras, instrucciones que se tienen aisladas entre sí, sin seguir una organización u orden en específico.
- Anidado: Es la técnica de incorporar llamadas a funciones o procedimientos dentro de otros, mediante la inclusión de diversos niveles de paréntesis.
- Filtrar: Seleccionar valores desde un documento XML.
- Mezclar: Integrar valores desde múltiples fuentes.
- Transformar valores desde un esquema a otro.
- minOccurs: especifica el número mínimo de veces que un elemento puede ocurrir

- maxOccurs: especifica el número máximo de veces que un elemento puede ocurrir

Introducción

Como se sabe, XML es el acrónimo de Extensible Markup Language, o que es lo mismo: Lenguaje extensible de marcado. Para el intercambio de datos entre aplicaciones en internet independiente del formato de almacenamiento de los mismos. Es lógico que las consultas entre aplicaciones se expresan como consultas contra los datos en formato XML, muchas aplicaciones requieren el almacenamiento de datos XML. Para comprender XML es importante entender sus raíces como un lenguaje de marcas de documentos.

En función a las necesidades actuales de las empresas, se propone un programa integral en la formación del ingeniero en computación (Universidad Autónoma del Estado de México, 2010); de acuerdo con el plan de estudios se abordan cuatro unidades de competencia para componer la Unidad de Aprendizaje Bases de Datos:

1. Bases teóricas y terminología usada en el diseño e implementación de una base de datos distribuida.
2. Fundamentos para generar consultas avanzadas utilizando el lenguaje SQL.
3. Aspectos de Bases de Datos XML
4. Definición de una Base de Datos Orientada a Objetos

Elementos de XML, Atributos XML.

El constructor fundamental en un documento XML es el **elemento**. Un elemento es sencillamente un par de etiquetas de inicio y finalización coincidentes y todo el texto que aparece entre ellas. Se pueden definir como bloques de construcción de un XML Los elementos pueden comportarse como contenedores para texto, elementos, atributos, objetos de soporte o de todas ellas.

Cada documento XML contiene uno o más elementos, el alcance de lo que son o bien delimitado por las etiquetas inicial y final, o de elementos vacíos, de una etiqueta de elemento vacío. Los documentos XML deben tener un unico elemento **raíz** que abarque el resto de los elementos en el documento. En el ejemplo de la Ilustración

22 el elementos <banco> forma el elemento raiz. Ademas, los elementos en un documento XML se deben anidar adecuadamente.

```
</banco>
    <cuenta>
        <número-cuenta> C-101 </número-cuenta>
        <nombre-sucursal> Centro </nombre-sucursal>
        <saldo> 500 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-201 </número-cuenta>
        <nombre-sucursal> Galapagar </nombre-sucursal>
        <saldo> 900 </saldo>
    </cuenta>
```

Ilustración 22 Representación de XML de información bancaria.

Por ejemplo:

<cuenta> … <saldo> … </saldo> … </cuenta>

Esta anidado adecuadamente, mientras que

<cuenta> … <saldo> … </cuenta> … </saldo>

No esta adecuadamente anidado.

Aunque el anidamiento es una propiedad intuitiva, la debemos definir más formalmente. Se dice que el texto aparece en el contexto de un elemento si aparece entre la etiqueta de inicio y la etiqueta de finalización de dicho elemento. Las etiquetas están anidadas adecuadamente si toda etiqueta de inicio tiene una única etiqueta de finalización coincidente que está en el contexto del mismo elemento padre.

Además de los elementos, XML especifica la noción de **atributo.** Por ejemplo, el tipo de cuenta se puede representar como atributo, como en la Ilustración 23. Los

atributos de un elemento aparecen como pares *nombre = valor* antes del cierre “>” de una etiqueta.

```
...
    <cuenta tipo-cuenta = «corriente»>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
...
```

Ilustración 23 . Uso de atributos

Los atributos le permiten añadir información sobre un elemento mediante pares de nombre-valor. Los atributos se suelen utilizar para definir propiedades de elementos que no se consideran el contenido del elemento, aunque en algunos casos (por ejemplo, el elemento img HTML) el contenido del elemento se determina mediante valores de atributo.

Además, los atributos pueden aparecer solamente una vez en una etiqueta dada, al contrario que las subelementos, que pueden estar repetidos. Es importante la distinción entre subelemento y atributo; un atributo es implícitamente texto que no aparece en el documento impreso o visualizado. Sin embargo, en la aplicación de bases de datos y de intercambio de datos de XML esta distinción es menos relevante y la elección de representar los datos como un atributo o un sub-elemento es frecuentemente arbitraria.

Los atributos deben tener un nombre y un valor. No se permite un nombre sin un valor. Un elemento no puede tener dos atributos con el mismo nombre. Como XML no considera importante el orden en que aparecen los atributos dentro de un elemento, es posible que el analizador XML no lo conserve.

Al igual que los nombres de los elementos, los nombres de los atributos distinguen mayúsculas y minúsculas y deben empezar por una letra o un carácter de subrayado. El resto del nombre puede contener letras, números, guiones, caracteres de subrayado y puntos.

Una nota sintáctica final es que un elemento de la forma <elemento> </elemento>, que no contiene sub elementos o texto, se puede abreviar como <elemento/> los elementos abreviados pueden, no contener atributos (Abraham Silberschatz, 2002).

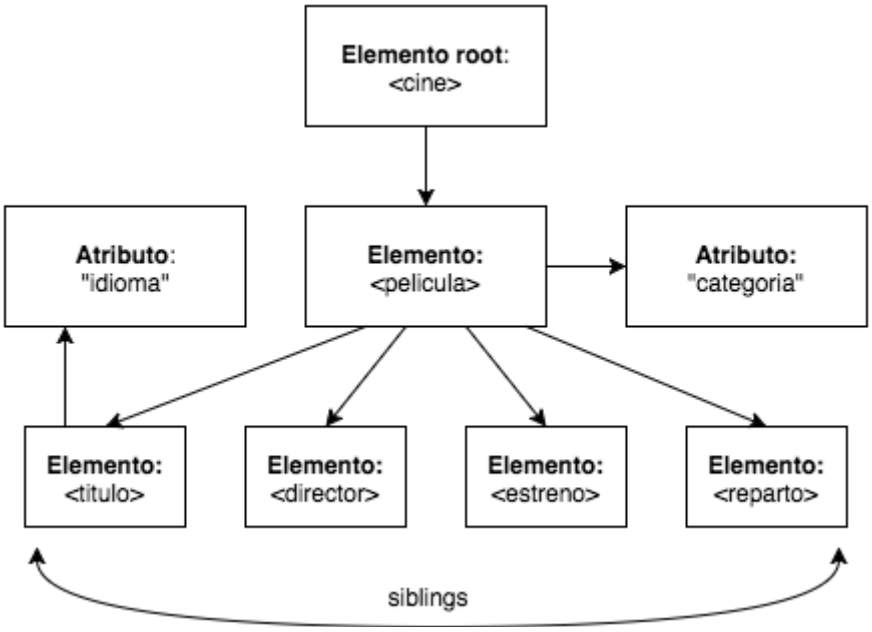

Ilustración 24 Representación de los atributos

Documentos XML

Definición: es una parte opcional de un documento XML. La definición de tipos de documento (Document Type Definition, DTD) es una parte opcional del documento XML. El propósito principal es similar al de un esquema, restringir el tipo de información presente en el documento. Sin embargo, DTD no restringe en realidad los tipos en el sentido básico como entero o cadena. Cada declaración está en la forma de una expresión normal para las subelementos de un elemento (Silberschatz, Korth, & Sudarshan, 2002).

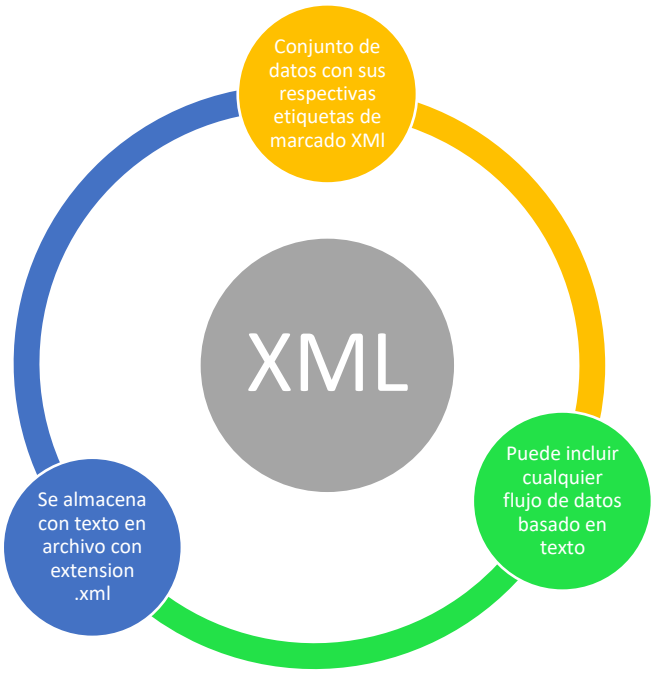

Ilustración 25 Documento XML

Estructura de un documento

Está formado por datos de caracteres y marcado, el marcado lo forman etiquetas

Prólogo
<?xml version =”1.0” encoding=”ISO-8859-1” standalone=”no”?>
<! DOCTYPE persona SYSTEM “persona.dtd”

Cuerpo
<persona>
<nombre> Luis</nombre>
<apellidos> Perez</apellidos>
</persona>

Nombre del atributo — Contenido del elemento

<autor país = “España” > Jose Ramon </autor>

Nombre de elemento — Etiqueta de fin

Valor de atributo

Componentes de un documento XML

En un documento XML existen los siguientes componentes (Universidad a Distancia de Madrid, 2001):

- ✓ Elementos: Pieza lógica del marcado, se representan con una cadena de texto (dato) encerrada entre etiquetas. Pueden existir elementos vacíos (
). Los elementos pueden contener atributos.
- ✓ Instrucciones: Ordenes especiales para ser utilizadas por la aplicación que procesa <?xml-stylesheet type=”text/css” href=”estilo.css”>
- ✓ Las instrucciones XML: Comienzan por <? Y terminan por ?>
- ✓ Comentarios: Información que no forma parte del documento. Comienzan por <! y terminan por -->
- ✓ Declaraciones de tipo: Especifican información acerca del documento: <!DOCTYPE persona SYSTEM “persona.dtd”>
- ✓ Secciones CDATA: Se trata de un conjunto de caracteres que no deben ser interpretados por el procesador: <![CDATA[Aquí se puede meter cualquier carácter, como < , & , >, Sin que sean interpretados como marcación]]>

Ejemplo 1

En los archivos de datos que se muestran en los ejemplos siguientes, <tab> indica un carácter de tabulación en un archivo de datos, y <return> indica un retorno de carro.

A. Ordenar campos de datos de caracteres igual que columnas de tabla

En el ejemplo siguiente se muestra un archivo de formato XML que describe un archivo de datos que contiene tres campos de datos de caracteres. El archivo de formato asigna el archivo de datos a una tabla que contiene tres columnas. Los campos de datos se corresponden uno a uno con las columnas de la tabla.

Tabla (fila): Person (Age int, FirstName varchar(20), LastName varchar(30))

Archivo de datos (registro): Age<tab>Firstname<tab>Lastname<return>

El siguiente archivo de formato XML lee del archivo de datos a la tabla.

En el elemento <RECORD>, el archivo de formato representa los valores de datos de los tres campos como datos de caracteres. Para cada campo, el atributo TERMINATOR indica el terminador que sigue al valor de datos.

Los campos de datos se corresponden uno a uno con las columnas de la tabla. En el elemento <ROW>, el archivo de formato asigna la columna Age al primer campo, la columna FirstName al segundo campo y la columna LastName al tercer campo.

```
<?xml version="1.0"?>
<BCPFORMAT
xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RECORD>
    <FIELD ID="1" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="12"/>
    <FIELD ID="2" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="20" COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="3" xsi:type="CharTerm" TERMINATOR="\r\n"
      MAX_LENGTH="30"
      COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
  </RECORD>
  <ROW>
    <COLUMN SOURCE="1" NAME="age" xsi:type="SQLINT"/>
    <COLUMN SOURCE="2" NAME="firstname" xsi:type="SQLVARYCHAR"/>
    <COLUMN SOURCE="3" NAME="lastname" xsi:type="SQLVARYCHAR"/>
  </ROW>
</BCPFORMAT>
```

B. Ordenar campos de datos y columnas de tabla de forma diferente

En el ejemplo siguiente se muestra un archivo de formato XML que describe un archivo de datos que contiene tres campos de datos de caracteres. El archivo de formato asigna el archivo de datos a una tabla que contiene tres columnas que están ordenadas de forma diferente a los campos del archivo de datos.

Tabla (fila): Person (Age int, FirstName varchar(20), LastName varchar(30))
Archivo de datos (registro): Age<tab>Lastname<tab>Firstname<return>
En el elemento <RECORD>, el archivo de formato representa los valores de datos de los tres campos como datos de caracteres.

En el elemento <ROW>, el archivo de formato asigna la columna Age al primer campo, la columna FirstName al tercer campo y la columna LastName al segundo campo.

```
<?xml version="1.0"?>
<BCPFORMAT
xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RECORD>
    <FIELD ID="1" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="12"/>
    <FIELD ID="2" xsi:type="CharTerm" TERMINATOR="\t" MAX_LENGTH="20"
      COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="3" xsi:type="CharTerm" TERMINATOR="\r\n"
      MAX_LENGTH="30" COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
  </RECORD>
  <ROW>
    <COLUMN SOURCE="1" NAME="age" xsi:type="SQLINT"/>
    <COLUMN SOURCE="3" NAME="firstname" xsi:type="SQLVARYCHAR"/>
    <COLUMN SOURCE="2" NAME="lastname" xsi:type="SQLVARYCHAR"/>
  </ROW>
</BCPFORMAT>
```

C. Omitir un campo de datos

En el ejemplo siguiente se muestra un archivo de formato XML que describe un archivo de datos que contiene cuatro campos de datos de caracteres. El archivo de formato asigna el archivo de datos a una tabla que contiene tres columnas. El segundo campo de datos no se corresponde con ninguna columna de la tabla.

Tabla (fila): Person (Age int, FirstName Varchar(20), LastName Varchar(30))

Archivo de datos (registro):

Age<tab>employeeID<tab>Firstname<tab>Lastname<return>

En el elemento <RECORD>, el archivo de formato representa los valores de datos de los cuatro campos como datos de caracteres. Para cada campo, el atributo TERMINATOR indica el terminador que sigue al valor de datos.

En el elemento <ROW>, el archivo de formato asigna la columna Age al primer campo, la columna FirstName al tercer campo y la columna LastName al cuarto campo.

<?xml version="1.0"?>

```
<BCPFORMAT
xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RECORD>
    <FIELD ID="1" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="12"/>
    <FIELD ID="2" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="10"
      COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="3" xsi:type="CharTerm" TERMINATOR="\t"
      MAX_LENGTH="20"
      COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="4" xsi:type="CharTerm" TERMINATOR="\r\n"
      MAX_LENGTH="30"
      COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
  </RECORD>
  <ROW>
    <COLUMN SOURCE="1" NAME="age" xsi:type="SQLINT"/>
    <COLUMN SOURCE="3" NAME="firstname" xsi:type="SQLVARYCHAR"/>
    <COLUMN SOURCE="4" NAME="lastname" xsi:type="SQLVARYCHAR"/>
  </ROW>
</BCPFORMAT>
```

D. Asignar <FIELD> xsi:type a <COLUMN> xsi:type

En el ejemplo siguiente se muestran tipos de campos diferentes y sus asignaciones a columnas.

```
<?xml version = "1.0"?>
<BCPFORMAT
xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <RECORD>
      <FIELD xsi:type="CharTerm" ID="C1" TERMINATOR="\t"
            MAX_LENGTH="4"/>
      <FIELD xsi:type="CharFixed" ID="C2" LENGTH="10"
         COLLATION="SQL_LATIN1_GENERAL_CP1_CI_AS"/>
      <FIELD xsi:type="CharPrefix" ID="C3" PREFIX_LENGTH="2"
         MAX_LENGTH="32" COLLATION="SQL_LATIN1_GENERAL_CP1_CI_AS"/>
      <FIELD xsi:type="NCharTerm" ID="C4" TERMINATOR="\t"
         MAX_LENGTH="4"/>
      <FIELD xsi:type="NCharFixed" ID="C5" LENGTH="10"
         COLLATION="SQL_LATIN1_GENERAL_CP1_CI_AS"/>
      <FIELD xsi:type="NCharPrefix" ID="C6" PREFIX_LENGTH="2"
         MAX_LENGTH="32" COLLATION="SQL_LATIN1_GENERAL_CP1_CI_AS"/>
      <FIELD xsi:type="NativeFixed" ID="C7" LENGTH="4"/>
   </RECORD>
   <ROW>
      <COLUMN SOURCE="C1" NAME="Age" xsi:type="SQLTINYINT"/>
      <COLUMN SOURCE="C2" NAME="FirstName" xsi:type="SQLVARYCHAR"
      LENGTH="16" NULLABLE="NO"/>
      <COLUMN SOURCE="C3" NAME="LastName" />
      <COLUMN SOURCE="C4" NAME="Salary" xsi:type="SQLMONEY"/>
      <COLUMN SOURCE="C5" NAME="Picture" xsi:type="SQLIMAGE"/>
      <COLUMN SOURCE="C6" NAME="Bio" xsi:type="SQLTEXT"/>
      <COLUMN SOURCE="C7" NAME="Interest"xsi:type="SQLDECIMAL"
      PRECISION="5" SCALE="3"/>
   </ROW>
</BCPFORMAT>
```

E. Asignar datos XML a una tabla

En el ejemplo siguiente se crea una tabla vacía de dos columnas (t_xml), en la que la primera columna se asigna al tipo de datos int y la segunda columna se asigna al tipo de datos xml.

CREATE TABLE t_xml (c1 int, c2 xml)

El formato XML carga un archivo de datos en una tabla t_xml.

```
<?xml version="1.0"?>
<BCPFORMAT xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <RECORD>
  <FIELD ID="1" xsi:type="NativePrefix" PREFIX_LENGTH="1"/>
  <FIELD ID="2" xsi:type="NCharPrefix" PREFIX_LENGTH="8"/>
 </RECORD>
 <ROW>
  <COLUMN SOURCE="1" NAME="c1" xsi:type="SQLINT"/>
  <COLUMN SOURCE="2" NAME="c2" xsi:type="SQLNCHAR"/>
 </ROW>
</BCPFORMAT>
```

F. Importar campos de longitud fija o de ancho fijo

En el siguiente ejemplo se describen campos fijos de 10 o 6 caracteres cada uno. El archivo de formato representa estas longitudes y anchos de campo como LENGTH="10" y LENGTH="6", respectivamente. Cada fila de los archivos de datos termina con una combinación de retorno de carro y avance de línea, {CR}{LF}, que el archivo de formato representa como TERMINATOR="\r\n".

```
<?xml version="1.0"?>
<BCPFORMAT
       xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RECORD>
    <FIELD ID="1" xsi:type="CharFixed" LENGTH="10"/>
    <FIELD ID="2" xsi:type="CharFixed" LENGTH="6"/>
    <FIELD ID="3" xsi:type="CharTerm" TERMINATOR="\r\n"
  </RECORD>
  <ROW>
    <COLUMN SOURCE="1" NAME="C1" xsi:type="SQLINT" />
    <COLUMN SOURCE="2" NAME="C2" xsi:type="SQLINT" />
  </ROW>
</BCPFORMAT>
```

Datos Semiestructurados

Modelo de datos diseñado para afrontar problemas de integración de información.

Este tipo de modelos permite la especificación en los que cada elemento del mismo tipo puede contener atributos diferentes. Esto lo diferencia de los modelos de datos estructurados, en los que todos los elementos de datos de un tipo dado deben tener el mismo conjunto de atributos.

Integración de información (Silberschatz, Korth, & Sudarshan, 2002):

- ✓ Datos que se relacionan y existen en diferentes lugares y podrían, en principio, trabajar juntos.
- ✓ Pero en distintas bases de datos difieren en:
    1. Modelo (relacional de objetos)
    2. Esquema (normalizado / sin normalizar)
    3. Terminología (¿son los consultores
    4. empleados? ¿Jubilados? ¿Subcontratistas?)
    5. Convenciones (metros vs pies)

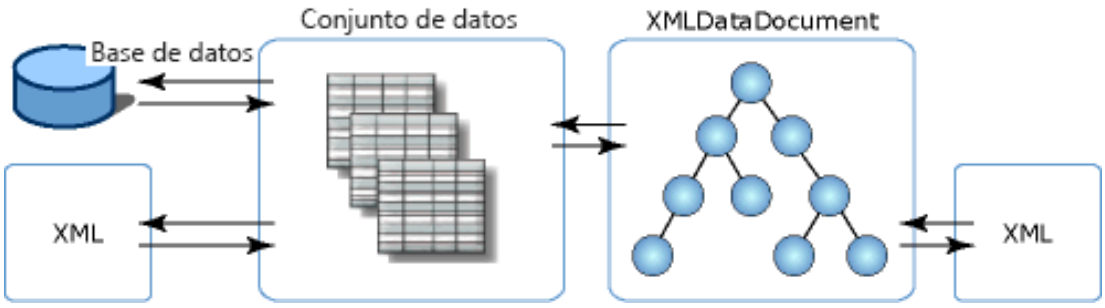

Ilustración 26 Esquema sobre integración de datos a XML y Base de Datos

Datos semiestructurados

- ✓ Propósito: representar los datos de fuentes independientes de una manera más flexible que en los modelos relacional u orientada a objetos.
- ✓ Pensar en objetos, pero que cada tipo de objeto se ajuste a su propio negocio, no en “clases”.
- ✓ Etiquetas para indicar el significado de subestructuras.

Ejemplo con grafos sobre Datos Semiestructurados.

- ❖ Nodos = Objetos

- Etiquetas sobre los arcos (atributos, relaciones)
- Valores atómicos en nodos hoja (nodos sin arcos salientes)
- Flexibilidad: Sin restricciones en:
    a) Etiquetas fuera de un nodo
    b) Número de sucesores con una etiqueta dada

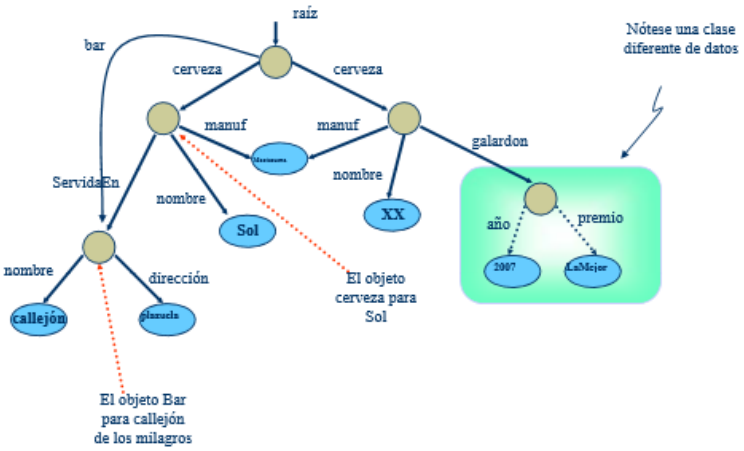

Ilustración 27 Grafo de datos (Sosa, 2001)

Definiciones de tipo documento (DTD)

Es una parte opcional de un documento XML. El propósito de principal de un DTD es similar al de un esquema: Restringir el tipo de información presente en el documento en donde solo restringe el aspecto de subelementos y atributos en un elemento (Silberschatz, Korth, & Sudarshan, 2002).

El DTD es una lista de reglas que indican el patrón de sub elementos que aparecen en un elemento.

```
</banco>
    <cuenta>
        <número-cuenta> C-101 </número-cuenta>
        <nombre-sucursal> Centro </nombre-sucursal>
        <saldo> 500 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-sucursal> Navacerrada </nombre-sucursal>
        <saldo> 400 </saldo>
    </cuenta>
    <cuenta>
        <número-cuenta> C-201 </número-cuenta>
        <nombre-sucursal> Galapagar </nombre-sucursal>
        <saldo> 900 </saldo>
    </cuenta>
    <cliente>
        <nombre-cliente> González </nombre-cliente>
        <calle-cliente> Arenal </calle-cliente>
        <ciudad-cliente> La Granja </ciudad-cliente>
    </cliente>
    <cliente>
        <nombre-cliente> López </nombre-cliente>
        <calle-cliente> Mayor </calle-cliente>
        <ciudad-cliente> PeguEtrerosos </ciudad-cliente>
    </cliente>
    <impositor>
        <número-cuenta> C-101 </número-cuenta>
        <nombre-cliente> González </nombre-cliente>
    </impositor>
    <impositor>
        <número-cuenta> C-201 </número-cuenta>
        <nombre-cliente> González </nombre-cliente>
    </impositor>
    <impositor>
        <número-cuenta> C-102 </número-cuenta>
        <nombre-cliente> López </nombre-cliente>
    </impositor>
</banco>
```

```
<!DOCTYPE banco [
    <!ELEMENT banco ( (cuenta|cliente|impositor)+)>
    <!ELEMENT cuenta ( número-cuenta nombre-sucursal saldo )>
    <!ELEMENT cliente ( nombre-cliente calle-cliente ciudad-cliente )>
    <!ELEMENT impositor ( nombre-cliente número-cuenta )>
    <!ELEMENT número-cuenta ( #PCDATA )>
    <!ELEMENT nombre-sucursal ( #PCDATA )>
    <!ELEMENT saldo( #PCDATA )>
    <!ELEMENT nombre-cliente( #PCDATA )>
    <!ELEMENT calle-cliente( #PCDATA )>
    <!ELEMENT ciudad-cliente( #PCDATA )>
]>
```

Ilustración 28 Ejemplo de un DTD tipo banco

Cada declaración está en la forma de una expresión normal para los sub-elementos de un elemento un ejemplo de esto sería como lo que se muestra en la ilustración 28. Así, un elemento bancario consiste en uno o más elementos cuenta, cliente o impositor; el operador | especifica «o» mientras que el operador + especifica «uno o más». Aunque no se muestra aquí, el operador * se usa para especificar «cero o más» mientras que el operador ? se usa para especificar un elemento opcional (es decir, «cero o uno»).

Existen otros dos tipos de declaraciones especiales que son empty (vacío) que dice que un elemento no tiene ningún contenido y any (cualquiera) que indica que no hay restricción sobre subelementos del elemento puede ser subelemento del elemento. La ausencia de una declaración para un sub-elemento es equivalente a declarar explícitamente el tipo como any.

Los atributos permitidos para cada elemento también se declaran en la DTD. Al contrario que los sub-elementos no se impone ningún orden a los atributos. Los atributos se pueden especificar del tipo CDATA, ID, IDREF o IDREFS.

- ✓ **CDATA :** Simplemente dice que el atributo contiene datos de caracteres

la siguiente línea de una DTD especifica que el elemento cuenta tiene un atributo del tipo tipo-cuenta con valor predeterminado corriente.

<!ATTLIST cuenta tipo-cuenta CDATA «corriente» >

Los atributos siempre deben tener una declaración de tipo y una declaración predeterminada. La declaración predeterminada puede consistir en un valor predeterminado para el atributo o #REQUIRED, queriendo esto decir que se debe especificar un valor para el atributo en cada elemento, #IMPLIED, lo que significa que no se ha proporcionado ningún valor predeterminado.

- ✓ **ID:** proporciona un identificador único para el elemento; un valor que tiene un atributo ID de un elemento no debe estar presente en ningún otro elemento del mismo documento.
- ✓ **IDREF:** es una referencia a un elemento; el atributo debe contener un valor que aparezca en el atributo ID de algún elemento en el documento.
- ✓ **IDREFS:** permite una lista de referencias, separadas por espacios.

Un ejemplo de DTD son las relaciones de la cuenta de un cliente se representan mediante los atributos ID e IDREFS en lugar de los registros impositor como se muestra en la ilustración 28. El elemento cuenta usan número-cuenta como su atributo identificador; para realizar esto se ha hecho que número-cuenta sea un atributo de cuenta en lugar de un sub-elemento. Los elementos cliente tienen un nuevo atributo identificador denominado cliente-ID. Además, cada elemento cliente contiene un atributo cuentas del tipo IDREFS, que es una lista de identificadores de las cuentas que posee el cliente. Cada elemento cuenta tiene un atributo tenedores del tipo IDREFS, que es una lista de propietarios de la cuenta (Silberschatz, Korth, & Sudarshan, 2002).

```
<!DOCTYPE banco-2 [
    <!ELEMENT cuenta (sucursal, saldo)>
    <!ATTLIST cuenta
        número-cuenta ID #REQUIRED
        tenedores IDREFS #REQUIRED >
    <!ELEMENT cliente (nombre-cliente, calle-cliente, ciudad-cliente)>
    <!ATTLIST cliente
        cliente-id ID #REQUIRED
        cuentas IDREFS #REQUIRED >
    ... declaraciones para sucursal, saldo, nombre-cliente,
        calle-cliente y ciudad-cliente ...
] >
```

Ilustración 29 Ejemplo de un ID y IDREFS

Las definiciones de tipos de documentos están fuertemente conectadas con la herencia del formato del documento XML. Debido a esto no son adecuadas por varios motivos para servir como estructura de tipos de XML para aplicaciones de procesamiento de datos (Silberschatz, Korth, & Sudarshan, 2002).

Algunas limitaciones de las DTD como mecanismo de esquema:

- ✓ No se pueden declarar el tipo de elementos y atributos de texto individuales. Por ejemplo, el elemento saldo no se puede restringir para que sea un número positivo.
- ✓ Es difícil usar el mecanismo DTD para especificar conjuntos desordenados de subelementos. El orden es rara vez importante para el intercambio de datos.
- ✓ Hay una falta de tipos en ID e IDREF. Por ello no hay forma de especificar el tipo de elemento al cual se debería referir un atributo IDREF o IDREFS.

Esquema XML

Un intento de reparar muchas de estas deficiencias de DTD produjo un lenguaje de esquema más sofisticado el XMLSchema

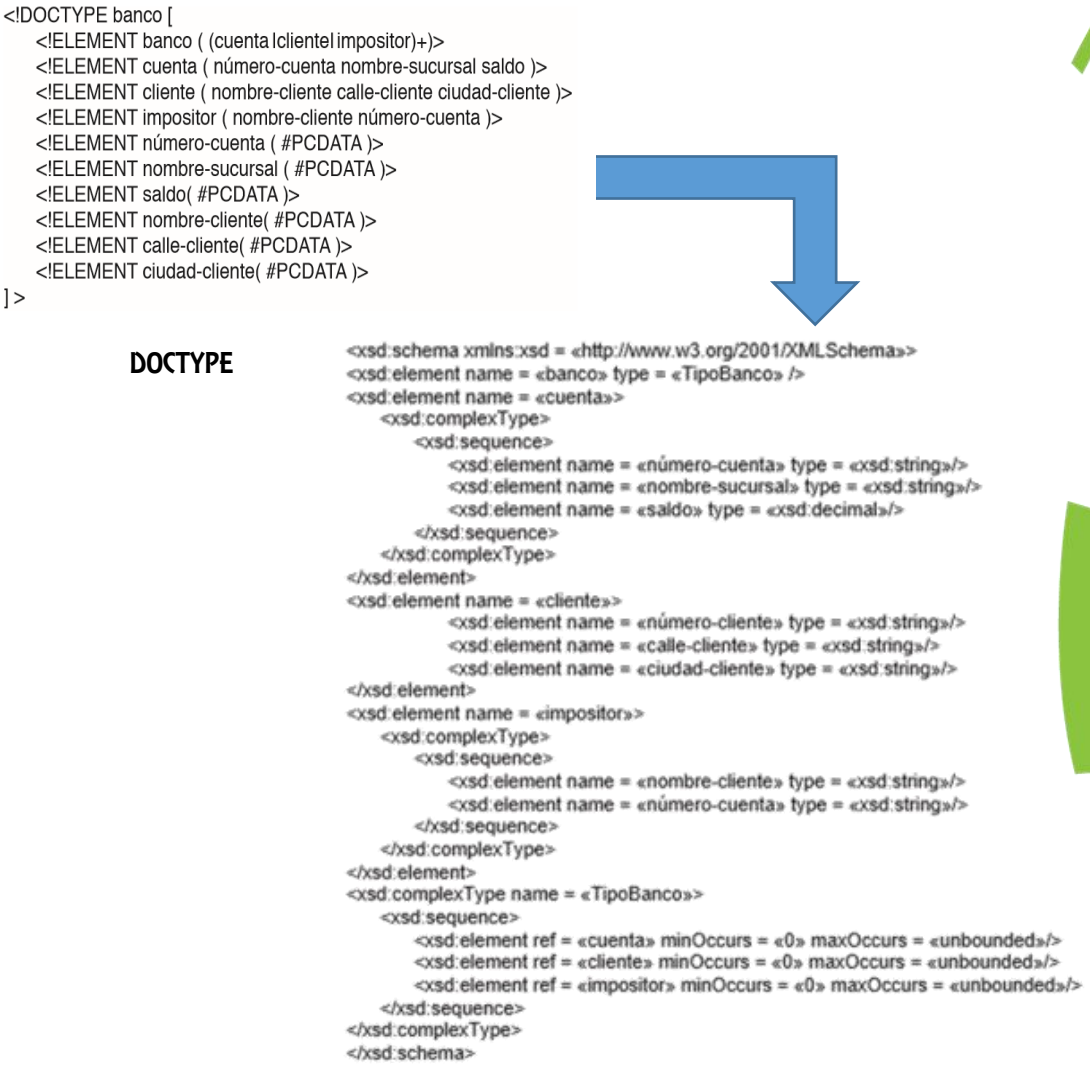

Ilustración 30 XML DOCTYPE A UN XML SCHEMA

Obsérvese el uso de los tipos xsd:string y xsd: decimal para restringir los tipos de los elementos de datos. Finalmente, el ejemplo es el que se muestra en la Ilustración 30 que define el tipo TipoBanco para contener cero o más apariciones de cada cuenta, cliente e impositor. XMLSchema puede definir el número mínimo y máximo de apariciones de subelementos mediante minOccurs y maxOccurs. El valor predeterminado para las apariciones máxima y mínima es 1, por lo que se tiene que especificar explícitamente para permitir cero o más cuentas, impositores y clientes (Silberschatz, Korth, & Sudarshan, 2002).

Entre los beneficios que ofrece XMLSchema res- pecto a las DTDs se encuentran los siguientes:

- Permite crear tipos definidos por el usuario
- Permite que el texto que aparece en los elementos esté restringido a tipos específicos tales como tipos numéricos en formatos específicos o incluso tipos más complicados como listas o uniones.
- Permite restringir los tipos para crear tipos especializados
- Permite la extensión de tipos complejos mediante el uso de una forma de herencia.
- Es un superconjunto de DTDs.
- Permite restricciones de unicidad y de clave externa.
- Está integrado con espacios de nombres para permitir a diferentes partes de un documento adaptarse a un esquema diferente

Sin embargo, el precio a pagar por estas características es que XMLSchema es significativamente más complicado que las DTDs

Lenguajes de Consulta para XML.

Un lenguaje de consulta para XML debería poseerlas cualidades comunes de los lenguajes de consulta de datos semiestructurados, tanto las puramente relacionales (operaciones de selección, filtrado, reducción, reestructuración y combinación) como similares a las de los lenguajes de consulta de bases de datos orientados a objetos, tales como la navegación.

A continuación, se explican con mayor detalle las principales particularidades deseables para la consulta de datos:

- ✓ **Operación de selección:** elija un documento o elemento basado en el contenido, estructura o atributos que satisfaga una condición específica. Estas consultas constan generalmente de 3 partes o cláusulas:
- ✓ **Patrón:** equipara elementos anidados en el documento de entrada y les asocia variables. Equivale a "from" en SQL, donde los patrones son tablas, vistas o alias.

En el caso de XML, un patrón es un elemento XML en el cual algunos de los elementos (nombres de etiqueta contenido, nombres de atributo o valores de atributo) hijo eventualmente sustituido por variables.

- ✓ **Filtro:** testea que las variables asociadas cum- plan las condiciones establecidas ("where" en SQL). El significado de los patrones y filtros en los lenguajes para datos semiestructurados es una relación (producto de cartesiano de todas las variables asignadas que satisface los patrones y filtros) que tiene una estructura plana y sin orden.
- ✓ **Constructor:** especifica el resultado en términos de las variables asociadas, es decir qué formato tiene de tener la respuesta. Se corresponde a la cláusula "seselda " de SQL. Las construcciones deberían poder incluir consultas anidadas, así como poder crear nuevos elementos.
- ✓ **Operación de filtrado:** extrae determinados elementos de los documentos conservando la jerarquía y secuencia.
- ✓ **Operación de reducción:** proyecta como salida la poda de los elementos especificados en la selección que satisfacen las condiciones, en vez de devolver un subárbol con todos los elementos y atributos.
- ✓ **Acción de reestructuración**, como por ejemplo la agrupación de datos relacionados y la ordenación.
- ✓ **Operación de combinación** de datos de diferentes porciones de documentos (correspondiente al "join" relacional) o combinación de diferentes partes del mismo documento ("semi-join").
- ✓ **Uso de funciones de agregación** gramo. Utilización de la cuantificación existencial y universal. marido. Operaciones de inserción, borrado y modificación. Además de estas características genéricas, para consultar documentos en XML sería también deseable disponer de:
- ✓ **Variables etiqueta o expresiones** de camino para permitir peticiones sin conocimiento preciso de la estructura del documento y acceso a datos anidados de forma arbitraria. El lenguaje de consulta debe poder usarse aun cuando no se conozca un esquema (dtd oxml schema) a priori.
- ✓ **Operadores de navegación** que simplifiquen el manejo de datos con referencias [atributos id, idref (s)]. -Manejo de tipos de datos, en particular los del XML.

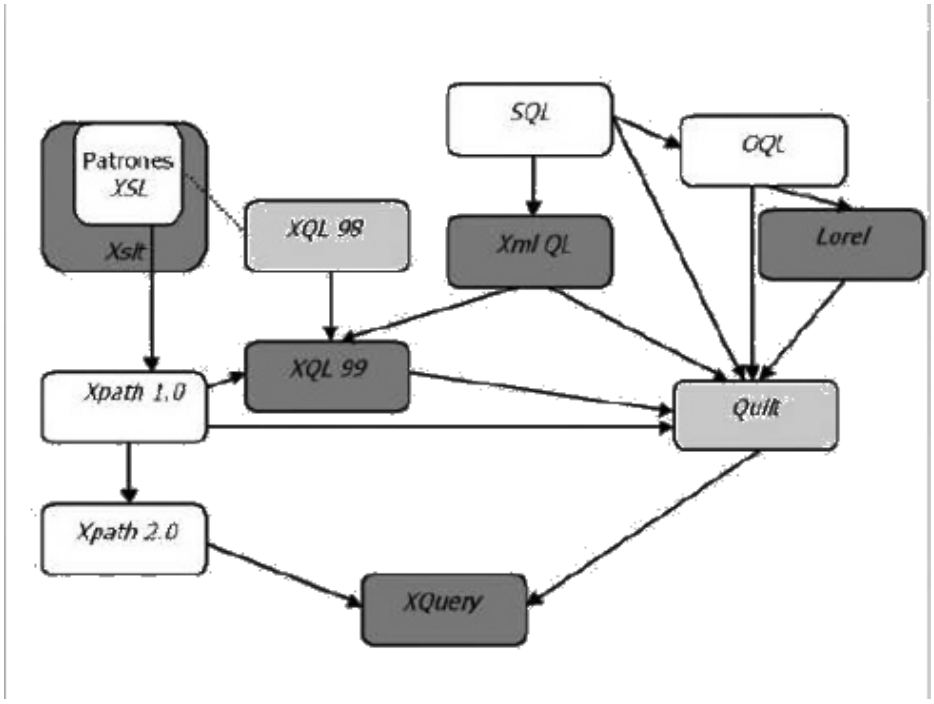

Ilustración 31 Query's Languajes for XML.

Un lenguaje de consulta para XML debería poseer las cualidades comunes de los lenguajes de consulta de datos semiestructurados, tanto las relacionales como aquellas de los lenguajes de consulta de bases de datos orientados a objetos. Los lenguajes de consulta para XML suavizan el problema del desconocimiento de la estructura exacta mediante la utilización de expresiones regulares de camino.

Desde el punto de vista de consulta de datos, todos los lenguajes estudiados poseen cláusulas de selección con patrón, filtro y constructor, excepto XQL que no posee esta última y no permite la creación de nuevos elementos. Todos toleran consultas anidadas y también expresiones de camino más o menos complejas.

XQuery es el más expresivo de todos los lenguajes si bien carece de las operaciones de reducción y de manipulación (inserción borrado y modificación). Lorel, al provenir de la comunidad de bases de datos, reúne la mayoría de las características estudiadas (incluidas las operaciones de manipulación), si bien no presenta las operaciones de reducción y filtrado, no soportando todos los tipos de datos de xml-schema. Por su parte, Xslt no posee operaciones de reestructuración de resultados (a excepción de la ordenación de nodos), de cuantificación universal y no per-mite el manejo de tipos de datos (si bien esto está en estudio para la próxima versión).

Xml-QL es menos expresivo que los anteriores al carecer de operaciones de filtrado, reducción, agrupa-ción de resultados, cuantificación universal, manejo de tipos de datos y de las operaciones de inserción, borrado y modificación.

XQL 99 añadió a XQL 98 la importante característica de realizar uniones inter-documentos. Sin embargo, todavía está falto de otras como la posibilidad de ordenación de resultados, la creación de nuevos elementos y el manejo de tipos de datos, además de no presentar la posibilidad de utilizar variables etiqueta. Sin embargo, XQL es más sencillo que xml-QL para consultas que requieran preservar la jerarquía, secuencia y anidamiento original, y por otra parte, algunas de las carencias de XQL mencionadas son extensiones del lenguaje propuestas para asemejarlo más a los lenguajes de datos semi-estructurados.

En la Ilustración 32 se puede ver la ordenación de los lenguajes analizados según su expresividad. En cuanto a las características ligadas a recuperación de información, los cinco lenguajes estudiados permiten realizar consultas teniendo en cuenta la inclusión estructural (indicada mediante expresiones de camino) y poseen índices posicionales que pueden utilizarse para consultas de inclusión posicional y de orden estructural. Algunos de los lenguajes poseen operadores específicos para expresar el orden estructural:
preceding, preceding-siblings, following, followingsiblings , en Xslt , y operadores before y after.

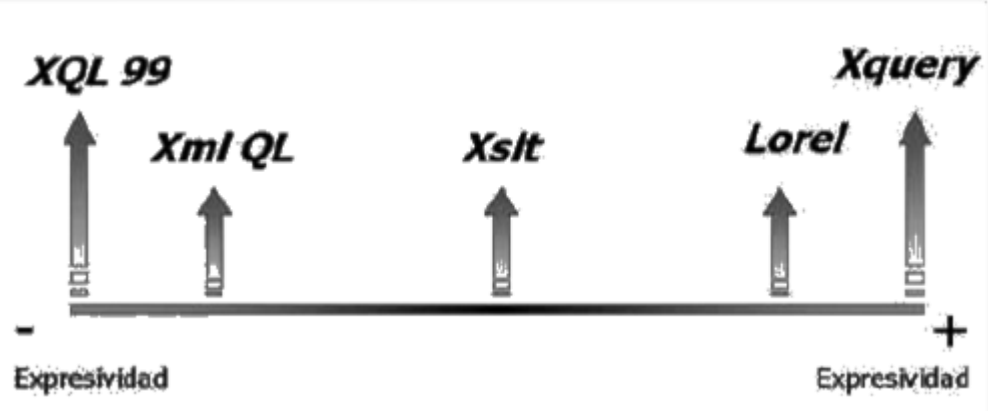

Ilustración 32 XQL 99 y XQuery

Todos permiten también operaciones sobre conjuntos, si bien XQL no puede expresar transformaciones de los resultados de la consulta. A pesar de estas características, los lenguajes presentan serias limitaciones para la recuperación de información ya que incluso la posibilidad más sencilla de búsqueda de una palabra en el texto se ha de realizar utilizando las expresiones regulares de camino, sin que exista una función u operador específico para ello. Tampoco permiten la búsqueda por similitud y por tanto ninguno ordena los resultados por relevancia, y también carecen de operadores de adyacencia, a excepción de ";" (inmediatamente precede) de XQL 99 para proximidad estructural. En cuanto a la búsqueda de patrones, Lorel y xml-QL son los únicos que incorporan el operador "like". Xql y Xslt proporcionan muchos operadores y funciones de texto, así como la posibilidad de comparar la posición de diferentes palabras.

XQuery hereda estas propiedades de XQL soportando además el uso de funciones externas. Sin embargo ninguno permite puntos de concordancia (match points) para presentar los puntos concretos del documento donde aparece el patrón bus-cado y permitir además visualizarlo en su contexto para asignarles una semántica correcta. Este aspecto es muy importante para investigadores del área de filología y lingüística y sí es considerado en otro lenguaje de consulta para documentos xml denominado Tequila-TX (Baeza-Yates, 2002).

Unidad de Competencia IV

Identificar e implementar un Sistema de Base de Datos Orientado a Objetos

Bases de Datos Orientadas a Objetos

Temario

- Modelo Orientado a Objetos
- Sistemas Gestores de Bases de Datos
- Object Store
- Lenguajes de creación y manipulación de objetos

Objetivos

- Identificar las ventajas y desventajas de las bases de datos relacionales y las bases de datos orientadas a objetos.
- Crear un Sistema de Base de Datos Orientado a Objetos
- Utilizar un lenguaje orientado a objetos para accesar a un SDBOO

Glosario

- Objeto: unidad dentro de un programa de computadora que consta de un estado y de un comportamiento, que a su vez constan respectivamente de datos almacenados y de tareas realizables durante el tiempo de ejecución.
- Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.
- Mensaje: Intercambio de solicitudes entre los objetos independientemente de los detalles concretos de su implementación.
- Clase: Es un modelo que define un conjunto de variables -el estado, y métodos apropiados para operar con dichos datos -el comportamiento
- Sistemas relacionales: Tipos de datos sencillos, lenguajes de consulta potentes, protección elevada.
- Sistemas relacionales orientados a objetos: Tipos de datos complejos, lenguajes de consulta potentes, protección elevada.

Modelo Orientado a Objetos

El paradigma orientado a objetos está basado en el encapsulamiento de los datos y del código relacionado con cada objeto en una sola unidad cuyo contenido no es visible desde el exterior. La interfaz entre cada objeto y el resto del sistema se define mediante un conjunto de mensajes permitidos. (Siberschatz, Korth, & Sudarshan, 2013)

En general, cada objeto está relacionado con:

- Un conjunto de variables que contiene los datos del objeto; las variables corresponden con los atributos del modelo E-R.
- Un conjunto de mensajes a los que responde; tener uno o varios.
- Un conjunto de métodos, cada uno de los cuales es código que implementa mensaje; el método devuelve un valor como respuesta al mensaje.

Hablando con rigor, en el modelo orientado objeto a objetos hay que expresar cada atributo de las entidades como una variable y un par de mensajes del objeto correspondiente. La variable se utiliza para guardar el valor del atributo, uno de los mensajes se utiliza para leer el valor del atributo y el otro mensaje para actualizar este valor. Sin embargo, en aras de la sencillez, muchos modelos orientados a objetos permiten que las variables se lean o actualicen de manera directa, sin necesidad de definir los mensajes para ello.

Clase objetos.

Generalmente en una base de datos hay muchos objetos similares. Por similar se entiende que corresponden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y del mismo tipo. Por tanto, los objetos parecidos se agrupan para formar una clase. Cada uno de estos objetos se denomina ejemplar de su clase. Todos los objetos de una clase comparten una definición común, pese a que se diferencien en los valores asignados a las variables.

El concepto de clase del modelo orientado a objetos se corresponde con el concepto de entidad del modelo E-R.

Ejemplo de clase de objetos en seudocódigo.

```
class empleado{
        /*Variables*/
                string nombre;
                string dirección;
                date fecha-alta;
                int sueldo;
        /*Mensajes*/
                int sueldo-anual( );
                string obtener-nombre( );
                string obtener-dirección( );
                int antigüedad( );
}
```

Las clases son gráficamente representadas por cajas con compartimientos para: Nombre de la clase, atributos y operaciones, métodos.

Utilicemos el seudocódigo anterior para crear un diagrama de clases.

Empleado
- nombre: string - direccion: string - fecha-alta: date - sueldo: int
+ sueldo-anual ( ) : int + obtener-nombre ( ) : string + obtener-direccion ( ) : string + antigüedad ( ) : int

Tabla 2 Objeto Empleado

El concepto de clases es parecido al concepto de los tipos abstractos de datos. Sin embargo, hay varios aspectos adicionales en el concepto de case respecto al de tipo de datos abstractos. Para representar estas propiedades adicionales, cada clase trata como si fuera un objeto.

Una clase incluye:

- ✓ Un tipo de variable cuyo valor es el conjunto de todos los objetos que son ejemplares de la clase.
- ✓ La implementación de un método para el mensaje nuevo, que crea un nuevo ejemplar de la clase.

Herencia

Los esquemas de las bases de datos orientadas a objetos suelen necesitar gran número de clases. Frecuentemente, sin embargo, varias clases son parecidas entre sí.

Para permitir la relación directa de los parecidos entre clases hay que ubicarlas en una jerarquía de especializaciones. El concepto de jerarquía de especializaciones es parecido al de especialización del modelo entidad-relación.

En la siguiente figura se muestra un ejemplo de jerarquía de especializaciones para un banco.

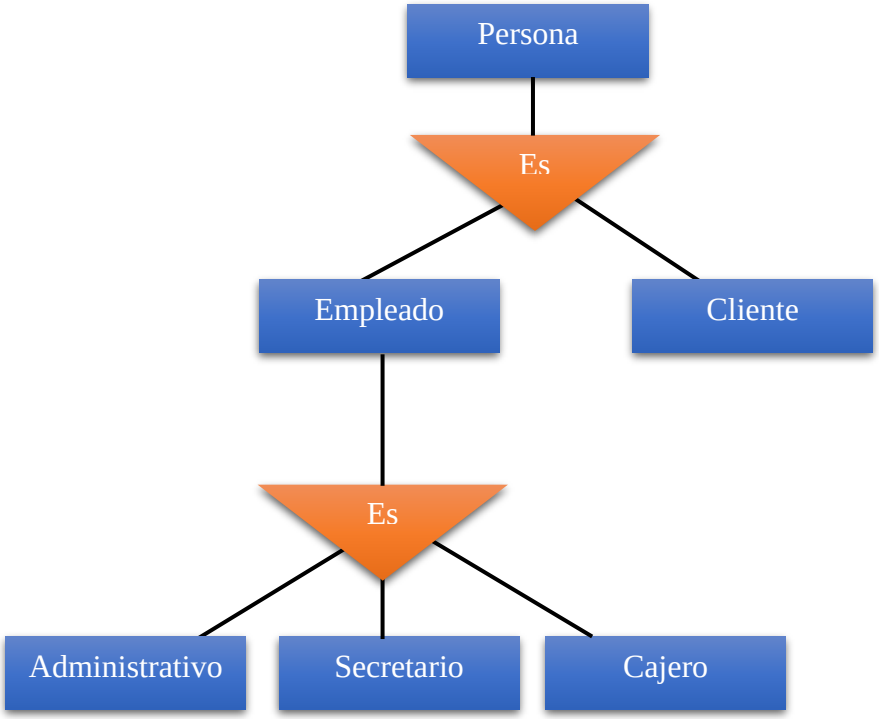

Ilustración 33 Jerarquía de Especialización de un banco

Los mensajes y los métodos se heredan de modo idéntico a las variables. Una ventaja importante en la herencia en los sistemas orientados a objetos es el concepto de posibilidad de sustitución: Cualquier método de una clase dada – por ejemplo, A (o una función que utilice un objeto de la clase A como argumento) puede ser llamado de igual modo con cualquier objeto perteneciente a cualquier subclase B de A. Esta

característica lleva a la reutilización de código, dado a que no hace falta escribir los mensajes, métodos y funciones para los objetos de la clase B.

Herencia múltiple

La herencia múltiple permite a las clases heredar variables y métodos de múltiples superclases. La relación entre clases y subclases se representa mediante un grafo acíclico dirigido (GAD) en el que las clases pueden tener más de una superclase.

Ejemplo.

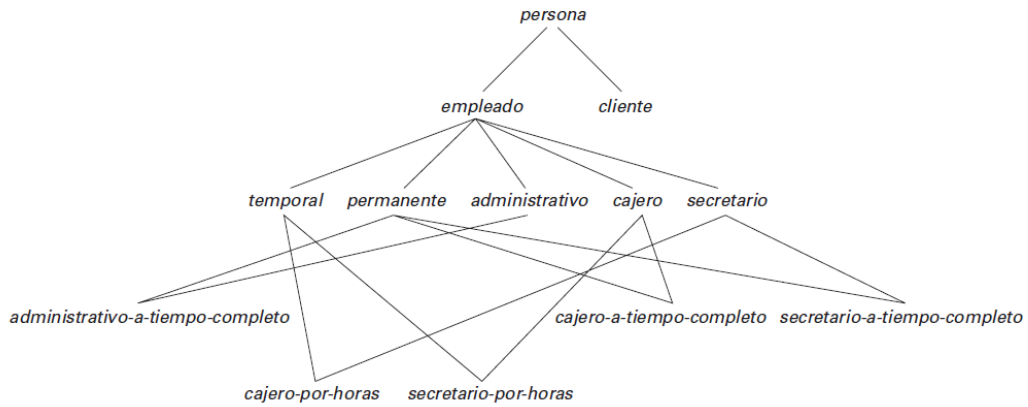

Ilustración 34 Relación entre clases

Identidad de los objetos

Los objetos de las bases de datos orientadas a objetos suelen corresponder a entidades del sistema modelado por la base de datos. Las entidades conservan su identidad, aunque algunas de sus propiedades cambien con el tiempo. De manera parecida, los objetos conservan su identidad, aunque los valores de las variables o las definiciones de los métodos cambien total o parcialmente con el tiempo. Este concepto de identidad no se aplica a las tuplas de las bases de datos relacionales.

La identidad de los objetos es el concepto más potente en el que suele hallarse en los lenguajes de programación o en los modelos de datos que no se basan en la programación orientada a objetos.

A continuación, se ilustran varios ejemplos de identidad.

- ✓ Valor: Se utiliza un valor de datos como identidad
- ✓ Nombre: Se utiliza como identidad un nombre proporcionado por el usuario.

- ✓ Incorporada: Se incluye el concepto de identidad en el modelo de datos o en el lenguaje de programación y no hace falta que el usuario proporcione ningún identificador.

Sistemas Gestores de Bases de Datos Orientadas a Objetos

Estos sistemas integran la tecnología de las bases de datos con el paradigma orientado a objetos que se ha desarrollado en el área de los lenguajes de programación y de la ingeniería de software. Esta tendencia en su mayor parte ha estado controlada por los desarrollos industriales, incluso cuando no hay aún un funcionamiento teórico consolidado para los lenguajes y modelos orientados a objetos.

En este apartado nos dedicaremos a exponer cuáles son las características que debe poseer un Sistema Gestor de Bases de Datos Orientadas a Objetos (OODBMS). Para ello, nos basaremos en "The Object-Oriented Database System Manifesto", propuesto por Malcolm Atkinson. En dicho manifiesto se hace una distinción de las características que debe poseer todo sistema orientado a objetos, dividiéndolas en tres grupos: aquellas que el sistema está obligado a cumplir para ser cualificado como un sistema de bases de datos orientado a objetos, aquellas que son opcionales y que pueden ser añadidas con el fin de hacer el sistema más eficiente y, por último, las opciones abiertas, en las cuales el diseñador puede hacer el número de cambios que desee. Detallaremos los tres grupos a continuación:

Características obligatorias: las reglas de Oro

Todo sistema de bases de datos orientado a objetos debe satisfacer dos criterios: ser un DBMS y ser un sistema orientado a objetos. El primero de los dos se reduce a cinco características: persistencia, gestión de almacenamiento secundario, concurrencia, recuperación y facilidad en la realización de consultas. El segundo se traduce en las siguientes características: objetos complejos, identificación de objetos, encapsulación, tipos o clases, herencia, ocultamiento combinado con ligadura tardía, extensibilidad y completitud computacional.

Características opcionales

- ✓ Herencia Múltiple.
- ✓ Comprobación de tipos e inferencia de tipos.
- ✓ Distribución.
- ✓ Diseño de transacciones.
- ✓ Versiones.

Opciones abiertas

Cualquier sistema que satisface las "reglas de oro", pude etiquetarse como un sistema de bases de datos orientado a objetos. A pesar de ello, quedan a la libre elección de los implementadores muchas opciones de diseño.

- ✓ Paradigma de programación.
- ✓ Sistema de representación.
- ✓ Sistema de tipos.
- ✓ Uniformidad.

Object Store

Es un sistema de gestión de bases de datos orientado a objetos que proporciona una interfaz de lenguaje estrechamente integrada a las características tradicionales de sistema de gestión de bases de datos de almacenamiento persistente, gestión de transacciones (control de concurrencia y recuperación), acceso a datos distribuidos y consultas asociativas. ObjectStore se diseñó para proporcionar una interfaz programática unificada tanto para los datos asignados de forma persistente (es decir, datos que van más allá de la ejecución de un programa de aplicación) como para los datos asignados de manera transitoria (es decir, datos que no sobreviven más allá de la ejecución de una aplicación): velocidad de acceso para datos persistentes generalmente igual a la de una referencia en memoria de un puntero a datos transitorios. (Siberschatz, Korth, & Sudarshan, 2013):

- ✓ Los diferentes objetos del mismo tipo pueden ser persistentes o transitorios dentro del mismo programa. Existen varias motivaciones para nuestro

objetivo de hacer que ObjectStore esté estrechamente integrado con el lenguaje de programación.

- ✓ Acceder a datos en el mismo formato en el que existe en la aplicación
- ✓ Cree y modifique objetos en lugar de tablas, columnas, filas y tuplas utilizando las interfaces ObjectStore C ++, Java o ActiveX.
- ✓ Con ObjectStore, no se necesita traducción ni copia. Los datos persistentes son como los datos ordinarios de transferencia de datos (transitorios): una vez que se obtiene un puntero, el usuario puede simplemente usarlo de la manera ordinaria. ObjectStore se encarga automáticamente del bloqueo y realiza un seguimiento de lo que se ha modificado.
- ✓ Las referencias a los objetos transitorios y persistentes son manejadas por las mismas secuencias de código máquina. Otras arquitecturas requieren referencias a objetos potencialmente persistentes para ser manejados en software, y esto es necesariamente más lento.
- ✓ Cómo es el almacenamiento ObjectStore
- ✓ El servidor es el proceso de ObjectStore que controla principalmente el almacenamiento de objetos. Con la ayuda del proceso del cliente y el proceso del administrador de caché, el servidor puede administrar bases de datos para múltiples aplicaciones cliente. Estas aplicaciones pueden estar en uno o varios hosts.

Lenguajes de creación y manipulación de objetos

Un lenguaje de programación es un lenguaje formal que es diseñado para realizar procesos que pueden ser llevados a cabo por computadoras. El lenguaje de programación puede usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos. También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos.

Aunque existen muchas clasificaciones, en general se puede distinguir entre dos clases de lenguajes: los Lenguajes Naturales (español, inglés, alemán, etc.), y los lenguajes artificiales o formales (matemático, lógico, computacional, etc.). Los lenguajes artificiales sí se aprenden de manera voluntaria y consciente.

Los programadores escriben instrucciones en diversos lenguajes de programación. Las computadoras pueden entender directamente alguno de ellos, pero otros requieren pasos de traducción intermedios. Hoy en día se utilizan cientos de lenguajes de computadora.

¿Qué es una BDOO?

Se trata de datos inteligente. Soporta el paradigma orientado a objetos almacenando datos y métodos, y no sólo datos. Está diseñada para ser eficaz desde el punto de vista físico y para almacenar objetos complejos. Es más segura ya que no permite tener acceso a los datos, debido a que para poder entrar se tiene que hacer por los métodos que haya implementado el programador. Las BDOO se desarrollan al describir en primer lugar los tipos de objetos importantes del dominio. Estos tipos de objetos determinan las clases que conformarán la definición de la BDOO.

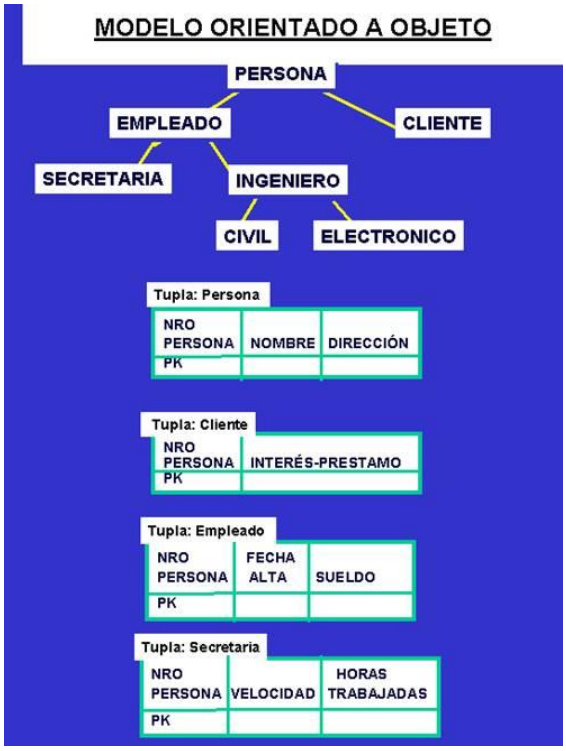

Ilustración 35 Modelo Orientado a Objeto (Martín-Palomino, 2005)

Ilustración 36 Componentes de la programación Orientada a Objetos

Ejemplos de Lenguajes de Creación

Lenguaje Máquina:

100001010101010 100100101010100 100011100101110

Lenguaje de Nivel Bajo (Ensamblador)

LOAD R1, (B) LOAD R2, (C) ADD R1, R2 STORE (A), R1

Lenguajes de Alto Nivel

A = B + C;

Las técnicas, vistas anteriormente en las que se basa la programación orientada a objetos (como el encapsulamiento, abstracción, etc) ya eran conocidas, desde hace ya varios años, sin embargo no todos los lenguajes proporcionan todas las facilidades para escribir programas orientados a objetos. Existen discrepancias de cuales deben de ser estas facilidades y se pueden agrupar en las siguientes:

- ✓ Manejo de memoria automático, incorporándose el concepto del recoletor de basura, con lo que la memoria utilizada por objetos cuya utilidad ha terminado es liberada por mecanismos propios del lenguaje, sin intervención del programador.
- ✓ Abstracción de datos a través del lenguaje.

- ✓ Estructura modular en objetos, tomando como base sus estructuras de datos.
- ✓ Clases, Herencia y polimorfismo que puedan ser manipuladas a través del lenguaje.

Entre los lenguajes orientados a objetos destacan los siguientes:

- ✓ Smalltalk
- ✓ Objective-C
- ✓ C++
- ✓ Ada 95
- ✓ Java
- ✓ Ocaml
- ✓ Python
- ✓ Delphi
- ✓ Lexico (en castellano)
- ✓ C#
- ✓ Eiffel
- ✓ Ruby
- ✓ ActionScript
- ✓ Visual Basic
- ✓ PHP
- ✓ PowerBuilder
- ✓ Clarion
- ✓ Simula67

Existen lenguajes como C++ y otros lenguajes, como OOCOBOL, OOLISP, OOPROLOG y Object REXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación ya existentes.

La sintaxis (lo equivalente a la gramática en un lenguaje natural) de un programa escrito en JAVA es casi, casi igual a uno escrito en el lenguaje de programación C++.

Generalmente las instrucciones terminan con un punto y coma; con las excepciones que maneja también en C/C++.

Los tipos de datos en JAVA son los siguientes tres:

- ✓ *Primitivos.*- Son unidades de información tales como caracteres, números o valores boléanos (lógicos).
- ✓ *Clases del sistema.*- No son clases y no tienen métodos propios.
- ✓ *Clases definidas por el usuario.*- Como su nombre lo indica son hechas por el usuario.

Los tipos primitivos en JAVA son (Hérnandez, 2002):

- ✓ *bolean.*- Los cuales pueden tener valores de falso o verdadero (trae/false).
- ✓ *char.*-Es un carácter de 16 bits.
- ✓ *byte, short.*- Cuenta con 8, 16,32 y 64 bits
- ✓ *int, long.*- Tiene valores entero y enteros largos.
- ✓ *Flota, double.*- Son números de punto flotante (notación científica) de 32 y 64 bits.

Dentro de la sintaxis de JAVA se encuentran los siguientes elementos:

- ✓ *Comentarios.*
- ✓ *Declaraciones.*
- ✓ *Bloques de código.*
- ✓ *Estructura de archivos fuentes.*
- ✓ *Identificadores.*
- ✓ *Palabras claves*
- ✓ *Literales.*
- ✓ *Expresiones y Operadores*

Conclusión

Los sistemas distribuidos abarcan una cantidad de aspectos considerables, sistemas operativos, comunicaciones, modelos de programación, etc., lo que hace que sus beneficios se pueden traducir en complejidades al momento de su implantación.

- ✓ Existen ciertos aspectos que requieren cuidado especial ya que pueden pasar de ser una ventaja a una desventaja, por ejemplo, el manejo de fallos, el control de la concurrencia, etc.

- ✓ Existen muchos temas de investigación relacionados con los sistemas distribuidos, en la sección de Desafíos se presentan algunos ejemplos.
- ✓ Es importante señalar que muchas tecnologías están en constante desarrollo y maduración, esto requiere de un estudio a profundidad de los factores que intervienen en cada aspecto de los sistemas distribuidos antes de apostar por alguna tecnología en especial.
- ✓ Es claro que la evolución constante en la tecnología sigue impulsando y estableciendo nuevos retos en el desarrollo de los sistemas distribuidos situación que se ve casi imposible de revertir

Tablas e Ilustraciones

Ilustración 1 Tipos de Fragmentación.....12
Ilustración 2 Características de las réplicas. (Siberschatz, Korth, & Sudarshan, 2013) 13
Ilustración 3 Transacción Distribuida. (Peter & Coronel, 2004).....14
Ilustración 4. Primera fase del protocolo de compromiso (2FC).....15
Ilustración 5 Segunda fase del protocolo de compromiso (2FC).....16
Ilustración 6 Pasos para el procesamiento de consultas.....18
Ilustración 7 Ejemplo de una BDD. (Peter & Coronel, 2004).....19
Ilustración 8 Memoria Compartida. (Peter & Coronel, 2004).....19
Ilustración 9 . Disco Compartido. (Peter & Coronel, 2004).....20
Ilustración 10 Almacenamiento con memoria propia. (Peter & Coronel, 2004).....20
Ilustración 11 Almacén de Datos.....24
Ilustración 12 Metodología para la creación de un almacén de datos 25
Ilustración 13 Esquema de definición de datos.....27
Ilustración 14 Metadatos de un archivo Mp3 28
Ilustración 15 Tipos de Metadatos.....28
Ilustración 16 Metadatos de Usuario Final 29
Ilustración 17 Arquitectura de un Data Warehouse.....30
Ilustración 18 Estructura de Data Warehouse de una Capa.....31
Ilustración 19 Segunda capa de una estructura Data Warehouse 31
Ilustración 20 Tercera capa de una estructura Data Warehouse.....32
Ilustración 21 Data marts.....33
Ilustración 22 Representación de XML de información bancaria. 39
Ilustración 23 . Uso de atributos 40
Ilustración 24 Representación de los atributos 41
Ilustración 25 Documento XML.....42
Ilustración 26 Esquema sobre integración de datos a XML y Base de Datos 48
Ilustración 27 Grafo de datos (Sosa, 2001).....49
Ilustración 28 Ejemplo de un DTD tipo banco 50
Ilustración 29 Ejemplo de un ID y IDREFS 52
Ilustración 30 XML DOCTYPE A UN XML SCHEMA.....53
Ilustración 31 Query's Languajes for XML.....56
Ilustración 32 XQL 99 y XQuery 57

Ilustración 33 Jerarquía de Especialización de un banco.....63
Ilustración 34 Relación entre clases.....64
Ilustración 35 Modelo Orientado a Objeto (Martín-Palomino, 2005).....68
Ilustración 36 Componentes de la programación Orientada a Objetos.....69

Referencias

Abelló, A., & Rollón, E. (2006). *Diseño y Administracion de Bases de Datos* . Barcelona: Edicions UPC .

Abraham Silberschatz, H. F. (2002). *Fundamentos de Bases de Datos.* McGraw-Hill.

ARZALUZ, M. D. (2015). *Bases de Datos Avanzadas.* Estado de México: UAEM.

Ayala, J. (2015). *Apuntes de Fundamentos de Bases de Datos.* Obtenido de http://ri.uaemex.mx/bitstream/handle/20.500.11799/33944/secme-19274.pdf?sequence

Baeza-Yates, A. D. (2002). Lenguajes de consulta para XML: un analisis comparativo. *El profesional de la Información*.

Bermúdez, J. C. (2014). *Diseño de elementos software con tecnologías basadas en componentes.* IC Editorial.

Camps Paré, R., Casillas Santillán, L. A., Costal Costa, D., Gibert Ginestà, M., Martín Escofet, C., & Pérez Mora, O. (2005). *Bases de Datos.* Barcelona: Eureca Media, SL.

Coronel, C., & Morris, S. (2011). *Bases de Datos* . Mexico, DF: CENGAGE Learning.

ELMASRI, N. .. (2001). *Fundamentos de Sistemas de Bases de Datos.*

Elmasri, R. A. (2007). *Fundamentos de Sistemas de Bases de Datos.* Madrid: Redes Informaticas.

Escuela de Educacion Tecnica No. 2. (2003). *Normalizacion de Base de Datos.* Buenos Aires.

Frade, D. O., & Castillo, J. N. (2007). Estado actual de las tecnologías de bodega de datos y OLAP aplicadas a bases de datos espaciales OLAP aplicadas a bases de datos espaciales. *Ingeniería e investigación*, 58-67.

Fuentes, L. (2010). *Incorporación de elementos de inteligencia de negocios.*

Hérnandez, L. A. (2002). Programación Orientada a Objetos. *Programación Orientada a Objetos*, 61.

López, C. P. (2008). *Mineria de datos: Técnicas y herramientas.*

Marques, M. (2011). *Bases de Datos* . Castelló de la Plana.

Martín-Palomino, P. (2005). Bases de datos Orientadas a Objetos. *Bases de datos Orientadas a Objetos*, 11.

Peter, R., & Coronel, C. (2004). *Sistemas de Bases de Datos: diseño, implementación y administración.* . México D.F.: Thomson, cop.

Raquel Abella, L. C. (1999). *Sistema Data Warehousing.*

Sarniento, J. C. (2011). *Construccion y poblamiento de un datawarehouse basado en paradigma de bases de datos objeto relacional.*

Senn, J. (1992). *Sistemas de Información.* México: Iberoaméricana.

Siberschatz, A., Korth, H., & Sudarshan, S. (2013). *Fundamentos de Bases de Datos.* España: McGraw-Hill .

Silberschatz, A., Korth, H., & Sudarshan, S. (2002). *Fundamentos de Bases de Datos.* Aravaca(Madrid): McGRAW-HILL/INTERAMERICANA DE ESPAÑA.

Silva, F. C. (2012). ARQUITECTURA DE UNA BODEGA. *SENA - Servicio Nacional de Aprendizaje*.

Sosa, D. V. (2001). *XML-Extensible Markup Languaje Introduction + Datos Semi-estructurados.* Tamaulipas: cinvestav.

Teran, D. M. (2014). *Administración Estratégica de la función informática.* S.A. MARCOMBO.

Universidad a Distancia de Madrid. (03 de julio de 2001). *Area de programacion y desarrolo*. Obtenido de Manual de XML: http://www.mundolinux.info/que-es-xml.htm

Universidad Autónoma del Estado de México. (2010). *Programa de Estudios por Competencias Bases de Datos Avanzadas.*