

UDS

ANTOLOGIA

BASE DE DATOS II

INGENIERÍA EN SISTEMAS COMPUTACIONALES

OCTAVO CUATRIMESTRE

Marco Estratégico de Referencia

ANTECEDENTES HISTORICOS

Nuestra Universidad tiene sus antecedentes de formación en el año de 1979 con el inicio de actividades de la normal de educadoras “Edgar Robledo Santiago”, que en su momento marcó un nuevo rumbo para la educación de Comitán y del estado de Chiapas. Nuestra escuela fue fundada por el Profesor de Primaria Manuel Albores Salazar con la idea de traer Educación a Comitán, ya que esto representaba una forma de apoyar a muchas familias de la región para que siguieran estudiando.

En el año 1984 inicia actividades el CBTiS Moctezuma Ilhuicamina, que fue el primer bachillerato tecnológico particular del estado de Chiapas, manteniendo con esto la visión en grande de traer Educación a nuestro municipio, esta institución fue creada para que la gente que trabajaba por la mañana tuviera la opción de estudiar por las tarde.

La Maestra Martha Ruth Alcázar Mellanes es la madre de los tres integrantes de la familia Albores Alcázar que se fueron integrando poco a poco a la escuela formada por su padre, el Profesor Manuel Albores Salazar; Víctor Manuel Albores Alcázar en septiembre de 1996 como chofer de transporte escolar, Karla Fabiola Albores Alcázar se integró como Profesora en 1998, Martha Patricia Albores Alcázar en el departamento de finanzas en 1999.

En el año 2002, Víctor Manuel Albores Alcázar formó el Grupo Educativo Albores Alcázar S.C. para darle un nuevo rumbo y sentido empresarial al negocio familiar y en el año 2004 funda la Universidad Del Sureste.

La formación de nuestra Universidad se da principalmente porque en Comitán y en toda la región no existía una verdadera oferta Educativa, por lo que se veía urgente la creación de una institución de Educación superior, pero que estuviera a la altura de las exigencias de los jóvenes que tenían intención de seguir estudiando o de los profesionistas para seguir preparándose a través de estudios de posgrado.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de cuarenta

alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el Corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y Educativos de los diferentes Campus, Sedes y Centros de Enlace Educativo, así como de crear los diferentes planes estratégicos de expansión de la marca a nivel nacional e internacional.

Nuestra Universidad inició sus actividades el 18 de agosto del 2004 en las instalaciones de la 4ª avenida oriente sur no. 24, con la licenciatura en Puericultura, contando con dos grupos de cuarenta alumnos cada uno. En el año 2005 nos trasladamos a nuestras propias instalaciones en la carretera Comitán – Tzimol km. 57 donde actualmente se encuentra el campus Comitán y el corporativo UDS, este último, es el encargado de estandarizar y controlar todos los procesos operativos y educativos de los diferentes campus, así como de crear los diferentes planes estratégicos de expansión de la marca.

MISIÓN

Satisfacer la necesidad de Educación que promueva el espíritu emprendedor, aplicando altos estándares de calidad Académica, que propicien el desarrollo de nuestros alumnos, Profesores, colaboradores y la sociedad, a través de la incorporación de tecnologías en el proceso de enseñanza-aprendizaje.

VISIÓN

Ser la mejor oferta académica en cada región de influencia, y a través de nuestra Plataforma Virtual tener una cobertura Global, con un crecimiento sostenible y las ofertas académicas innovadoras con pertinencia para la sociedad.

VALORES

- Disciplina
- Honestidad
- Equidad
- Libertad

ESCUDO



El escudo de la UDS, está constituido por tres líneas curvas que nacen de izquierda a derecha formando los escalones al éxito. En la parte superior está situado un cuadro motivo de la abstracción de la forma de un libro abierto.

ESLOGAN

“Mi Universidad”

ALBORES



Es nuestra mascota, un Jaguar. Su piel es negra y se distingue por ser líder, trabaja en equipo y obtiene lo que desea. El ímpetu, extremo valor y fortaleza son los rasgos que distinguen.

BASE DE DATOS II

Objetivo de la materia:

Capacitar al alumno en el conocimiento de los diferentes aspectos de control de concurrencia, definirá estrategias de recuperación adecuadas y diseñará esquemas de seguridad e integridad de bases de datos centralizadas y distribuidas.

UNIDAD I RECUPERACIÓN

- I.1 Concepto
- I.2. Transacciones
- I.3 Condiciones de terminación de una transacción
- I.4 Caracterización de transacciones
- I.5 Formalización del concepto de transacción
- I.6 Propiedades de las transacciones
- I.7 Tipos de transacciones
- I.8 Fallas de transacción
- I.9 Fallas de sistemas
- I.10 Falas en el medio
- I.11 DML (lenguaje de manipulación de datos) con un dbms comercial.

UNIDAD II CONCURRENCIA

- 2.1. Definición
- 2.2. problemas que se presentan (actualización, perdida, etc.)
- 2.3 Control de concurrencia en bases de datos
- 2.4 Bloqueos
 - 2.4.1 Bloqueo mortal
 - 2.4.2 Soluciones
 - 2.4.3 Bloqueo de dos fases (2pl)

- 2.4.4 Time-stamp
- 2.5 Seriabilidad
- 2.6 Dead lock
 - 2.6.1 Causas del deadlock
- 2.7 Técnicas para prevenirlo
- 2.8 Técnicas para deshacerlo
- 2.9 Recuperación

UNIDAD III SEGURIDAD

- 3.1. Concepto
- 3.2. Identidad y autenticación
- 3.3. Matriz de autorización
- 3.4. Definición de un esquema de seguridad
- 3.5. Mecanismos e vista para implantación de seguridad
- 3.6. Bases de datos estadísticas
- 3.7. Encriptamiento de datos
- 3.8. Seguridad empleando un DML con un DBMS comercial

UNIDAD IV BASES DE DATOS DISTRIBUIDAS

- 4.1. Introducción
- 4.2. Estructura de un sistema distribuido
- 4.3. Procesamiento de consulta
- 4.4. Propagación de actualizaciones
- 4.5. Control de concurrencia

Contenido

UNIDAD I RECUPERACIÓN

MANEJO DE TRANSACCIONES DISTRIBUIDAS 10

1.2. TRANSACCIONES..... 11

1.3 CONDICIONES DE TERMINACIÓN DE UNA TRANSACCIÓN 13

1.4 CARACTERIZACIÓN DE TRANSACCIONES..... 14

1.5 FORMALIZACIÓN DEL CONCEPTO DE TRANSACCIÓN 14

1.6 PROPIEDADES DE LAS TRANSACCIONES 16

1.7 TIPOS DE TRANSACCIONES..... 17

1.8 FALLAS DE TRANSACCIÓN 18

1.9 FALLAS DE SISTEMAS 18

1.10 FALAS EN EL MEDIO 18

1.11 DML (lenguaje de manipulación de datos) con un DBMS comercial. 19

UNIDAD II CONCURRENCIA 24

2.1. DEFINICIÓN..... 24

2.2. PROBLEMAS QUE SE PRESENTAN (ACTUALIZACIÓN, PERDIDA, ETC.) 25

2.3 CONTROL DE CONCURRENCIA EN BASES DE DATOS 29

2.4 BLOQUEOS 31

 2.4.1 BLOQUEO MORTAL 32

 2.4.2 SOLUCIONES 33

 2.4.3 BLOQUEO DE DOS FASES (2PL)..... 33

 2.4.4 TIME-STAMP..... 35

2.5 SERIABILIDAD..... 35

2.6 DEAD LOCK 36

 2.6.1 CAUSAS DEL DEADLOCK 37

2.7 TÉCNICAS PARA PREVENIRLO 38

2.8 TÉCNICAS PARA DESHACERLO..... 40

2.9 RECUPERACIÓN..... 42

UNIDAD III SEGURIDAD 43

3.1. CONCEPTO 43

3.2. IDENTIDAD Y AUTENTIFICACIÓN 48

3.3. MATRIZ DE AUTORIZACIÓN 53

3.4. DEFINICIÓN DE UN ESQUEMA DE SEGURIDAD 59

Migración 64

Monitoreo	66
3.5. MECANISMOS DE VISTA PARA IMPLANTACIÓN DE SEGURIDAD	68
3.6. BASES DE DATOS ESTADÍSTICAS	69
3.7. ENCRIPAMIENTO DE DATOS.....	72
3.8. SEGURIDAD EMPLEANDO UN DML CON UN DBMS COMERCIAL	77
UNIDAD IV BASES DE DATOS DISTRIBUIDAS	82
4.1. INTRODUCCIÓN.....	82
4.2. ESTRUCTURA DE UN SISTEMA DISTRIBUIDO.....	87
4.3. PROCESAMIENTO DE CONSULTA	93
4.4. PROPAGACIÓN DE ACTUALIZACIONES.....	94
4.5. CONTROL DE CONCURRENCIA.....	101
BIBLIOGRAFÍA BÁSICA Y COMPLEMENTARIA.....	109

UNIDAD I RECUPERACIÓN MANEJO DE TRANSACCIONES DISTRIBUIDAS

Hasta este momento, las primitivas básicas de acceso que se han considerado son las consultas (queries). Sin embargo, no se ha discutido qué pasa cuando, por ejemplo, dos consultas tratan de actualizar el mismo elemento de datos, o si ocurre una falla del sistema durante la ejecución de una consulta. Dada la naturaleza de una consulta, de lectura o actualización, a veces no se puede simplemente reactivar la ejecución de una consulta, puesto que algunos datos pueden haber sido modificados antes la falla. El no tomar en cuenta esos factores puede conducir a que la información en la base de datos contenga datos incorrectos. El concepto fundamental aquí es la noción de “ejecución consistente” o “procesamiento confiable” asociada con el concepto de una consulta. El concepto de una transacción es usado dentro del dominio de la base de datos como una unidad básica de cómputo consistente y confiable.

1.1 CONCEPTO

Una transacción es una colección de acciones que hacen transformaciones consistentes de los estados de un sistema preservando la consistencia del sistema. Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y supresiones de información. Por supuesto, se quiere asegurar que la base de datos nunca entra en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.

Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro

lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.



I.2. TRANSACCIONES

Una transacción es una colección de acciones que hacen transformaciones consistentes de los estados de un sistema preservando la consistencia del sistema. Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y supresiones de información. Por supuesto, se quiere asegurar que la base de datos nunca entra en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.

Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

Ejemplo I: Considere la siguiente consulta en SQL para incrementar el 10% del presupuesto del proyecto CAD/CAM de la base de datos de ejemplo.

UPDATE J

```
SET BUDGET = BUDGET*I.I
WHERE JNAME = "CAD/CAM"
```

Esta consulta puede ser especificada, usando la notación de SQL, como una transacción otorgándole un nombre:

```
Begin_transaction ACTUALIZA_PRESUPUESTO
begin
UPDATE J
SET BUDGET = BUDGET*I.I
WHERE JNAME = "CAD/CAM"
end.
```

Ejemplo 2: Considere una agencia de reservaciones para líneas aéreas con las siguientes relaciones:

```
FLIGHT( FNO, DATE, SRC, DEST, STSOLD, CAP )
CUST( CNAME, ADDR, BAL )
FC( FNO, DATE, CNAME, SPECIAL )
```

Una versión simplificada de una reservación típica puede ser implementada mediante la siguiente transacción:

```
Begin_transaction Reservación
begin
input( flight_no, date, customer_name );
EXEC SQL UPDATE FLIGHT
SET STSOLD = STSOLD + I
WHERE FNO = flight_no
AND DATE = date
EXEC SQL INSERT
```

```

INTO FC( FNAME, DATE, CNAME, SPECIAL )
VALUES (flight_no, date, customer_name, null )
output("reservación terminada");
end.

```

I.3 CONDICIONES DE TERMINACIÓN DE UNA TRANSACCIÓN

Una transacción siempre termina, aun en la presencia de fallas. Si una transacción termina de manera exitosa se dice que la transacción hace un commit (se usará el término en inglés cuando no exista un término en español que refleje con brevedad el sentido del término en inglés). Si la transacción se detiene sin terminar su tarea, se dice que la transacción aborta. Cuando la transacción es abortada, su ejecución es detenida y todas sus acciones ejecutadas hasta el momento son deshechas (undone) regresando a la base de datos al estado antes de su ejecución. A esta operación también se le conoce como rollback.

Ejemplo 3: Considerando de nuevo el Ejemplo 2, veamos el caso cuando no existen asientos disponibles para hacer la reservación.

```

Begin_transaction Reservación
begin
input( flight_no, date, customer_name );
EXEC SQL SELECT STSOLD, CAP
INTO temp1, temp2
FROM FLIGHT
WHERE FNO = flight_no AND DATE = date
if temp1 = temp2 then
output( "no hay asientos libres" )
Abort
else

```

```

EXEC SQL UPDATE FLIGHT
SET STSOLD = STSOLD + I
WHERE FNO = flight_no AND DATE = date
EXEC SQL INSERT
INTO FC( FNAME, DATE, CNAME, SPECIAL )
VALUES (flight_no, date, customer_name, null )
Commit
output("reservación terminada");
endif
end.

```

I.4 CARACTERIZACIÓN DE TRANSACCIONES

Observe que en el ejemplo anterior las transacciones leen y escriben datos. Estas acciones se utilizan como base para caracterizar a las transacciones. Los elementos de datos que lee una transacción se dice que constituyen el conjunto de lectura (RS). Similarmente, los elementos de datos que una transacción escribe se les denomina el conjunto de escritura (WS). Note que los conjuntos de lectura y escritura no tienen que ser necesariamente disjuntos. La unión de ambos conjuntos se le conoce como el conjunto base de la transacción ($BS = RS \cup WS$).

I.5 FORMALIZACIÓN DEL CONCEPTO DE TRANSACCIÓN

Sea $O_{ij}(x)$ una operación O_j de la transacción T_i la cual trabaja sobre alguna entidad x . $O_j \in \{\text{read, write}\}$ y O_j es una operación atómica, esto es, se ejecuta como una unidad indivisible. Se denota por $OS_i = \bigcup_j O_{ij}$ al conjunto de todas las operaciones de la transacción T_i . También, se denota por N_i la condición de terminación para T_i , donde, $N_i \in \{\text{abort, commit}\}$.

La transacción T_i es un orden parcial, $T_i = \{S_i, <_i\}$, donde

$$I. S_i = OS_i \cup \{N_i\}$$

2. Para cualesquiera dos operaciones, $O_{ij}, O_{ik} \in OS_i$, si $O_{ij} = R(x)$ y $O_{ik} = W(x)$ para cualquier elemento de datos x , entonces, ó $O_{ij} <_i O_{ik}$ ó $O_{ik} <_i O_{ij}$
3. « $O_{ij} \in OS_i, O_{ij} <_i N_i$

Ejemplo 5. Considere una transacción simple T que consiste de los siguientes pasos:

Read(x)

Read(y)

$x \leftarrow x + y$

Write(x)

Commit

La especificación de su transacción de acuerdo con la notación formal que se ha introducido es la siguiente:

$$\hat{a} = \{ R(x), R(y), W(x), C \}$$

$$<_i = \{ (R(x), W(x)), (R(y), W(x)), (W(x), C), (R(x), C), (R(y), C) \}$$

Note que la relación de ordenamiento especifica el orden relativo de todas las operaciones con respecto a la condición de terminación. Esto se debe a la tercera condición de la definición de transacción. También note que no se define el ordenamiento entre cualquier par de operaciones, esto es, debido a que se ha definido un orden parcial.

I.6 PROPIEDADES DE LAS TRANSACCIONES

La discusión en la sección previa clarifica el concepto de transacción. Sin embargo, aún no se ha dado ninguna justificación para afirmar que las transacciones son unidades de procesamiento consistentes y confiables. Las propiedades de una transacción son las siguientes:

1. **Atomicidad.** Se refiere al hecho de que una transacción se trata como una unidad de operación. Por lo tanto, o todas las acciones de la transacción se realizan o ninguna de ellas se lleva a cabo. La atomicidad requiere que, si una transacción se interrumpe por una falla, sus resultados parciales deben ser deshechos. La actividad referente a preservar la atomicidad de transacciones en presencia de abortos debido a errores de entrada, sobrecarga del sistema o interbloqueos se le llama recuperación de transacciones. La actividad de asegurar la atomicidad en presencia de caídas del sistema se le llama recuperación de caídas.
2. **Consistencia.** La consistencia de una transacción es simplemente su correctitud. En otras palabras, una transacción es un programa correcto que lleva la base de datos de un estado consistente a otro con la misma característica. Debido a esto, las transacciones no violan las restricciones de integridad de una base de datos.
3. **Aislamiento.** Una transacción en ejecución no puede revelar sus resultados a otras transacciones concurrentes antes de su commit. Más aún, si varias transacciones se ejecutan concurrentemente, los resultados deben ser los mismos que si ellas se hubieran ejecutado de manera secuencial (seriabilidad).
4. **Durabilidad.** Es la propiedad de las transacciones que asegura que una vez que una transacción hace su commit, sus resultados son permanentes y no pueden ser borrados de la base de datos. Por lo tanto, los DBMS aseguran que los resultados de una transacción sobrevivirán a fallas del sistema. Esta propiedad motiva el aspecto de recuperación de bases de datos, el cual trata sobre como recuperar la base de datos a un estado consistente en donde todas las acciones que han hecho un commit queden reflejadas.

I.7 TIPOS DE TRANSACCIONES

Las transacciones pueden pertenecer a varias clases. Aun cuando los problemas fundamentales son los mismos para las diferentes clases, los algoritmos y técnicas que se usan para tratarlas pueden ser considerablemente diferentes. Las transacciones pueden ser agrupadas a lo largo de las siguientes dimensiones:

1. **Áreas de aplicación.** En primer lugar, las transacciones se pueden ejecutar en aplicaciones no distribuidas. Las transacciones que operan en datos distribuidos se les conoce como transacciones distribuidas. Por otro lado, dado que los resultados de una transacción que realiza un commit son durables, la única forma de deshacer los efectos de una transacción con commit es mediante otra transacción. A este tipo de transacciones se les conoce como transacciones compensatorias. Finalmente, en ambientes heterogéneos se presentan transacciones heterogéneas sobre los datos.
2. **Tiempo de duración.** Tomando en cuenta el tiempo que transcurre desde que se inicia una transacción hasta que se realiza un commit o se aborta, las transacciones pueden ser de tipo batch o en línea. Estas se pueden diferenciar también como transacciones de corta y larga vida. Las transacciones en línea se caracterizan por tiempos de respuesta muy cortos y por acceder una porción relativamente pequeña de la base de datos. Por otro lado, las transacciones de tipo batch toman tiempos relativamente largos y acceden grandes porciones de la base de datos.
3. **Estructura.** Considerando la estructura que puede tener una transacción se examinan dos aspectos: si una transacción puede contener a su vez subtransacciones o el orden de las acciones de lectura y escritura dentro de una transacción.

I.8 FALLAS DE TRANSACCIÓN

El sistema debe estar preparado para recuperarse no sólo de fallas puramente locales, como la aparición de una condición de desborde dentro de una transacción, sino también de fallas globales, como podría ser la interrupción del suministro eléctrico al CPU. Las fallas locales son las que afectan sólo a la transacción en donde ocurrió. Por el contrario, las fallas globales, afectan a varias -y casi siempre a todas- las transacciones que se estaban efectuando en el momento de la falla, por lo cual tienen implicaciones importantes en el sistema.

I.9 FALLAS DE SISTEMAS

Por ejemplo, interrupción del servicio eléctrico, estas afectan a todas las transacciones que se estaban ejecutando pero no afectan a la base de datos. Las fallas de sistema se conocen también como caídas (crash) suaves. El problema aquí es que se pierda el contenido de memoria principal, en particular, las áreas de almacenamiento temporal o buffers. Si esto ocurre, no se conocerá el estado preciso de la transacción que se estaba ejecutando en el momento de la falla, esta transacción jamás se podrá completar con éxito por lo que será preciso anularla cuando se reinicie el sistema.

Además, puede ocurrir que sea necesario volver a ejecutar algunas transacciones que sí se realizaron con éxito antes de la falla pero cuyas modificaciones no lograron efectuarse sobre la base de datos porque no lograron ser transferidas de los buffers de la base de datos a la base de Datos física (en disco).

I.10 FALAS EN EL MEDIO

Una falla de los medios de almacenamiento es un percance en el cual se destruye físicamente alguna porción de la DB. La recuperación de una falla semejante implica en esencia cargar

de nuevo la DB a partir de una copia de respaldo y utilizar después la bitácora para realizar de nuevo todas las transacciones terminadas desde que se hizo esa copia de respaldo. No hay necesidad de anular las transacciones inconclusas en el momento de la falla, porque por definición todas las modificaciones de esas transacciones ya se anularon de todas maneras.

La parte de restauración de la utilería servirá entonces para recrear la DB después de una falla de los medios de almacenamiento a partir de una copia de respaldo especificada. Por ejemplo una falla en el controlador de disco o un aterrizaje de cabeza en el disco, estas fallas sí causan daños a la base de datos o a una porción de ella y afecta, al menos, a las transacciones que están haciendo uso de esa porción. Las fallas de los medios de almacenamiento se llaman caídas duras. La Recuperación de una falla semejante implica, en esencia, cargar de nuevo la base de datos a partir de una copia de respaldo (database backup) y después utilizar la bitácora, o system log, para realizar de nuevo todas las transacciones terminadas desde que se hizo esa copia para respaldo. No hay necesidad de anular todas las transacciones inconclusas en el momento de la falla, porque por definición esas transacciones ya se anularon (se destruyeron) de todas maneras.

1.11 DML (lenguaje de manipulación de datos) con un DBMS comercial.

Un sistema de gestión de base de datos consta de varios componentes, todos los cuales contribuyen al buen funcionamiento del software. Los elementos básicos que lo conforman son tres: el diccionario de datos, el lenguaje de definición de datos y el lenguaje de manipulación de datos.

- **Diccionario de datos:** consiste en una lista de metadatos que reflejan las características de los diversos tipos de datos incluidos en la base de datos. Además, estos metadatos informan sobre los permisos de uso de cada registro y su representación física. De esta manera, el diccionario proporciona toda la información relevante sobre los datos almacenados.

- **Lenguaje de definición de datos:** el lenguaje de definición de datos, también llamado lenguaje de base de datos o DDL (data definition language), sirve para estructurar el contenido de la base de datos. Gracias a este lenguaje, es posible crear, modificar y eliminar objetos individuales, como referencias, relaciones o derechos de usuario.
- **Lenguaje de manipulación de datos:** mediante el lenguaje de manipulación de datos o DML (data manipulation language), se pueden introducir nuevos registros en la base de datos, así como eliminar, modificar y consultar los que ya contiene. Este lenguaje también permite comprimir y extraer los datos.

Tareas, funciones y propiedades del sistema gestor de base de datos

El sistema de gestión de base de datos es el componente más importante de un sistema de base de datos. Sin él, no sería posible administrar, controlar o supervisar la base de datos. Este software también es responsable de gestionar todos sus permisos de lectura y escritura. Un término que suele utilizarse mucho para resumir las funciones y propiedades de las transacciones de los sistemas gestores de base de datos es **ACID**, siglas de los términos en inglés *atomicity, consistency, isolation* y *durability* (es decir, atomicidad, consistencia, aislamiento y permanencia). Estos cuatro conceptos engloban los requisitos más importantes de un SGBD:

- La atomicidad o **integridad** describe la propiedad de “todo o nada” de los SGBD, por la que todas las fases de una transacción deben finalizarse por completo y en el orden correcto para que esta sea válida.
- La **consistencia** implica que las transacciones completadas no afecten la estabilidad de la base de datos, lo que requiere supervisarlas constantemente.
- El **aislamiento** es la propiedad que asegura que las transacciones no obstaculicen a las demás, de lo que, por lo general, se encargan algunas funciones de bloqueo.
- La **permanencia** implica que todos los datos queden almacenados permanentemente en el SGBD, no solo después de una transacción correcta, sino también o especialmente en caso de error o caída del sistema. Los registros de las

transacciones, donde quedan anotados todos los procesos del SGBD, son fundamentales para garantizar la permanencia.

En la siguiente tabla, te mostramos una clasificación distinta de las funciones y propiedades de los sistemas gestores de base de datos, que va más allá del modelo ACID.

¿Qué tipos de SGBD existen?

El objetivo de instalar un sistema gestor de base de datos es administrar los registros de la mejor manera posible. Como ya hemos mencionado, existen varios modelos para ello, que difieren básicamente en la manera en que **se estructuran los datos**. Por lo tanto, decidirse por un DBMS siempre implica decantarse por un **modelo de base de datos** concreto. Existen los siguientes modelos de bases de datos:

- Relacional
- Jerárquica
- De red
- Orientada a objetos
- Orientada a documentos

El más común y popular es el modelo de base de datos relacional, en el que los datos se estructuran en **filas de tabla**. La ventaja de este modelo radica en la posibilidad de crear diferentes relaciones entre las filas y presentarlas en columnas. El procedimiento es diferente al del **modelo de base de datos jerárquico**, donde los diferentes datos se organizan en relaciones padre-hijo, en una **estructura similar a la de un árbol**.

Otros enfoques para organizar los datos son el **modelo de base de datos de red**, donde los datos, como el nombre indica, se estructuran en forma de red, o el modelo de bases de datos orientada a objetos, en el que no solo importa la relación entre los registros de datos, sino también el concepto de la **herencia**: esto significa que los objetos pueden transferir algunos de sus atributos a otros objetos, lo que se regula a través del SGBD.

Por su parte, el modelo de base de datos orientado a documentos permite almacenar los registros de datos en diferentes documentos.

Sistema gestor de base de datos: resumen de ventajas e inconvenientes

Los SGBD, el pilar de todas las bases de datos, presentan varias ventajas y puntos fuertes, aunque, como cualquier otro software, también tienen algunos inconvenientes, como puedes ver en la siguiente lista:

Ventajas de los sistemas gestores de base de datos:

- Gestión fácil de grandes conjuntos de datos
- Acceso sencillo y eficaz a los datos almacenados
- Gran flexibilidad
- Integridad y consistencia de los datos
- Control de acceso del usuario (seguridad y protección de datos)
- Alta disponibilidad

Inconvenientes de los sistemas gestores de base de datos:

- Inversión inicial relativamente elevada (incluidos costes de hardware adicionales)
- Bastante menos eficaz para el software especial
- Se requieren empleados cualificados (administradores de bases de datos)

- Mayor vulnerabilidad por el hecho de centralizar los datos

Ejemplos de SGBD: sistemas más populares

De entre los numerosos sistemas gestores de bases de datos que existen, estos son los 15 más populares y utilizados:

- Microsoft Access (relacional)
- Microsoft SQL Server (relacional)
- MySQL (relacional)
- Oracle Database (relacional)
- OrientDB (orientado a documentos)
- CouchDB (orientado a documentos)
- Db2 de IBM (relacional)
- IMS de IBM (jerárquico)
- IBM Informix (relacional)
- MariaDB (relacional)
- Sybase ASE (relacional)
- MongoDB (orientado a documentos)
- PostgreSQL (combina relacional y orientado a objetos)
- Firebird (relacional)
- InterSystems Caché (combina relacional y orientado a objetos)
- InterSystems IRIS (combina relacional y orientado a objetos)

UNIDAD II CONCURRENCIA

2.1. DEFINICIÓN

Concurrencia se refiere al hecho de que los Sistemas Administradores de Base de Datos permiten que muchas transacciones accedan a una misma Base de Datos a la vez.

Cuando existen varios usuarios intentando modificar los datos al mismo tiempo, se necesita establecer algún tipo de control para que dichas modificaciones de un usuario no interfieran en las de los otros, a este sistema se le denomina control de concurrencia.

En este informe podremos ver algunos de los problemas que se presentan cuando la concurrencia no se controla y algunos de los mecanismos de bloqueo que nos permiten manejar la concurrencia en las transacciones. De esta manera, los sistemas de control de concurrencia deben garantizar la consistencia de transacciones que se ejecutan de manera concurrente.

En el campo informático, el termino concurrencia se refiere a la capacidad de los Sistemas de Administración de Base de Datos, de permitir que múltiples procesos sean ejecutados al mismo tiempo, y que también puedan interactuar entre sí.

Los procesos concurrentes pueden ser ejecutados realmente de forma simultánea, sólo cuando cada uno es ejecutado en diferentes procesadores. En cambio, la concurrencia es simulada si sólo existe un procesador encargado de ejecutar todos los procesos, simulando la concurrencia, ocupándose de forma alternada de uno y otro proceso a muy pequeños intervalos de tiempo. De esta manera simula que se están ejecutando a la vez.

Algunos casos de concurrencia, pueden ser:

- La multiprogramación, ya que el tiempo del procesador es compartido dinámicamente por varios procesos.
- Las aplicaciones estructuradas, donde la programación estructurada se implementa como un conjunto de procesos concurrentes.

- También se tiene que la misma estructura recién mencionada es utilizada en el diseño de los sistemas operativos, los cuales se implementan como un conjunto de procesos.

Debido a que los procesos concurrentes en un sistema pueden interactuar entre otros también en ejecución, el número de caminos de ejecución puede ser extremadamente grande, resultando en un comportamiento sumamente complejo. Las dificultades asociadas a la concurrencia han sido pensadas para el desarrollo de lenguajes de programación y conceptos que permitan hacer la concurrencia más manejable.

2.2. PROBLEMAS QUE SE PRESENTAN (ACTUALIZACIÓN, PERDIDA, ETC.)

Existen tres formas en las que una transacción, aunque sea correcta por sí misma, puede producir una respuesta incorrecta si alguna otra transacción interfiere con ella en alguna forma.

Consideremos que la transacción que interfiere también puede ser correcta; lo que produce el resultado incorrecto general es el intercalado sin control entre las operaciones de las dos transacciones correctas.

Los tres problemas son:

- El problema de la Actualización Perdida
- El problema de la Dependencia No Confirmada
- El problema del Análisis Inconsistente

Veamos algunos ejemplos para entender cómo es que los programas pueden interferir con otros. Tomaremos como ejemplo las transacciones de las cuentas de un banco, supongamos que tenemos un programa llamado Depositar, el cual deposita dinero en una cuenta.

Procedure Depositar(Cuenta, Monto)**begin**

```

    Start;
    temp := Leer(Cuentas[Cuenta]);
    temp := temp + Monto;
    Escribir(Cuentas[Cuenta],temp);
    Commit;

```

end

Supongamos que la cuenta **7** tiene un saldo de **\$1000** y que el cliente **1** deposita **\$100** en la cuenta **7** en el mismo instante en el que el cliente **2** deposita **\$100000** en la cuenta **7**. Cada uno de los clientes llama al procedimiento Depositar de tal manera que se crea una transacción para realizar su actualización. La ejecución concurrente de estos depósitos produce una secuencia de lecturas y escrituras en la base de datos, tales como:

```

Leer1(Cuentas[7])   devuelve el valor de $1000
Leer2(Cuentas[7])   devuelve el valor de $1000
Escribir2(Cuentas[7], $101000)
Commit2
Escribir1(Cuentas[7], $1100)
Commit1

```

El resultado de esta ejecución es que la Cuenta[7] contiene **\$1100**. A pesar que el depósito del cliente **2** concluyó satisfactoriamente, la interferencia con la ejecución de Depósito del cliente **1** causó que el depósito del cliente **2** se pierda. Este fenómeno de **actualización perdida** ocurre cuando dos transacciones, mientras intentan modificar un dato, ambas leen el valor antiguo del elemento antes que ninguna haya modificado su valor.

Otro problema del control de concurrencia se ilustra con el siguiente programa, llamado ImprimirSuma, el cual imprime la suma de los saldos de dos cuentas.

Procedure ImprimirSuma(Cuenta1, Cuenta2)

begin

```

    Start;
    temp1 := Leer(Cuentas[Cuenta1]);
    output(temp1);
    temp2 := Leer(Cuentas[Cuenta2]);
    output(temp2);
    temp1 := temp1 $$ temp2;
    output(temp1);
    Commit;

```

end

Supongamos que las cuentas **8** y **9** tiene un saldo de **\$200** cada una, y que el cliente **3** imprime los saldos de las cuentas **8** y **9** (utilizando ImprimirSuma) en el mismo instante en el que el cliente **4** transfiere **\$100** de la cuenta **8** a la cuenta **9** (utilizando Transferir). La ejecución concurrente de estas dos transacciones puede originar la siguiente ejecución de operaciones de la base de datos.

Leer4(Cuentas[8]) devuelve el valor de **\$200**

Escribir4(Cuentas[8], **\$100**)

Leer3 (Cuentas[8]) devuelve el valor de **\$100**

Leer3 (Cuentas[9]) devuelve el valor de **\$200**

Leer4 (Cuentas[9]) devuelve el valor de **\$200**

Escribir4 (Cuentas[9], **\$300**)

Commit4

Commit3

El procedimiento Transferir interfiere con ImprimirSuma en esta ejecución, causando que ImprimirSuma imprima el valor de **\$300**, la cual no es la suma correcta de los saldos de las cuentas **8** y **9**. El procedimiento ImprimirSuma no capturó los **\$100** en tránsito de la cuenta **8** a la cuenta **9**. Es importante recalcar que, a pesar de la interferencia, Transferir todavía asigna los valores correctos en la base de datos.

Este tipo de interferencia se denomina **análisis inconsistente** que ocurre cuando una transacción lee un dato antes que otra transacción lo actualice y lea otro dato después que la misma transacción lo ha actualizado. Es decir, la extracción (lectura) de los datos sólo percibe algunos de los resultados de la transacción de actualización.

Para conseguir el objetivo del control de concurrencia se utiliza un sincronizador. Un sincronizador es un programa (o una colección de ellos) que controla la ejecución concurrente de las transacciones; su función radica en ordenar las operaciones que forman parte de las transacciones que se requieren ejecutar concurrentemente, de tal manera que la ejecución resultante sea correcta. Usando tres operaciones básicas (ejecución, rechazo y retraso de una operación) el sincronizador puede controlar el orden en el cual las operaciones son ejecutadas por el Administrador de Datos. Cuando éste recibe una operación de la transacción (mediante el TM), usualmente trata de pasarla al DM inmediatamente, si no produce alguna ejecución incorrecta. Si el sincronizador decide que la ejecución de la operación puede producir resultados incorrectos, puede o retrasarla (en caso de que pueda procesar correctamente la operación más adelante) o rechazarla (si no es posible procesarla en el futuro de tal manera que produzca resultados correctos).

Por ejemplo, retomemos la ejecución concurrente de dos transacciones Depositar, que depositan **\$100** y **\$100,000** en la cuenta **7**

Leer1 (Cuentas[7]) devuelve el valor de **\$1000**

Leer2 (Cuentas[7]) devuelve el valor de **\$1000**

Escribir2 (Cuentas[7], **\$101000**)

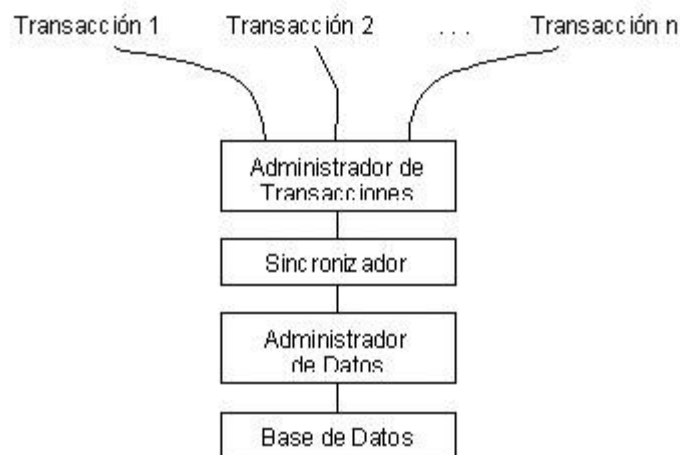
Commit2

Escribir1 (Cuentas[7], **\$1100**)

Commit1

Para evitar esta ejecución incorrecta, un sincronizador debe decidir rechazar Escribir1 provocando que la transacción T1 sea cancelada. En este caso, el usuario o el Administrador de Transacciones puede reenviar T1, la cual ahora se puede ejecutar sin interferir con T2. Como otra alternativa, el sincronizador puede prevenir la ejecución anterior retrasando Leer2 hasta que reciba y procese Escribir1 y de esta forma evitando que se rechace Escribir1 más adelante.

Por lo tanto, un DBMS que controla la ejecución concurrente de sus transacciones tiene una estructura como la siguiente :



2.3 CONTROL DE CONCURRENCIA EN BASES DE DATOS

En los ejemplos anteriores, los errores fueron causados por la ejecución intercalada de operaciones de transacciones diferentes. Los ejemplos no muestran todas las posibles formas en las que la ejecución de dos transacciones pueden interferir, pero sí ilustran dos de los problemas que surgen con frecuencia debido a la intercalación. Para evitar estos problemas, se deben controlar las intercalaciones entre transacciones.

El control de transacciones concurrentes en una base de datos brinda un eficiente desempeño del Sistema de Administración de Base de Datos, puesto que permite controlar la ejecución de transacciones que operan en paralelo, accediendo a información compartida y, por lo tanto, interfiriendo potencialmente unas con otras.

El objetivo de los métodos de control de concurrencia es garantizar la no inferencia o la propiedad de aislamiento de transacciones que se ejecutan de manera concurrente. Los distintos objetivos atacan el problema garantizando que las transacciones se ejecuten en un plan que sea serializable, es decir, que el resultado sea equivalente a el resultante de ejecutar un plan en serie.

El criterio de clasificación más común de los algoritmos de control de concurrencia es el tipo de primitiva de sincronización. Esto resulta en dos clases: aquellos algoritmos que están basados en acceso mutuamente exclusivo a datos compartidos (bloqueos) y aquellos que intentan ordenar la ejecución de las transacciones de acuerdo a un conjunto de reglas (protocolos). Sin embargo, esas primitivas se pueden usar en algoritmos con dos puntos de vista diferentes: el punto de vista pesimista que considera que muchas transacciones tienen conflictos con otras, o el punto de vista optimista que supone que no se presentan muchos conflictos entre transacciones.

Los algoritmos pesimistas sincronizan la ejecución concurrente de las transacciones en su etapa inicial de su ciclo de ejecución. Los algoritmos optimistas retrasan la sincronización de las transacciones hasta su terminación. Ambos grupos de métodos, pesimistas y

optimistas, consisten de algoritmos basados en bloqueos y algoritmos basados en marcas de tiempo, entre otros.

Los protocolos basados en bloqueos son los más utilizados por los DBMS comerciales. Los demás tienen un alcance más teórico que práctico.

2.4 BLOQUEOS

Un bloqueo en general es cuando una acción que debe ser realizada está esperando a un evento. Para manejar los bloqueos hay distintos acercamientos: prevención, detección, y recuperación. También es necesario considerar factores como que hay sistemas en los que permitir un bloqueo es inaceptable y catastrófico, y sistemas en los que la detección del bloqueo es demasiado costosa.

En el caso específico de las bases de datos distribuidas usar bloqueo de recursos, peticiones para probar, establecer o liberar bloqueos requiere mensajes entre los manejadores de transacciones y el calendarizador. Para esto existen dos formas básicas:

- **Autónoma:** cada nodo es responsable por sus propios bloqueos de recursos.
 - Una transacción sobre un elemento con n replicas requiere $5n$ mensajes
 - Petición del recurso
 - Aprobación de la petición
 - Mensaje de la transacción
 - Reconocimientos de transacción exitosa
 - Peticiones de liberación de recursos

- **Copia Primaria:** un nodo primario es responsable para todos los bloqueos de recursos
 - Una transacción sobre un elemento con n copias requiere n mensajes
 - Una petición del recurso

- Una aprobación de la petición
- n mensajes de la transacción
- n reconocimientos de transacción exitosa
- Una petición de liberación de recurso

Podemos definir que dos operaciones entran en conflicto que debe ser resuelto si ambas acceden a la misma data, y una de ellas es de escritura y si fueron realizadas por transacciones distintas.

2.4.1 BLOQUEO MORTAL

En sistemas operativos, el bloqueo mutuo (también conocido como interbloqueo, traba mortal, deadlock, abrazo mortal) es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. A diferencia de otros problemas de concurrencia de procesos, no existe una solución general para los interbloqueos.

Todos los interbloqueos surgen de necesidades que no pueden ser satisfechas, por parte de dos o más procesos. En la vida real, un ejemplo puede ser el de dos niños que intentan jugar al arco y flecha, uno toma el arco, el otro la flecha. Ninguno puede jugar hasta que alguno libere lo que tomó.

En el siguiente ejemplo, dos procesos compiten por dos recursos que necesitan para funcionar, que sólo pueden ser utilizados por un proceso a la vez. El primer proceso obtiene el permiso de utilizar uno de los recursos (adquiere el lock sobre ese recurso). El segundo proceso toma el lock del otro recurso, y luego intenta utilizar el recurso ya utilizado por el primer proceso, por lo tanto, queda en espera. Cuando el primer proceso a su vez intenta

utilizar el otro recurso, se produce un interbloqueo, donde los dos procesos esperan la liberación del recurso que utiliza el otro proceso.

2.4.2 SOLUCIONES

El control de concurrencia y detección y manejo de bloqueos es un área de mucho estudio en las bases de datos distribuidas, a pesar de esto no hay ningún algoritmo aceptado para solucionar el problema. Esto se debe a varios factores de los cuales se consideran a los siguientes tres los más determinantes:

- La data puede estar duplicada en un BDD, por tanto, el manejador de la BDD es responsable de localizar y actualizar la data duplicada.
- Si un nodo falla o la comunicación con un nodo falla mientras se realiza una actualización, el manejador debe asegurarse de que los efectos se reflejen una vez el nodo se recupere del fallo.
- La sincronización de transacciones en sitios o nodos múltiples es difícil ya que los nodos no pueden obtener información inmediata de las acciones realizadas en otros nodos concurrentemente.

Para el control de bloqueos mutuos no se ha desarrollado ninguna solución viable y la forma más simple y que la mayoría de productos utilizan es la implementación de un tiempo máximo de espera en las peticiones de bloqueos.

Causa de estas dificultades más de 20 algoritmos de control de concurrencia se han propuesto en el pasado, y aun así siguen apareciendo nuevos. Una revisión bibliográfica muestra que la mayoría de los algoritmos son variantes del 2PL (2-phase locking o bloqueo de dos fases) o el algoritmo de time-stamp.

2.4.3 BLOQUEO DE DOS FASES (2PL)

El algoritmo 2PL utiliza bloqueos de lectura y escritura para prevenir conflictos entre operaciones. Consiste en los siguientes pasos para una transacción T:

- Obtiene bloqueo de lectura para un elemento L (bloqueo compartido)
- Obtiene bloqueo de escritura para un elemento E (bloqueo exclusivo)
- Lee el elemento L
- Escribe en el elemento E
- Libera el bloqueo de L
- Libera el bloqueo de E

Las reglas básicas para manejar los bloqueos son: transacciones distintas no pueden tener acceso simultáneamente a un elemento (lectura-escritura o escritura-escritura), y una vez se libere un bloqueo no se puede pedir otro, es decir, los bloqueos de la transacción crecerán mientras no libere ninguno y luego de liberar alguno solo puede liberar los demás.

Ejemplos del algoritmo 2PL son:

- La básica en la que se sigue el esquema previamente explicado con la variante que el bloqueo de escritura se pide para todas las copias del elemento.
- 2PL de copia primaria: en vez de pedir bloqueo para cada copia del elemento de escritura se le pide a una copia primaria o principal.
- 2PL de voto: se pide a todos los nodos que voten para ver si se concede el bloqueo.
- 2PL centralizado: el manejador de bloqueos está centralizado y todas las peticiones de bloqueo las maneja el.

Antes de implementar un algoritmo de control de concurrencia 2PL es necesario considerar distintos factores como cuál es la unidad atómica más pequeña que el sistema permite bloquear, cual es el intervalo de sincronización para todas las copias de un elemento, donde se deben colocar las tablas con la información de los bloqueos y por último que tan probable es que ocurra por los factores anteriores un bloqueo mutuo.

2.4.4 TIME-STAMP

Cada transacción realizada se le asigna un timestamp (literalmente: sello de tiempo) único en el nodo que se originó. Este sello se adjunta a cada petición de lectura y escritura. En el caso de que se dé un conflicto de que dos operaciones de escritura traten de acceder al mismo elemento, este se resuelve serializandolo respecto a los sellos que tengan. A pesar de que existen varios algoritmos de control de concurrencia basados en timestamps, muy pocos son utilizados en aplicaciones comerciales. Esto es en gran parte porque se requiere que el sistema distribuido cuente con un reloj sincronizado que es raro que se tenga implementado.

2.5 SERIABILIDAD

Una forma de evitar los problemas de interferencia es no permitir que las transacciones se intercalen. Una ejecución en la cual ninguna de dos transacciones se intercala se conoce como *serial*. Para ser más precisos, una ejecución se dice que es serial si, para todo par de transacciones, todas las operaciones de una transacción se ejecutan antes de cualquier operación de la otra transacción. Desde el punto de vista del usuario, en una ejecución serial se ve como si las transacciones fueran operaciones que el DBS procesa atómicamente. Las transacciones seriales son correctas dado que cada transacción es correcta individualmente, y las transacciones que se ejecutan serialmente no pueden interferir con otra.

Si el DBS procesara transacciones serialmente, significaría que no podría ejecutar transacciones concurrentemente, si entendemos concurrencia como ejecuciones intercaladas. Sin dicha concurrencia, el sistema usaría sus recursos en un nivel relativamente pobre y podría ser sumamente ineficiente.

Una solución puede ser ampliar el rango de ejecuciones permisibles para incluir aquellas que tienen los mismos efectos que las seriales. Dichas ejecuciones se conocen como *serializables*. Entonces, una ejecución es serializable si produce el mismo resultado y tiene el mismo efecto sobre la base de datos que alguna ejecución serial de las mismas transacciones. Puesto que las transacciones seriales son correctas, y dado que cada ejecución serializable tiene el mismo efecto que una ejecución serial, las ejecuciones serializables también son correctas.

Las ejecuciones que ilustran la actualización perdida y el análisis inconsistente no son serializables. Por ejemplo, ejecutando las dos transacciones de Depositar serialmente, en cualquier orden, da un resultado diferente al de la ejecución intercalada que pierde una actualización, por lo tanto, la ejecución intercalada no es serializable. Similarmente, la ejecución intercalada de Transferir e ImprimirSuma tiene un efecto diferente a los de cada ejecución serial de las dos transacciones, y por ello no es serializable.

2.6 DEAD LOCK

El *deadlock*, conocido también como abrazo mortal o bloqueo mutuo en programación, se refiere a la situación en la que dos o más tareas, como procesos o hilos, **se pausan esperándose la una a la otra para poder llevar a cabo la continuación o finalización de su trabajo.**

Esta problemática destaca como uno de los inconvenientes de mayor seriedad y delicadeza que puede presentarse en un entorno de multiprogramación, pues ocasiona que las labores implicadas se detengan y no puedan continuar de forma habitual.

Cabe destacar que estos procesos o labores implicadas en el Deadlock **se encuentran a la espera de un recurso determinado que no se les será asignado**, como puede ser un componente de CPU, de memoria o de entrada/salida.

De manera que se debe tener en cuenta que el bloqueo mutuo en programación **se relaciona con los procesos de asignación de recursos a los procesos para que pueda llevar a cabo su ejecución**. Así pues, en sistemas de multiprogramación, se busca que estos recursos sean compartidos; sin embargo, puede ocurrir que esto presente problemas y se presenten bloqueos en el sistema y sus labores.

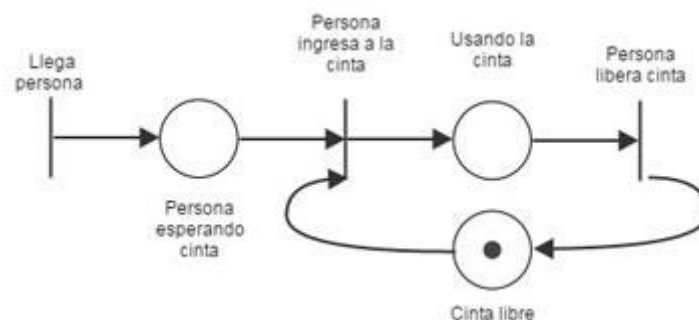
2.6.1 CAUSAS DEL DEADLOCK

Dentro de las características del *deadlock* en programación, se encuentra que este puede presentarse por diversas situaciones, como puede ser que **varios procesos compitan entre sí por la asignación de uno o más recursos determinados**. Esta destaca como la causa más común de los bloqueos mutuos.

Asimismo, un *deadlock* puede presentarse en caso de que a un proceso se le asigne la labor de esperar que suceda un evento, pero **el sistema no se encarga de incluir las opciones necesarias para notificar o señalar que dicho evento ocurrió**. Esto ocasionaría un bloqueo mutuo con un solo proceso. Cabe resaltar que, en caso de que el *deadlock* se presente por dicha causa, su detección sería altamente complicada.

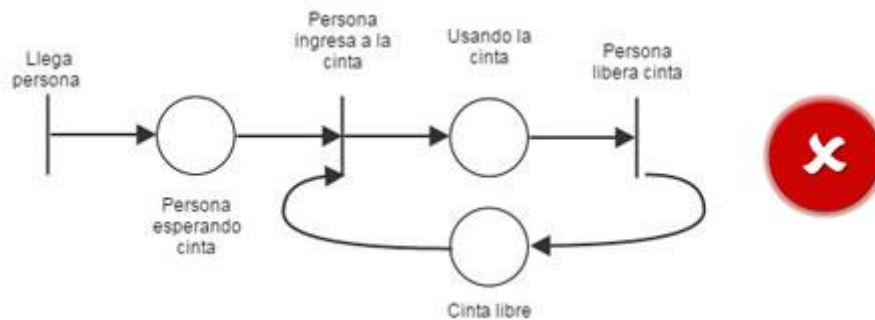
En sistemas operativos, el **deadlock**, es el bloqueo permanente de un conjunto de procesos en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos.

Al modelar una red, siempre se debe chequear que su ejecución no esté bloqueada por un mal diseño. Por ejemplo, consideremos la siguiente imagen:



Supongamos que no ponemos la marcación inicial en el sitio "Cinta libre". En ese caso, la transición "Persona ingresa a la cinta" no estará nunca habilitada, ya que es necesario que reciba un token por cada flecha de entrada.

En este caso, la red será INCORRECTA, ya que ninguna persona podrá usar la cinta.



2.7 TÉCNICAS PARA PREVENIRLO

La estrategia de prevención del interbloqueo consiste, a grandes rasgos, en diseñar un sistema de manera que esté excluida, a priori, la posibilidad de interbloqueo. Los métodos para prevenir el interbloqueo son de dos tipos. Los métodos indirectos consisten en impedir la aparición de alguna de las tres condiciones necesarias, antes mencionadas (condiciones 1 a 3). Los métodos directos consisten en evitar la aparición del círculo vicioso de espera (condición 4).

Se examinarán a continuación las técnicas relacionadas con cada una de las cuatro condiciones

Exclusión Mutua

En general, la primera de las cuatro condiciones no puede anularse. Si el acceso a un recurso necesita exclusión mutua, el sistema operativo debe soportar la exclusión mutua. Algunos recursos, como los archivos, pueden permitir varios accesos para lectura, pero sólo accesos exclusivos para escritura. Incluso en este caso, se puede producir interbloqueo si más de un proceso necesita permiso de escritura

Retención y Espera

La condición de retención y espera puede prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Esta solución resulta ineficiente por dos factores. En primer lugar, un proceso puede estar suspendido durante mucho tiempo, esperando que se concedan todas sus solicitudes de recursos, cuando de hecho podría haber avanzado con sólo algunos de los recursos. Y en segundo lugar, los recursos asignados a un proceso pueden permanecer sin usarse durante periodos considerables, tiempo durante el cual se priva del acceso a otros procesos

No apropiación

La condición de no apropiación puede prevenirse de varias formas. Primero, si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo, cuando sea necesario, junto con el recurso adicional. Por otra parte, si un proceso solicita un recurso que actualmente está retenido por otro proceso, el sistema operativo puede expulsar al segundo proceso y exigirle que libere sus recursos.

Este último esquema evitará el interbloqueo sólo si no hay dos procesos que posean la misma prioridad. Esta técnica es práctica sólo cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de una forma fácil, como es el caso de un procesador

Círculo Vicioso de Espera

La condición del círculo vicioso de espera puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso se le han asignado recursos de tipo R, entonces sólo podrá realizar peticiones posteriores sobre los recursos de los tipos siguientes a R en la ordenación. Para comprobar el funcionamiento de esta estrategia, se

asocia un índice a cada tipo de recurso. En tal caso, el recurso R_i , antecede a R_j , en la ordenación $s_i < s_j$.

Entonces, supóngase que dos procesos A y B se interbloquean, porque A ha adquirido R_i , y solicitado R_j , mientras que B ha adquirido R_j ;y solicitado R_i . Esta situación es imposible porque implica que $s_j < s_i$. Como en la retención y espera, la prevención del círculo vicioso de espera puede ser ineficiente, retardando procesos y denegando accesos a recursos innecesariamente

2.8 TÉCNICAS PARA DESHACERLO

Las estrategias de prevención del interbloqueo son muy conservadoras; solucionan el problema del interbloqueo limitando el acceso a los recursos e imponiendo restricciones a los procesos. En el lado opuesto, las estrategias de detección del interbloqueo no limitan el acceso a los recursos ni restringen las acciones de los procesos. Con detección del interbloqueo, se concederán los recursos que los procesos necesiten siempre que sea posible. Periódicamente, el sistema operativo ejecuta un algoritmo que permite detectar la condición de círculo vicioso de espera descrita en el punto 4 anterior. Puede emplearse cualquier algoritmo de detección de ciclos en grafos dirigidos

El control del interbloqueo puede llevarse a cabo tan frecuentemente como las solicitudes de recursos o con una frecuencia menor, dependiendo de la probabilidad de que se produzca el interbloqueo. La comprobación en cada solicitud de recurso tiene dos ventajas: Conduce a una pronta detección y el algoritmo es relativamente simple, puesto que está basado en cambios incrementales del estado del sistema. Por otro lado, tal frecuencia de comprobaciones consume un tiempo de procesador considerable

Una vez detectado el interbloqueo, hace falta alguna estrategia de recuperación. Las técnicas siguientes son posibles enfoques, enumeradas en orden creciente de sofisticación:

1. Abandonar todos los procesos bloqueados. Esta es, se crea o no, una de las soluciones más comunes, si no la más común, de las adoptadas en un sistema operativo.
2. Retroceder cada proceso interbloqueado hasta algún punto de control definido previamente y volver a ejecutar todos los procesos. Es necesario que haya disponibles unos mecanismos de retroceso y reinicio en el sistema. El riesgo de esta solución radica en que puede repetirse el interbloqueo original. Sin embargo, el no determinismo del procesamiento concurrente asegura, en general, que esto no va a pasar.
3. Abandonar sucesivamente los procesos bloqueados hasta que deje de haber interbloqueo. El orden en el que se seleccionan los procesos a abandonar seguirá un criterio de mínimo coste. Después de abandonar cada proceso, se debe ejecutar de nuevo el algoritmo de detección para ver si todavía existe interbloqueo
4. Apropiarse de recursos sucesivamente hasta que deje de haber interbloqueo. Como en el punto 3, se debe emplear una selección basada en coste y hay que ejecutar de nuevo el algoritmo de detección después de cada apropiación. Un proceso que pierde un recurso por apropiación debe retroceder hasta un momento anterior a la adquisición de ese recurso.

Para los puntos 3 y 4, el criterio de selección podría ser uno de los siguientes, consistentes en escoger el proceso con:

- La menor cantidad de tiempo de procesador consumido hasta ahora
- El menor número de líneas de salida producidas hasta ahora
- El mayor tiempo restante estimado
- El menor número total de recursos asignados hasta ahora
- La prioridad más baja Algunas de estas cantidades son más fáciles de medir que otras. El tiempo restante estimado deja lugar a dudas, especialmente. Además, aparte de las medidas de prioridad, no existe otra indicación del "coste" para el usuario frente al coste para el sistema en conjunto

2.9 RECUPERACIÓN

Un sistema que pretenda recuperarse del interbloqueo, debe invocar a un algoritmo de detección cuando lo considere oportuno (ej. periódicamente).

Formas de intentar la recuperación:

Terminación de procesos matando a todos los procesos implicados (drástico) matando a uno de los procesos ¿cuál? el que más recursos libere el que menos tiempo lleve en ejecución... Retrocediendo la ejecución de algún proceso (rollback) muy complicado de implementar y necesita que el programa esté diseñado para que pueda retroceder

Expropiación de recursos Selección de la víctima ¿Qué recursos y de que procesos se expropian? En ambos casos (terminación de procesos o expropiación de recursos) hay que tener cuidado de no provocar la inanición de proceso

UNIDAD III SEGURIDAD

3.1. CONCEPTO

La seguridad de datos, también conocida como seguridad de la información o seguridad informática, es un aspecto esencial de TI en organizaciones de cualquier tamaño y tipo. Se trata de un aspecto que tiene que ver con la protección de datos contra accesos no autorizados y para protegerlos de una posible corrupción durante todo su ciclo de vida.

Seguridad de datos incluye conceptos como encriptación de datos, tokenización y prácticas de gestión de claves que ayudan a proteger los datos en todas las aplicaciones y plataformas de una organización.

Hoy en día, organizaciones de todo el mundo invierten fuertemente en la tecnología de información relacionada con la ciberdefensa con el fin de proteger sus activos críticos: su marca, capital intelectual y la información de sus clientes.

En todos los temas de seguridad de datos existen elementos comunes que todas las organizaciones deben tener en cuenta a la hora de aplicar sus medidas: las personas, los procesos y la tecnología.

Ingeniería de la seguridad de datos

Pensar en seguridad de datos y construir defensas desde el primer momento es de vital importancia. Los ingenieros de seguridad tienen como objetivo proteger la red de las amenazas desde su inicio hasta que son confiables y seguras. Los ingenieros de seguridad diseñan sistemas que protegen las cosas correctas de la manera correcta. Si el objetivo de un ingeniero de software es asegurar que las cosas sucedan, el objetivo del ingeniero de seguridad es asegurar que las cosas (malas) no sucedan diseñando, implementando y probando sistemas completos y seguros.

La ingeniería de seguridad cubre mucho terreno e incluye muchas medidas, desde pruebas de seguridad y revisiones de código regulares hasta la creación de arquitecturas de seguridad y modelos de amenazas para mantener una red bloqueada y segura desde un punto de vista holístico.

Encriptación

Si la ingeniería de seguridad de datos protege la red y otros activos físicos como servidores, computadoras y bases de datos, la encriptación protege los datos y archivos reales almacenados en ellos o que viajan entre ellos a través de Internet. Las estrategias de encriptación son cruciales para cualquier empresa que utilice la nube y son una excelente manera de proteger los discos duros, los datos y los archivos que se encuentran en tránsito a través de correo electrónico, en navegadores o en camino hacia la nube.

En el caso de que los datos sean interceptados, la encriptación dificulta que los hackers hagan algo con ellos. Esto se debe a que los datos encriptados son ilegibles para usuarios no autorizados sin la clave de encriptación. La encriptación no se debe dejar para el final, y debe ser cuidadosamente integrada en la red y el flujo de trabajo existente para que sea más exitosa.

Detección de intrusión y respuesta ante una brecha de seguridad

Si en la red ocurren acciones de aspecto sospechoso, como alguien o algo que intenta entrar, la detección de intrusos se activará. Los sistemas de detección de intrusos de red (NIDS) supervisan de forma continua y pasiva el tráfico de la red en busca de un comportamiento que parezca ilícito o anómalo y lo marcan para su revisión. Los NIDS no sólo bloquean ese tráfico, sino que también recopilan información sobre él y alertan a los administradores de red.

Pero a pesar de todo esto, las brechas de seguridad siguen ocurriendo. Es por eso que es importante tener un plan de respuesta a una violación de datos. Hay que estar preparado para entrar en acción con un sistema eficaz. Ese sistema se puede actualizar con la frecuencia

que se necesite, por ejemplo, si hay cambios en los componentes de la red o surgen nuevas amenazas que deban abordarse. Un sistema sólido contra una violación garantizará que tienes los recursos preparados y que es fácil seguir un conjunto de instrucciones para sellar la violación y todo lo que conlleva, ya sea que necesites recibir asistencia legal, tener pólizas de seguro, planes de recuperación de datos o notificar a cualquier socio de la cuestión.

Firewall

¿Cómo mantener a visitantes no deseados y software malicioso fuera de la red? Cuando estás conectado a Internet, una buena manera de asegurarse de que sólo las personas y archivos adecuados están recibiendo nuestros datos es mediante firewalls: software o hardware diseñado con un conjunto de reglas para bloquear el acceso a la red de usuarios no autorizados. Son excelentes líneas de defensa para evitar la interceptación de datos y bloquear el malware que intenta entrar en la red, y también evitan que la información importante salga, como contraseñas o datos confidenciales.

Análisis de vulnerabilidades

Los hackers suelen analizar las redes de forma activa o pasiva en busca de agujeros y vulnerabilidades. Los analistas de seguridad de datos y los profesionales de la evaluación de vulnerabilidades son elementos clave en la identificación de posibles agujeros y en cerrarlos. El software de análisis de seguridad se utiliza para aprovechar cualquier vulnerabilidad de un ordenador, red o infraestructura de comunicaciones, priorizando y abordando cada uno de ellos con planes de seguridad de datos que protegen, detectan y reaccionan.

Pruebas de intrusión

El análisis de vulnerabilidad (que identifica amenazas potenciales) también puede incluir deliberadamente investigar una red o un sistema para detectar fallos o hacer pruebas de intrusión. Es una excelente manera de identificar las vulnerabilidades antes de tiempo y diseñar un plan para solucionarlas. Si hay fallos en los sistemas operativos, problemas con

incumplimientos, el código de ciertas aplicaciones u otros problemas similares, un administrador de red experto en pruebas de intrusión puede ayudarte a localizar estos problemas y aplicar parches para que tengas menos probabilidades de tener un ataque.

Las pruebas de intrusión implican la ejecución de procesos manuales o automatizados que interrumpen los servidores, las aplicaciones, las redes e incluso los dispositivos de los usuarios finales para ver si la intrusión es posible y dónde se produjo esa ruptura. A partir de esto, pueden generar un informe para los auditores como prueba de cumplimiento.

Una prueba de intrusión completa puede ahorrarte tiempo y dinero al prevenir ataques costosos en áreas débiles que no conoces. El tiempo de inactividad del sistema puede ser otro efecto secundario molesto de ataques maliciosos, por lo que hacer pruebas de intrusión con regularidad es una excelente manera de evitar problemas antes de que surjan.

Información de seguridad y gestión de eventos

Hay una línea aún más holística de defensa que se puede emplear para mantener los ojos en cada punto de contacto. Es lo que se conoce como Información de Seguridad y Gestión de Eventos (SIEM). SIEM es un enfoque integral que monitoriza y reúne cualquier detalle sobre la actividad relacionada con la seguridad de TI que pueda ocurrir en cualquier lugar de la red, ya sea en servidores, dispositivos de usuario o software de seguridad como NIDS y firewalls. Los sistemas SIEM luego compilan y hacen que esa información esté centralizada y disponible para que se pueda administrar y analizar los registros en tiempo real, e identificar de esta forma los patrones que destacan.

Estos sistemas pueden ser bastante complejos de configurar y mantener, por lo que es importante contratar a un experto administrador SIEM.

Ciberseguridad: HTTPS, SSL y TLS

Internet en sí mismo se considera una red insegura, lo cual es algo que puede asustar cuando nos damos cuenta que actualmente es la espina dorsal de muchas de las transacciones de

información entre organizaciones. Para protegernos de que, sin darnos cuenta, compartamos nuestra información privada en todo Internet, existen diferentes estándares y protocolos de cómo se envía la información a través de esta red. Las conexiones cifradas y las páginas seguras con protocolos HTTPS pueden ocultar y proteger los datos enviados y recibidos en los navegadores. Para crear canales de comunicación seguros, los profesionales de seguridad de Internet pueden implementar protocolos TCP/IP (con medidas de criptografía entrelazadas) y métodos de encriptación como Secure Sockets Layer (SSL) o TLS (Transport Layer Security).

El software anti-malware y anti-spyware también es importante. Está diseñado para supervisar el tráfico de Internet entrante o el malware como spyware, adware o virus troyanos.

Detección de amenazas en punto final

Se pueden prevenir ataques de ransomware siguiendo buenas prácticas de seguridad, como tener software antivirus, el último sistema operativo y copias de seguridad de datos en la nube y en un dispositivo local. Sin embargo, esto es diferente para organizaciones que tienen múltiple personal, sistemas e instalaciones que son susceptibles a ataques.

Los usuarios reales, junto con los dispositivos que usan para acceder a la red (por ejemplo, teléfonos móviles, ordenadores portátiles o sistemas TPV móviles), suelen ser el eslabón más débil de la cadena de seguridad. Se deben implementar varios niveles de protección, como tecnología de autorización que otorga acceso a un dispositivo a la red.

Prevención de pérdida de datos (DLP)

Dentro de la seguridad de punto final hay otra estrategia de seguridad de datos importante: la prevención de pérdida de datos (DLP). Esencialmente, esto abarca las medidas que se toman para asegurar que no se envían datos confidenciales desde la red, ya sea a propósito, o por accidente. Puede implementarse software DLP para supervisar la red y asegurarse de

que los usuarios finales autorizados no estén copiando o compartiendo información privada o datos que no deberían.

3.2. IDENTIDAD Y AUTENTIFICACIÓN

La gestión de identidad y acceso (abreviado: IAM o IdAM) es un modo de saber quién es un usuario y qué tiene permiso para hacer. IAM es como el portero de una discoteca con una lista de quién puede entrar, quién no y quién puede acceder a la zona VIP. IAM también se conoce como gestión de identidad (IdM).

En palabras más técnicas, IAM es un modo de administrar un conjunto dado de identidades digitales de los usuarios y los privilegios asociados con cada identidad. Es un término genérico que cubre una serie de productos diferentes, todos con la misma función básica. En una organización, IAM puede ser un producto único o una combinación de procesos, productos de software, servicios en la nube y hardware, que ofrecen a los administradores visibilidad y control sobre los datos de la organización a los que pueden acceder los usuarios individuales.

¿Qué es identidad en el contexto informático?

La identidad completa de una persona no puede cargarse y almacenarse en un ordenador, así que la "identidad" en un contexto informático indica un cierto conjunto de propiedades que, de forma conveniente, se pueden medir y registrar digitalmente. Piense en un carnet de identidad o en un pasaporte: el carnet de identidad no registra todos los datos acerca de una persona, pero sí que contiene suficientes características personales para que la identidad de una persona se pueda comparar rápidamente con los datos del carnet.

Para verificar la identidad, un sistema informático evaluará a un usuario para buscar características que sean específicas de los mismos. Si estos coinciden, se confirmará la identidad del usuario. Estas características también se conocen como "factores de autenticación", porque ayudan a autenticar a un usuario que dice ser quién es.

Los tres factores de autenticación más utilizados son:

- Algo que sabe el usuario
- Algo que tiene el usuario
- Algo que el usuario es

Algo que sabe el usuario: este factor es una información que solo debe tener un usuario, como la combinación de nombre de usuario y contraseña.

Imagina que John quiere revisar su correo electrónico del trabajo desde casa. Para hacerlo, en primer lugar, tendrá que iniciar sesión en su cuenta de correo electrónico demostrando identidad, ya si alguien que no fuera John accediera a su correo electrónico, se verían comprometidos los datos de la empresa.

John inicia sesión introduciendo su correo electrónico, *john@company.com* y la contraseña que solo él sabe, por ejemplo, "5jt*2)fl2?y". Además de John, supuestamente nadie sabe esta contraseña, por lo que el sistema de correo electrónico reconoce a John y le permite acceder a su cuenta de correo electrónico. Si alguien más intentara hacerse pasar por John introduciendo su dirección de correo electrónico "*john@company.com*", no podría acceder sin introducir la contraseña "5jt*2)fl2?y".

Algo que tiene el usuario: este factor se refiere a la posesión de un objeto físico otorgado a usuarios autorizados. El ejemplo más básico de este factor de autenticación es el uso de una llave para entrar en casa. Se supone que solo alguien que sea propietario, arrendatario o que de otra manera tenga permiso para entrar en la casa, contará con una llave.

En un contexto informático, el objeto físico podría ser un llavero con mando a distancia, un dispositivo USB o incluso un teléfono inteligente. Supongamos que la organización de John quisiera asegurarse de que todos los usuarios son quienes dicen ser mediante el uso de dos factores de autenticación en lugar de uno. Ahora, en lugar de simplemente introducir su contraseña secreta, el factor de algo que sabe el usuario, John tiene que mostrarle al sistema de correo electrónico que cuenta con un objeto que nadie más tiene. John es la única

persona en el mundo que posee su teléfono inteligente personal, así que el sistema de correo electrónico le envía un mensaje de texto con un código de uso único, que John tiene que introducir para demostrar que posee el teléfono.

Algo que el usuario es: esto hace referencia a una propiedad física del propio cuerpo. Un ejemplo habitual de este factor de autenticación es Face ID, una función incluida en muchos teléfonos inteligentes modernos. Otro ejemplo es el escaneado de huellas dactilares. Los escaneados de retina y los análisis de sangre son métodos menos comunes utilizados por algunas organizaciones de alta seguridad.

Imagina que la organización de John decide reforzar todavía más la seguridad haciendo que los usuarios verifiquen tres factores en lugar de dos (poco habitual). Ahora, John tendrá que introducir su contraseña, verificar la posesión de su teléfono inteligente y escanear su huella digital antes de que el sistema de correo electrónico confirme que se trata realmente de John.

En resumen: en el mundo real, la propia identidad es una compleja combinación de características personales, historia, ubicación y otros factores. En el mundo digital, la identidad de un usuario está compuesta de todos o algunos de los tres factores de autenticación, almacenados digitalmente en una base de datos de identidad. Para evitar que los impostores se hagan pasar por usuarios reales, los sistemas informáticos comprobarán la identidad de un usuario mediante el uso de la base de datos de identidad.

¿Qué es gestión de acceso?

"Access" hace referencia a qué datos puede ver un usuario y qué acciones puede llevar a cabo una vez hayan iniciado sesión. Cuando John inicia sesión en su correo electrónico, puede ver todos los correos electrónicos enviados y recibidos. Sin embargo, no debería poder ver los correos electrónicos enviados y recibidos por Tracy, su compañera de trabajo.

En otras palabras, solo porque se haya verificado la identidad de un usuario, no implica necesariamente que deberían poder tener acceso a lo que quieran dentro de un sistema o

red. Por ejemplo, un empleado de bajo nivel en una empresa debería poder tener acceso a su cuenta de correo electrónico corporativo, pero no debería poder acceder a los registros de nóminas o a la información confidencial de recursos humanos.

La gestión de acceso es el proceso de control y seguimiento del acceso. Cada usuario en un sistema tendrá diferentes privilegios en dicho sistema en función de sus necesidades individuales. Efectivamente, un contable necesita acceder y editar los registros de nóminas, así que una vez que se verifique su identidad, debería poder ver y actualizar esos registros, así como tener acceso a su cuenta de correo electrónico.

¿Por qué IAM es tan importante para la computación en nube?

En la computación en nube, se almacenan los datos de forma remota y se accede a ellos a través de Internet. Como los usuarios se pueden conectar a Internet desde casi cualquier ubicación y dispositivo, la mayoría de los servicios en la nube operan independientemente del dispositivo y la ubicación. Los usuarios ya no tienen que estar presentes en la oficina o utilizar un dispositivo propiedad de la empresa para acceder a la nube. De hecho, el teletrabajo cada vez es más habitual.

Como resultado, la identidad se convierte en el punto más importante a la hora de controlar el acceso, no el perímetro de la red. * La identidad del usuario, y no su dispositivo o ubicación, determina a qué datos pueden acceder en la nube y si cuentan en general con acceso.

Para entender por qué es tan importante la identidad, aquí hay una ilustración. Supongamos que un ciberdelincuente quiere acceder a archivos confidenciales del centro de datos corporativos de una empresa. En la época anterior a la adopción generalizada de la computación en nube, el ciberdelincuente tendría que superar el firewall de la empresa que protege la red interna, o acceder físicamente al servidor entrando de manera ilegal en el edificio o sobornando a un empleado interno. El objetivo principal del delincuente sería superar el perímetro de red.

Sin embargo, con la computación en la nube, los archivos confidenciales se almacenan en un servidor remoto en la nube. Ya que los empleados de la empresa necesitan acceder a los archivos, lo hacen iniciando sesión mediante un navegador o una aplicación. Si un ciberdelincuente quiere acceder a los archivos, solo necesita las credenciales de inicio de sesión de los empleados (nombre de usuario y contraseña) y una conexión a Internet; el delincuente no tiene que superar un perímetro de red.

IAM ayuda a prevenir los [ataques basados en identidad](#) y las fugas de datos que provengan de escaladas de privilegios (cuando un usuario no autorizado tiene demasiado acceso). Por tanto, los sistemas de IAM son fundamentales para la computación en nube y para gestionar equipos en remoto.

**El perímetro de la red hace referencia a los extremos de una red interna; es un límite virtual que separa la red interna gestionada y segura del Internet no seguro y sin controlar. Todos los ordenadores de una oficina, más los dispositivos conectados como las impresoras de la oficina, están dentro de este perímetro, pero un servidor en remoto en un centro de datos que cruza el mundo no lo está.*

¿Dónde encaja IAM en una pila de implementación en nube/arquitectura de nube?

A menudo, IAM es un servicio en la nube que los usuarios deben atravesar para poder acceder al resto de la infraestructura en la nube de una organización. También se puede implementar en una red interna en las instalaciones de la organización. Por último, algunos proveedores públicos de nube pueden agrupar IAM con sus otros servicios.

En cambio, las empresas que utilizan una arquitectura de multinube o nube híbrida pueden utilizar un proveedor independiente para IAM. Desacoplar IAM de sus otros servicios en la nube públicos o privados les proporciona más flexibilidad: pueden mantener su identidad y acceder a su base de datos si cambian de proveedor de nube.

¿Qué es un proveedor de identidad (IdP)?

Un proveedor de identidad (IdP) es un producto o servicio que ayuda a gestionar la identidad. Un IdP suele gestionar el proceso de inicio de sesión. Los proveedores de Inicio de sesión único (SSO) entran en esta categoría. Los IdP pueden formar parte de un marco IAM, pero generalmente no ayudan con la gestión de acceso de usuarios.

¿Qué es la Identidad como servicio (IDaaS)?

La identidad como servicio (IDaaS) es un servicio en la nube que verifica la identidad. Es una oferta de [SaaS](#) de los proveedores de nube, un modo de externalizar de forma parcial la gestión de identidad. En algunos casos, IDaaS e IdP son esencialmente intercambiables, pero en otros casos, el proveedor de IDaaS ofrece funciones adicionales además de la verificación y gestión de identidad. Dependiendo de las funciones que ofrezca el proveedor de IDaaS, IDaaS puede formar parte de un marco de IAM o constituir todo el sistema de IAM.

3.3. MATRIZ DE AUTORIZACIÓN

La matriz de autorización ayuda a proteger el acceso a los datos en el sistema SAP mediante los objetos de autorización. La matriz concede autorización sólo después de completar verificaciones complejas con múltiples condiciones. La matriz de autorización también utiliza términos descriptivos y técnicos para facilitar la confiabilidad y eficiencia de la auditoría. Cada autorización da permiso para realizar una o varias tareas dentro del sistema SAP.

Perfiles compuestos

La matriz concede autorización basada en el papel del individuo en la organización. Perfiles compuestos determinan estas funciones. Un individuo puede tener múltiples roles dentro de la organización, y un perfil compuesto puede contener múltiples seudónimos. Esto permite que una persona realice las operaciones del negocio asociadas con múltiples

funciones. Una organización debería, sin embargo, tener cuidado de no crear roles que resultan en medidas de seguridad redundantes en el sistema. Por esta razón, las empresas deben evaluar periódicamente todo perfil compuesto.

Configuración de seguridad

SAP desarrolló base para organizaciones con muchos empleados. La configuración de seguridad y administración de base utiliza un proceso de múltiples fases, y este proceso ayuda a garantizar la adecuada seguridad, integridad de la información y privacidad de las personas dentro de la organización. La configuración de seguridad también reconoce e implementa la autenticación de usuario. Esto ayuda a asegurar la integridad del sistema regulando un acceso seguro y que requieren autenticación de usuario válido para acceder a la aplicación.

Asignación de auditoría y supervisión

Base utiliza un generador de perfil para automáticamente generar y asignar perfiles de autorización a las personas. Los administradores autorizados crear estos perfiles para aumentar o limitar roles dentro del sistema. Un sistema de auditoría seguimiento de la autorización de cada perfil creado en el sistema. El sistema de vigilancia, por el contrario, garantiza el cumplimiento de las actividades de usuario. Según SAP, las organizaciones deben iniciar estas auditorías sobre una base semanal, mensual, trimestral y anual.

¿Qué es una matriz de control de accesos?

Como ya lo mencionamos, formalmente, es una herramienta para conocer y administrar los accesos y los permisos que los empleados tienen a la información de tu empresa.

Aunque su definición puede quedar un poco ambigua todavía, no temas, es solo un documento con filas y columnas donde defines a qué aplicaciones puede ingresar tu equipo y qué pueden hacer una vez dentro (ver, editar, administrar, etc.). En general, ayuda a construir pautas para limitar y proteger el acceso a los datos críticos y confidenciales de una empresa.

Te preguntará: ¿Por qué necesito una si todos en mi equipo son de confianza? La matriz de control de accesos es un requisito deseable para cumplir con normativas de seguridad de la información como ISO 27001, aunque no es obligatorio.

De todas formas, te aconsejamos implementarla ya que tarde o temprano un cliente te la pedirá en sus cuestionarios de ciberseguridad. Que no se te caiga un contrato por algo tan simple.

	GSUITE	SuperAdmin	Groups Admin	Service Admin	Help Desk Admin	Administrator	User Managment	Member /Standard User	SLACK	Admin	Workspace Owner	Primary Owner	Member / Final User	Guest	ASANA	Admin	Billing Owner	Member	Guest	AWS	UserAdmin	Security Admin	Operator User	
Alta Gerencia																								
CEO													x					x						
COO													x					x						
CTO						x	x			x								x				x		
CFO													x					x						
CTIO													x					x						
Tecnología																								
Developers													x					x						
Lider Tecnico										x			x					x						
Operaciones TI													x					x						
Responsable de Infraestructura													x					x						

Ejemplo de Matriz de Control de Accesos

Cómo asignar los accesos y permisos al equipo

¿Te suena familiar? Al diseñar tu matriz, es muy probable que quieras asignar los accesos basándote en las personas, en lugar de basarte en sus roles. Este es un error muy frecuente y es fuertemente recomendable evitarlo para que la asignación de accesos sea todavía más fácil.

La recomendación es, entonces, asignar los accesos y permisos del personal en función de sus roles y responsabilidades.

Por ejemplo, quienes tengan el rol de Analista de Sistemas tendrán “X” accesos, quienes tengan el rol de Programador tendrán “Y” accesos, y así con cada puesto.

6 pasos para crear una matriz de accesos basada en roles

Ahora sí, pasemos a la acción. Si quieres crear tu propia matriz de accesos desde cero sigue estos pasos:

I. Listar los roles


En una tabla de excel haz una lista de los roles que existen en la empresa. Para ello puedes ayudarte del organigrama o de las descripciones de trabajo de cada puesto. Esta información irá en las filas de la matriz.

	GSUITE	SuperAdmin	Groups Admin	Service Admin	Help Desk Admin	Administrator	User Management	Member /Standard User	SLACK	Admin	Workspace Owner	Primary Owner	Member / Final User	Guest
Alta Gerencia														
CEO													X	
COO													X	
CTO														
CFO													X	
CTIO													X	
Tecnología														
Developers													X	
Lider Tecnico										X			X	
Operaciones TI													X	
Responsable de Infraestructura													X	

Roles en Matriz de Control de Accesos

2. Agregar las aplicaciones que usa la empresa

En las columnas de la matriz agrega las aplicaciones y bases de datos que usa tu startup. Lo recomendable es listarlas todas, pero si no, como mínimo, deben figurar las que tienen información crítica o las que hayas definido en el alcance de este proyecto.



	GSUITE	SuperAdmin	Groups Admin	Service Admin	Help Desk Admin	Administrator	User Management	Member / Standard User	SLACK	Admin	Workspace Owner	Primary Owner	Member / Final User	Guest
Alta Gerencia														
CEO														x
COO														x
CTO														
CFO														x
CTIO														x
Tecnología														
Developers														x
Lider Tecnico										x				x
Operaciones TI														x
Responsable de Infraestructura														x

Activos bajo gestión en la Matriz de Control de Accesos

3. Crear el modelo de accesos y permisos para cada rol

Ahora es momento de definir un modelo a seguir de permisos y accesos. Para ello, marca con una cruz los permisos que idealmente deberían tener cada uno de los roles.

	CSUITE	SuperAdmin	Groups Admin	Service Admin	Help Desk Admin	Administrator	User Management	Member / Standard User	SLACK	Admin	Workspace Owner	Primary Owner	Member / Final User	Guest	ASANA	Admin	Billing Owner	Member	Guest	AWS	UserAdmin
Alta Gerencia																					
CEO													x					x			
COO													x					x			
CTO																		x			x
CFO													x					x			
CTIO													x					x			
Tecnología																					
Developers													x					x			
Lider Tecnico									x				x					x			
Operaciones TI													x					x			
Responsable de Infraestructura													x					x			

Control de Accesos

4. Conocer el panorama real de accesos y permisos

Revisemos cómo está la situación en la realidad. Dirígete a cada aplicación y exporta la lista de usuarios con acceso para comparar con tu matriz y poder resaltar aquellos permisos que debas corregir o agregar.

Hackmetrix Insight: recuerda que puedes buscar cómo exportar estas listas en la documentación de cada aplicación. En el caso de las bases de datos, puedes obtener este reporte mediante una consulta estándar.

5. Configurar y corregir los accesos

Toma todo lo que resaltaste y comienza a corregir o agregar los accesos y permisos. ¡Recuerda modificarlos tanto en la aplicación como en la matriz!

6. Controlar periódicamente

En el caso de los startups, recomendamos realizar controles a la matriz con una frecuencia mínima semestral o trimestral, porque muy probablemente en tres o seis meses, ya no tengas la misma cantidad de aplicaciones, o incluso la misma cantidad de empleados.

3.4. DEFINICIÓN DE UN ESQUEMA DE SEGURIDAD

La seguridad ha sido muy importante para la preservación de los datos, sin embargo, este ha pasado de utilizarse para datos clasificados de ciertos sectores privados, a tener una aplicación en diversas áreas personales. Por ello, se hace impredecible que las necesidades de seguridad potenciales sean tomadas en cuenta y se determinen para todo tipo de aplicaciones.

Espejo

¿Qué es espejo?

Se conoce como copia espejo (en inglés data mirroring) al procedimiento de protección de datos y de acceso a los mismos en los equipos informáticos implementado en la tecnología de RAID 1.

Consiste en la idea básica de tener dos discos duros conectados. Uno es el principal y en el segundo se guarda la copia exacta del principal, almacenando cualquier cambio que se haga en tiempo real en las particiones, directorios, etc, creando imágenes exactas, etc.

De esta forma se consigue tener 2 discos duros idénticos y que permiten, si todo está bien configurado, que ante el fallo del disco principal, el secundario tome el relevo, impidiendo la caída del sistema y la pérdida de los datos almacenados.

En el «mirroring» en una base de datos tenemos un servidor principal/primario que mantiene la copia activa de la base de datos (BD accesible). Otro servidor de espejo que mantiene una copia de la base de datos principal y aplica todas las transacciones enviadas

por el Servidor Principal (en el que no se podrá acceder a la BD). Y un servidor testigo/arbitro que permite recuperaciones automáticas ante fallos, monitoriza el servidor principal y el de espejo para en caso de caída cambiar los roles (servidor opcional, no es obligatorio).

Beneficios del espejo

Además de proporcionar una copia adicional de los datos con el fin de redundancia en caso de fallo de hardware, la duplicación de disco puede permitir que cada disco se acceda por separado para los propósitos de lectura.

En determinadas circunstancias esto puede mejorar significativamente el rendimiento ya que el sistema puede elegir para cada lectura que disco puede buscar más rápidamente a los datos requeridos. Esto es especialmente importante cuando hay varias tareas que compiten por los datos en el mismo disco, y el «trashing» (donde el cambio entre tareas ocupa más tiempo que la tarea en sí) se puede reducir. Esta es una consideración importante en las configuraciones de hardware que frecuentemente tienen acceso a los datos en el disco.

Anuncios

INFORMA SOBRE ESTE ANUNCIO

En algunas implementaciones, el disco reflejado se puede dividir fuera y se utiliza para la copia de seguridad de datos, permitiendo que el primer disco permanezca activo. Sin embargo, la fusión de los dos discos se puede requerir un período de sincronización en su caso escribir la actividad I/O ha ocurrido con el disco duplicado.

Replicación

¿Qué es?

La replicación es un mecanismo utilizado para propagar y diseminar datos en un ambiente distribuido, con el objetivo de tener mejor performance y confiabilidad, mediante la reducción de dependencia de un sistema de base de datos centralizado.

La réplica proporciona una manera rápida y confiable de diseminar la información corporativa entre múltiples localizaciones en un ambiente de negocio distribuido, permitiendo distribuir la información de manera confiable.

Beneficios

Con la replicación se pueden llegar a obtener dos mejoras importantes:

1. Por un lado, se garantiza que el servicio ofrecido por la aplicación, no se vea interrumpido en caso de que se dé un fallo en alguna de las réplicas.

Además, el tiempo necesario para restablecer el servicio en la aplicación podría llegar a ser grande en algunos tipos de fallo.

2. Por otra parte, la capacidad de servicio se ve incrementada cuando las peticiones efectuadas por los clientes únicamente implican consultas.

Normalmente y dependiendo del caso, hay más lectura que escritura en una base de datos, por lo que tener varios nodos solo procesando la lectura puede traer un gran beneficio de rendimiento en una base de datos muy consultada.

Un esclavo estando casi sincrónicamente actualizado puede ser útil en caso de que el nodo maestro caiga, este puede reemplazarlo y así no detener el servicio.

Al haber múltiples copias de los datos disponibles en el sistema se dispone de un mecanismo excelente de recuperación cuando existan fallos en nodos.

Aunque esta es más precisa, también se puede usar para procesos que necesiten leer datos, generando bloqueos, al hacerlo sobre un esclavo esto no interviene en el funcionamiento

de todo el sistema, es muy usado para hacer copias de seguridad, o extraer grandes cantidades de datos para generar estadísticas.

Métodos de respaldo de un DBMS

Los medios a utilizar dependerán del tipo de computadora, cantidad de información a almacenar, tiempo disponible para realizar el respaldo, costos y obviamente de la tecnología disponible al momento.

Comandos para respaldo de datos

Copias de seguridad completa

```
BACKUP DATABASE NOMBRE_BASE_DATOS
```

```
TO DISK = “ < Ruta_Absoluta | Ruta_Relativa >Nombre_Archivo.bak”
```

```
[WITH FORMAT]
```

```
[NAME = ‘Nombre Lógico del Respaldo’]
```

Copias de seguridad diferencial

```
BACKUP DATABASE NOMBRE_BASE_DATOS
```

```
TO DISK = “ < Ruta_Absoluta | Ruta_Relativa >Nombre_Archivo”
```

```
WITH DIFFERENTIAL
```

```
[NAME = ‘Nombre Logico del Respaldo’]
```

Establece una nueva base diferencial para los datos.

Copias de seguridad de registro de transacciones

```
BACKUP LOG NOM_BASE_DATOS
```

TO DISK = '< Ruta_Absoluta | Ruta_Relativa >Nom_Respaldo_Log.trn'

BACKUP LG NOM_BASE_DATOS

TO DISK = '< Ruta_Absoluta | Ruta_Relativa >Nom_Respaldo_Log.trn'

WITH NO_TRUNCATE

Métodos de recuperación de un DBMS

Existen diferentes tipos de respaldos posibles:

Los tipos de copia de seguridad a realizar, dependiendo la cantidad de archivos que guardemos en la copia, podemos hablar de diferentes tipos de copias de seguridad.

Se realiza un respaldo total del disco, se respalda la totalidad de las bases de datos y la totalidad de las operaciones que se mantienen en línea (online). Si se realiza diariamente, ante cualquier problema solamente se debe recuperar el respaldo del día anterior.

Respaldos globales (full back-up):

Se respalda sólo una parte de la información (solamente una aplicación, una plataforma, los datos críticos o las bases nuevas, etc.) Como se ve, existen varios criterios para optar qué parte respaldar. Las consideraciones realizadas para el respaldo global valen aquí solamente para las partes respaldadas.

Respaldos parciales

Administración de base de datos

Se combina con respaldos globales o parciales. Se respalda solamente las modificaciones que han ocurrido desde el último respaldo. Para realizar una recuperación se debe adicionar al último respaldo global todos los respaldos incrementales sucesivos. Es un procedimiento de respaldo ágil y que ocupa poco espacio. El procedimiento de recuperación es complejo.

Respaldos incrementales

Similar al anterior. Se respalda las modificaciones que han ocurrido desde el último respaldo global o parcial. Para realizar una recuperación se debe adicionar al último respaldo global solamente el último respaldo diferencial. Es un procedimiento de respaldo relativamente ágil y que ocupa poco espacio, con un procedimiento de recuperación de sólo dos etapas.

Respaldos diferenciales

Se realiza una copia exactamente igual al mismo tiempo de todos los datos procesados, por ejemplo, cuando el sistema de discos espejados es utilizado. Recuperar los datos es simplemente direccionarse a otra unidad de disco.

Respaldos simultáneos

Los tipos posibles de respaldo pueden combinarse en forma conveniente en la modalidad de copia, como ser, por ejemplo:

- copia simple
- copia doble
- copia generacional

Modalidad de copia

Migración

¿Qué es la migración en base de datos?

Una migración de BD es un proceso que se realiza para mover o trasladar los datos almacenados de un origen de datos a otro.

Dicha migración se requiere llevar a cabo cuando es necesario mover un esquema dentro del mismo servidor, o de un servidor a otro, así como para actualizar la versión del software, y hacer un cambio de manejador de bases de datos por el de otro fabricante o para cambiarlo a una plataforma de cómputo distinta.

Motivos para realizar una migración

Existen diversos motivos para hacer una migración:

Migración de datos

La migración de datos tiene su fundamento en la ampliación un sistema de gestión de base. En este contexto, se trata de exportar los datos a un nuevo sistema con mayor capacidad o más funciones adicionales. Estos cambios llevan consigo una adaptación de todos los datos de una base de datos a otra. Por tanto siempre que se producen cambios de un sistema de gestión a otro, se habla inevitablemente de los procesos de migración de datos.

Inconsistencias

Una solución es permitir inconsistencias temporales de este tipo y proveer un mecanismo de notificación para determinar qué objetos son inconsistentes y lograr así optimizar la migración de datos.

*Mejorar el desempeño de la base de datos.

*Cumplir con nuevos requerimientos de usuario de la aplicación o políticas de seguridad.

*La compatibilidad con otras aplicaciones.

*La actualización de versiones.

*La estandarización de la tecnología de información en la organización.

*Facilitar el intercambio de datos entre procesos.

*La reducción de costos que se puede tener al cambiar por software libre.

*El aumento en el volumen de datos.

*Nuevos procesos de negocio.

*Mejoras en la seguridad o el control de la información entre otros escenarios posibles.

- Limpieza de las tablas de la Base de Datos.
- Consolidación de las tablas de la Base de Datos.
- Mapeado de las tablas de Origen y Destino.
- Definición de Formato de Origen y Destino.
- Ruta de Saltos de Formato en la Migración.
- Definición del Set de Caracteres de Origen y Destino.
- Comprobación de los delimitadores de campos.
- Migración de Prueba.
- Evaluación y Comprobación de Errores.
- Depuración Final. Pasos para llevar a cabo una Migración de datos

Monitoreo

El nivel de monitoreo es requerido con fines de auditoría. Ambos favorecen el buen funcionamiento de la base de datos, ya que se encargan de supervisar y analizar cada uno de los procesos y actividades para confirmar si se ajustan de acuerdo a los criterios establecidos.

Definición

Monitoreo (Monitorizar): Es observar el curso de uno o varios parámetros para detectar posibles anomalías.

Su origen se encuentra en monitor, un aparato que toma imágenes de instalaciones filmadoras o sensores y que permite visualizar información en una pantalla. El monitor, por lo tanto, ayuda a controlar o supervisar una situación.

Monitoreo general de un DBMS

Los centros de datos han ido creciendo cada vez más, adquiriendo mayor complejidad de acuerdo a las necesidades, transformando los datos en información virtual almacenada en las bases de datos.

Monitoreo de logs

Existen unos archivos denominados log, donde de manera particular quedan registrados todos los accesos que los diferentes usuarios de los sistemas de información hacen a los equipos computacionales y por supuesto a los programas de todo tipo, operacional, de comunicaciones, de base de datos, aplicativos.

Monitoreo de modos de operación

Los sistemas de monitoreo de servidores físicos y virtuales han ido evolucionando con el correr de los años. En muy poco tiempo y al ir adquiriendo experiencia, surgieron nuevas necesidades y, en paralelo, se descubrieron funcionalidades que no se tenían en cuenta.

Un Sistema Manejador de Bases de Datos permiten el almacenamiento, modificación, extracción y análisis de la información en una base de datos.

Por lo tanto, permite el monitoreo de la información almacenada. Este proceso es fundamental para apoyar la responsabilidad delegada al analista por la organización.

Cuando los analistas monitorean, se enfocan en los cambios del entorno para analizar si, entre las circunstancias detectadas mediante la búsqueda, se perfila hacia una tendencia inusual.

Monitoreo de espacio en disco

Anteriormente, los desarrolladores de estos sistemas de monitorización, comenzaron a ver que se podía controlar el espacio en disco, la cantidad de memoria RAM consumida y el porcentaje de CPU utilizado mediante unos simples comandos contra el sistema operativo de la maquina en cuestión. Por lo que, se agregaron controles a los servicios más comunes para luego pasar a los más específicos, como los programas de monitoreo.

3.5. MECANISMOS DE VISTA PARA IMPLANTACIÓN DE SEGURIDAD

Los mecanismos de seguridad son también llamadas herramientas de seguridad y son todos aquellos que permiten la protección de los bienes y servicios informáticos. Con estos mecanismos es con lo que se contesta la última pregunta de la metodología de la seguridad informática: ¿Cómo se van a proteger los bienes?

Estos mecanismos pueden ser algún dispositivo o herramienta física que permita resguardar un bien, un software o sistema que de igual manera ayude de algún modo a proteger un activo y que no precisamente es algo tangible, o una medida de seguridad que se implemente, por ejemplo, las políticas de seguridad.

Los mecanismos también reciben el nombre de controles ya que dentro de sus funciones se encuentran el indicar la manera en que se deben ejecutar las acciones que permitan resguardar la seguridad y se eviten vulnerabilidades en la misma.

- Finalmente, los mecanismos pueden clasificarse de acuerdo con el objetivo principal de los mismos en:
- Mecanismos preventivos. Como su nombre lo dice, son aquellos cuya finalidad consiste en prevenir la ocurrencia de un ataque informático. Básicamente se concentran en el monitoreo de la información y de los bienes, registro de las actividades que se realizan en la organización y control de todos los activos y de quienes acceden a ellos.
- Mecanismos detectores. Son aquellos que tienen como objetivo detectar todo aquello que pueda ser una amenaza para los bienes. Ejemplos de éstos son las personas y equipos de monitoreo, quienes pueden detectar cualquier intruso u anomalía en la organización.
- Mecanismos correctivos. Los mecanismos correctivos se encargan de reparar los errores cometidos o daños causados una vez que se ha cometido un ataque, o en otras palabras, modifican el estado del sistema de modo que vuelva a su estado original y adecuado.
- Mecanismos disuasivos. Se encargan de desalentar a los perpetradores de que cometan su ataque para minimizar los daños que puedan tener los bienes.

3.6. BASES DE DATOS ESTADÍSTICAS

Una **base de datos estática** es aquella cuya función principal es el almacenamiento y registro de datos fijos. Es decir, guarda información que no se va a modificar ni editar con el tiempo.

Se trata de un tipo de bases de datos de solo lectura. Su implementación se suele realizar con el objetivo de registrar datos históricos para poder comparar su evolución a lo largo del tiempo.

Por ejemplo, se puede crear una base de datos con las ventas totales de una categoría de productos durante una serie de meses, para comparar la evolución de las ventas a lo largo del tiempo.

Las bases de información estática son ampliamente utilizadas en el campo de la estadística y suelen estar orientadas a la toma de decisiones de índole empresarial.

En contraposición a las bases estáticas están las bases dinámicas, las cuáles sí permiten la edición, actualización o eliminación de datos a lo largo del tiempo.

Características y aspectos a tener en cuenta

- Vista la definición del párrafo anterior, se puede decir que las principales **características de las bases de datos estáticas** son las siguientes:
- Son bases de datos de solo lectura. Es decir, están diseñadas para agregar datos fijos que no se pueden modificar con el tiempo.
- Se utilizan fundamentalmente para almacenar datos históricos o hechos invariables.
- Se suelen combinar diferentes bases de datos estáticas realizadas en diferentes periodos para analizar la evolución de los datos en el tiempo.
- Por ello, son muy usadas para hacer estudios de mercado, investigaciones estadísticas y otros proyectos relacionados con el Business Intelligence.

Ventajas y desventajas

Ventajas

- Entre las **ventajas de las bases de datos estáticas** se pueden citar las siguientes:
- Son bases de datos que tienen total validez a lo largo del tiempo. Es decir, los datos siguen siendo válidos por mucho tiempo que pase.
- No necesitan mantenimiento de ningún tipo, ya que los datos no son editables.
- Las consultas suelen ser sencillas ya que se basan en datos fijos. Las relaciones entre tablas también son fáciles de realizar.
- Se pueden aplicar con numerosos objetivos, pero son especialmente útiles a la hora de comparar estadísticas, o la evolución de cualquier dato en el tiempo.

Desventajas

- Por su parte, las principales **desventajas de las bases de datos estáticas** son:
- No sirven para almacenar datos que han de ser modificados en el tiempo. Por ejemplo, el precio de un producto que es susceptible de cambiar.
- Son poco flexibles y se limitan a una serie de datos muy concretos.

5 ejemplos de base de datos estáticas

Existen numerosos **ejemplos de bases de datos estáticas**. Pero pongamos algún ejemplo que las ilustre mejor:

Las bases de datos del INE (Instituto Nacional de Estadística) que recogen los datos anuales sobre natalidad, ocupación laboral, sueldos medios y un largo etcétera.

Bases de datos que recogen el censo de una población a lo largo de los años.

Una base de datos que recoge los ingresos mensuales de una tienda en diferentes tablas a lo largo de los meses.

Bases de datos que se refieren a hechos históricos o efemérides.

Una base de datos que recoge la tasa de conversión de diferentes estrategias de marketing.

Diferencia entre base de datos estáticas y dinámicas

En la clasificación entre bases de datos según la variabilidad de la información hay dos grandes grupos: las bases de datos estáticas y las dinámicas.

La principal diferencia entre ambas es que, mientras las bases de datos estáticas son de solo lectura y no permite modificar o añadir datos, las bases de datos dinámicas son mucho más flexibles y sí permiten editar, actualizar o borrar datos.

Por ello, las bases estáticas se suelen emplear para información que es inamovible en el tiempo, mientras que las dinámicas se emplean para crear bases con datos que son susceptibles de variar con el tiempo.

Esto ha sido todo por nuestra parte. Si hay algo que no te ha quedado claro no dudes en plantear tus dudas en los comentarios.

3.7. ENCRIPAMIENTO DE DATOS

La **encriptación de datos** es un proceso de codificación mediante el cual se altera el contenido de la información haciéndola ilegible, de esta manera se consigue mantener la confidencialidad de la información mientras viaja del emisor al receptor. En el actual mundo digital, lo que se altera son los bits que conforman las cadenas de datos de archivos, documentos, emails, imágenes, etc.

Aunque nos pueda parecer que la encriptación es un procedimiento relacionado con el mundo digital e Internet, puesto que la gran mayoría de datos e información que viajan por la Red lo hacen de manera encriptada, lo cierto es que existe desde mucho antes de que existieran los ordenadores e incluso que la luz eléctrica, ya que desde las primeras civilizaciones se han ido desarrollando formas para proteger el contenido de los mensajes. Una de esas primeras formas es cifrado César, creado por el emperador romano para cifrar los mensajes militares.

En el cifrado César se altera el orden de las letras, sustituyendo cada letra por la que se encuentre tres posiciones más adelante en el alfabeto (aunque el valor de la posición se puede variar).

En la actualidad, si bien existen **tipos de encriptación de datos**, todos ellos recurren a un algoritmo matemático para modificar el contenido de los datos y generar la clave o claves que se usarán para desencriptarlo.

¿Encriptar y cifrar es lo mismo?

¿Cifrar o encriptar? Es más que probable que os hayáis encontrado con los dos términos alguna vez, de hecho, un poco más arriba nosotros hemos empleado el término «cifrar», por lo tanto, cabe preguntarse si son lo mismo.

Y la respuesta es sí, encriptar y cifrar son sinónimos, si bien «encriptar» es la traducción literal del inglés «encrypt» y, aunque su uso ya está admitido en el Diccionario de la RAE, «cifrar» es la traducción más correcta del término. En cualquier caso, podéis utilizarlos indistintamente.

¿Para qué sirve encriptar datos?

Encriptar archivos, documentos, mensajes, emails, imágenes o cualquier tipo de datos que circulen por la Red sirve para protegerlos de miradas indiscretas, es decir, para garantizar que la información viaja segura y se mantiene confidencial. Además, también puede servir para garantizar que no ha sido alterada o modificada y para acreditar el origen de la misma.

Por ese motivo encontramos tantos **ejemplos de encriptación de datos** en nuestro día a día, especialmente si usamos ciertas aplicaciones y navegamos por Internet.

Por ejemplo, los mensajes de WhatsApp viajan cifrados de extremo a extremo, lo que quiere decir que ningún actor intermedio, ni siquiera el servidor, tiene acceso al contenido de los mismos. O cuando introducimos información en una página web con certificado HTTPS, esos datos se cifran automáticamente. También tenemos un ejemplo de cifrado en los certificados digitales, que debemos usar para poder acceder a determinados servicios o áreas de usuario de la administración.

Con Windows 10 y otros programas podemos encriptar no solo archivos o documentos, también el disco duro o encriptar un pendrive en el que guardemos información confidencial.

La importancia de la encriptación de datos reside en que es una medida de seguridad para garantizar la confidencialidad de la información, viaje esta por Internet o esté almacenada en un ordenador o unidad de memoria externa.

Además, de acuerdo al RGPD, si la base de datos de nuestra empresa está encriptada y sufrimos una brecha de seguridad que afecte a los datos personales almacenados en ella, no tendremos obligación de notificar dicho incidente de seguridad ni a la AEPD ni a los interesados afectados, puesto que pese a que se haya podido extraer información, esta no será legible para los ciberdelincuentes.

¿Cómo funciona la encriptación de datos?

En la encriptación de datos actual se utiliza un algoritmo matemático para modificar el contenido que se quiere cifrar (dependiendo del sistema de cifrado, se usan diferentes algoritmos), para ello se genera una clave o claves que establecen la forma en que se «desordena la información» cuando se cifra y que después se emplea para descifrarla, es decir, volver a ordenarla.

Cuanto mayor sea la longitud de la clave, que se mide en bits, más seguro resultará el cifrado. Las longitudes habituales son de 128 y 256 bits para las claves privadas y de hasta 2048 bits para las públicas.

Actualmente, existen diferentes programas y sistemas de cifrado que realizan esta labor de forma automática, por lo que cuando los usamos para encriptar cualquier tipo de

información, solo debemos preocuparnos de guardar las claves de cifrado en un lugar seguro y, de ser necesario, guardar una copia de seguridad de dichas claves.

Tipos de encriptación de datos

Podemos hablar de tres **tipos de encriptación**:

Encriptación simétrica: En este tipo de cifrado, los datos se encriptan usando una única clave secreta de cifrado, de manera que emisor y receptor comparten una copia de esa clave. Los algoritmos utilizados en el cifrado simétrico se basan en operaciones sencillas como sustitución o permutación. Actualmente no es el método de encriptación más seguro e infalible, pero se sigue utilizando. Algunos sistemas de encriptación simétricos son:

- DES (estándar de encriptación de datos)
- Triple DES
- AES (estándar de encriptación avanzado, que sustituyó al DES)
- Blowfish
- Twofish

Encriptación asimétrica: En este tipo de cifrado se emplean dos claves diferentes, una privada y otra pública, teniendo emisor y receptor un par cada uno. La clave pública debe ser conocida por ambas partes, mientras que la privada es secreta. De esta manera, la clave pública se utiliza para cifrar los mensajes, pero solo la clave secreta puede descifrarlos. Para que el cifrado asimétrico sea completamente seguro, es necesario completarlo con la autenticación de las partes, como hacen, por ejemplo, los certificados digitales. Ejemplo de sistema de encriptación asimétrica es:

RSA

Encriptación híbrida: Se trata de un sistema que combina técnicas de cifrado simétrico y de cifrado asimétrico, utilizando este último para transferir la clave simétrica por un canal no protegido. Este tipo de encriptación se usa, por ejemplo en estándares de cifrado como:

- PGP (Pretty Good Privacy)
- GnuPG
- S/MIME

¿Cuándo debemos encriptar los datos?

Debemos encriptar los datos cuando queramos mantener la seguridad y confidencialidad de los mismos o cuando tengamos el mandato legal de hacerlo (puesto que hay leyes que obligan a cifrar determinada información). Si estamos manejando información sensible, secreta o que no queramos que pueda ser interceptada por terceros (por la razón que sea), deberemos encriptar los datos recurriendo a alguno de los métodos de cifrado existentes.

Por lo tanto, siempre que sea necesario garantizar la privacidad de los datos, tendremos que encriptar estos.

¿Obligan el RGPD y la LOPDGDD a encriptar los datos?

Hemos mencionado en el punto anterior que hay leyes que obligan a encriptar determinados datos; en el ámbito de la protección de datos, el RGPD y la LOPDGDD no obligan siempre a recurrir al cifrado como medida de seguridad, pero sí la entiende como una de las medidas de seguridad adecuada para garantizar la privacidad y confidencialidad de la información en el ámbito digital.

¿Cuándo puede ser obligatorio encriptar los datos?

El Reglamento (UE) 2016/679 y la Ley 3/2018, por tanto, obligan a encriptar los datos cuando se trate de datos sensibles, sean datos recabados para fines policiales sin el consentimiento de los interesados, sean datos derivados de casos de violencia de género,

cuando así lo imponga una ley o si el resultado de la evaluación de impacto de un tratamiento de datos recomienda hacerlo como medida de seguridad, para reducir o eliminar un riesgo.

Otras leyes que obligan a la encriptación de datos cuando se tratan datos personales son:

- La Ley de Prevención de Blanqueo de Capitales
- La Ley de Autonomía del Paciente
- Real Decreto 3/2010 que regula el Esquema Nacional de Seguridad en el ámbito de la Administración Pública

3.8. SEGURIDAD EMPLEANDO UN DML CON UN DBMS COMERCIAL

La creación de reflejo de la base de datos es una estrategia sencilla que ofrece las siguientes ventajas:

Incrementa la disponibilidad de una base de datos. Si se produce un desastre en el modo de alta seguridad con conmutación automática por error, la conmutación por error pone en línea rápidamente la copia en espera de la base de datos, sin pérdida de datos. En los demás modos operativos, el administrador de bases de datos tiene la alternativa del servicio forzado (con una posible pérdida de datos) para la copia en espera de la base de datos. Para obtener más información, vea Conmutación de roles, más adelante en este tema.

Aumenta la protección de los datos. La creación de reflejo de la base de datos proporciona una redundancia completa o casi completa de los datos, en función de si el modo de funcionamiento es el de alta seguridad o el de alto rendimiento. Para obtener más información, vea Modos de funcionamiento, más adelante en este tema.

Un asociado de creación de reflejo de la base de datos que se ejecute en SQL Server 2008 Enterprise o en versiones posteriores intentará resolver automáticamente cierto tipo de errores que impiden la lectura de una página de datos. El socio que no puede leer una página, solicita una copia nueva al otro socio. Si la solicitud se realiza correctamente, la copia sustituirá a la página que no se puede leer, de forma que se resuelve el error en la mayoría de los casos. Para obtener más información, vea Reparación de página automática (grupos de disponibilidad/creación de reflejo de base de datos).

Mejora la disponibilidad de la base de datos de producción durante las actualizaciones. Para minimizar el tiempo de inactividad para una base de datos reflejada, puede actualizar secuencialmente las instancias de SQL Server que hospedan los asociados de creación de reflejo de la base de datos. Esto incurrirá en el tiempo de inactividad de solo una conmutación por error única. Esta forma de actualización se denomina actualización gradual. Para obtener más información, vea Instalar un Service Pack en un sistema con un tiempo de inactividad mínimo para bases de datos reflejadas.

Lenguaje de Definición de Datos (DDL)

Es un lenguaje de programación para definir estructuras de datos, proporcionado por los sistemas gestores de bases de datos, en este caso PostgreSQL. En inglés, **Data Definition Language**, de ahí sus siglas **DDL**. Te recuerdo que si necesitas un amplio conocimiento en DDL deberías ver nuestro **Curso de Sentencias DDL, DML, DCL y TCL**. Si estás empezando y quieres conocer Postgre, quizás te interese nuestro **Curso de introducción a PostgreSQL**.

Empieza ahora y apúntate ya para aprender más sobre PostgreSQL.
¿Quieres ser un profesional trabajando como Administrador de Bases PostgreSQL?
¿Ampliar tus conocimientos? Mira nuestro **listado de Carreras para PostgreSQL**.

Este lenguaje **permite a los programadores** de un sistema gestor de base de datos, como Postgres, **definir las estructuras que almacenarán los datos** así como los procedimientos o funciones que permitan consultarlos.

Para definir las estructura disponemos de tres sentencias:

- **CREATE**, se usa para crear una base de datos, tabla, vistas, etc.
- **ALTER**, se utiliza para modificar la estructura, por ejemplo, añadir o borrar columnas de una tabla.
- **DROP**, con esta sentencia, podemos eliminar los objetos de la estructura, por ejemplo, un índice o una secuencia.

Lenguaje de Manipulación de Datos (DML)

También es un lenguaje proporcionado por los sistemas gestores de bases de datos. En inglés, **Data Manipulation Language (DML)**.

Utilizando instrucciones de SQL, **permite a los usuarios introducir datos para posteriormente realizar tareas de consultas o modificación** de los datos que contienen las Bases de Datos.

Los elementos que se utilizan para manipular los datos, son los siguientes:

- **SELECT**, esta sentencia se utiliza para realizar consultas sobre los datos.
- **INSERT**, con esta instrucción podemos insertar los valores en una base de datos.
- **UPDATE**, sirve para modificar los valores de uno o varios registros.
- **DELETE**, se utiliza para eliminar las filas de una tabla.

Todos estos lenguajes forman parte del lenguaje SQL en general. Es decir, no son aplicables únicamente a PostgreSQL sino a todos los gestores de bases de datos relacionales tales como Oracle SQL, MySQL o SQL Server. Si quieres aprender más sobre ello, **empieza ya y apúntate a nuestra academia de PostgreSQL** donde tienes todo el contenido en español.

Si quieres convertirte en DBA de PostgreSQL mira nuestros **cursos de PostgreSQL: DBA & Developers**

Lenguaje de Control de Datos (DCL)

Es un lenguaje que incluye una serie de comandos SQL. Como los anteriores, es proporcionado por los sistemas gestores de bases de datos. Sus siglas son **DCL** por su nombre en inglés, **Data Control Language**.

Estos comandos **permiten al Administrador del sistema gestor de base de datos, controlar el acceso a los objetos**, es decir, podemos otorgar o denegar permisos a uno o más roles para realizar determinadas tareas.

Los comandos para controlar los permisos son los siguientes:

- **GRANT**, permite otorgar permisos.
- **REVOKE**, elimina los permisos que previamente se han concedido.

En nuestro curso de **SQL Developer** te explicamos personalmente estos comandos, su uso y algunos trucos de cuando, cómo y por qué se deben de utilizar.

Sentencias prácticas

Cómo habéis visto en el video, las diferencias entre DDL, DML y DCL radican en los diferentes usos que hacemos de cada lenguaje. Si quieres ampliar la información sobre una

de las sentencias más utilizadas en el mundo SQL, aquí tienes acceso a una entrada en la que vemos cómo podemos hacer **uso de la instrucción INSERT**.

UNIDAD IV BASES DE DATOS DISTRIBUIDAS

4.1. INTRODUCCIÓN

Una Base de Datos Distribuida es una colección de datos que pertenecen lógicamente a un solo sistema, pero se encuentra físicamente distribuido en varios computadores o servidores de datos en una red de computadoras. Un sistema de bases de datos distribuidas se compone de un conjunto de sitios lógicos, conectados entre sí, mediante algún tipo de red de comunicaciones, en el cual:

Cada sitio lógico puede tener un sistema de base de datos.

Los sitios han sido diseñados para trabajar en conjunto, con el fin de que un usuario de cualquier posición geográfica pueda obtener acceso a los datos desde cualquier punto de la red tal como si todos los datos estuvieran almacenados en la posición propia del usuario. Entonces, la llamada “Base de Datos Distribuida” es en realidad una especie de “objeto virtual”, cuyos componentes se almacenan físicamente en varias “bases de datos reales” ubicadas en diferentes sitios. En esencia es la unión lógica de esas diferentes bases de datos.

En otras palabras, cada sitio tiene sus propias “bases de datos reales” locales, sus propios usuarios locales, sus propios SGBD y programas para la administración de transacciones y su propio administrador de comunicación de datos. Así pues, el sistema de bases de datos distribuidas puede considerarse como una especie de sociedad entre los diferentes SGBD individuales locales. Un nuevo componente de software en cada sitio realiza las funciones de sociedad necesarias; y es la combinación de este nuevo componente y el SGBD ya existente constituyen el llamado Sistema de Administración o Gestión de Bases de Datos Distribuidas - SGBDD. (En inglés DDBMS, Distributed DataBase Management System).

Desde el punto de vista del usuario final, un sistema distribuido deberá ser idéntico a un sistema no distribuido. Los usuarios de un sistema distribuido se comportan en su manipulación de información exactamente como si el sistema no estuviera distribuido.

Todos los problemas de los sistemas distribuidos son de tipo interno o a nivel de realización, no pueden existir problemas de tipo externo o a nivel del usuario final.

Los datos que se encuentran distribuidos en varios sitios y que están interconectados por una red de comunicaciones tienen capacidad de procesamiento autónomo de transacciones y hacer procesos locales. Cada sitio realiza la ejecución de al menos una transacción global, la cual requiere accesos a datos en diversos sitios.

Los principios fundamentales de un sistema de datos distribuido son: (Estos doce postulados no son todas independientes entre sí, ni tienen toda la misma importancia que cada usuario le otorgue. Sin embargo, sí son útiles como fundamento para entender la tecnología distribuida y como marco de referencia para caracterizar la funcionalidad de sistemas distribuidos específicos.)

1. **Autonomía local:** Los sitios o posiciones de un sistema distribuido deben ser autónomos. La autonomía local significa que todas las operaciones en un sitio determinado se controlan en ese sitio; ningún sitio A deberá depender de algún otro sitio B para su buen funcionamiento (pues de otra manera el sitio A podría ser incapaz de trabajar, aunque no tenga en sí problema alguno, si cae el sitio B). La autonomía local significa que existe un propietario y un administrador local de los datos, con responsabilidad local: todos los datos pertenecen a una base de datos local, aunque los datos sean accesibles desde algún sitio distante. Todo el manejo de la seguridad y la integridad de los datos se efectúan con control de la instalación y administración local.
2. **No dependencia de un sitio central :** La no dependencia de un sitio central, sería lo ideal pero si esto no se logra la autonomía local completa se vería comprometida. La dependencia de un sitio central no es práctica al menos por las siguientes razones: en primer lugar, estos sitios podrían generar un cuello de botella, y en segundo lugar, el sistema sería vulnerable; si el sitio central sufriera un desperfecto, todo el sistema dejaría de funcionar.

3. Operación continua: En un sistema distribuido, lo mismo que en uno no distribuido, nunca debería haber necesidad de apagar o dejar de funcionar. Es decir, el sistema nunca debería necesitar apagarse para que se pueda realizar alguna operación, como añadirse un nuevo sitio o instalar una versión mejorada del SGBD en un sitio ya existente.
4. Independencia con respecto a la localización : La independencia con respecto a la localización, permite que los usuarios finales no sepan donde están almacenados físicamente los datos, sino que trabajen como si todos los datos estuvieran almacenados en su propio sitio local. La independencia con respecto a la localización es deseable porque simplifica los sistemas de información de los usuarios y sus actividades en la terminal. Esto hace posible la migración de datos de un sitio a otro sin anular la validez de ninguno de esos sistemas o actividades. Esta función de migración permite modificar la distribución de los datos dentro de la red, en respuesta a cambios en los requerimientos de desempeño.
5. Independencia con respecto a la fragmentación: Un sistema tiene fragmentación de datos solo si es posible dividir una relación en partes o “fragmentos” para propósitos de almacenamiento físico. La fragmentación es deseable por razones de desempeño: los datos pueden almacenarse en la localidad donde se utilizan con mayor frecuencia, de manera que la mayor parte de las operaciones sean solo locales y se reduzca el tráfico en la red de cómputo. Existen en esencia dos clases de fragmentación, la fragmentación horizontal y la fragmentación vertical; estos tipos de fragmentación son correspondientes a las operaciones relacionales de restricción y proyección, respectivamente. Un fragmento puede ser cualquier sub relación que pueda derivarse de la relación original mediante operaciones de restricción y proyección; la reconstrucción de la relación originada a partir de los fragmentos se hace mediante operaciones de reunión y unión (reunión en el caso de fragmentación vertical, y la unión en casos de fragmentación horizontal).

6. Independencia de réplica: Un sistema maneja réplica de datos si una relación dada se puede representar en el nivel físico mediante varias copias réplicas, en muchos sitios distintos. La réplica es deseable al menos por dos razones: en primer lugar, puede producir un mejor desempeño (las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con sitios remotos); en segundo lugar, también puede significar una mejor disponibilidad (un objeto estará disponible para su procesamiento en tanto esté disponible por lo menos una copia, al menos para propósitos de recuperación). La desventaja principal de las réplicas es cuando se pone al día un cierto objeto copiado, deben ponerse al día todas las réplicas de ese objeto. La réplica debe ser “transparente para el usuario final”, un sistema que maneja la réplica de los datos deberá ofrecer también una independencia de réplica (conocida también como transparencia de réplica); es decir, los usuarios deberán comportarse como si sólo existiera una copia de los datos.
7. Procesamiento distribuido de consultas: Este manejo de datos en las consultas permite las consultas eficientes desde diferentes usuarios con las características que determine el sistema; la consulta de datos es más importante en un sistema distribuido que en uno centralizado. Lo esencial es que, en una consulta donde están implicados varios sitios, habrá muchas maneras de trasladar los datos en la red de cómputo para satisfacer la solicitud, y es crucial encontrar una estrategia suficiente. Por ejemplo, una solicitud de unión de una relación Rx almacenada en el sitio X y una relación Ry almacenada en el sitio Y podría llevarse a cabo trasladando Rx a Y o trasladando Ry a X, o trasladando las dos a un tercer sitio Z.
8. Manejo distribuido de transacciones: Este manejo tiene dos aspectos principales, el control de recuperación y el control de concurrencia, cada uno de los cuales requiere un tratamiento más amplio en el ambiente distribuido. En un sistema distribuido, una sola transacción puede implicar la ejecución de programas o procesos en varios sitios (en particular puede implicar actualizaciones en varios sitios). Por esto, cada transacción está compuesta de varios agentes, donde un

agente es el proceso ejecutado en nombre de una transacción dada en determinado sitio. Y el sistema necesita saber cuándo dos agentes son parte de la misma transacción. Es importante aclarar que no puede haber un bloqueo mutuo entre dos agentes que sean parte de la misma transacción.

9. Independencia con respecto al equipo: Las instalaciones de cómputo en el mundo real por lo regular incluyen varias máquinas de diferentes marcas comerciales como IBM, DELL, HP, SUN, entre otras; por esta razón existe una verdadera necesidad de poder integrar los datos en todos esos sistemas y presentar al usuario “una sola imagen del sistema”. Por tanto, conviene ejecutar el mismo SGBD en diferentes equipos, y además lograr que esos diferentes equipos se integren en un sistema distribuido.
10. Independencia con respecto al sistema operativo: Es necesario y conveniente no sólo de poder ejecutar el mismo SGBD en diferentes equipos, sino también poder ejecutarlo en diferentes sistemas operativos y lograr que una versión LINUX y una WINDOWS participen todas en el mismo sistema distribuido.
11. Independencia con respecto a la red: Si el sistema puede manejar múltiples sitios, con equipos distintos y diferentes sistemas operativos, resulta obvia la conveniencia de manejar también varios tipos de redes de comunicación distintas.
12. Independencia con respecto al SGBD: En la independencia con respecto a su manejo, se requiere que los SGBD en los diferentes sitios manejen toda la misma interfaz; no necesitan ser por fuerza copias del mismo sistema.

4.2. ESTRUCTURA DE UN SISTEMA DISTRIBUIDO

Un sistema distribuido de base de datos consiste en un conjunto de localidades, cada una de las cuales mantiene un sistema de base de datos local. Cada localidad puede procesar transacciones locales, o bien transacciones globales entre varias localidades, requiriendo para ello comunicación entre ellas.

Las localidades pueden conectarse físicamente de diversas formas, las principales son:

- Red totalmente conectada
- Red prácticamente conectada
- Red con estructura de árbol
- Red de estrella
- Red de anillo

Las diferencias principales entre estas configuraciones son:

Coste de instalación: El coste de conectar físicamente las localidades del sistema.

Coste de comunicación: El coste en tiempo y dinero que implica enviar un mensaje desde a localidad A a la B.

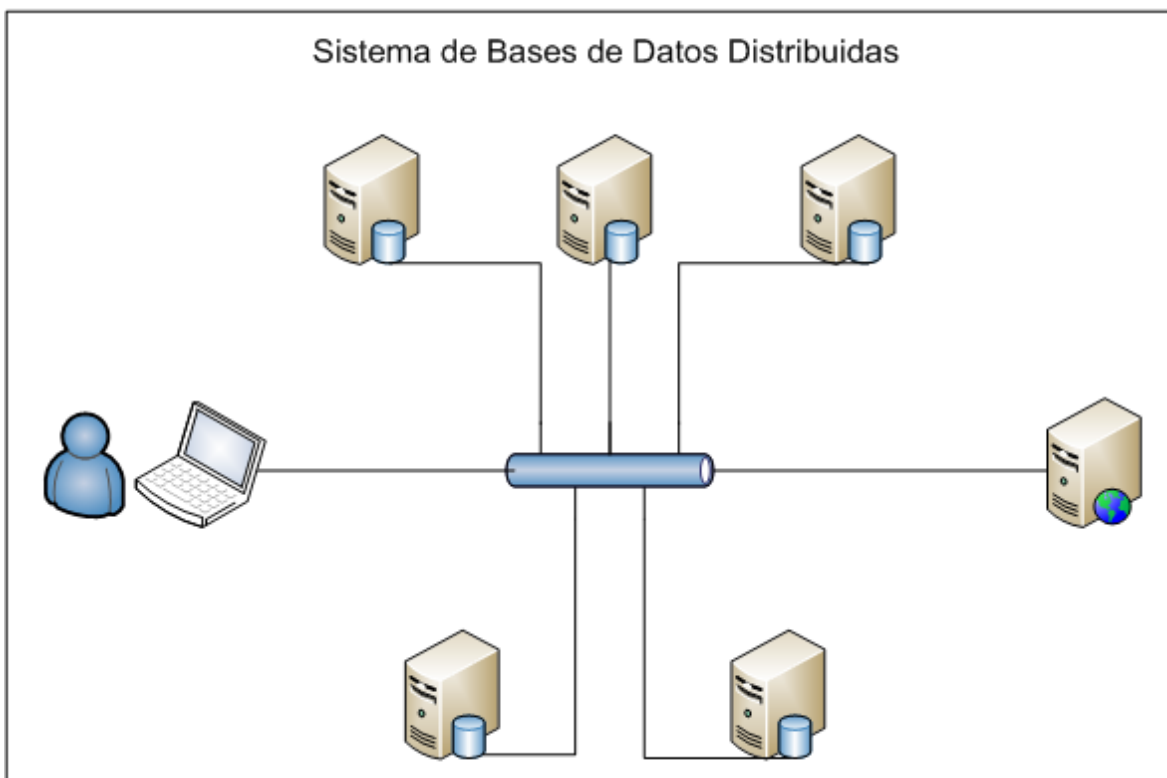
Fiabilidad: La frecuencia con que falla una línea de comunicación o una localidad.

Disponibilidad: La posibilidad de acceder a información a pesar de fallos en algunas localidades o líneas de comunicación.

Las localidades pueden estar dispersas, ya sea por un área geográfica extensa (a lo largo de un país), llamadas redes de larga distancia; o en un área reducida (en un mismo edificio), llamadas redes de área local. Para las primeras se utilizan en la comunicación líneas

telefónicas, conexiones de microondas y canales de satélites; mientras que para las segundas se utiliza cables coaxiales de banda base o banda ancha y fibra óptica.

Una Base de Datos Distribuida (BDD) es, una base de datos construida sobre una red de computadores. La información que estructura la base de datos esta almacenada en diferentes sitios en la red, y los diferentes sistemas de información que las utilizan accesan datos en distintas posiciones geográficas.



Por ende una Base de Datos Distribuida es una colección de datos que pertenecen lógicamente a un solo sistema, pero se encuentra físicamente distribuido en varios computadores o servidores de datos en una red de computadoras. Un sistema de bases de

datos distribuidas se compone de un conjunto de sitios lógicos, conectados entre sí, mediante algún tipo de red de comunicaciones, en el cual:

Cada sitio lógico puede tener un sistema de base de datos.

Los sitios han sido diseñados para trabajar en conjunto, con el fin de que un usuario de cualquier posición geográfica pueda obtener acceso a los datos desde cualquier punto de la red tal como si todos los datos estuvieran almacenados en la posición propia del usuario. Entonces, la llamada "Base de Datos Distribuida" es en realidad una especie de "objeto virtual", cuyos componentes se almacenan físicamente en varias "bases de datos reales" ubicadas en diferentes sitios. En esencia es la unión lógica de esas diferentes bases de datos.

En otras palabras, cada sitio tiene sus propias "bases de datos reales" locales, sus propios usuarios locales, sus propios SGBD y programas para la administración de transacciones y su propio administrador de comunicación de datos. Así pues, el sistema de bases de datos distribuidas puede considerarse como una especie de sociedad entre los diferentes SGBD individuales locales. Un nuevo componente de software en cada sitio realiza las funciones de sociedad necesarias; y es la combinación de este nuevo componente y el SGBD ya existente constituyen el llamado Sistema de Administración o Gestión de Bases de Datos Distribuidas - SGBDD. (En inglés DDBMS, Distributed DataBase Management System).

Desde el punto de vista del usuario final, un sistema distribuido deberá ser idéntico a un sistema no distribuido. Los usuarios de un sistema distribuido se comportan en su manipulación de información exactamente como si el sistema no estuviera distribuido. Todos los problemas de los sistemas distribuidos son de tipo interno o a nivel de realización, no pueden existir problemas de tipo externo o a nivel del usuario final.

Los datos que se encuentran distribuidos en varios sitios y que están interconectados por una red de comunicaciones tienen capacidad de procesamiento autónomo de transacciones

y hacer procesos locales. Cada sitio realiza la ejecución de al menos una transacción global, la cual requiere accesos a datos en diversos sitios.

Los principios fundamentales de un sistema de datos distribuido son:

1. Autonomía local.
2. No dependencia de un sitio central.
3. Operación continúa.
4. Independencia con respecto a la localización.
5. Independencia con respecto a la fragmentación.
6. Independencia de réplica.
7. Procesamiento distribuido de consultas.
8. Manejo distribuido de transacciones.
9. Independencia con respecto al equipo.
10. Independencia con respecto al sistema operativo.
11. Independencia con respecto a la red.
12. Independencia con respecto al SGBD.

Estos doce postulados no son todas independientes entre sí, ni tienen toda la misma importancia que cada usuario le otorgue. Sin embargo, sí son útiles como fundamento para entender la tecnología distribuida y como marco de referencia para caracterizar la funcionalidad de sistemas distribuidos específicos.

1. Autonomía local: Los sitios o posiciones de un sistema distribuido deben ser autónomos. La autonomía local significa que todas las operaciones en un sitio determinado se controlan en ese sitio; ningún sitio A deberá depender de algún otro sitio B para su buen funcionamiento (pues de otra manera el sitio A podría ser incapaz de trabajar, aunque no tenga en sí problema alguno, si cae el sitio B). La autonomía local significa que existe un propietario y un administrador local de los datos, con responsabilidad local: todos los datos pertenecen a una base de datos local, aunque los datos sean accesibles desde algún sitio distante. Todo

el manejo de la seguridad y la integridad de los datos se efectúan con control de la instalación y administración local.

2. No dependencia de un sitio central: La no dependencia de un sitio central, sería lo ideal pero si esto no se logra la autonomía local completa se vería comprometida. La dependencia de un sitio central no es práctica al menos por las siguientes razones: en primer lugar, estos sitios podrían generar un cuello de botella, y en segundo lugar, el sistema sería vulnerable; si el sitio central sufriera un desperfecto, todo el sistema dejaría de funcionar.
3. Operación continúa: En un sistema distribuido, lo mismo que en uno no distribuido, nunca debería haber necesidad de apagar o dejar de funcionar. Es decir, el sistema nunca debería necesitar apagarse para que se pueda realizar alguna operación, como añadirse un nuevo sitio o instalar una versión mejorada del SGBD en un sitio ya existente.
4. Independencia con respecto a la localización: La independencia con respecto a la localización, permite que los usuarios finales no sepan dónde están almacenados físicamente los datos, sino que trabajen como si todos los datos estuvieran almacenados en su propio sitio local. La independencia con respecto a la localización es deseable porque simplifica los sistemas de información de los usuarios y sus actividades en la terminal. Esto hace posible la migración de datos de un sitio a otro sin anular la validez de ninguno de esos sistemas o actividades. Esta función de migración permite modificar la distribución de los datos dentro de la red, en respuesta a cambios en los requerimientos de desempeño.
5. Independencia con respecto a la fragmentación: Un sistema tiene fragmentación de datos solo si es posible dividir una relación en partes o "fragmentos" para propósitos de almacenamiento físico. La fragmentación es deseable por razones de desempeño: los datos pueden almacenarse en la localidad donde se utilizan con mayor frecuencia, de manera que la mayor parte de las operaciones sean solo locales y se reduzca el tráfico en la red de cómputo. Existen en esencia dos clases de fragmentación, la fragmentación horizontal y la fragmentación vertical; estos tipos de fragmentación son correspondientes a las operaciones relacionales de restricción y proyección, respectivamente. Un fragmento puede ser cualquier sub relación que pueda derivarse de la relación original mediante

operaciones de restricción y proyección; la reconstrucción de la relación originada a partir de los fragmentos se hace mediante operaciones de reunión y unión (reunión en el caso de fragmentación vertical, y la unión en casos de fragmentación horizontal).

6. Independencia de réplica: Un sistema maneja réplica de datos si una relación dada se puede representar en el nivel físico mediante varias copias réplicas, en muchos sitios distintos. La réplica es deseable al menos por dos razones: en primer lugar, puede producir un mejor desempeño (las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con sitios remotos); en segundo lugar, también puede significar una mejor disponibilidad (un objeto estará disponible para su procesamiento en tanto esté disponible por lo menos una copia, al menos para propósitos de recuperación). La desventaja principal de las réplicas es cuando se pone al día un cierto objeto copiado, deben ponerse al día todas las réplicas de ese objeto. La réplica debe ser "transparente para el usuario final", un sistema que maneja la réplica de los datos deberá ofrecer también una independencia de réplica (conocida también como transparencia de réplica); es decir, los usuarios deberán comportarse como si sólo existiera una copia de los datos.
7. Procesamiento distribuido de consultas: Este manejo de datos en las consultas permite las consultas eficientes desde diferentes usuarios con las características que determine el sistema; la consulta de datos es más importante en un sistema distribuido que en uno centralizado. Lo esencial es que, en una consulta donde están implicados varios sitios, habrá muchas maneras de trasladar los datos en la red de cómputo para satisfacer la solicitud, y es crucial encontrar una estrategia suficiente. Por ejemplo, una solicitud de unión de una relación R_x almacenada en el sitio X y una relación R_y almacenada en el sitio Y podría llevarse a cabo trasladando R_x a Y o trasladando R_y a X , o trasladando las dos a un tercer sitio Z .
8. Manejo distribuido de transacciones: Este manejo tiene dos aspectos principales, el control de recuperación y el control de concurrencia, cada uno de los cuales requiere un tratamiento más amplio en el ambiente distribuido. En un sistema distribuido, una sola transacción puede implicar la ejecución de programas o

procesos en varios sitios (en particular puede implicar actualizaciones en varios sitios). Por esto, cada transacción está compuesta de varios agentes, donde un agente es el proceso ejecutado en nombre de una transacción dada en determinado sitio. Y el sistema necesita saber cuándo dos agentes son parte de la misma transacción. Es importante aclarar que no puede haber un bloqueo mutuo entre dos agentes que sean parte de la misma transacción.

9. Independencia con respecto al equipo: Las instalaciones de cómputo en el mundo real por lo regular incluyen varias máquinas de diferentes marcas comerciales como IBM, DELL, HP, SUN, entre otras; por esta razón existe una verdadera necesidad de poder integrar los datos en todos esos sistemas y presentar al usuario "una sola imagen del sistema". Por tanto, conviene ejecutar el mismo SGBD en diferentes equipos, y además lograr que esos diferentes equipos se integren en un sistema distribuido.
10. Independencia con respecto al sistema operativo: Es necesario y conveniente no sólo de poder ejecutar el mismo SGBD en diferentes equipos, sino también poder ejecutarlo en diferentes sistemas operativos y lograr que una versión LINUX y una WINDOWS participen todas en el mismo sistema distribuido.
11. Independencia con respecto a la red: Si el sistema puede manejar múltiples sitios, con equipos distintos y diferentes sistemas operativos, resulta obvia la conveniencia de manejar también varios tipos de redes de comunicación distintas.
12. Independencia con respecto al SGBD: En la independencia con respecto a su manejo, se requiere que los SGBD en los diferentes sitios manejen toda la misma interfaz; no necesitan ser por fuerza copias del mismo sistema.

4.3. PROCESAMIENTO DE CONSULTA

El proceso de consultas en bases de datos relacionales deja al programador de aplicaciones en un escenario distinto al anterior; la razón es el empleo de lenguajes de especificación: “si se utiliza un lenguaje de especificación el programador no tiene que diseñar ni generar un método para ejecutar la especificación o consulta requerida”, es decir el programador es introducido en un escenario “no procedural”, “no está obligado a

crear métodos ni procedimientos para obtener los datos, sólo a especificar los datos que requiere”.

Ejemplo: si en un programa de aplicación se inserta una instrucción SQL del tipo:

```
SELECT N°Matricula,Nombre,Asignatura,Nota FROM notas WHERE curso= “3°”.
```

Lo único que está aportando el programador es la especificación de los datos requeridos, pero a diferencia de la obtención de datos en un ambiente de archivos convencionales no especifica el algoritmo o método de obtención.

4.4. PROPAGACIÓN DE ACTUALIZACIONES

Asegúrese de que tiene autorización SYSADM.

Asegúrese de que todas las bases de datos locales que desea actualizar están catalogadas.

Asegúrese de que ha realizado una copia de seguridad de las bases de datos tal como se indica en Tareas previas a la actualización para servidores Db2.

Asegúrese de que ha instalado Db2 versión 11.1 y ha actualizado la instancia a Db2 versión 11.1.

Restricciones

Revise los pasos de Restricciones de actualización para servidores Db2 para la actualización de base de datos.

Procedimiento

Para actualizar una base de datos Db2 a Db2 versión 11.1:

Inicie sesión en el servidor Db2 como propietario de la instancia o como usuario con autorización SYSADM.

Opcional: Renombre o suprima los archivos de registro de **db2diag** para que se creen nuevos archivos. Asimismo, elimine o mueva a otro directorio los archivos de vuelco, los archivos de interrupción y los archivos de anotaciones cronológicas de alerta existentes en el directorio que indica el parámetro **diagpath**. De esta forma, los archivos sólo contendrán información relacionada con el proceso de actualización que le ayudará a determinar y entender cualquier problema que pudiera producirse durante la actualización de la base de datos.

Opcional: Emita el mandato **db2 LIST DATABASE DIRECTORY** para asegurarse de que la base de datos está en la lista de todas las bases de datos catalogadas en la instancia actual.

Actualice la base de datos utilizando el mandato **UPGRADE DATABASE** :

```
db2 UPGRADE DATABASE database-alias USER username USING password
```

donde *database-alias* es el nombre o el alias de la base de datos que desea actualizar y el nombre de usuario y la contraseña para autenticar un usuario con autorización SYSADM.

Para evitar la sobrecarga que supone una revinculación automática, considere la posibilidad de utilizar el parámetro **REBINDALL**, que especifica que se ejecute el mandato **REBIND** de todos los paquetes durante la actualización.

Si el mandato **UPGRADE DATABASE** falla y devuelve el mensaje de error SQLI704N con un código de razón que describe la causa de la anomalía, busque este código de error de SQL y determine la acción que se debe realizar de la lista de posibles soluciones para cada código de razón.

Una de las causas más comunes de la incorrecta ejecución de la actualización es que el espacio del archivo de anotaciones cronológicas es insuficiente, en cuyo caso se devuelve el error siguiente:

```
SQLI704N Database upgrade failed. Reason code "3".
```

Debe aumentar el tamaño del archivo de registro y volver a ejecutar el mandato **UPGRADE DATABASE** . Después de haberse completado la actualización de la base de datos, restablezca el valor de los parámetros de configuración de base de datos **logfilsiz**, **logprimary** y **logsecond**.

El mandato **UPGRADE DATABASE** devuelve otros códigos de error para casos específicos que no reciben el soporte de la actualización de la base de datos. Estos casos se describen en las restricciones de actualización.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI243W , debe descartar o renombrar SYSTOOLS.DB2LOOK_INFO .

De lo contrario, las sentencias ALTER TABLE y COPY SCHEMA no se ejecutarán.

Compruebe si existe la tabla SYSTOOLS.DB2LOOK_INFO ejecutando este mandato:

```
db2 "SELECT tabname, tabschema, definer FROM syscat.tables
```

```
WHERE tabschema = 'SYSTOOLS' AND tabname = 'DB2LOOK_INFO'"
```


Si ha creado esta tabla, cambie el nombre de ésta mediante la ejecución de la sentencia RENAME:

```
db2 RENAME SYSTOOLS.DB2LOOK_INFO TO new-table-name
```

Si no ha creado esta tabla, elimínela mediante la ejecución del mandato DROP:

```
db2 DROP TABLE SYSTOOLS.DB2LOOK_INFO
```

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y graba el mensaje de aviso ADM7535W con todos los detalles en el registro de notificaciones de administración, el mandato no ha podido renovar los atributos de espacio de tabla en la tabla de catálogo.

Sin embargo, la base de datos se ha actualizado correctamente.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y escribe el mensaje de aviso ADM4003E con todos los detalles en el registro de notificaciones de administración, el mandato no ha podido actualizar los catálogos o índices de Db2 Text Search debido a un error en un procedimiento almacenado.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y graba el mensaje de aviso ADM7534W con todos los detalles en el registro de notificaciones de administración, el mandato no ha podido renovar los atributos de espacio de tabla en la tabla de catálogo.

Sin embargo, la base de datos se ha actualizado correctamente.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y graba el mensaje de aviso ADM4102W en el registro de notificaciones de administración, califique o delimite con comillas los identificadores denominados NULL en las sentencias SQL para evitar conflictos con la palabra clave NULL.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y graba el mensaje de aviso ADM4102W en el registro de notificaciones de administración, califique o delimite con comillas los identificadores denominados NULL en las sentencias SQL para evitar conflictos con la palabra clave NULL.

Si utiliza identificadores denominados NULL para los nombres de columnas, los nombres de parámetros de rutina o los nombres de variables en una sentencia de SQL que no se han cualificado por completo o delimitado mediante comas, puede que el nombre del identificador se resuelva, en su lugar, por la palabra clave NULL. Esto daría como resultado un cambio de comportamiento respecto a las versiones anteriores. Consulte Essentials de actualización para aplicaciones de base de datos para obtener más detalles.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de aviso SQLI499W y graba el mensaje de aviso ADM9516W en el registro de notificaciones de administración, verifique que el parámetro de configuración **indexrec** se haya establecido en RESTART y emita el mandato **RESTART DATABASE** para volver a crear los índices marcados como no válidos durante la actualización de la base de datos.

De otro modo, la recreación de los índices se iniciará cuando tenga lugar el primer acceso a la tabla, y puede que experimente una degradación no esperada del tiempo de respuesta.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de error SQL0473N , debe invertir la migración de la base de datos y volver a crear todos los tipos de datos definidos por el usuario que utilizan un nombre de tipo de datos incorporado del sistema con un nombre distinto que no está restringido.

Para evitar el error del mandato **UPGRADE DATABASE** , vuelva a crear estos tipos de datos definidos por el usuario durante Verificación de que las bases de datos están preparadas para la actualización.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de error DBT5512 , el mandato no ha podido actualizar la base de datos porque el ID de un objeto de gestión de carga de trabajo está en conflicto con un ID reservado por el sistema. Para actualizar la base de datos, lleve a cabo las siguientes acciones:

Genere las sentencias DDL para volver a crear los objetos de gestión de carga de trabajo emitiendo el mandato **db2look** con el parámetro **-wlm** .

Descarte todos los objetos de gestión de carga de trabajo de la base de datos.

Resuelva todos los problemas notificados por el mandato **db2ckupgrade** y bloquee la actualización de la base de datos.

Actualice la base de datos.

Vuelva a crear el objeto de gestión de carga de trabajo en la base de datos actualizada emitiendo las sentencias DDL que ha generado con el mandato **db2look** .

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de error SQLI700N , debe invertir la migración de la base de datos y volver a crear objetos de base de datos que

utilicen nombres de esquema restringidos con un nombre de esquema que no esté restringido.

Para evitar el error del mandato **UPGRADE DATABASE** , vuelva a crear estos objetos de base de datos durante Verificación de que las bases de datos están preparadas para la actualización

.

Si el mandato **UPGRADE DATABASE** devuelve el mensaje de error ADM4003E , actualice manualmente el catálogo e índices de Db2 Text Search.

Para obtener detalles,

consulte SYSTS_UPGRADE_CATALOG y SYSTS_UPGRADE_INDEX.

Compare los valores de configuración de la base de datos después de haber realizado la actualización con los valores de configuración que tenía antes de la actualización de la base de datos. Verifique que los valores y las informaciones sobre la base de datos siguientes sean los mismos:

- Valores de los parámetros de configuración de la base de datos
- Información de espacios de tablas
- Información de paquetes sólo para sus aplicaciones

No es necesario que compruebe la información sobre paquetes para los paquetes generados por el sistema. La información relacionada con los paquetes que genera el sistema puede cambiar después de la actualización.

Verifique que la actualización de las bases de datos ha sido satisfactoria. Establezca una conexión con las bases de datos actualizadas y emita una consulta sencilla:

```
db2 connect to sample
```

```
Database Connection Information
```

```
Database server      = DB2/AIX64 10.1.0
```

```
SQL authorization ID = TESTDB2
```

```
Local database alias = SAMPLE
```

```
db2 "select * from syscat.dbauth"
```

También existe la opción, si los archivos de ejemplo están instalados, de ejecutar el script `testdata.db2`:

```
cd samplefile-dir-clp
```

```
db2 connect to sample
```

```
db2 -tvf testdata.db2
```

donde *samplefile-dir-clp* es `DB2DIR/samples/clp` en Linux® y UNIX y `DB2DIR\samples\clp` en Windows, `DB2DIR` representa la ubicación especificada durante la instalación de Db2 versión 11.1 , y `sample` es el nombre de la base de datos.

4.5. CONTROL DE CONCURRENCIA

La mayoría de las bases de datos se utilizan en entornos multi-usuario, en los que muchos clientes utilizando la misma aplicación, o muchas aplicaciones cada una con uno o muchos clientes acceden a la misma base de datos. Cada una de esas aplicaciones enviará consultas al gestor, y normalmente cada hilo de ejecución será una transacción diferente.

En la mayoría de los sistemas operativos actuales, las diferentes tareas o hilos se ejecutan de forma intercalada (incluso en el caso de máquinas con varios procesadores). Es decir, el

sistema operativo decide por su cuenta cuando suspender una de las tareas y darle un poco de tiempo de ejecución a otra. Si hay tareas simultáneas o concurrentes sobre la misma base de datos, esta intercalación puede resultar en que las lecturas y escrituras de las diferentes tareas o aplicaciones en el medio físico se realicen en cualquier orden y secuencia.

El acceso simultáneo descrito puede dar como resultados información inconsistente o simplemente incorrecta, dependiendo de la mala o buena suerte que tengamos en la intercalación de las lecturas y escrituras simultáneas. Esta problemática ha llevado a diseñar e implementar diferentes estrategias de **control de concurrencia**, que se encargan de evitar todos esos problemas, de modo que los desarrolladores de las aplicaciones pueden “olvidarse” de ellos al escribir su código.

Por ejemplo, si tenemos una estructura de tablas relacional que incluye las siguientes:

PEDIDO(id, num-cliente, id-prod, cantidad, precio)

PRODUCTO(id-prod, nombre, ..., stock)

...

Pueden ocurrir diferentes problemas relacionados con la escritura simultánea con otras escrituras o lecturas, incluyendo los siguientes:

1. Dos sentencias **UPDATE** que actualicen un mismo producto decrementando el stock del mismo en una unidad podrían terminar en que una de ellas no se realizase. Si pensamos en un **UPDATE** como una secuencia de una lectura y una escritura, puede que ambos **UPDATE** hagan la lectura, por ejemplo, de un stock de 10, y

después las escrituras, decrementan ese dato, quedando el resultado en 9, mientras que lo correcto era un resultado de 8.

2. Supongamos una sentencia que primero comprueba que hay stock del producto P, y después inserta un nuevo PEDIDO de diez unidades del producto P, que tiene un stock de 10, seguido de un UPDATE al stock por esa cantidad. Puede que otra inserción de un pedido se ejecute antes del UPDATE pero después de la comprobación, haciendo quedar el stock del producto en negativo.

Existen varias técnicas para controlar la concurrencia. Los bloqueos son los más conocidos, aunque también se utiliza el control multi-versión y otras técnicas como las marcas de tiempo.

Los bloqueos como solución al problema de la concurrencia

Una forma de controlar la concurrencia es hacer que cada transacción deba adquirir un derecho de acceso exclusivo a cada fragmento de datos que necesite modificar. A estos “derechos” se les denomina bloqueos.

Bloqueos binarios

La forma más simple de bloquear es utilizar bloqueos binarios. En un bloqueo binario, cada transacción debe solicitar el bloqueo de cada fragmento de datos A que vaya a utilizar antes de acceder a él (sea para leerlo o escribirlo), mediante una operación bloquear(A). Deberá liberar todos los bloqueos, mediante una operación desbloquear(A) de modo que otras tareas puedan tomarlos.

Este sistema de bloqueos tiene una implementación muy simple, ya que solo requiere mantener una tabla que indica qué partes de los datos está bloqueada y por qué transacción.

Bloqueos de lectura/escritura

El sistema de bloqueos binarios es simple pero demasiado restrictivo, ya que no permite que dos transacciones que van a leer el mismo fragmento de datos A lo hagan simultáneamente, cuando en realidad, no puede haber problemas en varios lectores simultáneos. Los bloqueos de lectura/escritura hacen más débil la restricción permitiendo la siguiente compatibilidad de bloqueos.

	LECTURA	ESCRITURA
LECTURA	Compatible	Incompatible
ESCRITURA	Incompatible	Incompatible

En este caso, las operaciones que las transacciones deben realizar son tres: `desbloquear(A)` y `bloquear_para_lectura(A)` o `bloquear_para_escritura(A)`.

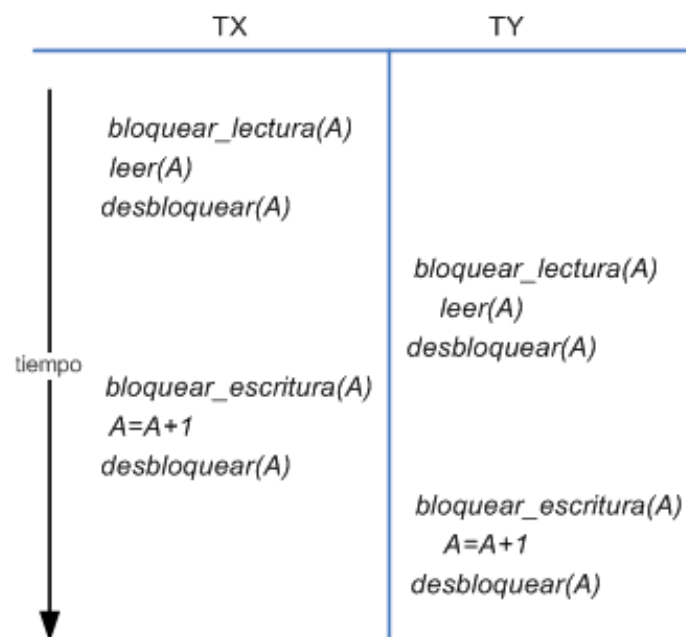
Nótese que esas llamadas se implementan de diferentes formas en diferentes gestores de bases de datos. Por ejemplo, en MySQL, tanto las solicitudes de bloqueos como las liberaciones se realizan mediante una sola llamada del API de los gestores de almacenamiento:

```
store_lock(THD *thd, THR_LOCK_DATA **to, enum thr_lock_type lock_type)
```


Cada llamada a `store_lock` utiliza el manejador de una tabla `thd` concreta, y se indica la información de los datos a bloquear mediante la variable `to`, y el tipo de bloqueo mediante `lock_type` (el número de tipos definidos en MySQL es muy grande, ya que cubre todos los tipos de bloqueo que implementan los múltiples gestores de almacenamiento disponibles).

Serialización de los bloqueos de lectura/escritura

La serialización de las operaciones de lectura y escritura consiste en ordenar esas operaciones para un conjunto de transacciones concurrentes de modo que los resultados de las operaciones sean correctos. Por ejemplo, si tenemos las siguientes transacciones X e Y, puede darse la siguiente situación.



En esa situación, la ejecución de TX y TY hace que el dato original sólo se haya incrementado una vez, cuando tenía que haberse incrementado dos veces. Sin embargo, si hubiésemos tenido suerte y todas las operaciones de TX se hubiesen realizado antes que las de TY, el resultado habría sido correcto.

La conclusión es que el mero mecanismo de los bloqueos garantiza el acceso exclusivo a un dato o fragmento de información (evitando ciertos problemas), pero los problemas asociados a la intercalación de las operaciones compuestas aún pueden darse.

Por lo tanto, hace falta alguna política o mecanismo de adquisición y liberación de bloqueos que permita hacer las operaciones serializables.

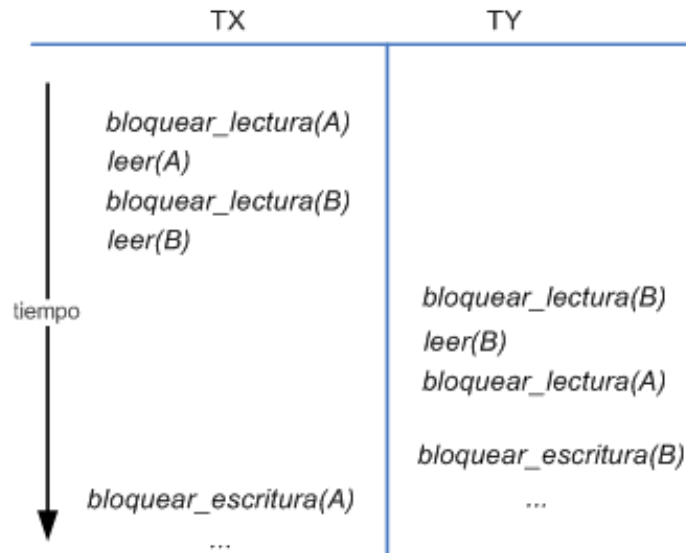
El bloqueo en dos fases permite la serialización

El protocolo de bloqueo en dos fases fuerza a las transacciones cuando todas las operaciones de adquisición de bloqueos (`bloquear_lectura`, `bloquear_escritura`) preceden a la primera operación de desbloqueo (`desbloquear`). Dicho de otro modo, primero hay que adquirir todos los bloqueos, y después se pueden liberar.

Cuando se utiliza el protocolo de bloqueo en dos fases, puede demostrarse que la ejecución será serializable.

Inconvenientes de los bloqueos y la serialización

Un problema del protocolo de bloqueo en dos fases es que puede llevar a situaciones de interbloqueo. La siguiente es una secuencia de operaciones que lleva al interbloqueo, pero cumple perfectamente el protocolo de bloqueo en dos fases.



El inconveniente está en que puede que en la fase de adquisición de bloqueos (fase de expansión), más de una transacción esté interesada en los mismos dos fragmentos de datos, y la mala suerte nos lleve a una situación en la que ambos quedan suspendidos, sin posibilidad de avanzar.

¿Bloqueos más grandes o más pequeños?

Un aspecto que aún no se ha tratado en la discusión anterior es cuán “grandes” son los elementos de datos que se deben bloquear. Una opción posible es bloquear tablas enteras. Esto hace que la gestión de los bloqueos sea simple y tenga poca sobrecarga (solo hay que guardar el estado de bloqueo de las N tablas). No obstante, esto impide que dos transacciones que van a manipular filas diferentes de una tabla puedan progresar en paralelo.

Una segunda opción es utilizar bloqueos al nivel de las filas. En este caso, se hacen mayores las posibilidades de concurrencia, pero por otro lado, hay que mantener mucha más información sobre los bloqueos (ya que el número de filas es en general muy grande respecto al número de tablas), y el servidor se sobrecarga más con la gestión de los bloqueos.

Hay gestores de bases de datos que permiten seleccionar el tipo de bloqueo que queremos para nuestra base de datos. Por ejemplo, en MySQL hay gestores de almacenamiento que ofrecen bloqueo a nivel de fila, y otros bloqueo a nivel de tabla.

No obstante lo anterior, hay sentencias que siempre producirán un bloqueo de tabla, como por ejemplo una sentencia `ALTER TABLE`.

Guardando “instantáneas” de los datos: el Control Multi-versión

El protocolo de bloqueo en dos fases limita considerablemente las posibilidades de concurrencia. Si observamos los problemas que causan los bloqueos no serializados, veremos que muchos de los problemas están en que una transacción lee un cierto dato y antes de escribir el resultado, otra transacción lee el dato “antiguo”. En ese momento, cada transacción trabaja con un estado de información inconsistente.

Para paliar esos problemas, pero permitir la mayor concurrencia posible, se han diseñado los protocolos de control multi-versión. La idea básica es que cuando una transacción modifica un dato, se crea una nueva versión del mismo, pero se guarda la anterior. De este modo, al acabar la ejecución de las transacciones, se puede utilizar para cada una de ellas la versión de los datos “que hace la ejecución correcta”, es decir, que hace la ejecución serializable.

BIBLIOGRAFÍA BÁSICA Y COMPLEMENTARIA

- <https://itm201533.webnode.es/products/tipos-de-chasis/>
- <https://www.ejemplos.co/10-ejemplos-de-perifericos-de-comunicacion/#ixzz6T42kuGU6>
- <https://www.caracteristicas.co/tecnologia/#ixzz6T3xKUIDH>
- <https://www.profesionalreview.com/ssd/>
- <https://concepto.de/placa-madre/#ixzz6SmKliI Ph>
- <http://www3.uji.es/~mmarques/f47/teoria/tema7.pdf>
- ISO/IEC 27001:2005 - Information technology -- Security techniques http://www.iso.org/iso/catalogue_detail?Csnumber=42103
- ISO/IEC 17799:2005 - Information technology -- Security techniques http://www.iso.org/iso/catalogue_detail?Csnumber=39612
- Malware - Ataque a la Base de Datos [en] <http://ataquebd.blogspot.mx/>
- Inyección de código SQL - MSDN – Microsoft [en] <http://msdn.microsoft.com/es-es/library/ms161953.aspx>
- Escolano F. “Inteligencia Artificial”, Editorial Paraninfo, 2003
- Aguilera L “Seguridad Informática” 2010, Madrid, Editorial Editex, S.A.
- <http://ict.udlap.mx/people/carlos/is341/bases10.html>
- (PDF) Principios Básicos de Seguridad en Bases de Datos. Available from: https://www.researchgate.net/publication/279983428_Principios_Basicos_de_Seguridad_en_Bases_de_Datos [accessed Dec 14 2018].
- <http://didepa.uaemex.mx/clases/Manuales/MySQL/MySQL-La%20biblia%20de%20mysql.pdf>
- <https://riunet.upv.es/bitstream/handle/10251/11166/memoria.pdf?sequence=1>
- <https://elvex.ugr.es/decsai/information-systems/slides/31%20Data%20Access%20-%20Distributed.pdf>
- <https://sistemascomputacionaleslahuelilpan.files.wordpress.com/2012/03/bases-dedatos-distribuidas.pdf>