UDS

Mi Universidad

LIBRO

Bases de Datos II

Ingeniería en Sistemas Computacionales

Octavo Cuatrimestre

Enero - Abril

Marco Estratégico de Referencia

Antecedentes históricos

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 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 tardes.

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 julio de 1996 como chofer de transporte escolar, Karla Fabiola Albores Alcázar se integró en la docencia en 1998, Martha Patricia Albores Alcázar en el departamento de cobranza 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, 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 del Grupo Educativo Albores Alcázar S.C. 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.

Estrategia de Enseñanza y aprendizaje

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.

Criterios de evaluación:

No	Concepto	Porcentaje
1	Trabajos Escritos	10%
2	Actividades Áulicas	20%
3	Trabajos en plataforma Educativa	20%
4	Examen	50%
Total de Criterios de evaluación		100%

INDICE

Unidad I

Recuperación

11. Concepto

1.2. Transacciones

1.3. Fallas de transacción

1.4. Fallas de sistemas

1.5. Falas en el medio

1.6. Recuperación empleando DML (lenguaje de manipulación de datos) con un DBMS comercial.

1.7. Integridad

1.7.1. Definición

1.7.2. Reglas de integridad

1.7.3. Reglas de integridad de Dominio

1.7.4. Reglas de integridad de relación

1.7.5. Integridad empleando DML con un DBMS comercial

Unidad II

Concurrencia

2.1. Definición

2.2. Problemas que se presentan (actualización, perdida, etc.)

2.3. Seriabilidad

2.4. Mecanismo de seguros

2.4.1. Tipos de seguros

2.4.2. Protocolos

2.4.3. Dead Lock

2.4.4. Técnicas para prevenirlo

2.4.5. Técnicas para deshacerlo

2.5. Etiquetas de tiempo

2.6. Operaciones de seguros empleando DML con un DBMS comercial

Unidad III

Seguridad

3.1. Concepto

3.2. Identidad y autentificación

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

Unidad I

Recuperación

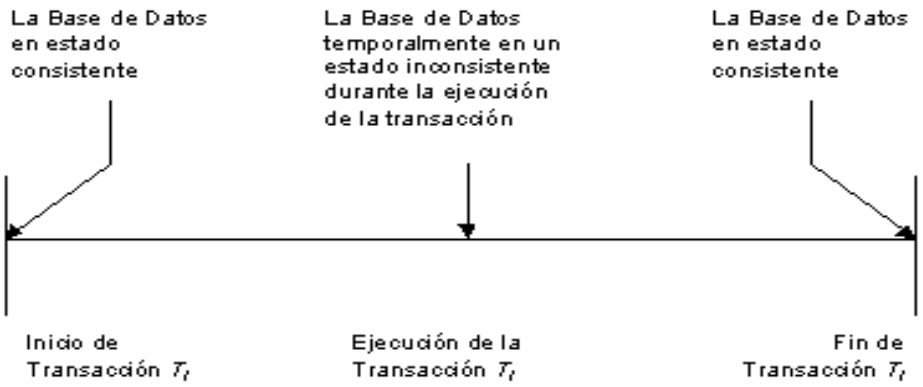
Manejo de transacciones distribuidas

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.

Definición de una transacción

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.



Un modelo de transacción

Ejemplo 1: 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*1.1

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*1.1

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 + 1

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.

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 esisten 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 + 1

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.

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 denominan 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 ☐ WS).

Ejemplo 4. Los conjuntos de lectura y escritura de la transacción del Ejemplo 3 son:

RS[Reservación] = { FLIGHT.STSOLD, FLIGHT.CAP }

WS[Reservación] = { FLIGHT.STSOLD, FC.FNO, FC.DATE, FC.NAME, FC.SPECIAL }

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 ☐ {read, write} y O_j es una operación atómica, esto es, se ejecuta como una unidad indivisible. Se denota por OS_i = ☐ $_j$ 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 ☐ {abort, commit}.

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

1. ☐ $_i$ = OS_i ☐ {N_i}
2. Para cualesquiera dos operaciones, O_{ij}, O_{ik} ☐ 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} ☐ 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 ☐ x + y

Write(x)

Commit

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

$\square$ = { 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.

Propiedades de las transacciones

La discusión en la sección previa clarifica el concepto de transacción. Sin embargo, aun 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.

Las transacciones proporcionan una ejecución atómica y confiable en presencia de fallas, una ejecución correcta en presencia de accesos de usuario múltiples y un manejo correcto de réplicas (en el caso de que se soporten).

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 diferencias también como transacciones de corta y larga vida. Las transacciones en línea se caracterizan por tiempos de respuesta muy cortos y por accesar una porción relativamente pequeña de la base de datos. Por otro lado, las transacciones de tipo batch toman tiempos relativamente largos y accesan 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.

Estructura de transacciones

Las transacciones planas consisten de una secuencia de operaciones primitivas encerradas entre las palabras clave begin y end. Por ejemplo,

Begin_transaction Reservación

. .

end.

En las transacciones anidadas las operaciones de una transacción pueden ser así mismo transacciones. Por ejemplo,

Begin_transaction Reservación

. . .

Begin_transaction Vuelo

end. {Vuelo}

Begin_transaction Hotel

end.

end.

Una transacción anidada dentro de otra transacción conserva las mismas propiedades que la de sus padres, esto implica, que puede contener así mismo transacciones dentro de ella. Existen restricciones obvias en una transacción anidada: debe empezar después que su padre y debe terminar antes que él. Más aún, el commit de una subtransacción es condicional al commit de su padre, en otras palabras, si el padre de una o varias transacciones aborta, las subtransacciones hijas también serán abortadas.

Las transacciones anidadas proporcionan un nivel más alto de concurrencia entre transacciones. Ya que una transacción consiste de varias transacciones, es posible tener más concurrencia dentro de una sola transacción. Así también, es posible recuperarse de fallas de manera independiente de cada subtransacción. Esto limita el daño a un parte más pequeña de la transacción, haciendo que costo de la recuperación sea menor.

En el segundo punto de vista se considera el orden de las lecturas y escrituras. Si las acciones de lectura y escritura pueden ser mezcladas sin ninguna restricción, entonces, a este tipo de transacciones se les conoce como generales. En contraste, si se restringe o impone que un dato deber ser leído antes de que pueda ser escrito entonces se tendrán transacciones restringidas. Si las transacciones son restringidas a que todas las acciones de lectura se realicen antes de las acciones de escritura entonces se les conoce como de dos pasos. Finalmente, existe un modelo de acción para transacciones restringidas en donde se aplica aún más la restricción de que cada par <read,write> tiene que ser ejecutado de manera atómica.

Ejemplo 6. Los siguientes son algunos ejemplos de los modelos de transacción mencionados antes.

General: T_1: { R(x), R(y), W(y), R(z), W(x), W(z), W(w), C}

Dos pasos: T_2: { R(x), R(y), R(z), W(x), W(y), W(z), W(w), C}

Restringida: T_3: { R(x), R(y), W(y), R(z), W(x), W(z), R(w), W(w), C}

Restringida y de dos pasos:

T_4: { R(x), R(y), R(z), R(w), W(y), W(x), W(z), W(w), C}

Acción: T_5: { [R(x), W(x)], [R(y), W(y)], [R(z), W(z)], [R(w), W(w)], C}

note que cada par de acciones encerrado en [ ] se ejecuta de manera atómica

Aspectos relacionados al procesamiento de transacciones

Los siguientes son los aspectos más importantes relacionados con el procesamiento de transacciones:

1. Modelo de estructura de transacciones. Es importante considerar si las transacciones son planas o pueden estar anidadas.
2. Consistencia de la base de datos interna. Los algoritmos de control de datos semántico tienen que satisfacer siempre las restricciones de integridad cuando una transacción pretende hacer un commit.
3. Protocolos de confiabilidad. En transacciones distribuidas es necesario introducir medios de comunicación entre los diferentes nodos de una red para garantizar la atomicidad y

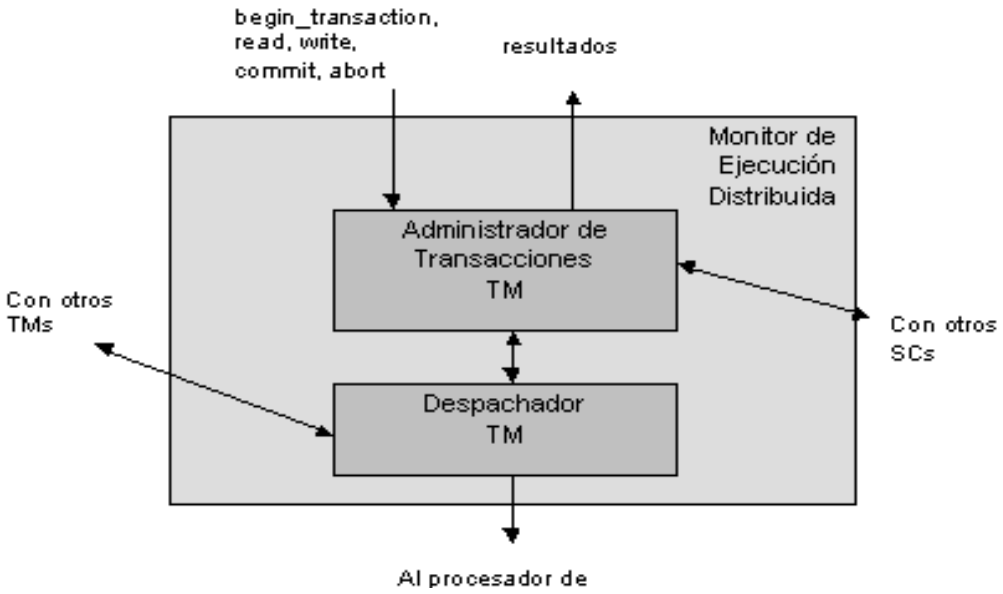
durabilidad de las transacciones. Así también, se requieren protocolos para la recuperación local y para efectuar los compromisos (commit) globales.

4. Algoritmos de control de concurrencia. Los algoritmos de control de concurrencia deben sincronizar la ejecución de transacciones concurrentes bajo el criterio de correctitud. La consistencia entre transacciones se garantiza mediante el aislamiento de las mismas.
5. Protocolos de control de réplicas**.** El control de réplicas se refiere a cómo garantizar la consistencia mutua de datos replicados. Por ejemplo se puede seguir la estrategia read-one-write-all (ROWA).

Incorporación del manejador de transacciones a la arquitectura de un smbd

El monitor de ejecución distribuida consiste de dos módulos: El administrador de transacciones (TM) y el despachador (SC). El administrador de transacciones es responsable de coordinar la ejecución en la base de datos de las operaciones que realiza una aplicación. El despachador, por otra parte, es responsable de implementar un algoritmo específico de control de concurrencia para sincronizar los accesos a la base de datos.

Un tercer componente que participa en el manejo de transacciones distribuidas es el administrador de recuperación local cuya función es implementar procedimientos locales que le permitan a una base de datos local recuperarse a un estado consistente después de una falla.

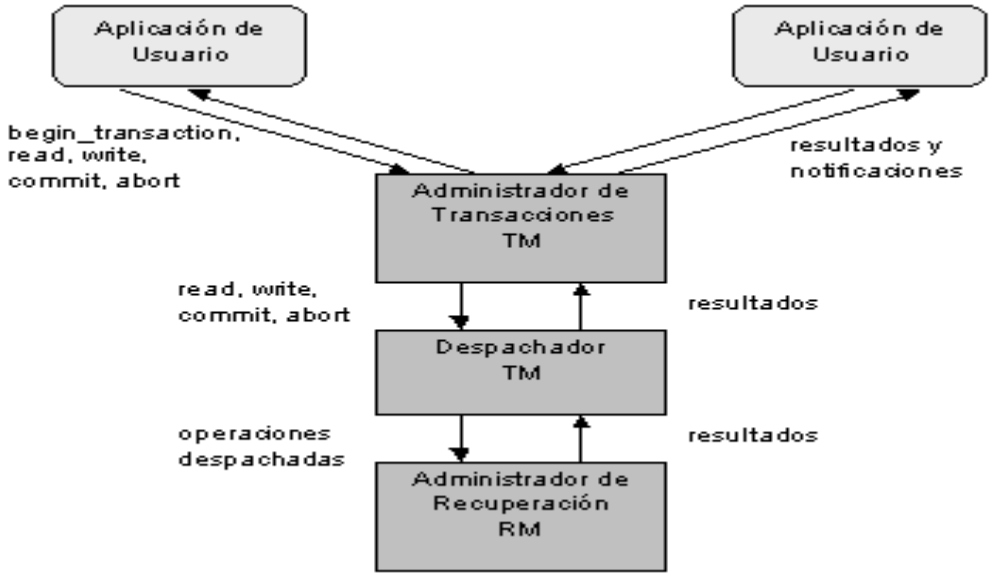


Un modelo de administrador de transacciones.

Los administradores de transacciones implementan una interfaz para los programas de aplicación que consiste de los comandos:

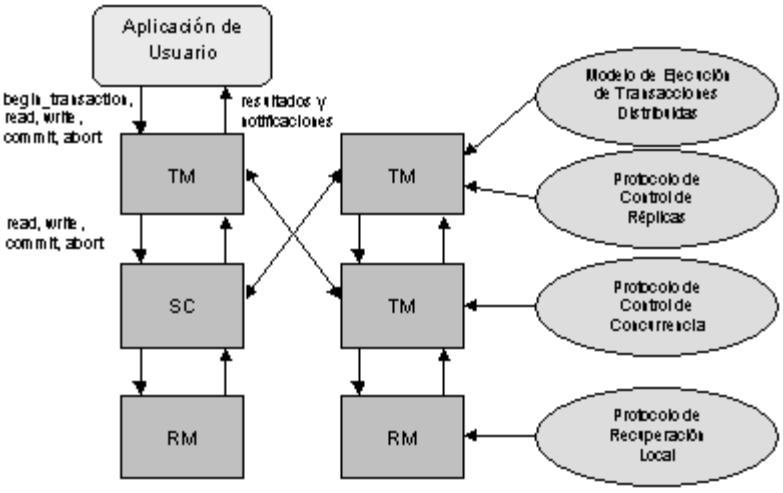
1. Begin_transaction.
2. Read.
3. Write.
4. Commit.
5. Abort.

Las modificaciones requeridas en la arquitectura para una ejecución distribuida.



Ejecución centralizada de transacciones.

Protocolos de comunicación necesarios para el manejo de transacciones distribuidas.



Ejecución distribuida de transacciones.

Anuncios

La integridad referencial es un sistema de reglas que utilizan la mayoría de las bases de datos relacionales para asegurarse que los registros de tablas relacionadas son válidos y que no se borren o cambien datos relacionados de forma accidental produciendo errores de integridad.

Primero repasemos un poco los tipos de relaciones.

Tipos de relaciones.

Entre dos tablas de cualquier base de datos relacional pueden haber dos tipos de relaciones, relaciones uno a uno y relaciones uno a muchos:

- Relación Uno a Uno: Cuando un registro de una tabla sólo puede estar relacionado con un único registro de la otra tabla y viceversa.

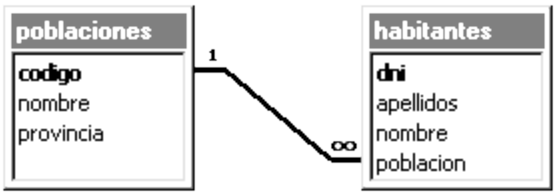
Por ejemplo: tenemos dos tablas una de profesores y otra de departamentos y queremos saber qué profesor es jefe de qué departamento, tenemos una relación uno a uno entre las dos tablas ya que un departamento tiene un solo jefe y un profesor puede ser jefe de un solo departamento.

- Relación Uno a Varios: Cuando un registro de una tabla (tabla secundaria) sólo puede estar relacionado con un único registro de la otra tabla (tabla principal) y un registro de la tabla principal puede tener más de un registro relacionado en la tabla secundaria, en este caso se suele hacer referencia a la tabla principal como tabla 'padre' y a la tabla secundaria como tabla 'hijo', entonces la regla se convierte en 'un padre puede tener varios hijos pero un hijo solo tiene un padre (regla más fácil de recordar).

Por ejemplo: tenemos dos tablas una con los datos de diferentes poblaciones y otra con los habitantes, una población puede tener más de un habitante, pero un habitante

pertenecerá (estará empadronado) en una única población. En este caso la tabla principal será la de poblaciones y la tabla secundaria será la de habitantes. Una población puede tener varios habitantes pero un habitante pertenece a una sola población. Esta relación se representa incluyendo en la tabla 'hijo' una columna que se corresponde con la clave principal de la tabla 'padre', esta columna es lo denominamos clave foránea (o clave ajena o clave externa).

Una clave foránea es pues un campo de una tabla que contiene una referencia a un registro de otra tabla. Siguiendo nuestro ejemplo en la tabla habitantes tenemos una columna población que contiene el código de la población en la que está empadronado el habitante, esta columna es clave ajena de la tabla habitantes, y en la tabla poblaciones tenemos una columna codigo de poblacion clave principal de la tabla.



- Relación Varios a Varios: Cuando un registro de una tabla puede estar relacionado con más de un registro de la otra tabla y viceversa. En este caso las dos tablas no pueden estar relacionadas directamente, se tiene que añadir una tabla entre las dos que incluya los pares de valores relacionados entre sí.

Por ejemplo: tenemos dos tablas una con los datos de *clientes* y otra con los *artículos* que se venden en la empresa, un cliente podrá realizar un pedido con varios artículos, y un artículo podrá ser vendido a más de un cliente.

No se puede definir entre *clientes* y *artículos*, hace falta otra tabla (por ejemplo una tabla de pedidos) relacionada con clientes y con artículos. La tabla pedidos estará relacionada con cliente por una relación uno a muchos y también estará relacionada con artículos por un relación uno a muchos.

Integridad referencial

Cuando se define una columna como clave foránea, las filas de la tabla pueden contener en esa columna o bien el valor nulo (ningún valor), o bien un valor que existe en la otra tabla, un error sería asignar a un habitante una población que no está en la tabla de poblaciones. Eso es lo que se denomina integridad referencial y consiste en que los datos que referencian otros (claves foráneas) deben ser correctos. La integridad referencial hace que el sistema gestor de la base de datos se asegure de que no hayan en las claves foráneas valores que no estén en la tabla principal.

La integridad referencial se activa en cuanto creamos una clave foránea y a partir de ese momento se comprueba cada vez que se modifiquen datos que puedan alterarla.

¿Cuándo se pueden producir errores en los datos?

- Cuando insertamos una nueva fila en la tabla secundaria y el valor de la clave foránea no existe en la tabla principal insertamos un nuevo habitante y en la columna *poblacion* escribimos un código de poblacion que no está en la tabla de poblaciones (una población que no existe).

- Cuando modificamos el valor de la clave principal de un registro que tiene 'hijos', modificamos el codigo de Valencia, sustituimos el valor que tenía (1) por un nuevo valor (10), si Valencia tenía habitantes asignados, qué pasa con esos habitantes, no pueden seguir teniendo el codigo de población 1 porque la población 1 ya no existe, en este caso hay dos alternativas, no dejar cambiar el codigo de Valencia o bien cambiar el codigo de población de todos los habitantes de Valencia y asignarles el código 10.

- Cuando modificamos el valor de la clave foránea, el nuevo valor debe existir en la tabla principal. Por ejemplo cambiamos la población de un habitante, tenía asignada la población 1 (porque estaba empadronado en valencia) y ahora se le asigna la población 2 porque cambia de lugar de residencia. La población 2 debe existir en la tabla de poblaciones.

- Cuando queremos borrar una fila de la tabla principal y ese registro tiene 'hijos', por ejemplo queremos borrar la población 1 (Valencia) si existen habitantes asignados a la población 1, estos no se pueden quedar con el valor 1 en la columna población porque tendrían asignada una población que no existe. En este caso tenemos dos alternativas, no dejar borrar la población 1 de la tabla de poblaciones, o bien borrarla y poner a valor nulo el campo poblacion de todos sus 'hijos'.

Asociada a la integridad referencial están los conceptos de actualizar los registros en cascada y eliminar registros en cascada.

Actualización y borrado en cascada

El actualizar y/o eliminar registros en cascada, son opciones que se definen cuando definimos la clave foránea y que le indican al sistema gestor qué hacer en los casos comentados en el punto anterior.

- Actualizar registros en cascada:

Esta opción le indica al sistema gestor de la base de datos que cuando se cambie un valor del campo clave de la tabla principal, automáticamente cambiará el valor de la clave foránea de los registros relacionados en la tabla secundaria.

Por ejemplo, si cambiamos en la tabla de poblaciones (la tabla principal) el valor 1 por el valor 10 en el campo codigo (la clave principal), automáticamente se actualizan todos los

habitantes (en la tabla secundaria) que tienen el valor 1 en el campo poblacion (en la clave ajena) dejando 10 en vez de 1.

Si no se tiene definida esta opción, no se puede cambiar los valores de la clave principal de la tabla principal. En este caso, si intentamos cambiar el valor 1 del codigo de la tabla de poblaciones, no se produce el cambio y el sistema nos devuelve un error o un mensaje que los registros no se han podido modificar por infracciones de clave.

- Eliminar registros en cascada:

Esta opción le indica al sistema gestor de la base de datos que cuando se elimina un registro de la tabla principal automáticamente se borran también los registros relacionados en la tabla secundaria.

Por ejemplo: Si borramos la población Onteniente en la tabla de poblaciones, automáticamente todos los habitantes de Onteniente se borrarán de la tabla de habitantes.

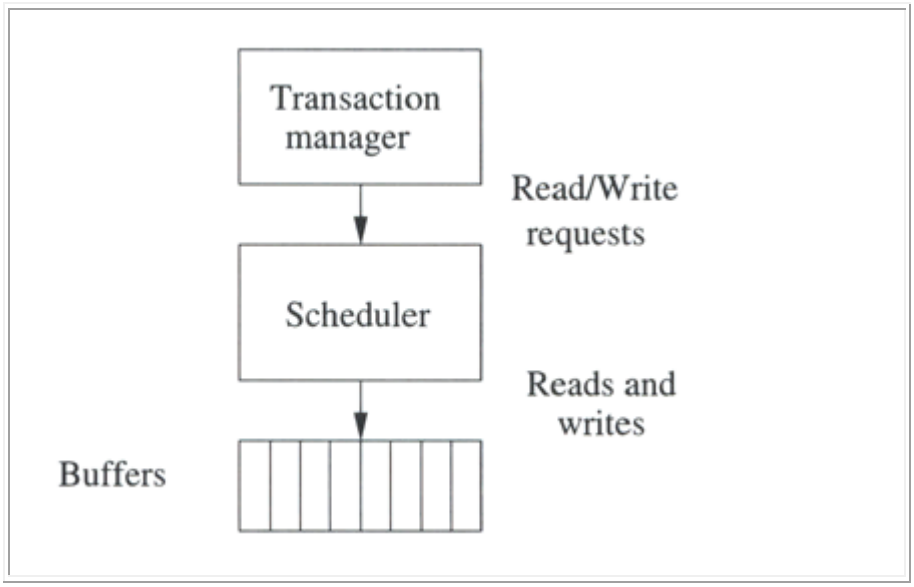
Si no se tiene definida esta opción, no se pueden borrar registros de la tabla principal si estos tienen registros relacionados en la tabla secundaria. En este caso, si intentamos borrar la población Onteniente, no se produce el borrado y el sistema nos devuelve un error o un mensaje que los registros no se han podido eliminar por infracciones de clave.

Unidad II

Control de concurrencia

El control de accesos concurrentes y específicamente de transacciones concurrentes es manejado por un módulo del dbms llamado "scheduler".

Es importante recordar que muchos de los datos de la base no se encuentran nada más en disco, sino tambien en los buffers de memoria, de ahí que el scheduler interactúa con ellos y en su defecto solicita la lectura de los datos del disco.



Scheduler del DBMS

El calendarizador crea agendas, secuencias ordenadas de las acciones tomadas por una o más transacciones.

El siguiente ejemplo muestra 2 transacciones cuya única característica de consistencia es que A=B ya que ambas operaciones son iguales para ambos elementos. Por otro lado sabemos que si las transacciones son ejecutadas aisladamente la consistencia se preservará.

T_1	T_2
READ(A,t)	READ(A,s)
t := t+100	s := s*2
WRITE(A,t)	WRITE(A,s)
READ(B,t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

2 transacciones

Métodos de control de concurrencia

Protocolos basados en técnicas de bloqueo

Un bloqueo es una operación usada para restringir las operaciones que se pueden aplicar sobre la base de datos. Existen varios tipos de bloqueo: binarios, compartidos, exclusivos, y bloqueos de certificación.

Bloqueos binarios

Poseen dos valores posibles, bloqueados y desbloqueados. Cada elemento de la base de datos tiene un bloqueo distinto. El bloqueo señala si una transacción está en ejecución sobre el elemento o está libre para que se pueda operar con él.

Bloqueos de lectura/escritura

Puede tener tres posibles posiciones: libre, bloqueado para lectura, y bloqueado para escritura. De esta forma, más de una transacción puede tener un mismo elemento de datos bloqueado para lectura, pero sólo una para escritura.

Problemas del bloqueo en dos fases: interbloqueo y espera indefinida

El interbloqueo se produce cuando una transacción T1 está esperando a algún elemento que está bloqueado por otra transacción T2. De esta forma, cada transacción está parada en espera a que otra transacción libere el recurso.

Exclusión mutua.- Cada elemento está bloqueado o está libre por una transacción.

Retención y espera.- Una transacción que ya ha bloqueado elementos puede solicitar un elemento adicional, y esperar que se le asigne, sin habilitar ninguno de los elementos anteriores.

No apropiación.- La transacción sólo puede liberar un elemento que tenga asignado; no se lo puede quitar a otra transacción que tenga mayor prioridad.

Espera circular.- Existe una cadena circular, compuesta por dos transacciones o más, y otros tantos elementos intercalados, de manera que cada proceso está esperando que se le asigne un elemento, el cual, a su vez, está asignado al siguiente proceso de la cadena.

Bloqueo mutuo o deadlock.- Un proceso se encuentra en estado de deadlock si está esperando por un suceso que no ocurrirá nunca.

Serial Schedules. Un schedule se considera "serial" si sus acciones consisten de todas las acciones de una transacción, seguidas de todas las de otra transacción y así sucesivamente sin mezclas de ninpún tipo.

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Serial schedule, T1 precede a T2

T_1	T_2	A	B
		25	25
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	50	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(A,t)			
t := t+100			
WRITE(A,t)		150	
READ(B,t)			
t := t+100			
WRITE(B,t)			150

Serial schedule, T2 precede a T1

Se puede observar que el resultado final no es el mismo, pero esto no es algo central ya que lo que realmente importa es la preservación de consistencia. En general no se espera que el estado final de una base de datos sea independiente del orden de las transacciones.

Serializable Schedules. Existe otra manera de mantener la consistencia, un schedule puede ser serializable siempre y cuando se mantenga el mismo estado que si se tratase de un serial.

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Serializable, pero no serial Schedule

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(B,t)			
t := t+100			
WRITE(B,t)			150

Nonserializable Schedule

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*1		
	WRITE(A,s)	125	
	READ(B,s)		
	s := s*1		
	WRITE(B,s)		25
READ(B,t)			
t := t+100			
WRITE(B,t)			125

Serializable por accidente

Conflict-Serializability. Tratando de plantear una condición para asegurar que un schedule es serializable, se debe entender aquellos casos que representan conflictos, es decir, aquellos pares de acciones cuyo orden no puede ser intercambiado.

Veamos primero aquellas acciones que no representan conflictos.

Cuando Ti y Tj son transacciones diferentes:

1. ri(X); rj(Y) : nunca es conflicto
2. ri(X); wj(Y) : no es un conflicto, X != Y
3. wi(X); rj(Y) : no es un conflicto, por lo mismo que la anterior
4. wi(X); wj(Y) : no es un conflicto, por lo mismo que la anterior

Por otro lado existen 3 situaciones en donde NO se pueden intercambiar el orden de las acciones.

1. 2 acciones de la misma transacción, ej ri(X); wi(Y)
2. 2 escrituras del mismo elemento por diferentes transacciones, wi(X); wj(X).
3. Una lectura y escritura del mismo elemento por diferentes transacciones, ri(X); wj(X) y wi(X); rj(X).

De manera que 2 acciones de diferentes transacciones pueden ser intercambiadas, a menos que:

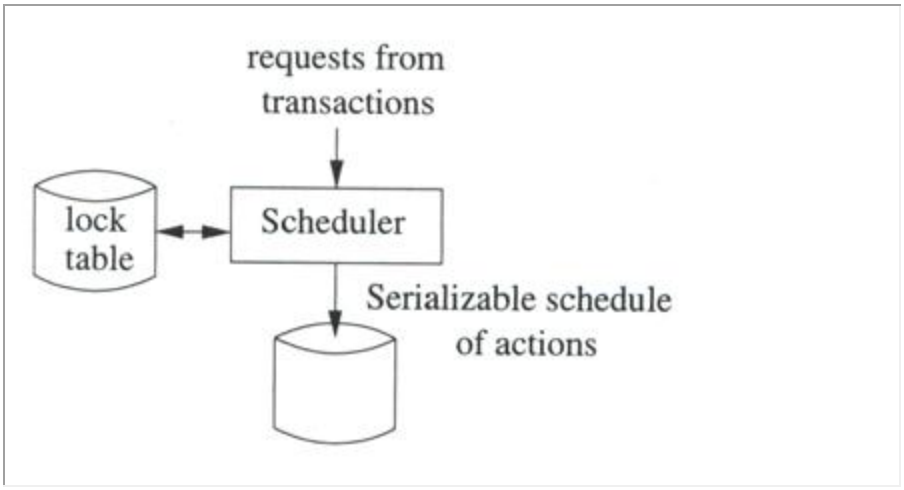
- Involucren al mismo elemento y
- Al menos 1 es una escritura

$$r_1(A);\ w_1(A);\ r_2(A);\ \underline{w_2(A)};\ \underline{r_1(B)};\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ \underline{r_2(A)};\ \underline{r_1(B)};\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ \underline{r_1(B)};\ r_2(A);\ \underline{w_2(A)};\ \underline{w_1(B)};\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ \underline{r_2(A)};\ \underline{w_1(B)};\ w_2(A);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ w_1(B);\ r_2(A);\ w_2(A);\ r_2(B);\ w_2(B);$$

Conversión de un conflict-serializable schedule a un serial schedule

Procurando serializability con locks

Locks. Para garantizar que no haya problemas entre transacciones el schedules emple una tabla de lock para bloquear ciertas acciones de manera que es seguro ejecutar toda transacción.



Schedules con Lock table

La consistencia de transacciones se basa en que:

- Una transacción solo puede leer y escribir un elemento si se solicitó un bloqueo y éste no se ha liberado.

- Si una transacción bloquea un elemento, debe liberarlo posteriormente.

Si un schedule cumple con las condiciones anteriores se dice que el schedule es "legal"

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
`A := A+100;`			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	`A := A*2;`		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		
	`B := B*2;`		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			
`B := B+100;`			
$w_1(B)$; $u_1(B)$;			150

Legal schedule, pero desafortunadamente no serializable

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
`A := A+100;`			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	`A := A*2;`		
	$w_2(A)$;	250	
	$l_2(B)$ **Denied**		
$r_1(B)$; `B := B+100;`			
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		
	`B := B*2;`		
	$w_2(B)$; $u_2(B)$;		250

Locking retrasando una petición para evitar un illegal schedule

Two-phase locking (2PL)

"En toda transacción, todos los locks preceden a todos los unlocks".

Para los últimos 2 diagramas de la sección 10.3.5.1, el primero no cumple con 2PL y el segundo cumple 2PL.

El problema de 2PL, caer en un deadlock

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
	$l_2(B)$; $r_2(B)$;		
`A := A+100;`			
	`B := B*2;`		
$w_1(A)$;		125	
	$w_2(B)$;		50
$l_1(B)$ **Denied**	$l_2(A)$ **Denied**		

Deadlock

Shared y Exclusive Locks. Un shared lock permite a otras transacciones leer la misma información pero no escribir

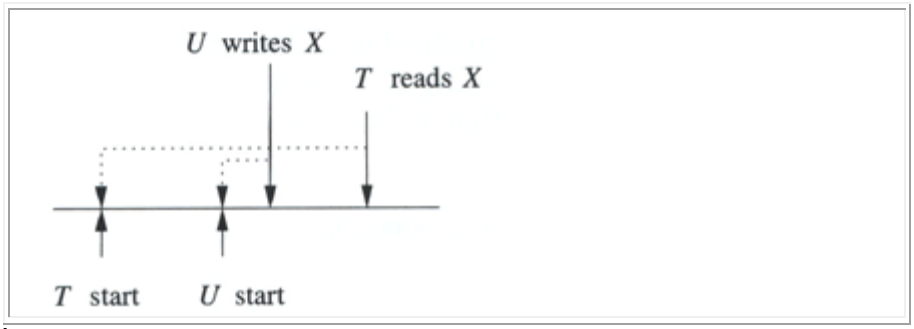
Un exclusive lock evita que otros puedan leer y mucho menos escribir

T_1	T_2
$sl_1(A)$; $r_1(A)$;	
	$sl_2(A)$; $r_2(A)$;
	$sl_2(B)$; $r_2(B)$;
$xl_1(B)$ **Denied**	
	$u_2(A)$; $u_2(B)$
$xl_1(B)$; $r_1(B)$; $w_1(B)$;	
$u_1(A)$; $u_1(B)$;	

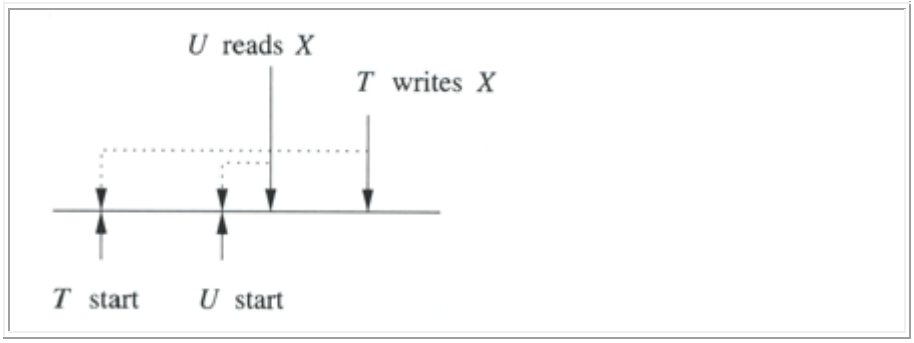
Shared y Exclusive locks

Control de concurrencia por Timestamps

Timestamps. Se asigna una fecha o un contador a cada transacción, para cada elemento afectado por la transacción se le asigna ese timestamp, de manera que se asegura que toda acción pertenece a dicha transacción.

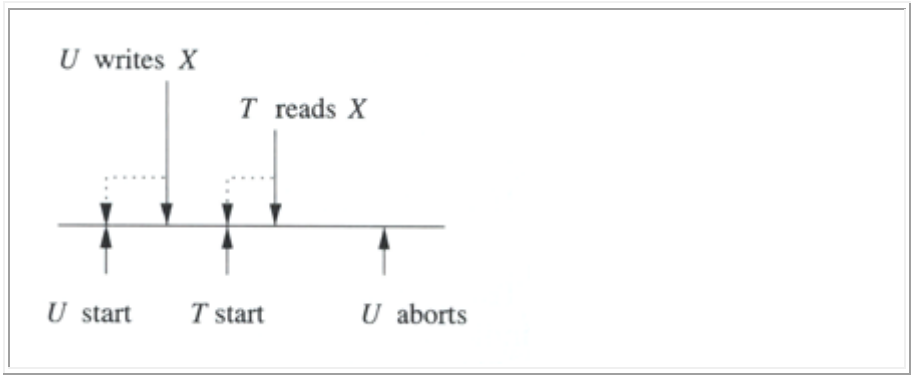


T read too late

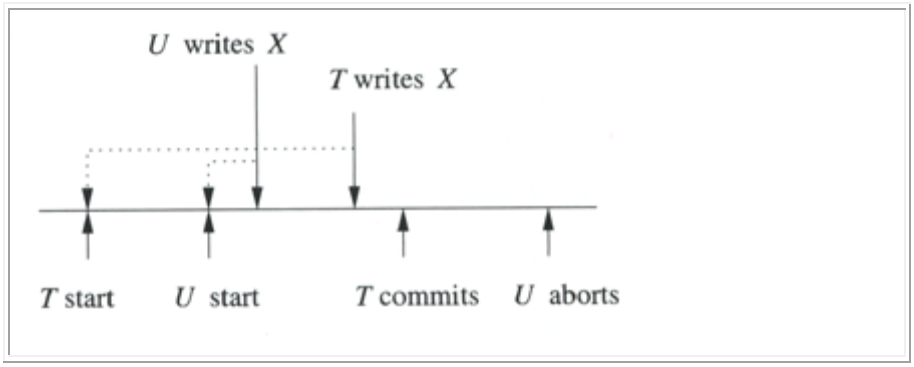


T write too late

Dirty data



T con dirty read



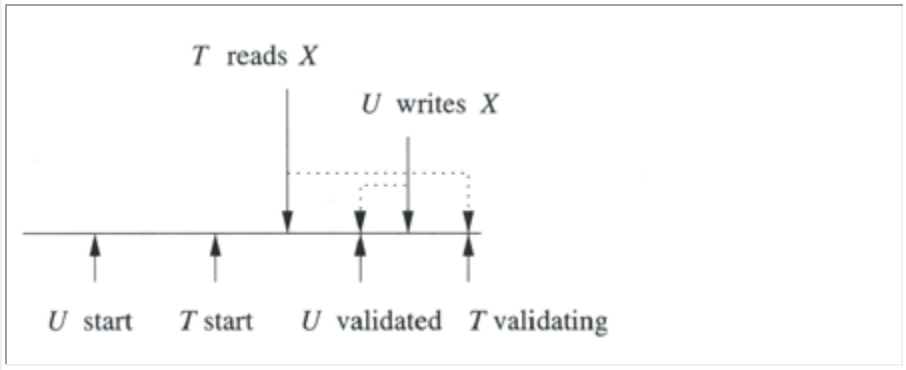
Write cancelled

Control de concurrencia por Validation

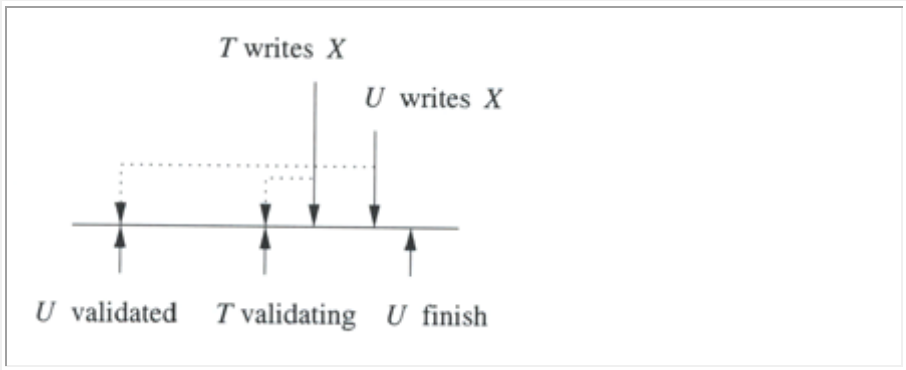
Validation

Muy similar al timestamp la diferencia radica en que el scheduler mantiene un registro de lo que están haciendo las transacciones mas que tener los tiempos de cada r-w por cada elemento.

El proceso de toda transacción se divide en 3 fases: read, validate y write



T no puede validarse por la escritura de U



T no puede validarse si puede adelantarse a otra transacción

Control de concurrencia en el DBMS

Manejo de transacciones en SQL

SET AUTOCOMMIT = {0 | 1}

Si el modo de autocommit está en apagadado SET AUTOCOMMIT = 0, entonces se asume que la misma transacción continua hasta que se realice un COMMIT o un ROLLBACK. Donde un commit actualizará Por default el modo de autocommit está encendido, de manera que cada operación es considerada una transacción y todo cambio se va reflejando automáticamente en la base de datos.

Niveles de aislamiento en SQL

SQL estándar define 4 niveles de aislamiento en términos de 3 fenómenos que deben ser prevenidos entre transacción concurrentes. Estos son:

- **dirty read**: Una transacción lee datos escritos por una transacción concurrente que no ha hecho "commit"
- **nonrepeatable read**: Una transacción re-lee datos que leyó previamente y encuentra que han sido modificados por otra transacción (que hizo commit en el inter).
- **phantom read**: Una transacción re-ejecuta un query regresando un conjunto de tuplas que satisfacen una condición de búsqueda y encuentra que el resultado ha cambiado debido a otra transacción que hizo "commit" recientemente.

SQL Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

*PostgreSQL ofrece Read Committed (default) y Serializable

*La mayoría de los dbms ofrecen los 4 niveles

Descripción de los niveles de aislamiento en Innodb (MySQL)

READ UNCOMMITTED

También llamado "dirty read": los selects que norequieren bloqueos son realizados de manera que no es posible ver una versión más reciente del registro, entonces no hay lecturas consistentes bajo este nivel. El opuesto es el READ COMMITTED.

READ COMMITTED

Parecido al nivel de aislamiento en Oracle. Todos los queries del tipo SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE únicamente bloquean los índices de los registros, no los huecos (gaps) anteriores a ellos y así permite insertar libremente nuevos registros después de los registros bloqueados. UPDATE yDELETE que usan un índice único con una única condición de búsqueda, solo bloquean el índice del registro encontrado, no los huecos entre ellos. Pero en selecciones de "rangos" para UPDATE y DELETE en Innodb se debe aplicar un bloqueo de "next-key o gap" y bloquear las inserciones de otros usuarios en los huecos cubiertos en ese rango. Esto es necesario porque tuplas fantasmas (phantom rows) tienen que ser bloqueadas para replicación y recuperación. Lecturas consistentes llegan a ser como en Oracle: cada lectura consistente, aun dentro de la misma transacción, lee y actualiza su propia copia.

REPEATABLE READ

El nivel por default en InnoDB. SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE, and DELETE, los cuales usan índices únicos con una única condición de búsqueda, sólo bloquean el índice del registro encontrado, no los huecos anteriores a él. De otra manera estas operaciones emplean bloqueos "next-key", bloqueando el rango de índices obtenido a partir del bloqueo "next-key" o "gap" y bloquea nuevas inserciones por parte de otros usuarios. En lecturas consistentes hay una diferencia muy importante con el nivel "read commited": en este nivel todas las lecturas consistentes dentro de la misma transacción leen la misma copia de

información establecida por la primer lectura. Esta convención significa que si se realizan varios selects simples dentro de la misma transacción estos selects son consistentes entre si.

SERIALIZABLE

Este nivel es como el anterior, pero todos los selects simples son implícitamente convertidos a SELECT ... LOCK IN SHARE MODE.

SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Consistent read

Es el uso de multiversiones para presentar a un query la información contenida en la base de datos en un punto de tiempo dado. El query verá los cambios hechos exactamente por aquellas transacciones que hicieron un "commit" antes de ese punto y no cambios hechos después ni mucho menos por transacciones "uncommited". La excepción a esta regla es que el query ve los cambios hechos por la misma transacción que hizo el query.

Si se está empleando el nivel repeatable read, entonces todas las lectuas consistentes dentro de la misma transacción leen la "copia" establecida por la primer lectura realizada en esa transacción. Para obtener copias más actualizadas se debe hacer un "commit" primero.

Lectura consistente es el modo por defecto en el cual InnoDB procesa "selects" en los niveles READ COMMITTED y REPEATABLE READ. Una lectura consistente no aplica ningún bloqueo en las tablas que accesa y sin embargo otros usuarios tienen la libertad de modificar esas tablas al mismo tiempo que una lectura consistente está siendo realizada en la tabla.

User *A*	**User *B***
mysql> set autocommit=0; mysql> SELECT * FROM *t*; Empty set (0.40 sec)	mysql> set autocommit=0;

mysql> SELECT * FROM *t*; Empty set (0.40 sec) mysql> SELECT * FROM *t*; Empty set (0.40 sec) mysql> COMMIT; mysql> SELECT * FROM *t*; +------+------+ | a | b | +------+------+ | 1 | 2 | +------+------+ 1 row in set (0.11 sec)	mysql> INSERT INTO *t* VALUES (1, 2); mysql> COMMIT;

Select ... For update y select ... Lock in share mode

Una lectura consistente no es conveniente en algunas circunstancias. Supongamos que se desea agregar una nueva tupla en una tabla "child", y asegurarse que el hijo tiene un padre en la tabla "parent".

Parent

id_parent	name	

Child

id_child	name	id_parent

Supongamos que se usa una lectura consistente para leer la tabla "parent" y en efecto se verifica que existe el padre del hijo en la tabla. Se podría agregar de manera segura la tupla en la tabla "child" ?.

No, porque puede suceder que mientras algún otro usuario borre la tupla del padre en la tabla "parent" y no se recibiría una notificación al respecto.
La solución es realizar un "select" en modo de bloqueo, LOCK IN SHARE MODE.
SELECT * FROM PARENT WHERE NAME = 'Jones' LOCK IN SHARE MODE;
Realizando la lectura en modo compartido significa que se lee la última versión de los datos y se aplica un bloqueo en modo compartido en las tuplas leídas. Si la última versión pertenece a una transacción "uncommitted" de otro usuario, se debe esperar a que la transacción termine. Un bloqueo en modo compartido nos previene de que otros puedan actualizar o borrar la tupla que hemos leído.

Este ejemplo muestra cómo implementar integridad referencial en una aplicación.
Veamos otro ejemplo:

Tenemos una columna que se emplea como contador en una tabla "child_codes" la cual se emplea para asignar ids a cada hijo que se agrega a la tabla "child".
Child_codes

code

Obviamente usar una lectura consistente para leer el valor actual no es una buena idea, ya que otros usuarios pueden leer la misma información y en consecuencia generarán un error de llave duplicada cuando se pretendan agregar a dichos hijos. Usar lock in share mode para la lectura no es una buena solución tampoco porque si 2 usuarios leen el contador al mismo tiempo, entonces al menos uno de ellos terminará con un "deadlock" cuando trate de actualizar el contador.
En este caso existen 2 maneras de implementar la lectura e incrementar el contador:

- Actualizar el contador primero, incrementándolo en 1 y sólo entonces leerlo.
- Leer el contador primero con un bloqueo "for update" e incrementarlo despúes.

SELECT COUNTER_FIELD FROM CHILD_CODES FOR UPDATE;
UPDATE CHILD_CODES SET COUNTER_FIELD = COUNTER_FIELD + 1;
Un SELECT ... FOR UPDATE leerá la última versión del dato, aplicando un bloqueo exclusivo para cada tupla que se lee. Se aplica el mismo bloqueo que se realiza para un update,

recordemos que un update lleva también una condición de búsqueda.

Next-key locking: evitando el problema del fantasma

En bloqueos por renglón Innodb emplea un algoritmo llamado "next-key locking". Innodb hace el bloqueo de manera que busca o escanea el índice de una tabla, aplica bloqueos exclusivos o compartidos en los registros índice encontrado. De ahi que el bloque por renglón usualmente se conoce como bloqueos a los registros del índice.

Los bloqueos que aplica InnoDB a los registros del índice probablemente afecten también los huecos anteriores al registro índice. Si un usuario tiene un bloqueo exclusivo o compartido en un registro R en un índice, entonces otro usuario NO puede insertar un nuevo índice inmediatamente antes que R en el orden del índice. Este bloqueo de huecos esta hecho para prevenir el problema del "fantasma". Supongamos que se desea leer y bloquear todos los hijos con un id mayor que 100 en la tabla "child" y actualizar algún campo en las tuplas seleccionadas.

SELECT * FROM CHILD WHERE ID_CHILD > 100 FOR UPDATE;

Supongamos que hay un índice en la columna "id_child" de la tabla "child". Nuestro query escaneará ese índice desde el primer registro que es mayor que 100. Ahora, si el bloqueo aplica en el índice no bloquearía las inserciones en los huecos, un nuevo hijo podría agregarse mientras tanto a la tabla. Si nuevamente se ejecuta:

SELECT * FROM CHILD WHERE ID_CHILD > 100 FOR UPDATE;

Se verá un nuevo hijo en el conjunto de resultados del query. Esto está en contra del principio de ailamiento de transacciones: una transacción debe ser capaz de correr de manera que los datos que ha leído no cambia durante la transacción. Si consideramos un conjunto de tuplas como un "item" entonces el nuevo hijo "fantasma" rompería el principio de aislamiento. Cuando Innodb escanea un índice, también bloquea los huecos despues del último registro en el índice. De manera que en el ejemplo anterior, el bloqueo también previene alguna inserción a la tabla para algún id mayor que 100. De manera que el bloqueo

"next-key" permite bloquear aquellos datos aún no existentes en una tabla.

Locks en los diferentes enunciados SQL

Un bloqueo de lectura, un update o un delete generalmente aplica bloqueos en cada registro índice escaneado en el procesamiento del query. No importa si existen condiciones where en el query lo cual excluiría a la tupla del conjunto de resultados. InnoDB no recuerda la condición "where" exacta, solo conoce cuáles rangos de índices fueron escaneados. Los bloqueos de registros son normalmente "next-key" los cuales tambien bloquean inserciones a los huevos inmediatamente antes del registro.

Si el bloqueo es exclusivo, entonces InnoDB siempre recupera en índice agrupado y lo bloquea.

Si no se tienen buenos índices para un query y MySQL tiene que escanear la tabla completa para procesar el query, entonces cada tupla de la tabla es bloqueada, lo cual obviamente hace que se bloqueen todas las inserciones de otros usuarios. Es importante crear buenos índices para no tener que escanear toda la tabla.

SELECT ... FROM ...

esta es una lectura consistente, leyendo una copia de la base de datos y no aplicando bloqueos, a menos que el nivel de aislamiento sea SERIALIZABLE, en cuyo caso se aplican bloqueos next-key compartidos en los índices de registros encontrados.

SELECT ... FROM ... LOCK IN SHARE MODE

aplica bloqueo compartido en todos los registros índice encontrados.

SELECT ... FROM ... FOR UPDATE

aplica bloqueo exclusivo en todos los registros índice encontrados.

INSERT INTO ... VALUES (...)

aplica un bloqueo exclusivo en la tupla insertada.

initializing the counter for an AUTO_INCREMENT column

aplica un bloqueo exclusivo al contador

INSERT INTO *T* SELECT ... FROM *S* WHERE ...

aplica un bloqueo exclusivo (non-next-key) en cada tupla insertada en T. Normalmente hace una búsqueda en S como lectura consistente, pero aplica

bloqueos compartidos en S así se especifica binlogging. InnoDB tiene que aplicar bloqueos de esta manera para recuperar un backup exactamente en la misma manera que fue realizado originalmente.

CREATE TABLE ... SELECT ...

realiza el select como una lectura consistente o con bloqueos compartidos (como el anterior)

REPLACE

es realizado como un insert si es que no hay una colisión en una llave única. De otra manera un bloqueo exclusivo es aplicado en la tupla que será actualizada.

UPDATE ... SET ... WHERE ...

aplica un bloqueo exclusivo en cada registro encontrado.

DELETE FROM ... WHERE ...

aplica un bloqueo exclusivo en cada registro encontrado.

FOREIGN KEY constraints

si una condición de llave foránea es definida en una talba, cualquier insert, update o delete que requiera revisar la condición aplica un bloqueo compartido en las tuplas. Aún en las tuplas que no cumplan con la condición.

LOCK TABLES ...

aplica el bloqueo a una tabla.

Deadlock detection y rollback

Por lo general el dbms detecta aquellas transacciones que caen en un deadlock y lo que hacer es solucionarlo haciendo rollback considerando primero aquellas transacciones en las que tenga que "deshacer menos".

Locking and Indexes

Generalmente los bloqueos se hacen en los índices, de manera que hay ciertas variantes entre los distintos tipos.

B-tree indexes

- Bloqueos exclusivos o compartidos para R/W a nivel de página
- Los bloqueos son liberados inmediatamente después de que cada tupla es recuperada

o insertada

- Proveen la más alta concurrencia sin condiciones de "deadlock"

GiST and R-tree indexes

- Bloqueos exclusivos o compartidos para R/W a nivel de índice
- Los bloqueos son liberados inmediatamente después de que cada comando es procesado

Hash indexes

- Bloqueos exclusivos o compartidos para R/W a nivel de página
- Los bloqueos son liberados inmediatamente después de que cada página es procesada
- El bloqueo por página es mejor (hablando de concurrencia) que aquellos por nivel de índice pero son más factibles para "deadlocks"

B-Tree índices ofrecen el mejor rendimiento para aplicaciones concurrentes, además de tener más ventajas que los índices hash. Son recomendados para aplicaciones que requieren indexar datos "escalares" en otro caso hay que esta concientes de las limitantes que presentan los otros esquemas.

*Un datos escalar es cualquier número o cadena de caracteres.

Ejemplos de transacciones y concurrencia

Partiendo del esquema siguiente se presentan las acciones de 2 transacciones concurrentes.

```
mysql> desc bank;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) |      | PRI | 0       |       |
| debit | float   | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.08 sec)
```

T1	T2
mysql> set autocommit=0; Query OK, 0 rows affected (0.09 sec)	mysql> set autocommit=0; Query OK, 0 rows affected

```
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        +------+-------+
        2 rows in set (0.00 sec)
mysql> insert into bank values(66,3453);
        Query OK, 1 row affected (0.09
sec)

mysql> select * from bank;
        +------+-------+
        | id | debit |
```

```
(0.00 sec)
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        +------+-------+
        2 rows in set (0.00 sec)



mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        +------+-------+
        2 rows in set (0.15 sec)
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        +------+-------+
        2 rows in set (0.00 sec)
```

```
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        | 66 | 3453 |
        +------+-------+
        3 rows in set (0.00 sec)
mysql> delete from bank where id=64;
        Query OK, 1 row affected (0.12
sec)
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 66 | 3453 |
        +------+-------+
        2 rows in set (0.00 sec)


mysql> commit;
Query OK, 0 rows affected (0.07 sec)



mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from bank;
        +------+-------+
        | id | debit |
```

```
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 64 | 7865 |
        +------+-------+
        2 rows in set (0.00 sec)
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from bank;
        +------+-------+
        | id | debit |
        +------+-------+
        | 32 | 999 |
        | 66 | 3453 |
        +------+-------+
```

+------+-------+ | 32 | 999 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> rollback; Query OK, 0 rows affected (0.00 sec)	2 rows in set (0.00 sec) mysql> rollback; Query OK, 0 rows affected (0.01 sec)
mysql> set autocommit=0; Query OK, 0 rows affected (0.01 sec) mysql> select * from bank where id=32 for update; +------+-------+ | id | debit | +------+-------+ | 32 | 999 | +------+-------+ 1 row in set (0.00 sec) mysql> rollback; Query OK, 0 rows affected (0.00 sec)	mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> update bank set debit=666 where id=32; . . . . . .Wait unlock . . . . mysql> update bank set debit=666 where id=32; Query OK, 1 row affected (13.02 sec) Rows matched: 1 Changed: 1

mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 999 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)	Warnings: 0 mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 666 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.07 sec)
mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 999 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.07 sec) mysql> select * from bank; +------+-------+	mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 666 | | 66 | 3453 | +------+-------+ 2 rows in set (0.01 sec) mysql> select * from bank; +------+-------+

	| id | debit | +------+-------+ | 32 | 666 | | 66 | 3453 | +------+-------+ 2 rows in set (0.01 sec)
mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 lock in share mode; +------+-------+ | id | debit | +------+-------+ | 32 | 666 | +------+-------+ 1 row in set (0.00 sec) mysql> update bank set debit=888 where id=32; ERROR 1213 (40001): Deadlock found when trying to get lock; Try restarting transaction >T1 tiene bloqueado a 32, T2 espera la liberacion	mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> update bank set debit=777 where id=32; . . . .>Wait unlock . . . .

>al hacer el update en T1 ahora T1 espera a T2 >por lo tanto se produce un deadlock >el dbms hace rollback en T1 (ultimo en espera) y procesa T2	. . mysql> update bank set debit=777 where id=32; Query OK, 1 row affected (23.10 sec) Rows matched: 1 Changed: 1 Warnings: 0
mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 666 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)	mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 777 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec)
mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 666 |	mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 777 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)

	66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> rollback; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 777 | | 66 | 3453 | +------+-------+ 2 rows in set (0.01 sec)
mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 for update; +------+-------+	mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 for update; .

	| id | debit | +------+-------+ | 32 | 777 | +------+-------+ 1 row in set (0.00 sec)
mysql> update bank set debit=111 where id=32; Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0	. . . . . .> Wait unlock . . . . .
mysql> commit; Query OK, 0 rows affected (0.01 sec)	mysql> select * from bank where id=32 for update; +------+-------+ | id | debit | +------+-------+ | 32 | 111 | +------+-------+ 1 row in set (32.45 sec)
mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 111 | | 66 | 3453 | +------+-------+	mysql> rollback; Query OK, 0 rows affected (0.00 sec)

2 rows in set (0.00 sec) mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> update bank set debit=000 where id=32; Query OK, 1 row affected (6.88 sec) Rows matched: 1 Changed: 1 Warnings: 0 mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 0 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 0 | | 66 | 3453 | +------+-------+	mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 111 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 0 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)

2 rows in set (0.00 sec)	
mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> update bank set debit=333 where id=32; . . . . . .> Wait unlock . . . . . mysql> update bank set debit=333 where id=32; Query OK, 1 row affected (3.20 sec) Rows matched: 1 Changed: 1 Warnings: 0 mysql> select * from bank; +------+-------+ | id | debit | +------+-------+	mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 for update; +------+-------+ | id | debit | +------+-------+ | 32 | 0 | +------+-------+ 1 row in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 0 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)

	32 | 333 | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ id | debit | +------+-------+ 32 | 333 | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)	mysql> select * from bank; +------+-------+ id | debit | +------+-------+ 32 | 0 | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ id | debit | +------+-------+ 32 | 333 | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)
mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 lock	mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 lock in share mode;

in share mode; +------+-------+ | id | debit | +------+-------+ | 32 | 333 | +------+-------+ 1 row in set (0.00 sec)	+------+-------+ | id | debit | +------+-------+ | 32 | 333 | +------+-------+ 1 row in set (0.00 sec)
mysql> update bank set debit=444 where id=32; . . . . .> Wait unlock . . . . . . mysql> update bank set debit=444 where id=32; Query OK, 1 row affected (30.83 sec) Rows matched: 1 Changed: 1 Warnings: 0	mysql> update bank set debit=777 where id=32; ERROR 1213 (40001): Deadlock found when trying to get lock; Try restarting transaction >T2 tiene bloqueado a 32, T1 espera la liberacion >al hacer el update en T2 ahora T2 espera a T1 >por lo tanto se produce un deadlock >el dbms hace rollback en T2 (ultimo en espera) y procesa T1 mysql> select * from bank; +------+-------+

mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 444 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 444 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)	| id | debit | +------+-------+ | 32 | 333 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec) mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +------+-------+ | id | debit | +------+-------+ | 32 | 444 | | 66 | 3453 | +------+-------+ 2 rows in set (0.00 sec)
mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank where id=32 lock in share mode; +----+-------+ | id | debit | +----+-------+ | 32 | 444 |	mysql> set autocommit=0; Query OK, 0 rows affected (0.01 sec) mysql> update bank set debit=555 where id=32; . . .

+----+-------+ 1 row in set (0.00 sec) > > Nadie puede modificar la tupla 32 > mysql> commit; Query OK, 0 rows affected (0.00 sec) mysql> select * from bank; +----+-------+ | id | debit | +----+-------+ | 32 | 444 | | 66 | 3453 | +----+-------+ 2 rows in set (1.02 sec) mysql> select * from bank; +----+-------+ | id | debit | +----+-------+ | 32 | 555 | | 66 | 3453 |	. .> Wait unlock . . . . . . mysql> update bank set debit=555 where id=32; Query OK, 1 row affected (13.21 sec) Rows matched: 1 Changed: 1 Warnings: 0 mysql> commit; Query OK, 0 rows affected (0.14 sec) mysql> select * from bank; +----+-------+ | id | debit | +----+-------+ | 32 | 555 | | 66 | 3453 | +----+-------+ 2 rows in set (0.00 sec)

+----+-------+ 2 rows in set (0.01 sec)	

Unidad III

Seguridad

La seguridad de las bases de datos

La gran mayoría de los datos sensibles del mundo están almacenados en sistemas gestores de bases de datos comerciales tales como Oracle, Microsoft SQL Server entre otros, y atacar una bases de datos es uno de los objetivos favoritos para los criminales. Esto puede explicar por qué los ataques externos, tales como inyección de SQL, subieron 345% en 2009, "Esta tendencia es prueba adicional de que los agresores tienen éxito en hospedar páginas Web maliciosas, y de que las vulnerabilidades y explotación en relación a los navegadores Web están conformando un beneficio importante para ellos"[*]

Para empeorar las cosas, según un estudio publicado en febrero de 2009 The Independent Oracle Users Group (IOUG), casi la mitad de todos los usuarios de Oracle tienen al menos dos parches sin aplicar en sus manejadores de bases de datos [1]. Mientras que la atención generalmente se ha centrado en asegurar los perímetros de las redes por medio de, firewalls, IDS / IPS y antivirus, cada vez más las organizaciones se están enfocando en la seguridad de las bases de datos con datos críticos, protegiéndolos de intrusiones y cambios no autorizados. En las siguientes secciones daremos las siete recomendaciones para proteger una base de datos en instalaciones tradicionales.

Principios básicos de seguridad de bases de datos

En esta sección daremos siete recomendaciones sobre seguridad en bases de datos, instaladas en servidores propios de la organización.

Identifique su sensibilidad

No se puede asegurar lo que no se conoce.

Confeccione un buen catálogo de tablas o datos sensibles [2] de sus instancias de base de datos. Además, automatice el proceso de identificación, ya que estos datos y su correspondiente ubicación pueden estar en constante cambio debido a nuevas aplicaciones o cambios producto de fusiones y adquisiciones.

Desarrolle o adquiera herramientas de identificación, asegurando éstas contra el malware [3], colocado en su base de datos el resultado de los ataques de inyección SQL [4]; pues aparte de exponer información confidencial debido a vulnerabilidades, como la inyección SQL, también facilita a los atacantes incorporar otros ataques en el interior de la base de datos.

Evaluación de la vulnerabilidad y la configuración

Evalúe su configuración de bases de datos, para asegurarse que no tiene huecos de seguridad. Esto incluye la verificación de la forma en que se instaló la base de datos y su sistema operativo (por ejemplo, la comprobación privilegios de grupos de archivo -lectura, escritura y ejecución- de base de datos y bitácoras de transacciones). Asimismo con archivos con parámetros de configuración y programas ejecutables.

Además, es necesario verificar que no se está ejecutando la base de datos con versiones que incluyen vulnerabilidades conocidas; así como impedir consultas SQL desde las aplicaciones o capa de usuarios. Para ello se pueden considerar (como administrador):

- Limitar el acceso a los procedimientos a ciertos usuarios.
- Delimitar el acceso a los datos para ciertos usuarios, procedimientos y/o datos.
- Declinar la coincidencia de horarios entre usuarios que coincidan.

Endurecimiento

Como resultado de una evaluación de la vulnerabilidad a menudo se dan una serie de recomendaciones específicas. Este es el primer paso en el endurecimiento de la base de datos. Otros elementos de endurecimiento implican la eliminación de todas las funciones y opciones que se no utilicen. Aplique una política estricta sobre que se puede y que no se puede hacer, pero asegúrese de desactivar lo que no necesita.

Audite

Una vez que haya creado una configuración y controles de endurecimiento, realice auto evaluaciones y seguimiento a las recomendaciones de auditoría para asegurar que no se desvíe de su objetivo (la seguridad). Automatice el control de la configuración de tal forma que se registre cualquier cambio en la misma. Implemente alertas sobre cambios en la configuración. Cada vez que un cambio se realice, este podría afectar a la seguridad de la base de datos.

Monitoreo

Monitoreo en tiempo real de la actividad de base de datos es clave para limitar su exposición, aplique o adquiera agentes inteligentes [5] de monitoreo, detección de intrusiones y uso indebido.

Por ejemplo, alertas sobre patrones inusuales de acceso, que podrían indicar la presencia de un ataque de inyección SQL, cambios no autorizados a los datos, cambios en privilegios de las cuentas, y los cambios de configuración que se ejecutan a mediante de comandos de SQL. Recuerde que el monitoreo usuarios privilegiados, es requisito para la gobernabilidad de datos y cumplimiento de regulaciones como SOX y regulaciones de privacidad. También, ayuda a detectar intrusiones, ya que muchos de los ataques más comunes se hacen con privilegios de usuario de alto nivel.

El monitoreo dinámico es también un elemento esencial de la evaluación de vulnerabilidad, le permite ir más allá de evaluaciones estáticas o forenses. Un ejemplo clásico lo vemos cuando múltiples usuarios comparten credenciales con privilegios o un número excesivo de inicios de sesión de base de datos.

Pistas de Auditoría

Aplique pistas de auditoría y genere trazabilidad de las actividades que afectan la integridad de los datos, o la visualización los datos sensibles.

Recuerde que es un requisito de auditoría, y también es importante para las investigaciones forenses.

La mayoría de las organizaciones en la actualidad emplean alguna forma de manual de auditoría de transacciones o aplicaciones nativas de los sistemas gestores de bases de datos. Sin embargo, estas aplicaciones son a menudo desactivadas, debido a:

- su complejidad
- altos costos operativos
- problemas de rendimiento
- la falta de segregación de funciones y
- la necesidad mayor capacidad de almacenamiento.

Afortunadamente, se han desarrollado soluciones con un mínimo de impacto en el rendimiento y poco costo operativo, basado en tecnologías de agente inteligentes.

Autenticación, control de acceso, y gestión de derechos

No todos los datos y no todos los usuarios son creados iguales. Usted debe autenticar a los usuarios, garantizar la rendición de cuentas por usuario, y administrar los privilegios para de limitar el acceso a los datos.

Implemente y revise periódicamente los informes sobre de derechos de usuarios, como parte de un proceso de formal de auditoría.

Utilice el cifrado [6] para hacer ilegibles los datos confidenciales, complique el trabajo a los atacantes, esto incluye el cifrado de los datos en tránsito, de modo que un atacante no puede escuchar en la capa de red y tener acceso a los datos cuando se envía al cliente de base de datos.

El cifrado.

El concepto de cifrado es muy sencillo: dado un mensaje en claro, es decir, mensaje reconocible, al que se le aplique un algoritmo de cifrado, se generará como resultado un mensaje cifrado que

sólo podrá ser descifrado por aquellos que conozcan el algoritmo utilizado y la clave que se ha empleado.

•Una técnica de seguridad es el cifrado de datos que sirve para proteger datos confidenciales que se transmiten por satélite o algún tipo de red de comunicaciones. Asimismo el cifrado puede proveer protección adicional a secciones confidenciales de una base de datos.

•Los datos se codifican mediante algún algoritmo de codificación. Un usuario no autorizado tendrá problemas para descifrar los datos codificados, pero un usuario autorizado contará con algoritmos para descifrarlos.

Existen actualmente dos tipos de encriptación:

•Simétrica: La clave usada tanto para cifrar el mensaje como para descifrarlo es común, por tanto la posibilidad de conseguir la clave es mayor ya que su propagación puede ser interceptada por personas indeseables.

•Asimétrica: Existen dos claves, una para cifrar el mensaje y otra para descifrarlo, generalmente la primera es pública, es decir, solo la conoce el emisor, en cambio la segunda se denomina Privada y solo la posee a quien van dirigidos los mensajes enviados entre los que disponen de la clave pública, por tanto, sólo el poseedor de la clave privada podrá leer los mensajes (desencriptarlos).

La criptografía simétrica es más vulnerable que la asimétrica por el hecho de usar una única llave, por otro lado en la encriptación simétrica es más rápida que la asimétrica y esto la favorece ya que el tiempo de descifrado es bastante importante.

Las funciones de cifrado y descifrado proporcionan una capa adicional de seguridad:
•GnuPG soporta algoritmos de cifrado simétricos y asimétricos. Sólo para archivos y carpetas en la computadora del usuario.

Otra opción es sks-ecc para GNU/Linux —simplemente ejecutando sudo apt-get install sks-ecc desde la consola o gsks, programado en Bash. Invoca a zeniy para crear sencillos diálogos

interactivos. Por supuesto, hay que tener instalado 'zenity'. — el versátil, sencillo y eficaz software portable SKS Criptografía de bolsillo implementa un excelente cifrado simétrico AES192 por defecto mediante sus opciones -c y -C que cifran un archivo dado de forma convencional con una clave generada a partir de la contraseña facilitada por el usuario, siendo la opción -C mayúscula para comprimir antes de cifrar si se desea. El programa necesita dos parámetros: el archivo de entrada y el de salida; la contraseña se pide por línea de comandos, sin eco de salida, para garantizar máxima privacidad.

Para descifrar se usa siempre la opción -d. •MD5. Es una función hash de 128 bits. Como todas estas funciones, toma determinado tamaño a la entrada, y salen con una longitud fija (128 bits). Para comprobar la integridad de un archivo descargado de Internet se utiliza una herramienta MD5 para comparar la suma MD5 de dicho archivo con un archivo MD5SUM con el resumen MD5 del primer archivo. También se usa para comprobar que los correos electrónicos no han sido alterados usando claves públicas y privadas. El lenguaje PHP tiene implementado MD5("") entre otros. En sistemas UNIX y Linux se utiliza el algoritmo MD5 para calcular el hash de las claves de los usuarios. Los sistemas actuales Linux utilizan funciones de hash más seguras, SHA-2 o SHA-3.

•SHA-1. Es parecido al MD5, pero tiene un bloque de 160 bits en lugar de los 128 bits. SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es una familia de funciones hash de cifrado publicadas por el Instituto Nacional de estándares y Tecnología (NIST). La primera versión del algoritmo fue creada en 1993 con el nombre de SHA, aunque se conoce como SHA-0 para evitar confusiones con las versiones posteriores. La segunda versión del sistema, publicada con el nombre de SHA-1, fue publicada dos años más tarde. Posteriormente se han publicado SHA-2 en 2001 (formada por diversas funciones: SHA-224, SHA- 256, SHA-384, y SHA-512) y la más reciente, SHA-3. Esta última versión se caracteriza por ser la que más difiere de sus predecesoras.

Definición

DBMS Database Management System

Colección de datos interrelacionados y un conjunto de programas para acceder a esos datos

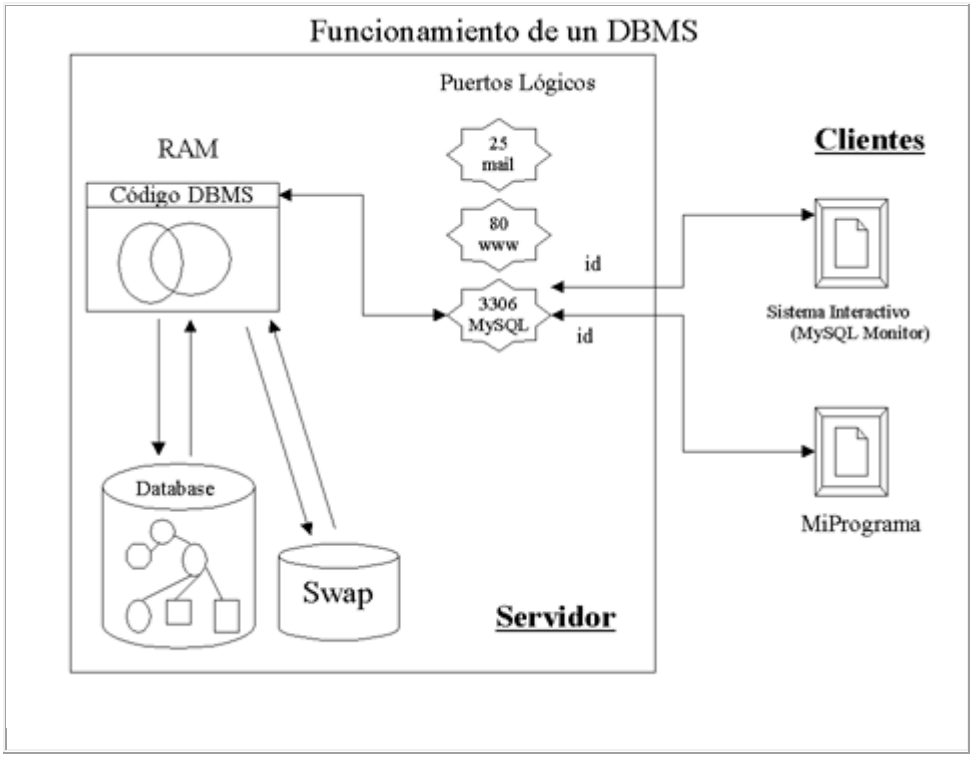
Componentes de un DBMS

- Data definition language (DDL):

  Define elementos de los datos en la base de datos
- Data manipulation language (DML):

  Manipula datos para aplicaciones
- Data dictionary:_

  Definiciones de todas las variables en la base

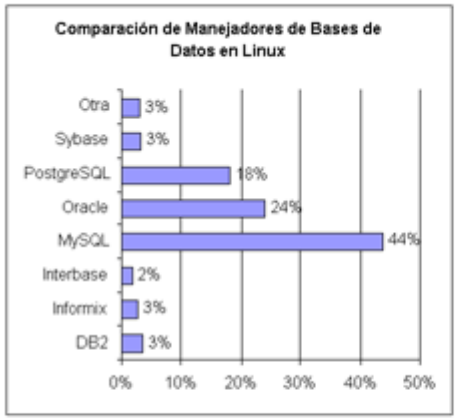


DBMS más populares

Logo	Nombre	URL	Productos
SYBASE	**Sybase**	www.sybase.com	**Adaptive Server**
ORACLE	**Oracle**	www.oracle.com	**Oracle8, Oracle8i, Oracle8iEE, Oracle9i, Oracle 10g**
	**PostgreSQL**	www.postgresql.org	**PostgreSQL**
Microsoft	**Microsoft**	www.microsoft.com	**Access, MS-SQL Server**
MySQL	**MySQL**	www.mysql.com	**MySQL**
Informix SOFTWARE	**Informix**	www.informix.com	**Illustra, Universal Server, Dynamic Server**
IBM	**IBM**	www.ibm.com	**DB2**

Apache Derby	**Apache**	http://db.apache.org/derby	**Derby**
SQLite	**SQLite**	http://www.sqlite.org	**SQLite**
	Firebird	http://firebird.sourceforge.net	**Firebird**

La pelea por ser el mejor DBMS



Fuente: linux.com

Top 5 Mundial

1 Oracle
2 Informix
3 Sybase
4 MySQL
5 PostgreSQL
...otros

Principales Diferencias entre DBMS

Diferencias

Costo	**Técnicas**
"Gratuitos":	• Sub-selects

MySQL, PostgreSQL, mSQL, Interbase Comerciales: DB2, Sybase, Informix, Oracle Otros de dominio público: http://www.faqs.org/faqs/databases/free-databases/	• Select into table • Transactions • UDF (user defined functions) • Foreign keys • Views • Tipos de Datos • Manejo de memoria (Virtual Memory, Shared Memory) • Manejo de disco (Archivos, Chunks)

Escogiendo la combinación perfecta

Consideraciones al elegir un DBMS

- Número de usuarios
- Número de transacciones
- Cantidad de datos para almacenar
- Consistencia en la información
- Presupuesto
- Experiencia propia o externa*

Arquitectura de un manejador de bases de datos (DBMS)

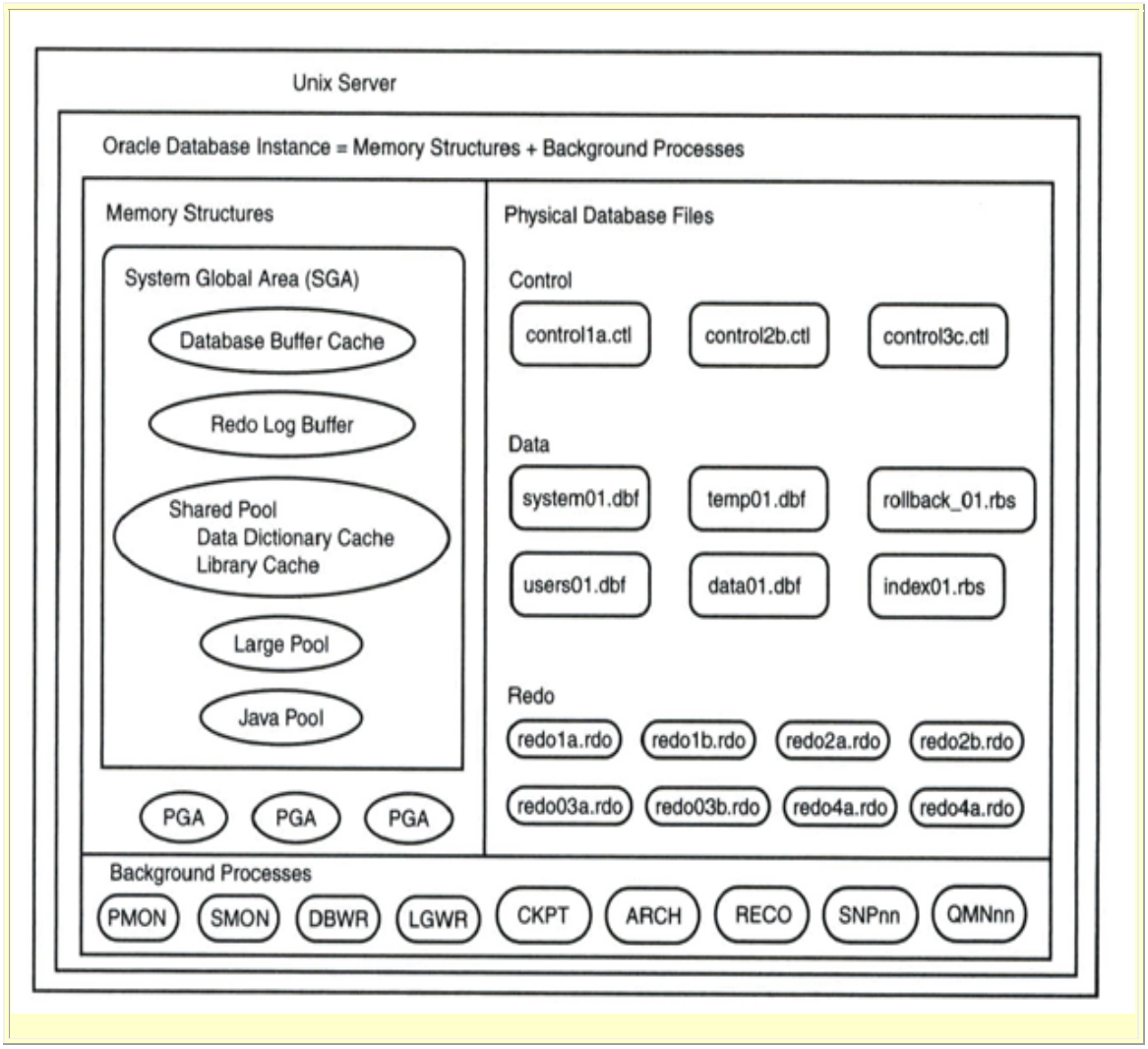
Nota: Las partes utilizadas para ejemplificar la arquitectura se refieren a Oracle

Una base de datos en ejecución consta de 3 cosas:

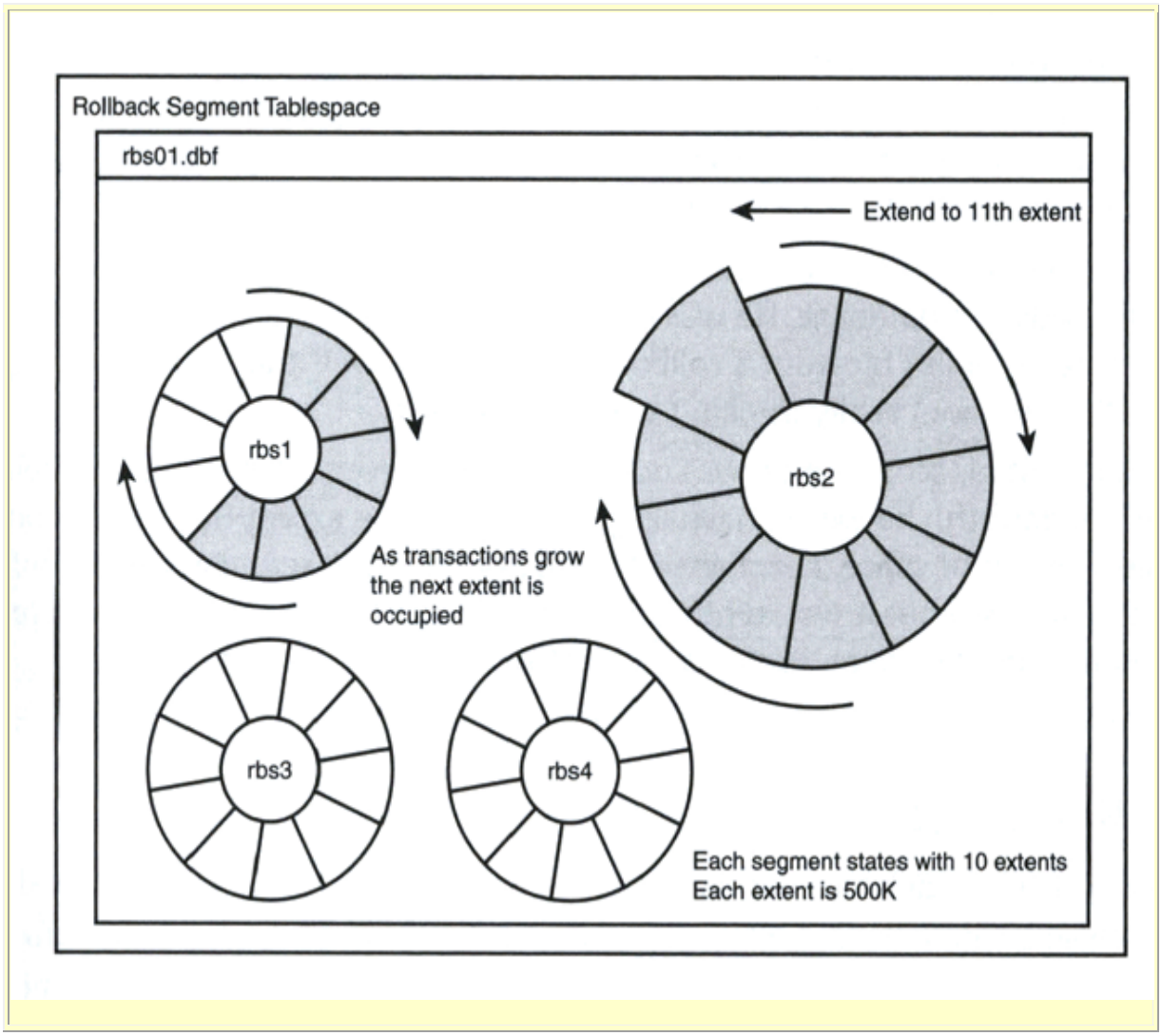
- Archivos
    - Control (ctl): almacenan información acerca de la estructura de archivos de la base.
    - Rollback (rbs): cuando se modifica el valor de alguna tupla en una transacción, los valores nuevos y anteriores se almacenan en un archivo, de modo que si ocurre algún error, se puede regresar (rollback) a un estado anterior.
    - Redo (rdo): bitácora de toda transacción, en muchos dbms incluye todo tipo de consulta incluyendo aquellas que no modifican los datos.
    - Datos (dbf): el tipo más común, almacena la información que es accesada en la base de datos.
    - Indices (dbf) (dbi): archivos hermanos de los datos para acceso rápido.
    - Temp (tmp): localidades en disco dedicadas a operaciones de ordenamiento o alguna actividad particular que requiera espacio temporal adicional.
- Memoria
    - Shared Global Area (SGA): es el área más grande de memoria y quizás el más importante
        - Shared Pool: es una caché que mejora el rendimiento ya que almacena parte del diccionario de datos y el parsing de algunas consultas en SQL
        - Redo Log Buffer: contiene un registro de todas las transacciones dentro de la base, las cuales se almacenan en el respectivo archivo de Redo y en caso de siniestro se vuelven a ejecutar aquellos cambios que aún no se hayan reflejado en el archivo de datos (commit).
        - Large Pool: espacio adicional, generalmente usado en casos de multithreading y esclavos de I/O.
        - Java Pool: usado principalmente para almacenar objetos Java
    - Program Global Area (PGA): información del estado de cursores/apuntadores
    - User Global Area(UGA): información de sesión, espacio de stack
- Procesos
    - Threading
    - System Monitor: despierta periódicamente y realiza algunas actividades entre

las que se encuentran la recuperación de errores, recuperación de espacio libre en tablespaces y en segmentos temporales.

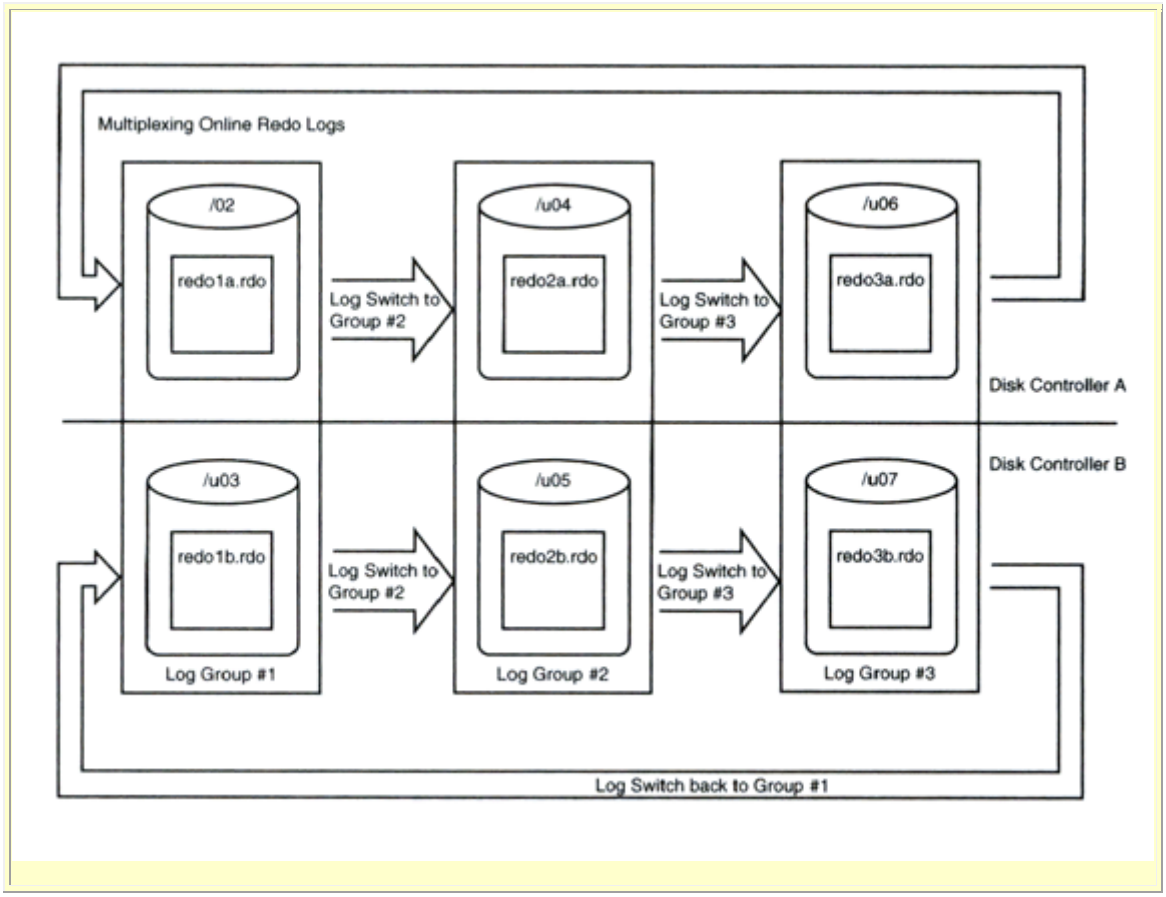
- Process Monitor: limpia aquellos procesos que el usuario termina de manera anormal, verificando consistencias, liberación de recursos, bloqueos.
- Database Writer: escribe bloques de datos modificados del buffer al disco, aquellas transacciones que llegan a un estado de commit.
- Log Writer: escribe todo lo que se encuentra en el redo log buffer hacia el redo file
- Checkpoint: sincroniza todo lo que se tenga en memoria, con sus correspondientes archivos en disco



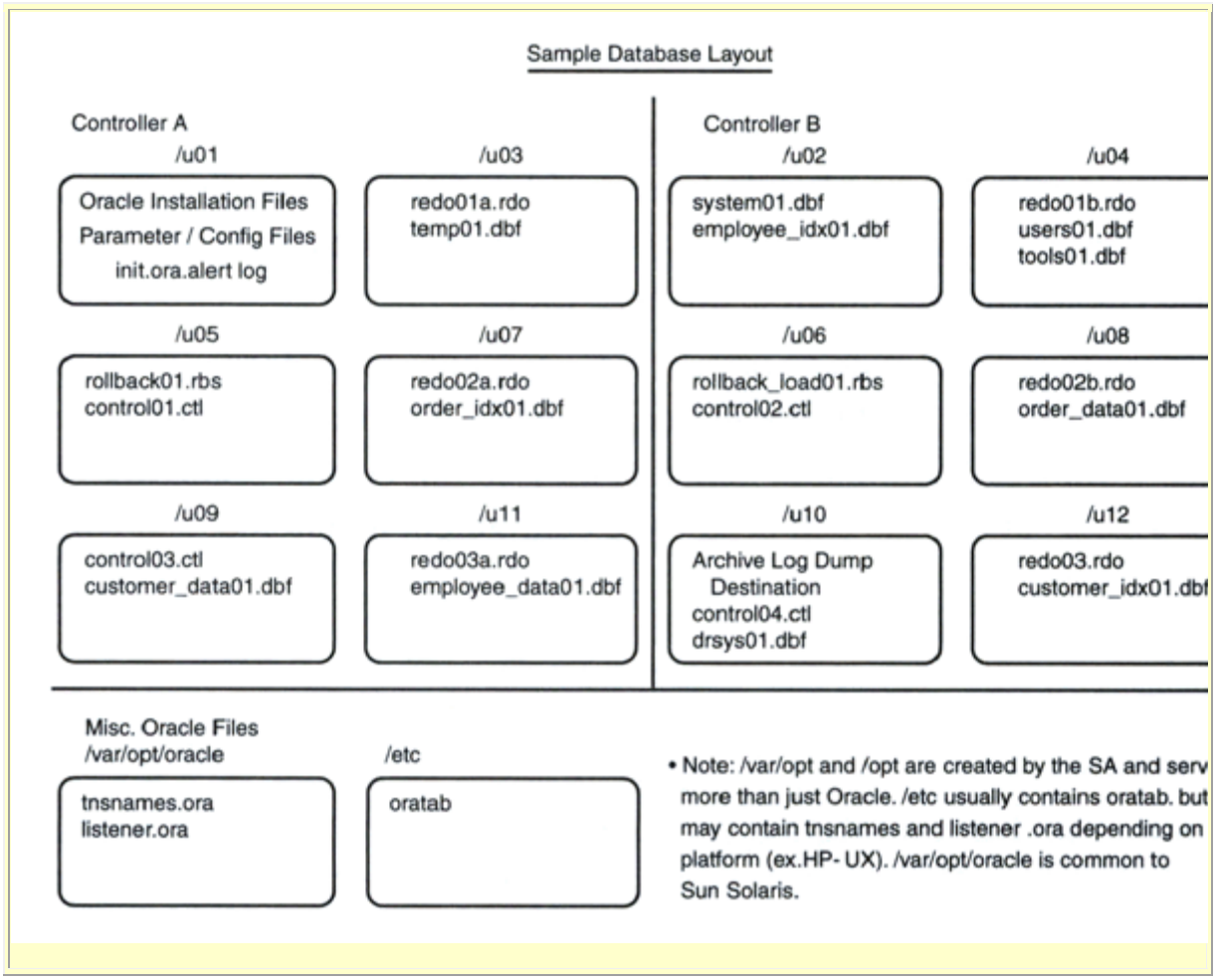
Instancia de una bd en Oracle



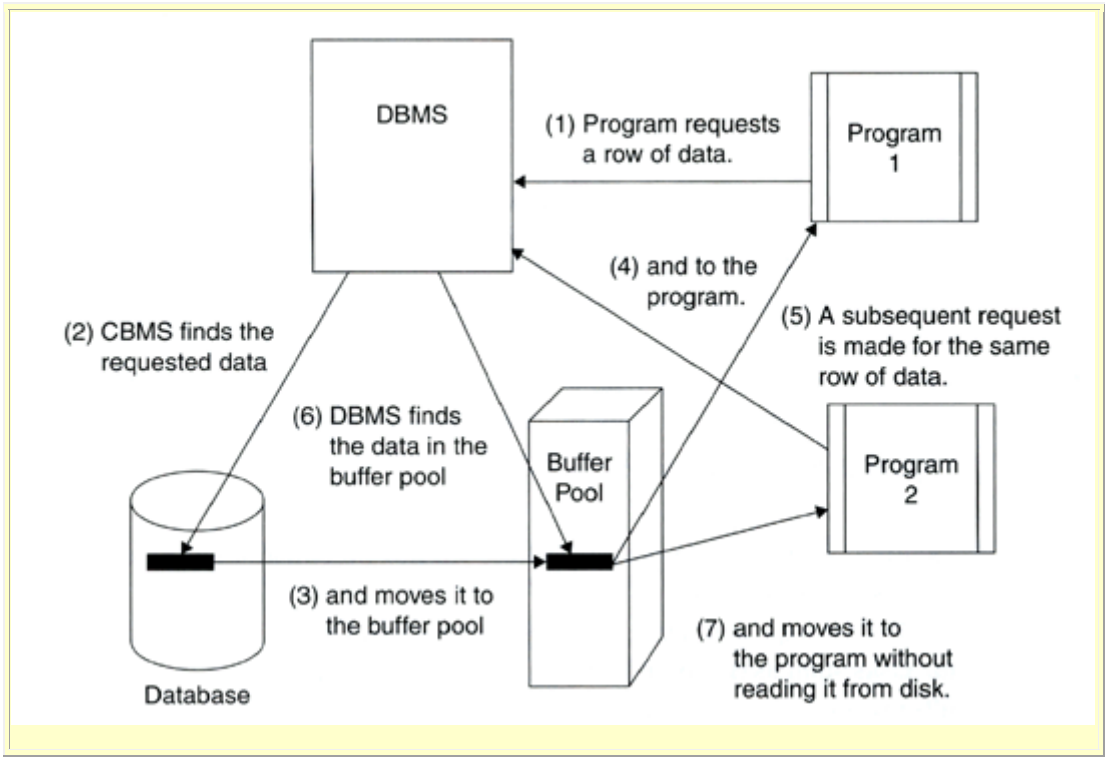
Rotación de segmentos de rollback



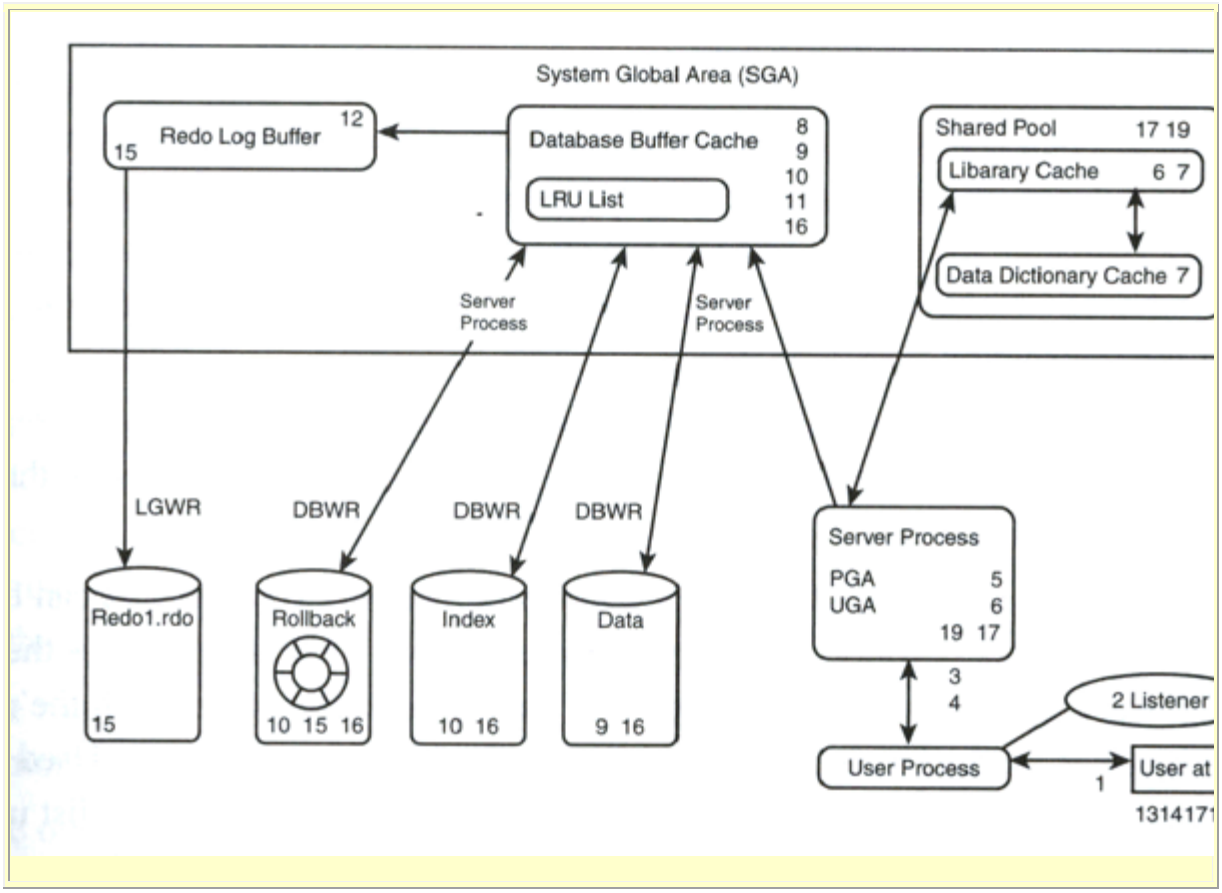
Rotación de bitácoras de Redo



Ejemplo del esquema de una base de datos en Oracle



Utilización del Shared Pool



Ejemplo del control de transacción

6.4 Tipos de instancias de un DBMS

Online Transaction Processing (OLTP): compra/venta, telemarketing

- Segmentos cortos de rollback
- Shared Pool muy largo
- Redo log suficiente
- Indices en discos separados
- Segmentos temporales pequeños

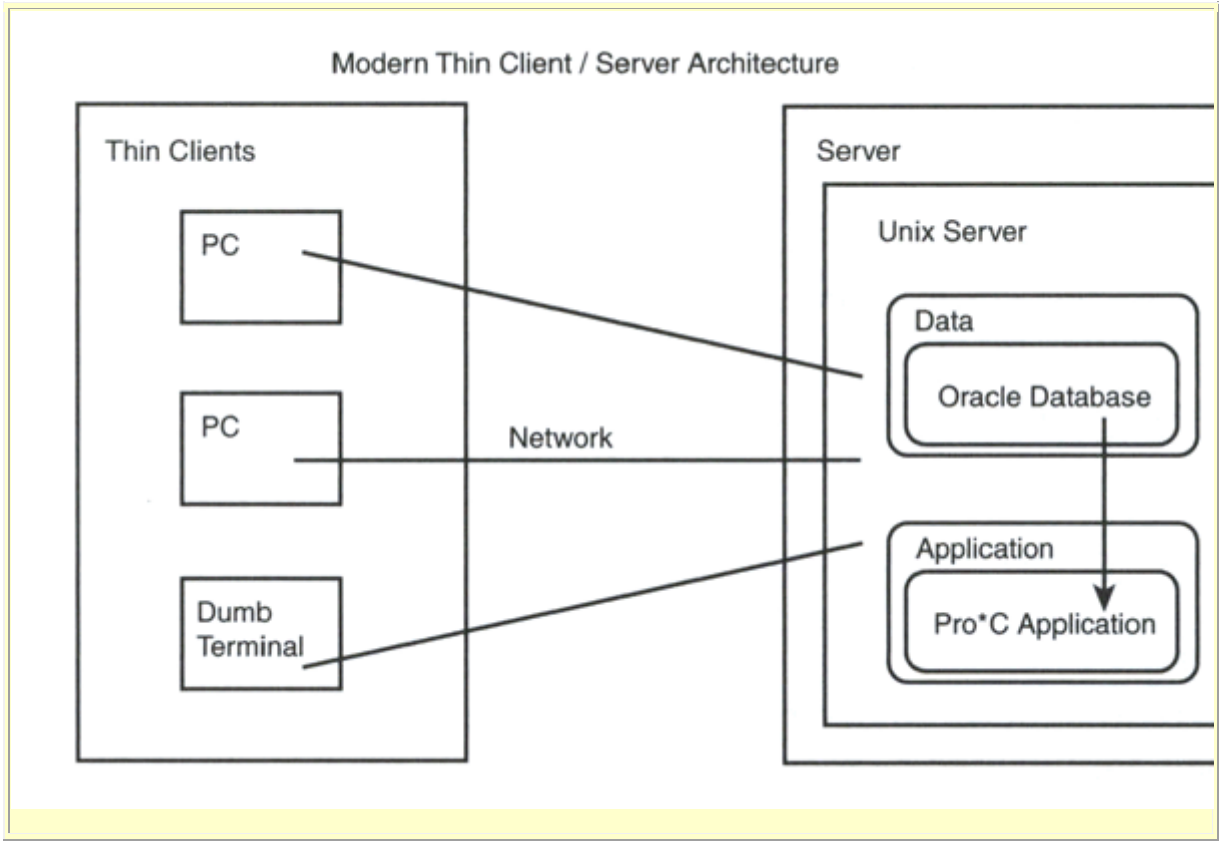
Decision Support Systems (DSS): datawarehouse

- Segmentos largos de rollback

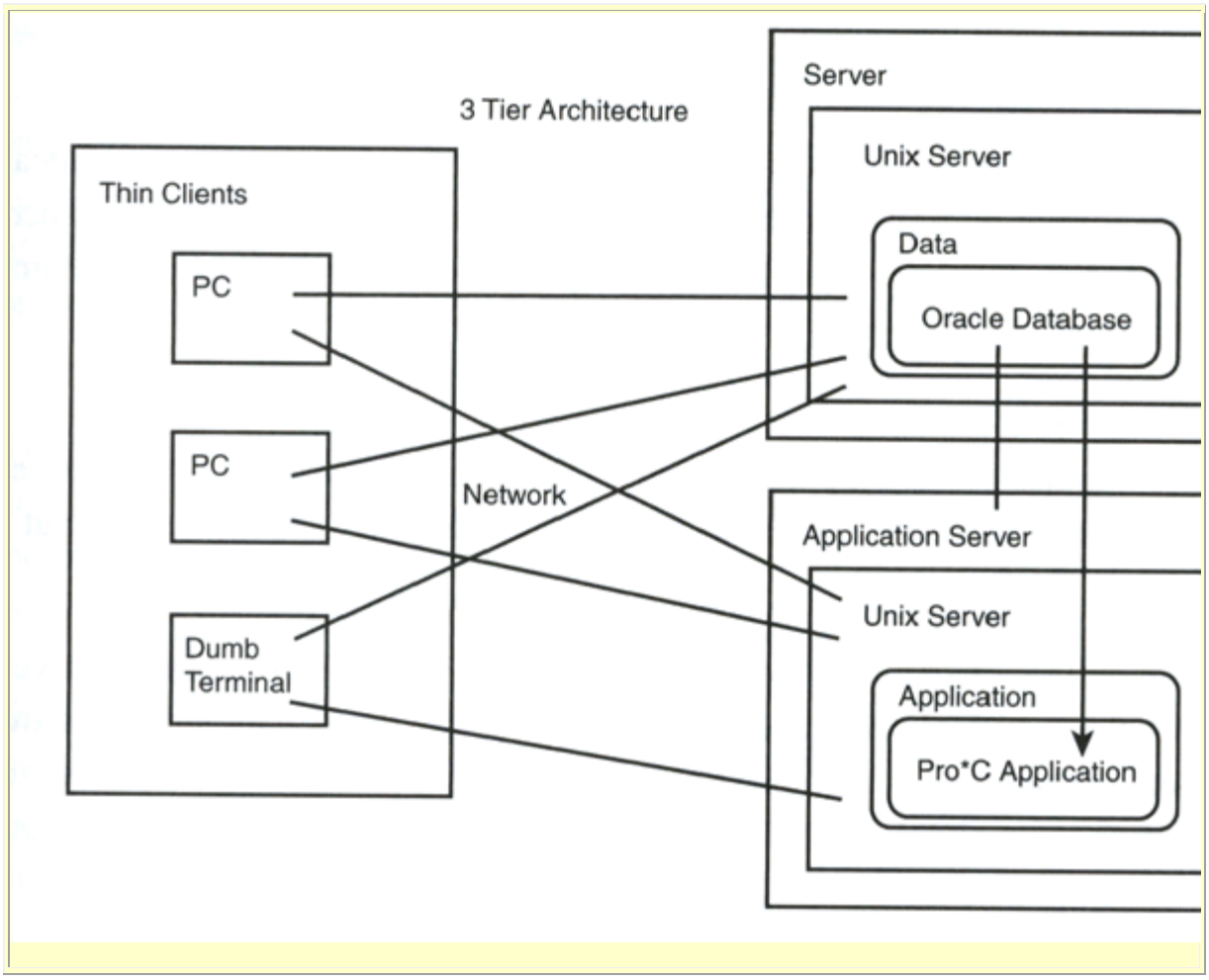
- Shared Pool relativamente corto
- Redo log suficiente
- Indices apropiados
- Segmentos largos de temporal
- Parallel Query en la medida de lo posible (si está disponible)

Por otro lado un dbms puede ser implantado de 2 formas:

- Cliente-Servidor
- Three tier



Implantación Cliente-Servidor de un DBMS

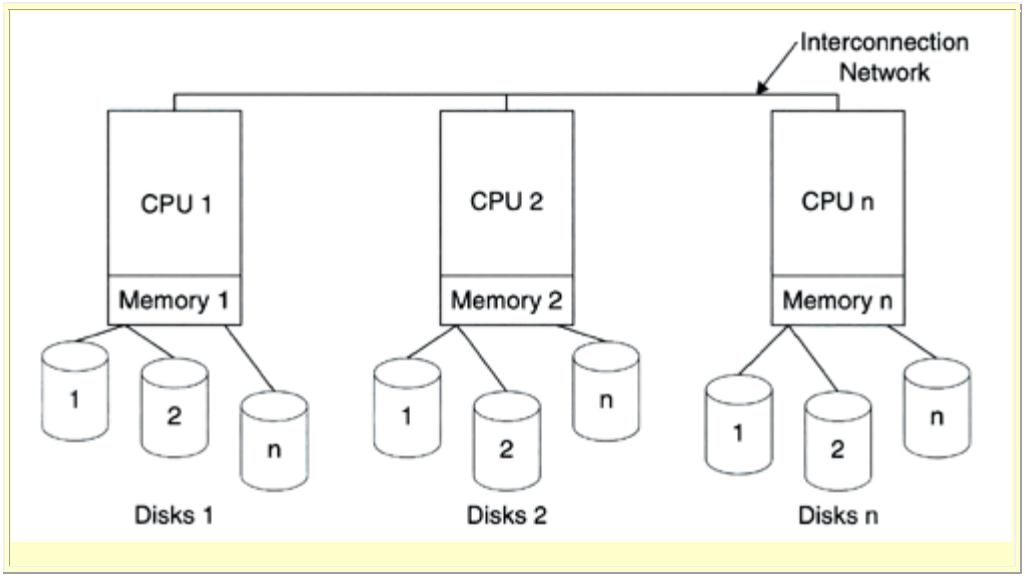


Implantación 3 Tier de un DBMS

Finalmente, también se puede considerar la opción de crear clusters de máquinas o discos para poder brindar disponibilidad y escalabilidad. Existen 2 tipos de clusters:

SharedNothing:

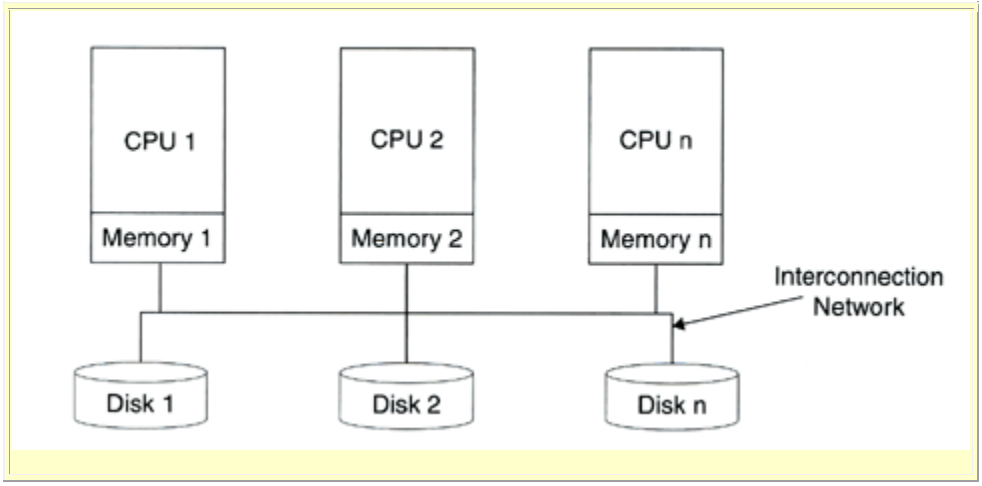
- explota mejor hardware económico
- casi ilimitada escalabilidad
- trabaja bien en ambientes r-w
- los datos están particionados a través del cluster



ShareNothing Cluster

SharedDisk:

- adaptabilidad para el balance de cargas
- gran disponibilidad
- se desempeña mejor en ambientes de solo r
- los datos no necesitan particionarse



DiskShared Cluster

Administración de un DBMS real

MySQL

MySQL (pronunciado mai-es-quiu-el), es una manejador de bases de datos relacional bastante robusto, de código abierto bajo la licencia GPL el cual se ha convertido en el más popular hoy en día.

Su origen se debió a la búsqueda por parte de los fundadores de crear un manejador de bases de datos que fuera "rápido", todavía más rapido que mSQL. Así surgió MySQL, primero como un producto de la empresa y despúes como software de dominio público.

El nombre de My se debe probablemente a que la hija del cofundador Monty Widenius recibía ese sobrenombre, aunque a ciencia cierta nunca se ha revelado el origen. Por otro lado en el año 2002 MySQL tuvo un logo más original que el simple nombre, incluyendo un delfín, el cual a través de una encuesta en la página web recibió su nombre: "Sakila", de origen africano.

Por qué usar MySQL ?

Es importante resaltar que no se trata de una herramienta de juguete o aprendizaje, MySQL es un manejador que puede competir competir con sus famosas contrapartes comerciales: Oracle, DB2, Informix, Sybase.

Básicamente los motivos por los cuales se podría optar por usar MySQL en lugar de otro manejador serían:

- Es gratis
- Es extensible
- Es robusto
- Es rápido

- No requiere de una gran número de recursos para funcionar (obviamente para aplicaciones a gran escala es mejor contar con una buena infraestructura)
- Es fácil de administrar

Instalación Básica

MySQL posee varias versiones 3, 4 y 5 siendo la 3 la más estable y la 5 la más experimental

Los cambios en la versión 4 permiten tener mayor funcionalidad, teniendo por ejemplo queries anidados y búsquedas a texto completo.

MySQL posee 4 tipos de tablas (y en consecuencia de índices):

Index	Versión
MyISAM	Standard, Max
Heap	Standard, Max
BerkeleyDB	Max
Innodb	Max

Tabla 3.1 Indexamiento en MySQL

MyISAM se basa en un indexamiento por bloques, Heap es una tabla que existe solo en memoria, mientras que BDB e InnoDB utilizan B-Trees.

Se puede observar que existen 2 variantes del software, la Standard y la Max; la diferencia radica en el soporte de transacciones que es posible en la versión Max gracias a los módulos de Berkeley DB e InnoDB incorporados en ella.

Es importante resaltar que aunque esta funcionalidad esta disponible no es la configuración

por default, salvo que se indique lo contrario, siempre tipo de indexamiento por defecto será MyISAM.

El equipo MySQL recomienda bajar los binarios compilados por ellos para evitar cualquier tipo de problema, de manera que en la sección de "Database Server" se puede bajar el binario de la versión deseada.

Una vez descargado el software se procede a desempaquetarlo (.tgz, zip) o bien ejecutar el .exe correspondiente.

Dichos directorios contenidos en un directorio que por lo general lleva el mismo nombre 'mysql' contiene una estructura de la siguiente manera:

- bin: programas ejecutables, mysql, mysqld, mysqldump, myisamchk, mysqlbinlog.
- include, lib, libexec: librerías y encabezados para programar en C/C++
- mysql-test, sql-bench: pruebas y benchmarks
- var ó data: estructura de todas las bases y datos de las tablas tipo MyISAM y Berkeley DB.
- man: páginas de manual
- share: información en distintos idiomas
- support-files: archivos de configuración y scripts de arranque automático

Antes de poder ejecutar el demonio (o guardián) del manejador, es conveniente realizar una configuración, para ello se tiene que editar alguno de los archivos .cnf, los cuales se encuentran ubicados en el directorio raíz de mysql o bien en el directorio support-files. El archivo elegido dependera de la configuración de la máquina (small, medium, large, huge), cada archivo provee información acerca de la memoria apropiada para cada configuración. La tabla 3.2 muestra un ejemplo de configuración para una arquitectura media.

Parte de la tarea de configuración es habilitar el soporte de InnoDB, configurando cada una de las variables de acuerdo a lo propuesto en 3.2, desde luego los aspectos más importantes serán los buffers

```
# Example mysql config file for medium systems.
#
# This is for a system with little memory (32M - 64M)
where MySQL plays
# a important part and systems up to 128M very MySQL is
used together with
# other programs (like a web server)
#
# You can copy this file to
# /etc/mf.cnf to set global options,
# mysql-data-dir/my.cnf to set server-specific options (in
this
# installation this directory is
/centia01/develop/database/mysql/var) or
# ~/.my.cnf to set user-specific options.
#
# One can in this file use all long options that the program
supports.
# If you want to know which options a program support,
run the program
# with --help option.
# The following options will be passed to all MySQL clients
[client]
#password = your_password
port = 3306
socket = /tmp/mysql.sock
# Here follows entries for some specific programs
# The MySQL server
[mysqld]
```

```
port = 3306
socket = /tmp/mysql.sock
skip-locking
set-variable = key_buffer=64M
set-variable = max_allowed_packet=256M
set-variable = table_cache=64
set-variable = sort_buffer=512K
set-variable = net_buffer_length=8K
set-variable = myisam_sort_buffer_size=16M
set-variable = max_connections=500
set-variable = interactive_timeout=604800
set-variable = wait_timeout=604800
log-bin
server-id = 1
set-variable=default_table_type=innodb
# Point the following paths to different dedicated disks
tmpdir = /var/tmp/
#log-update = /centia01/final/database/mysql/var/log-
catarina
# Uncomment the following if you are using BDB tables
set-variable = bdb_cache_size=4M
set-variable = bdb_max_lock=10000
# Uncomment the following if you are using Innobase
tables
innodb_data_file_path = ibdata1:30G:autoextend
innodb_data_home_dir = /database/mysql/innodb
innodb_log_group_home_dir = /database/mysql/innodb
innodb_log_arch_dir = /database/mysql/innodb
set-variable = innodb_mirrored_log_groups=1
```

```
set-variable = innodb_log_files_in_group=3
set-variable = innodb_log_file_size=5M
set-variable = innodb_log_buffer_size=16M
innodb_flush_log_at_trx_commit=1
innodb_log_archive=0
set-variable = innodb_buffer_pool_size=256M
set-variable = innodb_additional_mem_pool_size=256M
set-variable = innodb_file_io_threads=4
set-variable = innodb_lock_wait_timeout=50
#set-variable = innodb_force_recovery=3
[mysqldump]
quick
set-variable = max_allowed_packet=256M
[mysql]
prompt=(\u) [\d]>\_
no-auto-rehash
# Remove the next comment character if you are not
familiar with SQL
#safe-updates
[isamchk]
set-variable = key_buffer=20M
set-variable = sort_buffer=20M
set-variable = read_buffer=2M
set-variable = write_buffer=2M
[myisamchk]
set-variable = key_buffer=20M
set-variable = sort_buffer=20M
set-variable = read_buffer=2M
set-variable = write_buffer=2M
```

[mysqlhotcopy] interactive-timeout

Tabla 3.2 Archivo de Configuración my.cnf, my.ini

Arranque y Terminación

Arranque del manejador

Una vez configurado se puede iniciar el demonio del dbms a través del comando "mysqld" o alguna de sus variantes "safe_mysqld", "mysqld-max"

Esto se puede hacer desde cualquier terminal y se pueden pasar como parametros algunas de las mismas variables disponibles para configuración (en caso de necesitar alguna opción particular)

Terminación del manejador

Se puede hacer de 2 maneras

- a) Usando el comando "mysqladmin shutdown"
- b) Matando el proceso asociado

Creación de bases y cuentas de usuario

La tabla 3.3 muestra la manera en que cualquier usuario puede conectarse a la base llamada 'mysql' con el usuario 'root', para ello es indispensable contar con el password correspondiente; como se presentará más adelante la cuenta del super usuario 'root' se administra de manera similar que los demás usuario.

```
[digital@pcproal digital]$ mysql -u root -p mysql
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.0-alpha-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

(root) [mysql]>
```

Conexión con el dbms

La tabla muestra las tablas correspondientes a la base principal y que administra todos los usuarios, máquinas y bases permitidas en el manejador, recordando que se trata de un esquema relacional en el cual por ejemplo los usuarios se encuentran vinculados son bases de datos, de manera que no pueden existir problemas de seguridad al haber algun usuario malintencionado que pretenda modificar una base a la cual no tenga acceso.

```
(root) [mysql]> show databases;
+------------+
| Database   |
+------------+
| mysql      |
+------------+
1 rows in set (0.12 sec)

(root) [mysql]> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| help_category   |
| help_relation   |
| help_topic      |
| host            |
| tables_priv     |
| user            |
+-----------------+
9 rows in set (0.10 sec)
```

Tabla 3.4 Tablas de administración del dbms

Para cada tabla se puede emplear el comando 'desc' o 'describe' para analizar la estructura de cada tabla y apreciar la relación que tiene con las demás. Para dar de un alta un usuario, ejemplo de la tabla 3.5, se debe crear el usuario dentro de la tabla 'user', crear la base de datos y posteriormente asociar dicho usuario con la base en la tabla 'bd', todo lo anterior

utilizando instrucciones de SQL tradicionales.

```
(root) [mysql]> desc user;
+-----------------------+---------------------------------+-------------------+------+---
-+---------+-------+
| Field                 | Type                            | Collation         | Null | Key |
Default | Extra |
+-----------------------+---------------------------------+-------------------+------+---
-+---------+-------+
| Host                  | varchar(60) binary              | binary            |      | PRI |
|       |
| User                  | varchar(16) binary              | binary            |      | PRI |
|       |
| Password              | varchar(45) binary              | binary            |      |     |
|       |
| Select_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Insert_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Update_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Delete_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Create_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Drop_priv             | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Reload_priv           | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Shutdown_priv         | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| Process_priv          | enum('N','Y')                   | latin1_swedish_ci |      |
| N       |       |
| File_priv             | enum('N','Y')                   | latin1_swedish_ci |      |     |
N       |       |
```

Unidad IV

¿Qué son bases de datos distribuidas?

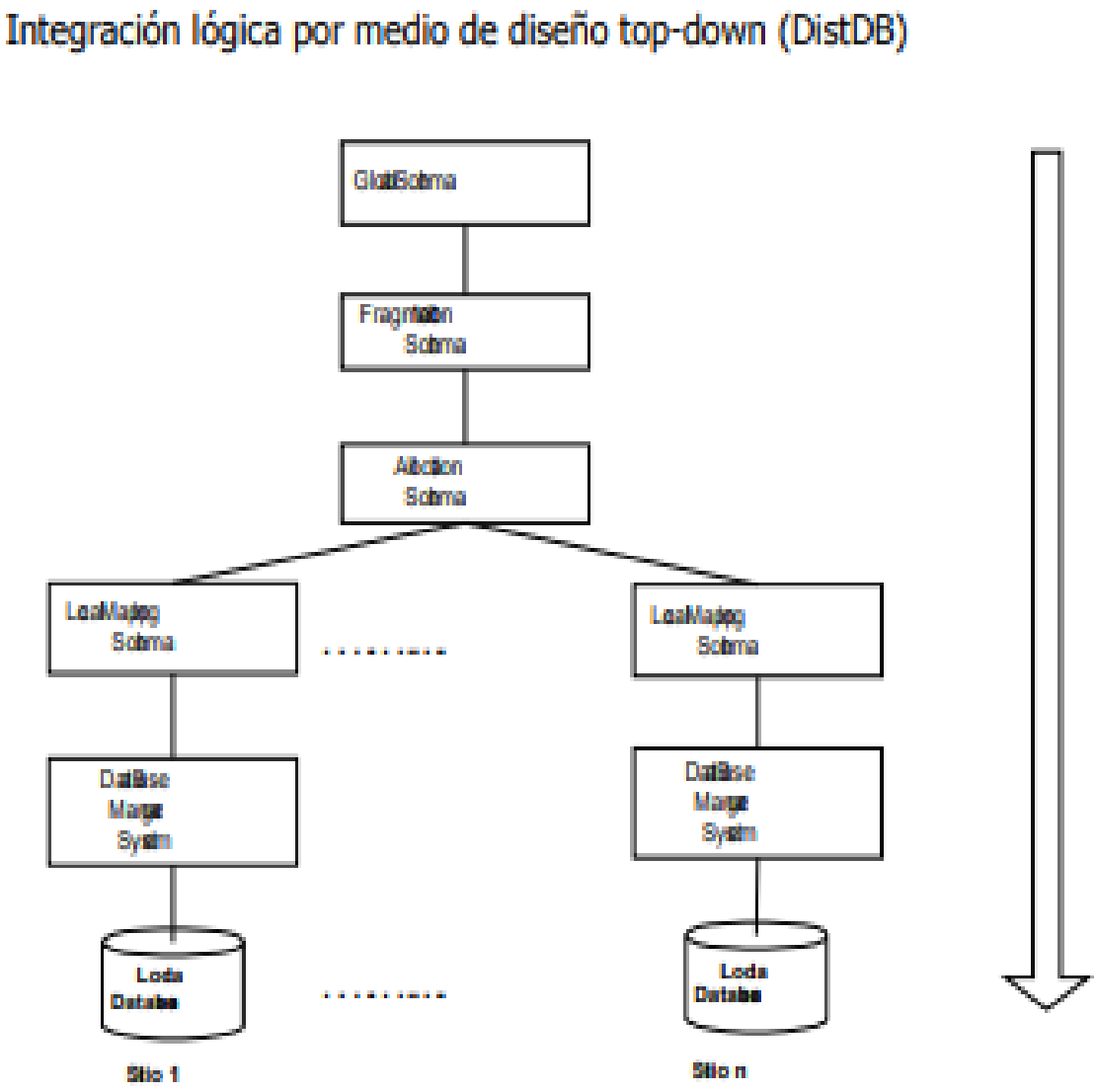
Son un grupo de datos que pertenecen a un sistema pero a su vez esta reapartido entre ordenadores de una misma red, ya sea a nivel local o cada uno en una diferente localizacion geografica, cada sitio en la red es autónomo en sus capacidades de procesamiento y es capaz de realizar operaciones locales y en cada uno de estos ordenadores debe estar ejecutandose una aplicación a nivel global que permita la consulta de todos los datos como si se tratase de uno solo.

Comparación

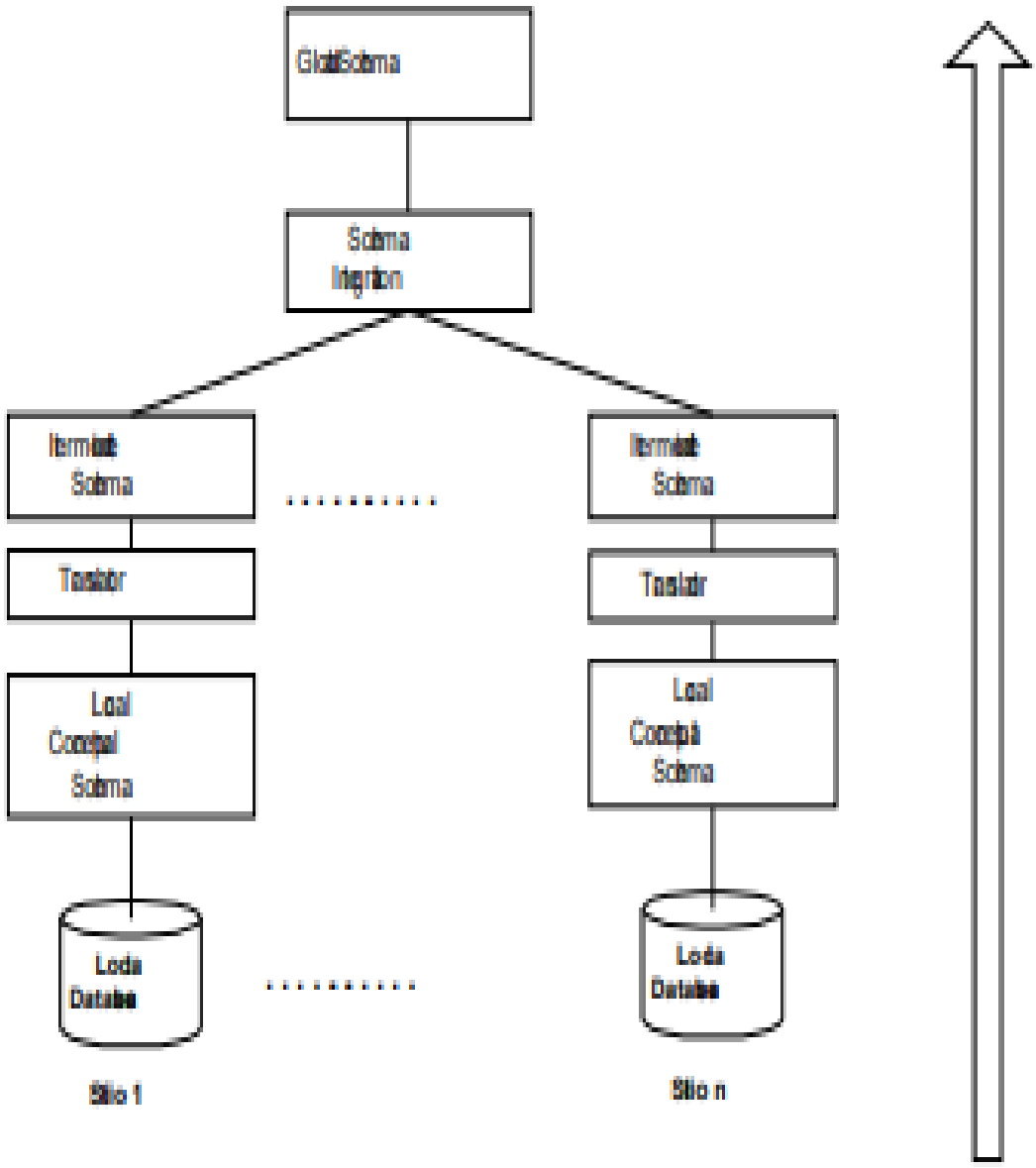
Centralizado	Distribuido
Control centralizado: un solo DBA	Control jerárquico: DBA global y DBA local
Independencia de Datos: Organización de los datos es transparente para el programador	Transparencia en la Distribución: Localización de los datos es un aspecto adicional de independencia de datos
Reducción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

-Para tener una base de datos distribuida debe cumplirse las condiciones de una Red Computacional. Una red de comunicación provee las capacicades para que un proceso ejecutandose en un sitio de la red envie y reciba mensajes de otro proceso ejecutandose en un sitio distinto. Parámetros a considerar incluyen: Retraso en la entrega de mensajes, Costo de transmisión de un mensaje y Confiabilidade de la red. Diferentes tipos de redes: point-to-point, broadcast, lan, wan.

Arquitectura de Base de Datos

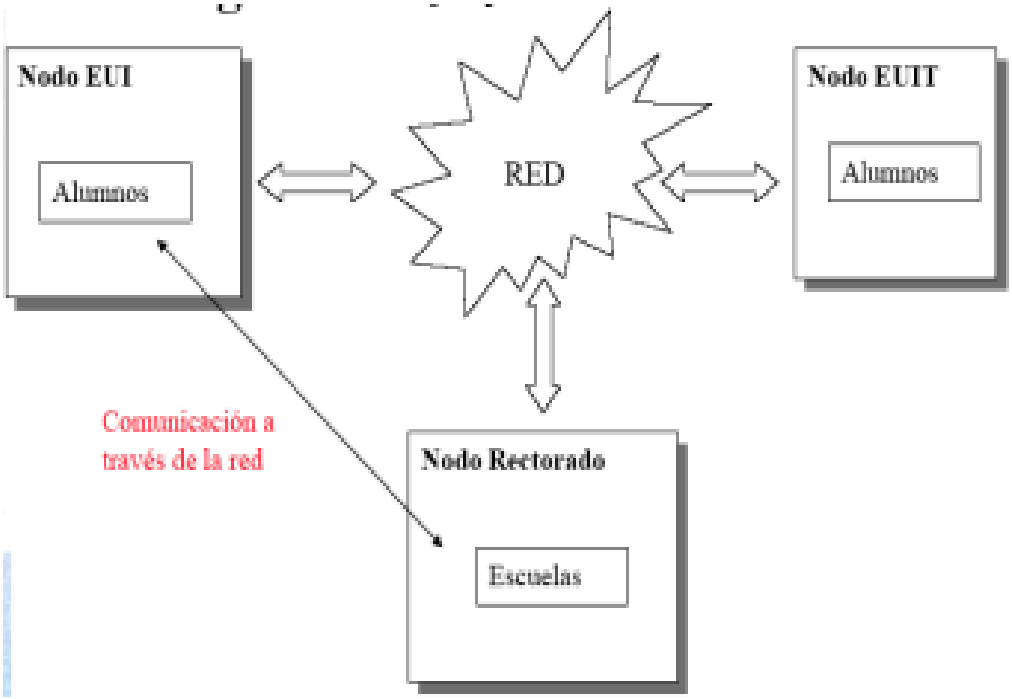


Integración lógica por medio de bottom-up (Multidatabase)



Global Schema: Define todos los datos que están incluidos en la bd distribuida tal como si la bd no fuera distribuida. Consiste de una definición de relaciones globales. -Fragmentation Schema: Traducción entre relaciones globales y fragmentos. (Una relación global puede consistir de varios fragmentos pero un fragmento está asociado con sólo una relación global) -Allocation Schema: Define el sitio (o sitios) en el cual un fragmennto está localizado. -Local Maping Schema: Traduce los fragamentos locales a los objetos que son manejados por el SMBD local Separación entre fragmentación y localización. -Transparencia de Fragmentación -Transparencia de Localización -Control explícito de redundancia -Independencia de BD locales

Ejemplo de una Base de Datos Distribuidas



Tipos de almacenamiento

1-Replica El sistema conserva varias copias o réplicas idénticas de unatabla. Cada réplica se almacena en un nodo diferente. Ventajas: Disponibilidad: El sistema sigue funcionando aún en caso de caída de uno de los nodos. Aumento del paralelismo: Varios nodos pueden realizar consultas en paralelo sobre la misma tabla. Cuantas más réplicas existan de la tabla, mayor serála posibilidad de que el dato buscado se encuentre en el nodo desde el que se realiza la consulta, minimizando con ello el tráfico de datos entre nodos. Inconveniente: Aumento de la sobrecarga en las actualizaciones: El sistema debe asegurar que todas las réplicas de la tabla sean consistentes. Cuandose realiza una actualización sobre una de las réplicas, los cambios deben propagarse a todas las réplicas de dicha tabla a lo largo del sistema distribuido.

2.-Fragmentación Existen tres tipos de fragmentacion la horizontal, la vertical y la mixta

1.-Fragmentación Horizontal Unatabla T se divide en subconjuntos, T1, T2, ...Tn. Los fragmentos se definen a través de una operación de selección y su reconstrucción se realizaráconuna operación de unión de los fragmentos componentes. Cada fragmento se sitúa en un nodo.

Pueden existir fragmentos no disjuntos: combinación de fragmentación y replicación.

2.-Fragmentación Vertical Una tabla T se divide en subconjuntos, T1, T2, ...Tn. Los fragmentos se definen a través de una operación de proyección. Cada fragmento debe incluir la clave primaria de la tabla. Su reconstrucción se realizará con una operación de join de los fragmentos componentes, pueden existir fragmentos no disjuntos: combinación de fragmentación y replicación.

3.-Fragmentación Mixta Como el mismo nombre indica es una combinación de las dos anteriores vistas he aquí un ejemplo apartir de una tabla fragmentada horizontalmente.

3.-Replica y Fragmentación Las técnicas de réplica y fragmentación se pueden aplicar sucesivamente a la misma relación de partida. Un fragmento se puede replicar y a su vez esa réplica ser fragmentada, para luego replicar alguno de esos fragmentos. 4.-Niveles de

Transparencia en una Base de Datos Distribuida El propósito de establecer una arquitectura de un sistema de bases de datos distribuidas es ofrecer un nivel de transparencia adecuado para el manejo de la información. La transparencia se define como la separación de la semántica de alto nivel de un sistema de los aspectos de bajo nivel relacionados a la implementación del mismo. Un nivel de transparencia adecuado permite ocultar los detalles de implementación a las capas de alto nivel de un sistema y a otros usuarios. El sistema de bases de datos distribuido permite proporcionar independencia de los datos. La independencia de datos se puede dar en dos aspectos: lógica y física. .1 Independencia lógica de datos. Se refiere a la inmunidad de las aplicaciones de usuario a los cambios en la estructura lógica de la base de datos. Esto permite que un cambio en la definición de un esquema no debe afectar a las aplicaciones de usuario. Por ejemplo, el agregar un nuevo atributo a una relación, la creación de una nueva relación, el reordenamiento lógico de algunos atributos

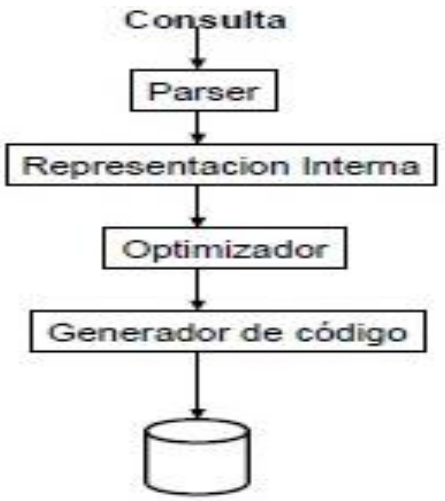
2 Independencia física de datos. Se refiere al ocultamiento de los detalles sobre las estructuras de almacenamiento a las aplicaciones de usuario. la descripción física de datos puede cambiar sin afectar a las aplicaciones de usuario. Por ejemplo, los datos pueden ser movidos de un disco a otro, o la organización de los datos puede cambiar. La transparencia al nivel de red se refiere a que los datos en un SBDD se accedan sobre una red de computadoras, sin embargo, las aplicaciones no deben notar su existencia. La transparencia al nivel de red conlleva a dos cosas: .1Transparencia sobre la localización de datos. el comando que se usa es independiente de la ubicación de los datos en la red y del lugar en donde la operación se lleve a cabo. Por ejemplo, en Unix existen dos comandos para hacer una copia de archivo. Cp se utiliza para copias locales y rcp se utiliza para copias remotas. En este caso no existe transparencia sobre la localización. .2Transparencia sobre el esquema de nombramiento. Lo anterior se logra proporcionando un nombre único a cada objeto en el sistema distribuido. Así, no se debe mezclar la información de la localización con en el nombre de un objeto. La transparencia sobre replicación de datos se refiere a que si existen réplicas de objetos de la base de datos, su existencia debe ser controlada por el sistema no por el usuario. Se debe tener en cuenta que cuando el usuario se encarga de manejar las réplicas en un sistema, el trabajo de éste es mínimo por lo que se puede obtener una

eficiencia mayor. Sin embargo, el usuario puede olvidarse de mantener la consistencia de las réplicas teniendo así datos diferentes. La transparencia a nivel de fragmentación de datos permite que cuando los objetos de la bases de datos están fragmentados, el sistema tiene que manejar la conversión de consultas de usuario definidas sobre relaciones globales a consultas definidas sobre fragmentos. Así también, será necesario mezclar las respuestas a consultas fragmentadas para obtener una sola respuesta a una consulta global. El acceso a una base de datos distribuida debe hacerse en forma transparente. En resumen, la transparencia tiene como punto central la independencia de datos. La responsabilidad sobre el manejo de transparencia debe estar compartida tanto por el sistema operativo, el sistema de manejo de bases de datos y el lenguaje de acceso a la base de datos distribuida. Entre estos tres módulos se deben resolver los aspectos sobre el procesamiento distribuido de consultas y sobre el manejo de nombres de objetos distribuidos.

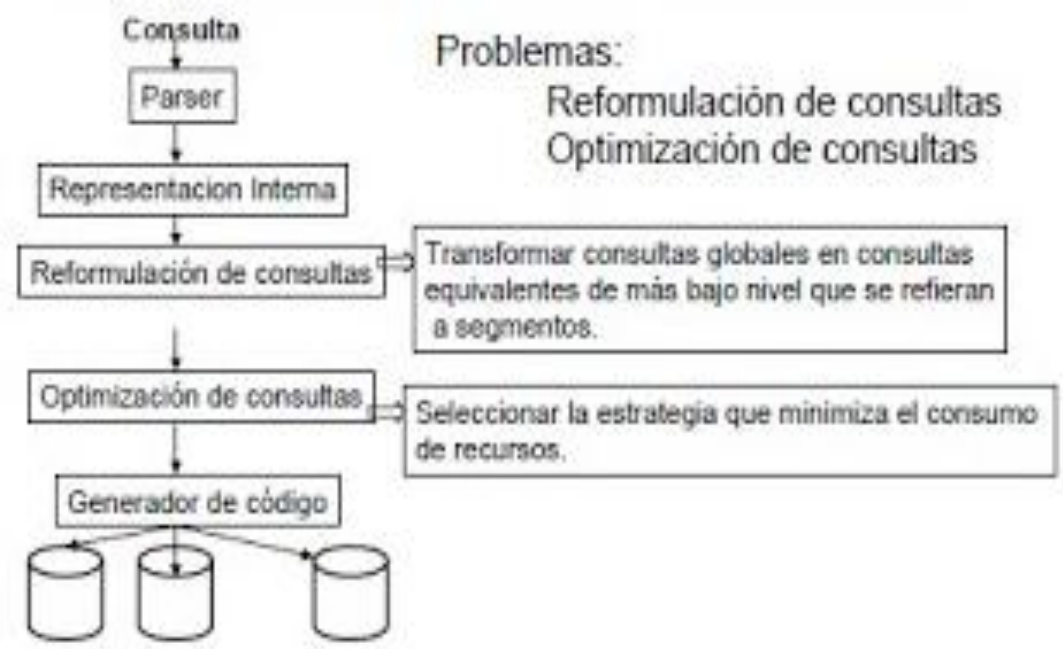
Metodologia de procesamiento de consultas distribuidas

Primeramente se debe de contar con heterogenidad de los datos, para que puedan ser usados para formular consultas. Tenemos los sigueintes ejemplos:

BD CENTRALIZADA



BD DISTRUIBUIDA



Así como también necesitamos contar con:

-Localización de los datos para generar reglas heuristicas

-Descomposición de consultas en paralelo en cada nodo

-Reducir la cantidad de datos a transferir en la red

OPTIMIZACION DE CONSULTAS DISTRIBUIDAS

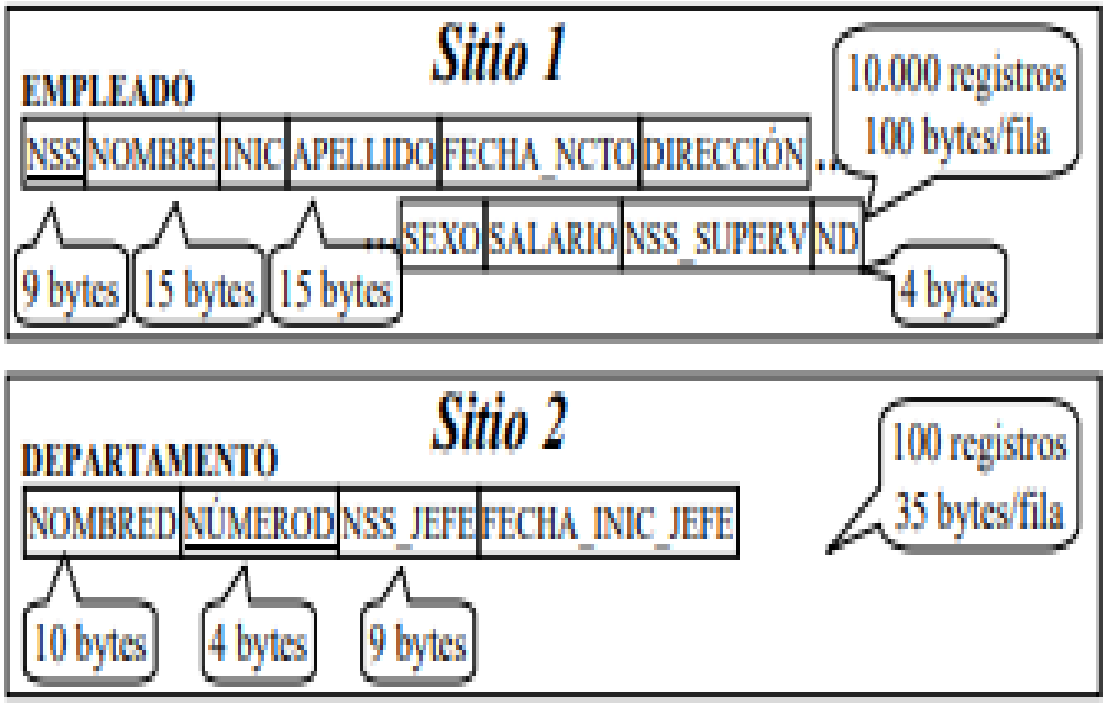
Para poder optimizar una consulta necesitamos tener claras las propiedades del algebra relacional para asegurar la reformulacion de la consulta, al optimizar una consulta obtenemos los siguientes beneficios:

- Minimizar costos

-Reducir espacios de comunicaciones

-Seguridad en envios de informacion

Algoritmos de optimización de consultas en SGBDD: reducir la cantidad de datos a transferir



SELECT Nombre, Apellido, NombreD FROM Empleado inner join Departamento on ND=NumeroD

Resultado 10.000 filas (una por empleado, si todo empleado tiene ND) de 40 bytes/fila

• La consulta se solicita en Sitio 3. Hay tres estrategias: – Transferir Empleado y Departamento a Sitio 3. Hacer allí la reunión: 10.000*100 + 100*35 = 1.003.500 bytes transferidos – Transferir Empleado a Sitio 2. Hacer allí la reunión y enviar el resultado a Sitio 3: 10.000*100 + 10.000*40 = 1.400.000 bytes transferidos – Transferir Departamento a Sitio 1. Hacer allí la reunión y enviar el resultado a Sitio 3: 100*35 + 10.000*40 = 403.500 bytes transferidos Æ mejor opción

Procesamiento de consultas en BDD: coste de transferir datos

• La misma consulta se solicita en Sitio 2. Dos estrategias: – Transferir Empleado a Sitio 2 y hacer allí la reunión: 10.000*100 = 1.000.000 bytes transferidos – Transferir Departamento a Sitio 1. Hacer allí la reunión y enviar el resultado a Sitio 2: 10.000*40 + 100*35 = 403.500 bytes transferidos Æ mejor opción • Operación de semirreunión (|X): – Es otra estrategia que a veces mejora los resultados – Se basa en transferir solamente las tuplas y atributos estrictamente necesarios

• En el caso de la consulta anterior, solicitada en Sitio 2, una estrategia con semirreunión puede ser: – Transferir R1=πNumeroD (Departamento) a Sitio 1: 4*100=400 bytes – R1 se reúne con Empleado en Sitio 1. Transferir a Sitio 2 R2=πNombre, Apellido, ND (R1|X| Empleado): 34*10.000 = 340.000 bytes – R2 se reúne con Departamento en Sitio 2 para obtener el resultado de la consulta. – Con esta estrategia se transfieren 340.400 bytes Æ mejor opción que las anteriores

Descomposición de actualizaciones y consultas

• SGBD sin transparencia de distribución: hay que indicar el sitio y la tabla sobre la que se realiza la consulta.

• SGBD sin transparencia de replicación: hay que mantener a mano la consistencia de los datos

• SGBD con transparencia de distribución, replicación y fragmentación: – La consulta o actualización se expresan como si se tratase de un SGBD centralizado – El SGBD se encarga de descomponer y dirigir a los fragmentos adecuados

BDD y cliente-servidor: arquitectura de 2 niveles

• Los SGBD totalmente distribuidos (transparentes) aun no son viables comercialmente

• En su lugar se han creado sistemas basados en cliente-servidor

• La forma habitual de dividir la funcionalidad del SGBD entre cliente y servidor ha sido la arquitectura de 2 niveles: – Servidor (o servidor SQL): donde se sitúa el SGBD. Una BDD se situaría en varios servidores.

– Clientes:

• Envían consultas/actualizaciones a servidores

• Tienen interfaces SQL, de usuario y funciones de interfaz del lenguaje de programación

• Consultan en el diccionario de datos la información sobre la distribución de la BD entre los servidores. Tienen módulos que descomponen consultas globales en varias locales a cada servidor

• Interacción cliente-servidor (arquitectura de 2 niveles): – El cliente analiza la consulta del usuario. La descompone en varias subconsultas y envía cada una a un servidor. (También puede hacerlo el usuario a mano)

– Cada servidor ejecuta su subconsulta y devuelve el resultado al cliente – El cliente combina los resultados recibidos y muestra al usuario el resultado de su consulta.

• En este enfoque al servidor se le llama máquina back-end (o subyacente) y al cliente máquina front-end (de la parte visible).

• Al servidor también se le llama servidor de transacciones y procesador de BD y al cliente procesador de aplicaciones

BDD y cliente-servidor: arquitectura de 3 niveles

Actualmente es más común utilizar una arquitectura en 3 niveles, sobre todo para aplicaciones web.

Las 3 capas son:

– Cliente (Presentación):

• Es la interfaz (interfaces web, formularios, …)

• Suelen usar navegadores web y lenguajes como HTML, JavaScript, PERL, …

• Gestiona las entradas, salidas y la navegación con páginas web estáticas o, cuando accede a BD, con páginas dinámicas (ASP, JSP,…)

– Servidor de aplicaciones (SA) (lógica de negocio):

• Incluye, por ejemplo, consultas basadas en datos introducidos por el usuario, o resultados de consultas a los que da formato y envía para su presentación

• Puede incluir otro tipo de funcionalidad como comprobaciones de seguridad o de la identidad

• Puede acceder a varias BD conectándose mediante ODBC, JDBC u otras técnicas – Servidor de BD (SBD):

• Procesa consultas y actualizaciones solicitadas por la capa de aplicación

• Puede devolver los resultados en formato XML

• La división de funcionalidad entre las 3 capas puede variar.

BDD y cliente-servidor: arquitectura de 3 niveles

• El servidor de aplicaciones (SA) también:

– Tiene acceso a diccionarios de datos para consultar cómo se distribuyen las BDD entre los servidores de BD (SBD)

– Puede incluir módulos para descomponer una consulta en varias subconsultas locales a cada SBD

• La interacción entre SA y SBD puede ser así:

– El SA construye una consulta utilizando datos tomados por el cliente. Descompone la consulta en varias subconsultas, cada una de ellas local a un SBD, y las envía a sus correspondientes SBD. – Cada SBD procesa sus consultas y envía los resultados al SA que las solicitó. Cada vez es más frecuente utilizar el formato XML para devolver los resultados.

– El SA combina los resultados para obtener el resultado de la consulta original. Dota al resultado de un formato, como HTML y lo envía al cliente para su presentación.

• El SA es responsable de (en arquitectura de 2 niveles lo es el cliente): – Generar un plan de ejecución distribuido y supervisar su ejecución.

– Garantizar la consistencia de las réplicas de datos.

– Asegurar la atomicidad de las transacciones globales (que no se puedan ejecutar "a medias", o sea que bien se ejecuta toda la transacción o no se ejecuta nada)

Diseño De Bdd: Sociedad Médica

• Cuenta con una oficina central y 3 centros médicos. Cada centro médico atiende ciertas especialidades. Una especialidad se puede atender en varios centros. Todo centro médico tiene al menos una especialidad.

• Actualmente utilizan la siguiente BD relacional centralizada:

• Diseñar esquemas de fragmentación, replicación y asignación de una BDD que tenga la mayor autonomía local, sabiendo que en los centros se necesita:

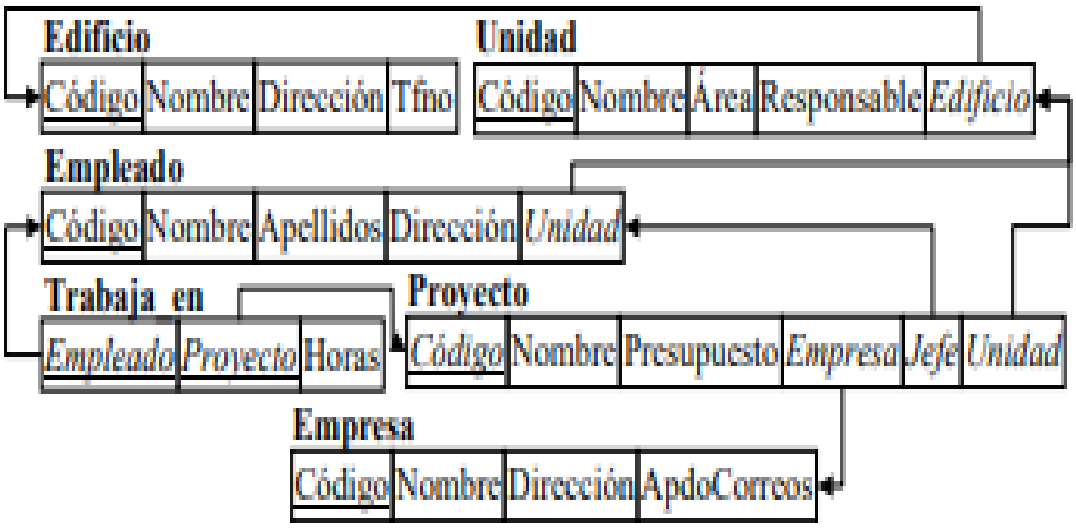
– En la oficina central (Centro.Código=0) la información para realizar las nóminas de todos los empleados.

– En el resto de centros, los horarios de sus consultas y la información de su personal y de sus especialidades.

Diseño de BDD: Servicios informáticos

• Se divide en cuatro edificios, cada uno con su servidor de BD, donde hay una o varias unidades de desarrollo. Cada una se aloja en un solo edificio. Los empleados y proyectos están asignados a una unidad de desarrollo, aunque un empleado puede trabajar en un proyecto que no sea de su unidad.

• Actualmente utilizan la siguiente BD relacional centralizada:



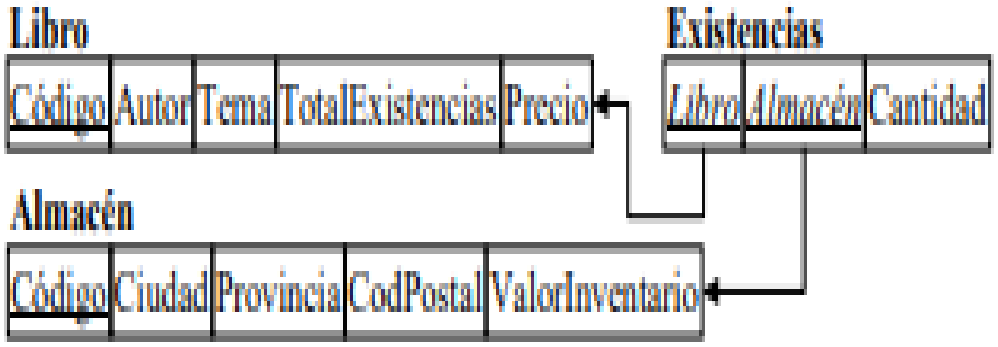
• Diseñar esquemas de fragmentación, replicación y asignación de una BDD sabiendo que :

– Cada unidad gestiona las nóminas de sus empleados y necesita la información de los proyectos que realiza.

– La unidad de Recursos humanos (edificio 0) realiza la facturación y usa todos los datos de las empresas. El resto de unidades sólo necesita el código y nombre de las empresas con las que trabaja.

– Hay proyectos internos, y por tanto sin factura, que tienen valor nulo en su campo Empresa.

Consultas en BDD: Libros



• Esquema de fragmentación:

– Libro_a: σPrecio<=20€(Libro)

– Libro_b: σPrecio>20€ and Precio<=50€(Libro)

– Libro_c: σPrecio>50€ and Precio<=100€(Libro)

– Libro_d: σPrecio>100€(Libro)

– Almacén_1: σCodPostal<=3500(Almacén)

– Almacén_2: σCodPostal>3500 and CodPostal<=70000(Almacén)

– Almacén_3: σCodPostal>70000(Almacén)

– Existencias_i: Existencias |×Almacén=CódigoAlmacén_i

• Esquema de replicación y asignación:

	Servidor 1	Servidor 2	Servidor 3
Libro	Libro_a Libro_d	Libro_a Libro_b	Libro_a, Libro_b Libro_c, Libro_d
Almacén	Almacén_1	Almacén_2	Almacén_3
Existencias	Existencias_1	Existencias_2	Existencias_3

• Qué subconsultas genera la ejecución de la siguiente consulta en el servidor 1: select Código, Total Existencias from Libro where Precio>15 and Precio

Control de concurrencia

El control de concurrencia trata con los problemas de aislamiento y consistencia del procesamiento de transacciones. El control de concurrencia distribuido de una DDBMS asegura que la consistencia de la base de datos se mantiene en un ambiente distribuido multi usuario. Si las transacciones son internamente consistentes, la manera más simple de lograr este objetivo es ejecutar cada transacción sola, una después de otra.

Algoritmos de control de concurrencia

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 (candados o bloqueos).

- Aquellos que intentar ordenar la ejecución de las transacciones de acuerdo a un conjunto de reglas (protocolos).

Basados en estampas de tiempo

Los algoritmos basados en estampas de tiempo no pretenden mantener la seriabilidad por exclusión mutua. En lugar de eso, ellos seleccionan un orden de serialización a prioridad y ejecutan las transacciones, de acuerdo a ellas. Para establecer este ordenamiento, el administrador de transacciones le asigna a cada transacción T1 una estampa de tiempo única t1 (T1) cuando ésta inicia. Una estampa de tiempo es un identificador simple que sirve para identificar cada transacción de manera única.

A diferencia de los algoritmos basados en candados, los algoritmos basados en marcas de tiempo no pretenden mantener la seriabilidad por la exclusión mutua. En su lugar eligen un orden de serializacion en primera instancia y ejecutan las transacciones de acuerdo a ese orden. En estos algoritmos cada transacción lleva asociada una marca de tiempo. Cada dato lleva asociado dos marcas de tiempo: uno de lectura y otro de escritura, que reflejan la marca de tiempo de la transacción que hizo la última operación de ese tipo sobre el dato. Para leer la marca de tiempo de escritura del dato, debe ser menor que el de la transacción, si no aborta. Para escribir las marcas de tiempo de escritura y lectura del dato, deben ser menores que el de la transacción, sino se aborta. Esta técnica está libre de Ínterbloqueos pero puede darse que haya que repetir varias veces la transacción. En los sistemas distribuidos se puede usar un mecanismo como, los relojes de Lamport para asignar marcas de tiempo. El conjunto de algoritmos pesimistas está formado por algoritmos basados en candados, algoritmos basados en ordenamiento por estampas de tiempo y algoritmos híbridos. Los algoritmos optimistas se componen por los algoritmos basados en candados y algoritmos basados en estampas de tiempo.

Algoritmos de cerradura o basados en candados

En los algoritmos basados en candados, las transacciones indican sus intenciones solicitando candados al despachador (llamado el administrador de candados) Los candados son de lectura, también llamados compartidos, o de escritura, también llamados exclusivos.

En sistemas basados en candados, el despachador es un administrador de candados. El administrador de transacciones le pasa al administrador de candados la operación sobre la

base de datos (lectura o escritura) e información asociada, como por ejemplo el elemento de datos que es accesado y el identificador de la transacción que está enviando la operación a la base de datos. El administrador de candados verifica si el elemento de datos que se quiere accesar ya ha sido bloqueado por un candado. Si el candado solicitado es incompatible con el candado con que el dato está bloqueado, entonces, la transacción solicitante es retrasada. De otra forma, el candado se define sobre el dato en el modo deseado y la operación a la base de datos es transferida al procesador de datos. El administrador de transacciones es informado luego sobre el resultado de la operación. La terminación de una transacción libera todos los candados y se puede iniciar otra transacción que estaba esperando el acceso al mismo dato.

Se usan cerraduras o candados de lectura o escritura sobre los datos. Para asegurar la secuencialidad se usa un protocolo de dos fases, en la fase de crecimiento de la transacción se establecen los cerrojos y en la fase de decrecimiento se liberan los cerrojos. Hay que tener en cuenta que se pueden producir ínterbloqueos. En los sistemas distribuidos el nodo que mantiene un dato se encarga normalmente de gestionar los cerrojos sobre el mismo.

Candados de dos fases:

En los candados de dos fases una transacción le pone un candado a un objeto antes de usarlo. Cuando un objeto es bloqueado con un candado por otra transacción, la transacción solicitante debe esperar. Cuando una transacción libera un candado, ya no puede solicitar más candados. En la primera fase solicita y adquiere todos los candados sobre los elementos que va a utilizar y en la segunda fase libera los candados obtenidos uno por uno.

Puede suceder que si una transacción aborta después de liberar un candado, otras transacciones que hayan accesado el mismo elemento de datos aborten también provocando lo que se conoce como abortos en cascada. Para evitar lo anterior, los despachadores para candados de dos fases implementan lo que se conoce como los candados estrictos de dos fases en los cuales se liberan todos los candados juntos cuando la transacción termina (con compromiso o aborta).

Candados de dos fases centralizados:

En sistemas distribuidos puede que la administración de los candados se dedique a un solo nodo del sistema, por lo tanto, se tiene un despachador central el cual recibe todas las solicitudes de candados del sistema. La comunicación se presenta entre el administrador de transacciones del nodo en donde se origina la transacción, el administrador de candados en el nodo central y los procesadores de datos de todos los nodos participantes. Los nodos participantes son todos aquellos en donde la operación se va a llevar a cabo.

Bibliografía básica y complementaria

- 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_detailcsnumber=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://sistemascomputacionalestlahuelilpan.files.wordpress.com/2012/03/bases-de-datos-distribuidas.pdf