



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 2

1. La administración de la base de datos.
2. Objetivos y funciones del administrador de la base de datos.
3. Gestión del almacenamiento.
4. Seguridad de la base de datos.
 - 4.1. Consideraciones generales de Seguridad.
 - 4.2. Seguridad en SQL.
5. Recuperación y Concurrencia.
 - 5.1. Recuperación de transacciones.
 - 5.2. Recuperación del sistema.
 - 5.3. Problemas de concurrencia.
 - 5.4. El bloqueo.





CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 2

Administración de Sistemas de Gestión de Bases de Datos. Funciones. Responsabilidades. Administración de datos.

1. LA ADMINISTRACIÓN DE LA BASE DE DATOS.

La administración de la BD se ocupa básicamente de asegurar que la información esté disponible para los usuarios y aplicaciones en el instante preciso y en la forma adecuada. Esta información ha de proporcionar dos características básicas: precisión y consistencia.

El ABD o DBA (Administrador de la Base de Datos) interactúa tanto con el sistema como con los usuarios para llevar a cabo las tareas que permitan la puesta a punto y mantenimiento de las bases de datos.

2. OBJETIVOS Y FUNCIONES DEL ADMINISTRADOR DE LA BASE DE DATOS.

- Comunicación con los usuarios de la base de datos y responsables del sistema.
 1. Detectar necesidades de uso, roles, privilegios, etc.
 2. Determinar la interfaz de conexión con otros sistemas ya existentes.
 3. Identificar recursos hardware y software disponibles en la organización y en su caso adoptarlos. Caso de no ser suficientes proponer su ampliación y/o mejora.
 4. Determinar la infraestructura de red existente y su posible ampliación en función de las nuevas demandas.
- Planificación, diseño e implementación de los sistemas de bases de datos.
 1. Obtener los cuadernos de carga elaborados por los analistas y diseñadores donde se recogen los detalles de implantación de la BD.

2. Desarrollar los scripts de creación de la BD.
 3. Desarrollar los scripts que permitan la puesta a punto de la BD.
- Establecimiento de normas y procedimientos.
 1. Establecimiento de normas y procedimientos uniformes para controlar la seguridad y la integridad de los datos de forma eficiente. Las normas se aplican particularmente al control del desarrollo y del uso de la programación y de las operaciones de la base de datos.
 2. En el área de la programación, las normas se establecen para asegurar que los programas se revisen y se prueben antes de ponerlos en producción.
 3. En el área de las operaciones, las normas pueden establecerse para mantener los diarios de las operaciones de las transacciones y se crean los procedimientos para la corrección de los errores, para el tratamiento de los puntos de control y para garantizar la copia de seguridad y la recuperación.
 - Mantener la integridad de los datos.
 1. Protección ante accidentes tales como los errores en la entrada de los datos o en la programación, del uso malintencionado de la base de datos y de los fallos del hardware o del software.
 2. Garantizar la actualización de los datos bajo los principios de atomicidad derivados del uso de transacciones.
 - Mantener la seguridad de los datos.
 1. Garantizar el acceso a aquellos usuarios y/o aplicaciones que estén reconocidos e identificados por el sistema.
 2. Determinar el nivel de seguridad exigible: SO, SGBD, o combinación de ambos.
 - Mantener la disponibilidad de los datos.
 1. Establecer los componentes que se han de mantener en sistemas redundantes con el fin de garantizar su uso continuado.
 2. Procedimientos de recuperación ante posibles pérdidas de información.
 3. Determinar la necesidad de gestionar el almacenamiento mediante redes independientes (SAN).
 4. Arranque parada de los procesos/servicios asociados a la BD.
 - Carga de la BBDD.
 1. Desarrollar y ejecutar los scripts que permitan llevar a cabo la «migración» de los datos normalmente existentes en algún otro sistema de almacenamiento de información (ficheros, otros SGBDs, etc.).

- Mantenimiento de la BBDD.

1. «Performance» de la BD. Adecuar los parámetros de la base a las exigencias cambiantes de cada momento: dispositivos de almacenamiento, buffers, índices, área de memoria... con el fin de garantizar la eficiencia en el acceso a la información.
2. Creación de nuevos usuarios, grupos, roles...
3. Modificación de objetos ya existentes, o en su caso creación de nuevos.
4. Optimización del diccionario de datos y consultas.
5. Políticas de salvaguarda y recuperación periódicas.

3. GESTIÓN DEL ALMACENAMIENTO.

Toda base de datos se graba en algún sistema de almacenamiento secundario constituido por uno o varios ficheros físicos soportados por el sistema de ficheros presente en la arquitectura hardware de soporte al SGBD. Cada fichero únicamente puede estar asociado a una base de datos. Para poder concretar los aspectos relativos a la gestión del almacenamiento tomaremos como referencia el SGBDR «Oracle».

Oracle establece los siguientes niveles de estructuras presentes en el sistema de almacenamiento:

- Ficheros: con un tamaño que el DBA establece en el momento de creación de la base de datos.
- Espacios de Tabla (TableSpaces). Se trata de unidades lógicas de almacenamiento en las que se divide una BD. Pueden estar almacenados en varios ficheros físicos, aunque Oracle los trata como una única unidad lógica.
- Segmentos: Agrupación de bloques Oracle no contiguos. Cada segmento suele estar pensado para especializarse en alguno de los tipos de informaciones o estructuras que ha de manejar la base de datos, por ejemplo puede haber segmentos de datos, rollback, índices,...
- Extensiones: conjunto de datos contiguos.

Una base de datos se almacena a nivel lógico como una serie de tablespaces que contiene los diferentes objetos que forman la propia base. Cada tablespace se subdivide en segmentos y estos últimos en extensiones. Por último es posible manejar la información contenida en los ficheros mediante bloques Oracle, cuya estructura y tamaño es distinto a la del bloque del sistema operativo.

Desde el punto de vista físico, la base de datos se almacena mediante una colección de ficheros representados en el sistema de almacenamiento secundario. La relación entre la estructura física y lógica se produce a dos niveles:

1. Entre ficheros y tablespaces.
2. Entre el bloque Oracle y el bloque del sistema operativo.

La gestión del almacenamiento es especialmente relevante en el momento de la propia creación de la base de datos. Por ello es necesario realizar las estimaciones de demanda de almacenamientos presentes y futuros que servirán de referencia al DBA para establecer el tamaño de los tablespaces y consecuentemente de los ficheros asociados.

4. SEGURIDAD DE LA BASE DE DATOS.

4.1. CONSIDERACIONES GENERALES DE SEGURIDAD.

Los objetos susceptibles de la aplicación de un mecanismo de seguridad pueden ser desde bases de datos completas hasta valores específicos de una fila y columna de una tabla. Los mecanismos de seguridad deben garantizar que los usuarios sólo pueden realizar aquellas operaciones para las que están autorizados sobre ciertos objetos particulares. Por otro lado, hay que tener en cuenta que distintos usuarios pueden tener diferentes tipos de autorizaciones sobre los mismos objetos.

Las medidas de seguridad de la información las toma el administrador de la base de datos, para lo que utiliza las herramientas proporcionadas por el lenguaje al efecto.

En el caso del SQL, el sistema cuenta con diferentes mecanismos implicados en el mantenimiento de la seguridad:

- El sistema de gestión de vistas.
- El subsistema de autorización, mediante el cual usuarios con derechos específicos pueden conceder de manera selectiva y dinámica esos derechos a otros usuarios, y su revocación.
- Control de transacciones.

4.2. SEGURIDAD EN SQL.

Las vistas pueden ser utilizadas con propósitos de seguridad, delimitando el acceso de los usuarios a ciertas porciones de la información. Veamos unos cuantos ejemplos que nos permiten distinguir distintas formas de aplicar seguridad a la base de datos:

```
CREATE VIEW $PROV_DE_MADRID
AS SELECT * FROM Proveedores
WHERE UPPER(CIUDAD) = 'MADRID';

/* Los usuarios de esta vista ven un subconjunto horizontal de la tabla */

CREATE VIEW $NOMBRE_CIUDAD
AS SELECT CodProv, Nombre, Ciudad FROM Proveedores;

/* Los usuarios de esta vista obtienen un subconjunto vertical de la tabla */

CREATE VIEW $MADRID_NOMBRE_CIUDAD
AS SELECT CodProv, Nombre, Ciudad FROM Proveedores
WHERE UPPER(CIUDAD) = 'MADRID';

/* Los usuarios de esta vista obtienen un subconjunto de filas y columnas */

CREATE VIEW $MYTABLES
AS SELECT * FROM SYSTABLES
WHERE CREATOR = USER;

/* Se obtiene una vista de todas las tablas cuyo propietario es el usuario */
```

Por lo tanto, las vistas pueden ser creadas utilizando la sintaxis siguiente:

```
CREATE VIEW [user.]view [ ( alias [, alias] ... ) ]  
AS query  
[WITH CHECK OPTION [CONSTRAINT constraint] ]
```

Es importante destacar que el uso de 'WITH CHECK OPTION' garantiza que se puedan insertar registros en la tabla, a través de la vista, sólo si se cumple la condición de restricción especificada (por ejemplo, en la primera vista se podrán insertar ciudades que no sean Madrid, a no ser que se utilice la opción 'WITH CHECK OPTION').

Para completar los mecanismos de seguridad, como ya hemos indicado, es necesario disponer de ciertas órdenes que nos permitan asignar a los usuarios permisos de acceso a tablas y vistas de la base de datos. Para ello se utilizan las sentencias GRANT y REVOKE. Su sintaxis es la siguiente:

```
GRANT database_priv [, database_priv] ...  
TO user [, user] ...  
[IDENTIFIED BY password [, password] ... ]  
  
Siendo database_priv = DBA | CONNECT | RESOURCE  
  
GRANT RESOURCE [ (quota [klm] ) ]  
ON tablespace  
TO { PUBLIC | user [, user] ... }  
  
Siendo quota el espacio en bytes.  
  
GRANT { object_priv [, object_priv] ... | ALL [PRIVILEGES] }  
ON [user.]object  
TO { user | PUBLIC } [, user] ...  
[WITH GRANT OPTION]  
  
Siendo object_priv = ALTER | DELETE | INDEX | INSERT | REFERENCES |  
SELECT | UPDATE. Si se utiliza UPDATE o REFERENCES, se pueden  
especificar columnas.  
  
REVOKE { [CONNECT] [, RESOURCE] [,DBA] }  
FROM user [, user]  
  
REVOKE space_privilege ON tablespace  
FROM user [, user]  
  
REVOKE {object_priv [, object_priv] ... | ALL [PRIVILEGES] }  
ON [user.]object  
FROM {user | PUBLIC} [,user] ...
```

5. RECUPERACIÓN Y CONCURRENCIA.

5.1. RECUPERACIÓN DE TRANSACCIONES.

Una transacción es una unidad lógica de trabajo. Dicho de otra forma, una transacción es una secuencia de operaciones en una base de datos mediante la cual un estado consistente de la base de datos se transforma en otro estado consistente, sin conservar por fuerza la consistencia en todos los estados intermedios.

El componente del sistema encargado de lograr este propósito es el gestor de transacciones, y las operaciones en SQL COMMIT y ROLLBACK son la clave de su funcionamiento: COMMIT indica que una transacción ha finalizado con éxito al SGBD, mientras que ROLLBACK indica al SGBD que debe recuperar su estado justo en el momento anterior a comenzar la transacción que no ha podido finalizar (en su defecto, retornará al último estado de la base de datos en el que se realizó un COMMIT).

La capacidad de realizar la recuperación se debe a que los SGBD mantienen un histórico de todas las operaciones de actualización realizadas en el sistema. Por tanto, si resulta necesario anular alguna modificación específica, el sistema puede utilizar la entrada concreta en el histórico para restaurar el valor original del objeto modificado.

Aun con todo este mecanismo de recuperación, es posible que el sistema caiga justo después de haber realizado el COMMIT, sin que los datos se hayan almacenado físicamente en el disco. Los SGBD deben garantizar el COMMIT incluso en este supuesto (siempre y cuando el histórico haya almacenado físicamente la transacción antes de ejecutar el comando COMMIT), y lo hacen de forma que cuando el sistema se arranca de nuevo toma las transacciones que hayan podido quedar a la espera del COMMIT y las confirma en el sistema. Este método se conoce como protocolo de bitácora de escritura avanzada.

Para asegurar la integridad de los datos se necesita que el sistema de base de datos mantenga las siguientes propiedades de las transacciones:

- Atomicidad. O todas las operaciones de la transacción se realizan adecuadamente en la base de datos, o ninguna de ellas.
- Consistencia. La ejecución aislada de la transacción (es decir, sin otra transacción que se ejecute concurrentemente) conserva la consistencia de la base de datos.
- Aislamiento. Aunque se ejecuten varias transacciones concurrentemente, el sistema garantiza que para cada par de transacciones T1 y T2, se cumple para T1 que T2 ha terminado su ejecución antes de que comience T1, o bien que T2 ha comenzado su ejecución después de que T1 termine. De este modo, cada transacción ignora al resto de las transacciones que se ejecuten concurrentemente en el sistema.
- Durabilidad. Tras la finalización con éxito de una transacción, los cambios realizados en la base de datos permanecen, incluso si hay fallos en el sistema.

Estas propiedades a menudo reciben el nombre de propiedades ACID; el acrónimo se obtiene de la primera letra de cada una de las cuatro propiedades en inglés (Atomicity, Consistency, Isolation, Durability).

En ausencia de fallos, todas las transacciones se completan con éxito. Sin embargo, una transacción puede que no siempre termine su ejecución con éxito. Una transacción de este tipo se denomina abortada. Si se pretende asegurar la atomicidad, una transacción abortada no debe tener efecto sobre el estado de la base de datos. Así, cualquier cambio que haya hecho la transacción abortada sobre la base de datos debe deshacerse. Una vez que se han deshecho los cambios realizados por la transacción abortada, se dice que la transacción se ha retrocedido. Una transacción que termina con éxito se dice que está comprometida. Una transacción comprometida que haya hecho modificaciones en la base de datos llevándola a un nuevo estado consistente, que permanece incluso si hay un fallo en el sistema.

Cuando una transacción se ha comprometido, no se pueden deshacer sus efectos abortándola. La única forma de deshacer los cambios de una transacción comprometida es ejecutando una transacción compensadora. Sin embargo, no siempre se puede crear dicha transacción compensadora. Por tanto, se deja al usuario la responsabilidad de crear y ejecutar transacciones compensadoras, y no las gestiona el sistema de base de datos.

Se puede precisar más lo que se entiende por terminación con éxito de una transacción, a través de un modelo simple abstracto de transacción. Una transacción debe estar en uno de los estados siguientes:

- Activa. El estado inicial. La transacción permanece en este estado durante su ejecución.
- Parcialmente comprometida. Después de ejecutarse la última instrucción.
- Fallida. Tras descubrir que no puede continuar la ejecución normal.
- Abortada. Después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción.
- Comprometida. Tras completarse con éxito.

5.2. RECUPERACIÓN DEL SISTEMA.

El sistema debe estar preparado para recuperarse no sólo de fallos puramente locales, sino también de fallos globales. Un fallo local afecta sólo a la transacción en la que se presentó el fallo, mientras que un fallo global afecta a todas las transacciones que se estaban realizando en el momento del fallo.

Ante un fallo global, el sistema deberá recuperarse de dos tipos de fallos:

- Fallos del sistema, que afectan a todas las transacciones que se están realizando, pero no dañan físicamente a la base de datos. También se conocen como caídas suaves.
- Fallos de los medios de almacenamiento, que causan daños a la base de datos o a una porción de ella, y afectan al menos a las transacciones que están utilizando esa porción. También se conocen como caídas duras.

5.3. PROBLEMAS DE CONCURRENCIA.

La mayoría de los SGBD son multiusuario. En estos sistemas se necesita un mecanismo de control de concurrencia para asegurar que ninguna transacción concurrente interfiera con las operaciones de las demás.

Son tres las situaciones en las que una transacción, aunque correcta en sí, puede producir de todos modos un resultado incorrecto debido a una interferencia por parte de alguna otra transacción:

1. El problema de la modificación perdida.
2. El problema de la dependencia no comprometida.
3. El problema del análisis inconsistente.

1. El problema de la modificación perdida.

Consideremos la siguiente situación:

TRANSACCIÓN X	TIEMPO	TRANSACCIÓN Y
Leer Fila 1	T1	
	T2	Leer Fila 1
Actualizar Fila 1	T3	
	T4	Actualizar Fila 1

La modificación realizada por la transacción X se pierde porque la transacción Y sobrescribe el resultado sobre el resultado anterior.

2. El problema de la dependencia no comprometida.

Este problema se presenta cuando se permite a una transacción leer (o modificar) un registro (fila) que ha sido puesto al día por otra transacción, y esta última todavía no la ha comprometido. Es evidente que existe la posibilidad de que nunca se comprometa, lo cual indicaría que los datos transitorios podrían no ser correctos y generar un error encadenado en otras transacciones.

3. El problema del análisis inconsistente.

Supongamos una situación inicial en la que se tengan los siguientes valores de columnas: Col1 = 40, Col2 = 50, Col3 = 30 y el siguiente cronograma de transacciones:

TRANSACCIÓN X	TIEMPO	TRANSACCIÓN Y
Seleccionar Col1 → 40 Suma = 40	T1	
Seleccionar Col2 → 50 Suma = 90	T2	
	T3	Seleccionar Col3 → 30
	T4	Actualizar Col3 ← 20
	T5	Seleccionar Col1 → 40
	T6	Actualizar Col1 ← 50
	T7	COMMIT
Seleccionar Col3 → 20 Suma = 110	T8	

Nos encontramos con la situación de que el resultado de una transacción ha interferido claramente en el resultado de otra de duración mayor. El resultado 110 es incorrecto, y por tanto decimos que X ha realizado un análisis inconsistente.

5.4. EL BLOQUEO.

El bloqueo es la técnica más usual que se utiliza para resolver problemas de concurrencia. La noción básica de bloqueo es simple: cuando una transacción requiere la seguridad de que algún objeto en la cual está interesada no cambiará de alguna manera no predecible sin que ella se dé cuenta, adquiere un bloqueo sobre ese objeto, con lo que ninguna transacción podrá leer ni modificar dicho objeto.

El bloqueo funciona del siguiente modo:

1. Primero suponemos la existencia de dos tipos de bloqueo, bloqueos exclusivos (X) y bloqueos compartidos (S), definidos en los siguientes dos puntos.
2. Si la transacción A tiene un bloqueo exclusivo (X) sobre el registro R, una solicitud por parte de una transacción B de cualquier tipo de bloqueo sobre R hará que B entre en un estado de espera. B espera hasta que A libere el bloqueo.
3. Si una transacción A tiene un bloqueo compartido (S) sobre el registro R:
 - a) Una solicitud por parte de una transacción B de bloqueo X sobre R hará que B entre en un estado de espera, y B espera hasta que se libere el bloqueo de A.
 - b) Una solicitud por parte de una transacción B de un bloqueo S sobre R será concedida, y B tendrá también ahora un bloqueo sobre R.

La matriz de compatibilidades que se puede dar es la siguiente:

	X	S	No block
X	N	N	S
S	N	S	S
No block	S	S	S

Explicación: consideremos un registro R, y que una transacción A tiene algún bloqueo sobre R tal y como lo indican las cabeceras de las columnas.

Supongamos que una transacción B emite una solicitud de bloqueo sobre R tal y como indica la primera columna. Una N indica un conflicto (B entra en un estado de espera), y S indica compatibilidad.

- Las solicitudes de bloqueo sobre registros por parte de las transacciones son implícitas en condiciones normales. Cuando una transacción lee con éxito un registro, adquiere de forma automática un bloqueo S sobre él, y cuando lo actualiza, adquiere un bloqueo X.
- Los bloqueos X se mantienen hasta el siguiente punto de sincronización. Lo normal es que los bloqueos S se mantengan de igual forma.

Teniendo en cuenta los bloqueos aplicables sobre los registros es posible resolver de forma eficiente los problemas de concurrencia anteriormente tratados:

- El problema de la modificación perdida:

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
Leer Fila 1 (Activar Bloqueo S)	T1	
	T2	Leer Fila 1 (Activar bloqueo S)
Actualizar Fila 1 (Solicitar bloqueo X) Espera Espera ...	T3	
	T4	Actualizar Fila 1 (Solicitar Bloqueo X) Espera Espera ...

Las operaciones de lectura llevan implícito un bloqueo S, y las de escritura un bloqueo X, con lo que siguiendo la secuencia en el tiempo, las dos transacciones quedan en un tiempo de espera infinito, y no se pierde ninguna modificación (porque no se realiza).

Esto, sin embargo, nos lleva a un problema que trataremos un poco más adelante: el bloqueo mutuo.

2. El problema de la dependencia no comprometida:

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
	T1	Actualizar Fila R (Activar bloqueo X)
Leer Fila R (Solicitar Bloqueo S) Esperar ...	T2	
	T3	Rollback (liberar el bloqueo X)
Continuar: Leer R (Adquirir bloqueo S)	T4	

TRANSACCIÓN 1	TIEMPO	TRANSACCIÓN 2
	T1	Actualizar Fila R (Activar bloqueo X)
Actualizar Fila R (Solicitar Bloqueo X) Esperar ...	T2	
	T3	Rollback (liberar el bloqueo X)
Continuar: Actualizar R (Adquirir bloqueo X)	T4	

Como se puede apreciar en las nuevas tablas que reproducen el caso del problema descrito en un apartado anterior, los bloqueos implícitos hacen que, dado que 2 ha realizado un bloqueo X sobre R, 1 no pueda leer el registro, y deba esperarse hasta que se llegue a un punto de sincronización (en este caso un rollback), y se libere el bloqueo. El problema queda de este modo resuelto.

3. El problema del análisis inconsistente.

TRANSACCIÓN X	TIEMPO	TRANSACCIÓN Y
Seleccionar Col1 → 40 (Activar bloqueo S sobre Col1) Suma = 40	T1	
Seleccionar Col2 → 50 (Activar bloqueo S sobre Col2) Suma = 90	T2	
	T3	Seleccionar Col3 → 30 (Activar bloqueo S sobre Col3)
	T4	Actualizar Col3 ← 20 (Activar bloqueo X sobre Col3)
	T5	Seleccionar Col1 → 40 (Activar bloqueo S sobre Col1)
	T6	Actualizar Col1 ← 50 (Solicitar bloqueo X sobre Col1) Esperar, ...
	T7	COMMIT
Seleccionar Col3 → 20 (Activar bloqueo X sobre Col3) Esperar, ...	T8	

De nuevo, se soluciona el problema del análisis inconsistente, puesto que ninguna transacción finaliza y no genera resultados erróneos. Sin embargo, vuelve a aparecer el problema del bloqueo mutuo que a continuación analizamos.

d) El bloqueo mutuo.

Hemos visto que los bloqueos son capaces de resolver los principales problemas de concurrencia, pero que al mismo tiempo pueden ocasionar problemas adicionales de transacciones que entran en un tiempo de espera infinito. Este problema, que ha sido claramente ilustrado en los ejemplos, se denomina bloqueo mutuo.

Si se presenta un bloqueo mutuo, es deseable que el sistema lo detecte y lo rompa. Para ello, es necesario elegir una de las transacciones que están paralizadas como víctima y cancelarla, con lo que se liberan sus bloqueos y permite continuar a la otra transacción.

El tratamiento que los sistemas dan a la transacción víctima una vez cancelada es variable. En algunos casos se reinician de forma automática las transacciones desde el principio, suponiendo que no se va a volver a repetir el caso de bloqueo mutuo. En otros casos, simplemente se avisa a la aplicación que maneja la transacción víctima que ha sido objeto de una cancelación debido a un bloqueo mutuo, y es responsabilidad del programador el tratar convenientemente estos problemas.

