# Eléments de base de la sécurité des bases de données

N. Boudjlida
UHP Nancy 1, LORIA, Campus scientifique, BP 239
54506 Vandœuvre Lès Nancy CEDEX (F)
Nacer.Boudjlida@loria.fr, http://www.loria.fr/~nacer

*Résumé* : Ce papier introduit des éléments de base pour la sécurité de fonctionnement des bases de données et de leurs systèmes de gestion. Ces élements sont constitués de concepts et de mécanismes fondés sur la réplication de données, la gestion des procédures de reprise en cas d'incident et la surveillance des activités opérées sur la base. Ces concepts et mécanismes sont délibérément présentés sous un angle applicatif, sans développement des théories sous-jacentes.

*Mots-Clés* : Serveur de données, transactions, sérialisabilité, reprise, audit, réplication.

*Abstract* : This paper introduces some basic elements for the security of databases and their management systems. These elements include concepts and mechanisms funded on data replication, recovery procedures and auditing the database activity. These concepts and mechanisms are deliberately introduced under an application perspective, without developing the underlying formal foundations.

*Keywords* : Data servers, transactions, serializability, revovery, auditing, replication.

## 1 Introduction

La sécurité des bases de données concerne tant le serveur de données que les bases existantes. Ainsi, outre les mécanismes de sécurité qui peuvent être mis en place sur un site par un administrateur système et réseau, la sécurité au sein d'un serveur de données peut combiner des mécanismes (i) de *reprise en cas d'incident* (paragraphe 2), (ii) de *duplication* (paragraphe 3) et (iii) de *surveillance des activités* réalisées ou tentées tant sur le serveur de données que sur les bases qu'il gère (paragraphe 4). D'autres types de concepts et de mécanismes, comme le cryptage d'informations ou la gestion des privilèges et des ressources d'un utilisateur d'une base, ne sont pas considérés ici (voir par exemple [4, 3]). Les trois types de mécanismes ci-dessus sont successivement examinés ici d'un point de vue opérationnel et ils devraient être mis en œuvre tant par les programmeurs que par les administrateurs de bases de données et de leurs systèmes de gestion (SGBD, également appelés ici serveurs de données). Dans chacun des cas, nous exposerons ou nous rappellerons quelques notions nécessaires à leur compréhension et nous présenterons leur mise en œuvre sur des SGBD représentatifs du commerce.

## 2 Transactions, sauvegardes et reprises

La reprise en cas d'incident (*recovery*) consiste à remettre une base de données en fonctionnement le plus rapidement possible et dans un état le plus proche possible de celui dans lequel elle était avant l'incident. Les incidents ou événements pouvant altérer l'état d'une base ou empêcher son utilisation vont de l'erreur de programmation à la détérioration des unités physiques de stockage. La reprise consiste alors à reconstruire une base saine à partir de son *histoire*, cette histoire étant enregistrée dans les journaux (*log*) de la base. Ce mécanisme étant intimement lié à ceux de gestion des transactions et des accès concurrents, nous commençons ce paragraphe par un rappel de concepts et de mécanismes de gestion des transactions et de reprise en cas d'incident (paragraphes 2.1 et 2.2) avant de présenter leur mise en œuvre dans Sybase (paragraphe 2.3) et Oracle (paragraphe 2.4).

### 2.1 Gestion des transactions

Une base de données devant pouvoir être simultanément accédée par plusieurs utilisateurs ou programmes (ces deux termes sont utilisés indifféremment ici), on trouvera dans un SGBD, un *sous-système de gestion des accès concurrents*

qui utilise des techniques très voisines de celles développées pour les systèmes d'exploitation (verrouillage, exclusion mutuelle). Nous appellerons *transaction* tout programme qui accède à la base ou qui en modifie le contenu, c'est-à-dire qui réalise une suite d'opérations de lecture et d'écriture dans la base. Un gérant de transactions a pour rôle de contrôler l'exécution de transactions et il doit assurer les propriétés dites *A.C.I.D.*, à savoir : *(i) Atomicité* : politique du "tout ou rien" ; *(ii) Cohérence* : satisfaction des contraintes d'intégrité ; *(iii) Isolation* : pas d'interférence entre les transactions et *(iv) Durabilité* : permanence des modifications correctement effectuées.

La propriété d'atomicité résulte en le schéma suivant d'exécution d'une transaction :

*Début Transaction*
$Action_1$ ; $Action_2$ ; $\cdots$ ; $Action_n$
*Si* toutes les $Action_i$ sont correctement exécutées
*Alors* Valider la transaction (*commit*)
*Sinon* Annuler ses effets (*rollback*)
*Finsi*
*Fin Transaction*

La *validation* consiste à rendre effectives les modifications apportées aux données. En SQL, elle est réalisée par la commande *commit*. Par contre, si on s'aperçoit que, pour une raison ou une autre, les modifications sont incorrectes, la commande *rollback* permet de défaire les actions de la transaction remettant ainsi la base dans l'état où elle était avant le début de la transaction. L'annulation d'une transaction peut être décidée par l'utilisateur ou par le gérant de transactions.

Par ailleurs, la gestion de la concurrence est fondée sur la notion de *sérialisabilité* qui stipule que l'exécution concurrente de n transactions doit être équivalente (avoir le même effet) à leur exécution séquentielle. D'un point de vue théorique, cette notion est fondée sur une relation d'ordre total entre les actions d'une transactions et partiel entre l'ensemble des actions des transactions concurrentes. Techniquement, cette relation d'ordre est concrétisée dans les SGBD par des techniques d'attente (techniques de verrouillage). Elle peut également être établie en associant une estampille (une date) à chaque transaction et à assurer que l'ordre "chronologique" d'exécution des transactions est respecté. Cette dernière technique n'étant pas implémentée, elle ne sera pas considérée ci-après (pour plus de détails, on pourra notamment consulter [1] ou [2, 3]). Voyons maintenant, en nous référant aux propriétés *ACID*, les problèmes que peut occasionner l'absence de contrôle de l'exécution de transactions.

### 2.1.1 (Non) "ACIDité" de transactions

**1.** *(Non) Atomicité* : Supposons qu'un programme veuille opérer un virement d'un montant $m$, d'un compte bancaire $C_1$ vers un compte $C_2$. La séquence d'actions élémentaires est : {*Début* : $x \longleftarrow Lire(C_1)$ ; $x \longleftarrow x - m$ ; $C_1 \longleftarrow Ecrire(x)$ ; $y \longleftarrow Lire(C_2)$ ; $y \longleftarrow y + m$ ; $C_2 \longleftarrow Ecrire(y)$ ; *Fin*}.

Si le programme s'arrête, pour une raison ou pour une autre, avant l'écriture de $y$ dans $C_2$, la base entre dans un état incohérent. De ce fait, une transaction sera considérée comme une *unité atomique d'actions*, c'est-à-dire qu'une transaction est considérée comme correctement achevée si elle a correctement exécuté toutes ses actions sur la base. Si tel n'est pas le cas, on veillera à remettre la base dans l'état où elle se trouvait *avant* le début de la transaction.

**2.** *(Non) Isolation* : Supposons que X soit un solde bancaire sur lequel une transaction T1 veut inscrire un crédit d'un montant N et une transaction T2 veut inscrire un débit d'un montant M. Les séquences d'actions élémentaires de T1 et T2 sont illustrées sur la figure 1 :

| *T1* | *T2* |
|---|---|
| T11 : $x \longleftarrow Lire(X)$ | T21 : $y \longleftarrow Lire(X)$ |
| T12 : $x \longleftarrow x - N$ | T22 : $y \longleftarrow y + M$ |
| T13 : $X \longleftarrow Ecrire(x)$ | T23 : $X \longleftarrow Ecrire(y)$ |

FIG. 1 – *Transactions et isolation*

Supposons que la valeur initiale de X dans la base soit égale à 100, que N vaille 10 et que M vaille 50. Si les actions des transactions sont exécutées dans l'ordre T11; T12; T21; T22; T13; T23, le solde X vaut 150 : l'effet de l'opération de débit (T12 : $x \longleftarrow x - N$) ayant été perdue.

Le problème vient du fait que T2 a accédé à X avant que T1 n'ait fini sa mise à jour. En d'autres termes, il faudrait que T1 travaille en *isolation* jusqu'à sa terminaison, c'est-à-dire sans interférence avec d'autres transactions qui veulent accéder aux mêmes objets qu'elle.

**3.** *(Non) Cohérence* : Considérons les transactions T1 et T2 de la figure 2 et supposons que la contrainte d'intégrité (A = B) est imposée sur la base.

| *Actions de T1* | *Actions de T2* |
|---|---|
| t11 : $A_{T1} \longleftarrow Lire(A)$ | t21 : $A_{T2} \longleftarrow Lire(A)$ |
| t12 : $A_{T1} \longleftarrow A_{T1} + 1$ | t22 : $A_{T2} \longleftarrow A_{T2} \times 2$ |
| t13 : $A \longleftarrow Ecrire(A_{T1})$ | t23 : $A \longleftarrow Ecrire(A_{T2})$ |
| t14 : $B_{T1} \longleftarrow Lire(B)$ | t24 : $B_{T2} \longleftarrow Lire(B)$ |
| t15 : $B_{T1} \longleftarrow B_{T1} + 1$ | t25 : $B_{T2} \longleftarrow B_{T2} \times 2$ |
| t16 : $B \longleftarrow Ecrire(B_{T1})$ | t26 : $B \longleftarrow Ecrire(B_{T2})$ |

FIG. 2 – *Transactions et cohérence*

Chaque transaction satisfait la contrainte, c'est-à-dire que celle-ci est vraie à l'issue de chaque transaction. Cependant leur exécution concurrente peut la violer. Par exemple, la séquence <t11; t12; t13; t21; t22; t23; t14; t15; t16; t24; t25; t26> est une exécution correcte, alors que <t21; t22; t11; t12; t23; t13; t14; t15; t16; t24; t25; t26> est une séquence incorrecte.

**4.** *(Non) Permanence (ou Durabilité) des modifications* : Le fait que des modifications de données par une transaction $T1$ ne sont pas répercutées dans la base avant leur utilisation par une transaction $T2$ peut occasionner les problèmes de *"non répétabilité des lectures"* et de *"tuples fantômes"*. Ceux-ci sont respectivement illustrés par les figures 3 et 4 sur lesquelles les actions sont temporellement ordonnées de haut en bas. Dans le premier cas, les impressions de T2 produiront des valeurs différentes alors que dans le second cas, la deuxième impression de $V$ rendra un ensemble vide.

| Transaction T1 | Transaction T2 |
|---|---|
| $a \longleftarrow Lire(A)$ | |
| | $b \longleftarrow Lire(A)$<br>*Imprimer(b)* |
| $a \longleftarrow a + 100$ | |
| $A \longleftarrow Ecrire(a)$ | |
| | $b \longleftarrow Lire(A)$<br>*Imprimer(b)* |

FIG. 3 – *Transactions et (non) répétabilité des lectures*

| Transaction T1 | Transaction T2 |
|---|---|
| Supprimer A = $\{a \mid a < 4000\}$ | $V \longleftarrow Lire(A), A = \{a \mid a < 4000\}$<br>*Imprimer(V)* |
| | $V \longleftarrow Lire(A), A = \{a \mid a < 4000\}$<br>*Imprimer(V)* |

FIG. 4 – *Transactions et tuples fantômes*

Pour remédier à ces divers problèmes, SQL-92 introduit quatre *niveaux d'isolation* des transactions, le niveau 3 étant celui par défaut. *Le niveau 0* autorise la lecture de valeurs incorrectes (*dirty reads*) : si une donnée $D$ de la base est en cours de modification par une transaction $T$, toute transaction autre que $T$ est autorisée à lire $D$ mais pas à modifier $D$. *Le niveau d'isolation 1* interdit la lecture de valeurs incorrectes. *Le niveau 2* empêche la relecture de données avant que ces données ne soient écrites dans la base (lectures "sales" ou *dirty reads*). Enfin, *le niveau 3* empêche le phénomène des tuples fantômes.

### 2.1.2 Techniques de contrôle de la concurrence

Comme nous l'avons déjà mentionné, il ne sera fait mention ici que des techniques usuellement implantées, c'est-à-dire celles de verrouillage. On peut considérer un verrou comme une variable associée à un granule et dont la valeur spécifie les opérations autorisées sur le granule, le granule étant l'unité de verrouillage et pouvant concrètement être la base de données, une relation ou une partie de relation, un tuple, un attribut, etc.

Ainsi, un *verrou binaire* est un verrou à deux états : quand il est apposé sur un granule, celui-ci est accessible ou inaccessible. Ce type de verrou ne permet, à l'évidence, qu'à une seule transaction de détenir un granule. Or, on peut, sans risques, autoriser des accès multiples en consultation (en lecture) sur un même granule. C'est ce que permet le verrou de type partagé : un verrouillage en mode partagé (*Share*) permet à plusieurs transactions de consulter simultanément la valeur d'un granule. Par contre, en mise à jour, il est nécessaire de verrouiller le granule en mode exclusif (*Exclusive*). Dans ce mode, seule une transaction $T$ peut, à un instant donné, détenir le granule, les autres sont mises en attente jusqu'à sa libération par $T$.

Un autre type de verrou est également offert par les SGBD ; il s'agit d'un *verrou d'intention :* une transaction peut demander un verrou de mise à jour (*Update*). Celui-ci correspond à un verrou de lecture avec *intention* d'écriture. Le tableau 1, où les lignes représentent les verrous apposés sur un granule et les colonnes ceux demandés sur le même granule, montre la compatibilité de ces différents types de verrous.

| | *Partagé* | *Exclusif* | *Mise à jour* |
|---|---|---|---|
| *Partagé* | Oui | Non | Non |
| *Exclusif* | Non | Non | Non |
| *Mise à jour* | Oui | Non | Non |

TAB. 1 – *Compatibilité des verrous partagés, exclusifs et mise à jour*

Les techniques de verrouillage garantissent la sérialisabilité des transactions lorsqu'elles se conforment au protocole de *verrouillage à deux phases*. Ce protocole consiste à opérer toutes les opérations de verrouillage avant la première opération de déverrouillage. La première phase est une *phase d'expansion* où tous les verrous sont apposés et la seconde est une *phase de rétrécissement* où les verrous sont libérés et où aucun nouveau verrou ne peut plus être apposé. Il est, par ailleurs, bien connu que ces techniques peuvent conduire à des situations *d'interblocage* (*deadlock*) et de *famine* (*livelock*).

*L'interblocage* est causé par des attentes mutuelles. Il se produit lorsqu'une transaction T1 détient un granule G1 et attend qu'une transaction T2 libère un granule G2, et qu'en même temps, T2 détient G2 et attend la libération de G1. La solution à ce type de situation est soit de l'empêcher de se produire soit de le détecter et d'y remédier. La *prévention* peut être réalisée en imposant à toute transaction de verrouiller en avance tous les éléments dont elle a besoin et en n'apposant aucun verrou si un de ces éléments n'est pas libre.

La *détection* consiste à tester périodiquement si une situation d'interblocage s'est produite. Cette détection se base sur un graphe d'attente dont les nœuds représentent les transactions et les arcs la relation *est_en_attente_de* : un interblocage existe si et seulement si le graphe comporte un cycle. Si une telle situation se produit, une des transactions impliquée dans l'interblocage est avortée, ses actions annulées et ses ressources libérées, débloquant ainsi une des transactions impliquées dans l'interblocage.

La situation de *famine* se produit lorsqu'une transaction est perpétuellement en attente alors que d'autres continuent à s'exécuter. C'est le cas, par exemple, lorsqu'un granule est verrouillé en mode partagé par une première transaction T1 : une transaction T2 désirant modifier ce granule est mise en attente tandis que de nouvelles transactions de lecture pourront elles acquérir un verrou partagé sur le granule et donc s'exécuter, augmentant ainsi le temps d'attente de T2.

## 2.2 Journalisation de transactions et reprises

Le précédent paragraphe a montré la nécessité de contrôler les accès concurrents à une base de données ainsi que celle de rétablir sa cohérence (on parlera de *reprise* en cas d'incident) dans le cas de sa violation suite à une fin anormale (ou échec) d'une transaction [5]. En fait, la reprise en cas d'incident va au-delà du simple échec des transactions. Elle est également requise dans le cas d'incidents "logiques" (panne du serveur de données ou du système d'exploitation hôte) ou "physiques" (défaillance du matériel, détérioration d'un support de stockage, etc.). Dans nombre de cas d'échec ou d'incident, le système pourra remettre la base en état s'il dispose de suffisamment d'informations sur les activités qui ont précédé le moment de l'incident : ces informations sont disponibles dans le *journal* (ou *log*) des transactions sous la forme d'une trace des opérations effectuées sur la base. Ce journal sert tant pour l'annulation des transactions en cas d'échec que pour la remise en état d'une base en cas d'incident d'une autre nature.

### 2.2.1 Journalisation et point de contrôle

Le journal d'une base est un (ou plusieurs) espace physique non volatile (disque, bande) associé à une base de données. Il doit être archivé périodiquement en synchronisation avec des sauvegardes de l'espace des données. Pour chaque transaction T, il comporte des informations du type :

– <identification de T, début> ;

– < identification de T, écriture, donnée concernée, ancienne valeur, nouvelle valeur> ;
– <identification de T, lecture, donnée concernée> ;
– <identification de T, *commit*>.

L'information <identification de T, *commit*> est inscrite dans le log au *point de validation* de la transaction T, c'est-à-dire à l'instant où toutes les actions de la transaction ont été réalisées avec succès et où les informations du journal ont été écrites. Au-delà de ce point, la transaction est considérée comme valide et les modifications qu'elle a effectuées deviennent visibles à d'autres transactions. Ainsi, si un incident se produit, le système est à même d'annuler les effets des transactions non validées (*rollback*) et de refaire (*rollforward*) éventuellement, les actions de celles validées. Dans le premier cas, il suffit de "rejouer l'histoire de la transaction en arrière", c'est-à-dire restituer les anciennes valeurs des données modifiées en partant de la fin du journal. Dans le second cas, il suffit de rejouer l'histoire "en avant".

Les *points de contrôle* sont d'autres types d'information du journal. Ils y sont inscrits périodiquement, la période pouvant être un intervalle de temps, un nombre déterminé d'opérations de mise à jour ou une combinaison des deux. À l'issue de chaque période, le système suspend les transactions en cours, inscrit leurs opérations de modification dans la base et inscrit un point de contrôle dans le journal. Concrètement, les points de contrôle sont les instants où un système réalise des écritures physiques de pages de sa mémoire cache vers l'espace disque. En effet, alors qu'une commande de validation (*commit*) ne déclenche que des écritures physiques des pages modifiées du cache du journal, la prise d'un point de contrôle (*checkpoint*) a pour effet, dans l'ordre : *(i)* suspendre les transactions actives, *(ii)* effectuer des écritures physiques des pages modifiées du cache du journal, et *(iii)* de celles modifiées du cache de données avant de *(iv)* relancer les transactions suspendues (voir la figure 5).

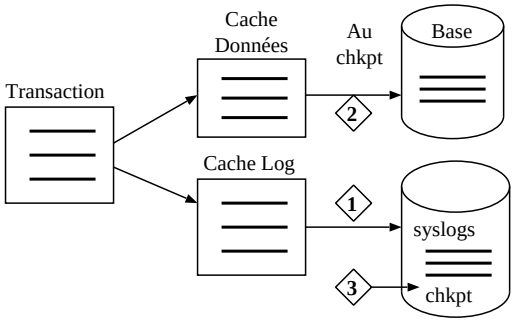


FIG. 5 – *Point de contrôle et journalisation*

### 2.2.2 Reprise en cas d'incident

La stratégie de reprise dépend de la gravité de l'incident qui la provoque. Si les dégâts sont importants (exemple : support physique détérioré), le processus de *"reprise à froid"* consiste à considérer une sauvegarde de la base, à la recharger puis à automatiquement exécuter les transactions marquées valides dans le journal, depuis la date de la sauvegarde jusqu'à la date de l'incident.

Si les dégâts ne sont pas catastrophiques, la *"reprise à chaud"* consiste à défaire (UNDO) certaines actions et à en refaire (REDO) d'autres, si nécessaire. Deux techniques sont principalement utilisées : la mise à jour différée et la mise à jour immédiate. Dans le premier cas, la mise à jour effective de la base n'a lieu qu'après que la transaction ait atteint son point de validation. Dit de manière très simplifiée, les modifications sont faites *"au brouillon"*, c'est-à-dire sur des copies des données de la base. Si la transaction échoue, "l'original" est inchangé et si elle réussit, on substitue la copie modifiée à l'original (*Shadow paging*).

Dans le cas de la *mise à jour immédiate*, les modifications sont d'abord journalisées puis répercutées dans la base avant d'atteindre le point de validation (*"Write Ahead Log Protocol"*). Ce protocole, implanté dans la plupart des SGBD du commerce, permet la reprise à chaud et la reprise à froid, même si un incident se produit entre l'écriture dans le journal et l'écriture dans la base.

**Exemple :** La figure 6 schématise un processus de sauvegarde des données et des journaux pendant une période d'activité. La sauvegarde du journal le vide de son contenu, ce qui n'est pas le cas de celui de la base, évidemment.

Au point de l'incident, une reprise à chaud consisterait à considérer les états B3 et J3 de la base et du journal, respectivement, et à défaire les actions des transactions qui n'ont pas atteint leur point de validation. Quant à la reprise à froid, elle consisterait à recharger une base cohérente en utilisant une de ses sauvegardes (par exemple B1), les journaux adéquats (J2 et J3, dans cet exemple) et à réexécuter automatiquement les transactions validées.

Voyons maintenant comment ces concepts et ces mécanismes se présentent dans les systèmes Sybase et Oracle.

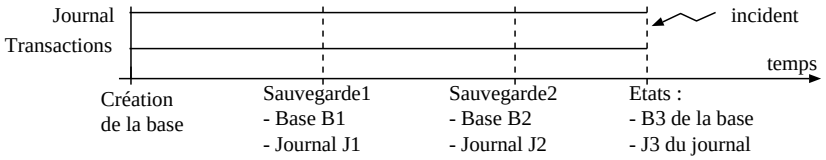


FIG. 6 – *Sauvegarde des données et des journaux*

## 2.3 Application au SGBD Sybase

### 2.3.1 Journalisation et points de contrôle

*Le journal des transactions*, un par base de données, est défini comme une ou plusieurs unités de stockage ou *device*. Il est géré comme une table (*syslogs*) selon le *"Write Ahead Log Protocol"* mentionné plus haut. Sa taille est fonction de l'activité et il est conseillé de la fixer à 10 à 25% de la taille de l'espace des données. D'autre part, afin d'en empêcher la saturation, on peut "borner" l'espace par une marque, appelée *threshold*, et associer une action à la marque (*threshold action*), l'action se déclenchant lorsque le remplissage du journal atteint ou dépasse la marque. Par exemple, une action simple de type alarme, permettrait à un administrateur ou à un responsable sécurité d'opérer une sauvegarde du journal.

En outre, bien que Sybase permette de ranger les données et les informations de journalisation dans le même espace (la même unité de stockage), il est, pour des raisons évidentes, plus prudent de les dissocier et même de les placer sur des unités disque différentes.

La fréquence des *points de contrôle* est fonction de la valeur d'un paramètre de configuration du serveur de données (*"recovery interval in minutes"*) et de l'activité constatée par le serveur.

### 2.3.2 Interblocage et Famine

Sybase utilise la technique classique de détection d'interblocage : gestion d'un graphe d'attente des transactions scruté périodiquement pour y rechercher les cycles éventuels. En cas de présence d'un cycle, la transaction ayant utilisé le moins de temps unité centrale est avortée, ses effets sur la base sont annulés, ses ressources sont libérées et son propriétaire est informé de la mort de sa transaction. La fréquence de scrutation du graphe d'attente est dépend d'un paramètre de configuration du serveur dont la valeur est exprimée en millisecondes.

Par ailleurs, pour éviter qu'une transaction de mise à jour ne soit en situation de *famine* du fait de transactions de lecture, Sybase gère des *demandes de verrous* qu'il octroie aux transactions de mise à jour. Après quatre tentatives infructueuses de satisfaction d'une demande d'apposition d'un verrou exclusif, toute nouvelle demande de verrou partagé est refusée (mise en attente). En quelque sorte, la demande de verrou exclusif devient prioritaire et la transaction de mise à jour pourra ainsi obtenir un verrou exclusif lorsque les transactions de lecture en cours, qui elles détenaient des verrous partagés, se seront achevées.

### 2.3.3 Sauvegardes et Reprises

Les sauvegardes (*dump database* ou *dump tran[saction]*) sont réalisées par un serveur dédié (*back_up server*) local ou distant. Dans l'exemple ci-dessous, les données de la base *MaBase* sont sauvegardées par le serveur de sauvegarde local sur les bandes *tape1* et *tape2* non préalablement déclarées comme unités de sauvegarde ; le journal l'est par un serveur de sauvegarde distant appelé *Backup_Distant*, sur une unité disque déclarée comme unité de sauvegarde de nom logique *DumpLog2MaBase* et consistant physiquement en le fichier *"/usr/. . . /Log2MaBase.dmp"*.

*dump transaction* MaBase *to* DumpLog2MaBase *at* Backup_Distant
*dump database* MaBase *to* "/dev/tape1" *stripe on* "/dev/tape2"

Sybase permet deux types de reprise : la reprise dite automatique et la reprise à froid. *Le premier type de reprise* a lieu à chaque relance du serveur et il a pour effets :

1. Lecture des *Logs* de chaque base existante ;
2. Annulation (*Rollback*) des transactions non validées ;
3. Réexécution (*Rollforward*) des transactions validées ;
4. Inscription d'un *ckeckpoint* dans le *Log* ;
5. Vidage de la base d'espaces temporaires ;
6. Recopie éventuelle d'objets de la base *model* (prototype des tables du dictionnaire d'une base) ;

7. Inscription du compte rendu de la relance dans un fichier de trace (*errorlog*) du serveur.

*La reprise "à froid"* quant à elle se conforme au scénario général décrit plus haut. On suppose disposer des sauvegardes des données et des journaux correspondants. Il suffit alors de recharger la base (*load database*) et le ou les journaux correspondants (*load tran*). Dans le cas où plusieurs unités de sauvegarde de données et de journaux sont nécessaires à la reprise, il faudra, évidemment, les spécifier dans le bon ordre chronologique. Les transactions journalisées validées sont exécutées au moment du chargement du ou des journaux.

**Exemple :**

*load database* MaBase *from* "/dev/tape1" *stripe on* "/dev/tape2"
*load transaction* MaBase *from* DumpLog2MaBase *at* Backup_Distant
*with until_time* "june 13 2006 12:12:12:222am"

## 2.4 Transactions et reprises sous Oracle

La journalisation des actions des transactions utilise deux supports : un ou plusieurs fichiers *redo logs* contenant les valeurs de données avant et après leur modification et des segments d'annulation (*rollback segments*) ou des espaces d'annulation (*Undo tablespaces*, depuis la version 9) contenant, pour les transactions non encore validées, les valeurs de données avant leur modification.

### 2.4.1 Journaux ou *redo logs*

Chaque base doit comporter au minimum deux fichiers *redo logs*. Ceux-ci sont gérés de façon circulaire, c'est-à-dire que lorsqu'ils sont saturés, la journalisation "recommence" sur le premier fichier, écrasant ainsi les enregistrements précédents. De ce fait, si on désire conserver les informations de ces journaux (ce qui est généralement souhaitable pour des bases de données applicatives), on doit définir une politique de sauvegarde. Celle-ci peut s'appuyer sur le mécanisme d'archivage automatique offert par Oracle, c'est-à-dire faire archiver par Oracle les *redo logs* au fur et à mesure de leur saturation, et réaliser des sauvegardes périodiques de ces archives, complétées éventuellement par des sauvegardes des *redo logs actifs* (c'est-à-dire ceux encore en cours d'utilisation). Il est important de noter que l'archivage automatique n'est pas synonyme de sauvegarde et que d'un point de vue politique de gestion, *il est important de coupler des sauvegardes périodiques avec les mécanismes d'archivage*, ces derniers présentant, *grosso modo*, l'intérêt de décharger un administrateur de la gestion de la saturation des journaux. *Le principe de l'archivage* est illustré par la figure 7 où on suppose disposer de trois *groupes* de fichiers *redo log* : *LGWR*, le processus d'écriture dans le journal, inscrit les enregistrements de journalisation dans le premier fichier. Lorsque celui-ci est saturé, *LGWR* poursuit ses écritures dans le suivant pendant que *ARCn*, le processus d'archivage, copie le premier dans un fichier archive, et ainsi de suite…

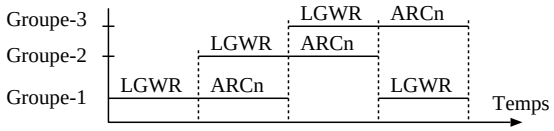


FIG. 7 – *Oracle : archivage automatique des* redo logs

### 2.4.2 Segments d'annulation (*rollback segments*) et transactions

Conjointement aux fichiers *redo logs*, Oracle maintient, dans des segments dits d'annulation, les valeurs de données avant leur modification. Ces données sont relatives aux transactions non encore validées. Ces segments servent alors pour effectuer des lectures "non sales" (*consistent read*), pour annuler les effets d'une transaction ou pour effectuer une reprise sur la base. Un segment d'annulation est un espace physique contenant un ensemble d'enregistrements appelés *"entrées d'annulation"* (*rollback entries*). Une entrée comporte, entre autres informations, l'identification du fichier de données et du bloc contenant la donnée modifiée ainsi que la valeur avant modification. Les entrées d'annulation d'une même transactions sont chaînées entre elles : l'annulation d'une transaction consiste alors à parcourir cette chaîne dans l'ordre inverse de sa construction en restorant les valeurs présentes dans le segment d'annulation.

### 2.4.3 Validation et annulation de transactions

Sous Oracle, le début d'une transaction est déterminé par la rencontre de la première instruction SQL exécutable et sa fin est déterminée par un des cas suivants :

1. Rencontre de l'instruction *commit* ou *rollback* "total", c'est-à-dire une annulation de tous les effets de la transaction et pas une annulation partielle (i.e. pas *rollback to savepoint*).
2. Rencontre d'une instruction de définition de données, comme *create, drop* ou *alter table* : si la transaction comportait des instructions de modification de données (*insert, update, ...* ), elle est validée puis l'instruction de définition de données est exécutée comme une transaction à part entière, c'est-à-dire comme s'il s'agissait d'une transaction à une seule instruction.
3. Déconnexion par l'utilisateur.
4. Arrêt en échec du processus utilisateur : la transaction est avortée dans ce cas.

*Lors de la validation d'une transaction*, Oracle effectue les actions suivantes :

1. Marquage de la transaction comme validée dans la table associant les transactions et leurs segments d'annulation (*rollback segments*) évoquée dans le paragraphe précédent.
2. Inscription d'un numéro de modification système (*SCN, System Change Number*) de la transaction dans cette table.
3. Transfert physique des enregistrements de journalisation des buffers de la zone globale système (*SGA*) vers le fichier *redo log* en ligne.
4. Écriture également du *SCN* dans le fichier *redo log* en ligne : la fin de cette opération d'écriture constitue le *point de validation* d'une transaction.
5. Libération des ressources verrouillées.
6. Marquage de la transaction comme achevée.

Quant à *l'échec d'une transaction*, il consiste simplement *(i)* à restaurer les anciennes valeurs des données à partir des segments d'annulation et *(ii)* à libérer les ressources verrouillées.

### 2.4.4 Interblocage et points de contrôle

Oracle applique un verrouillage de niveau tuple (par opposition à un verrouillage de la page contenant le tuple accédé) ou de niveau table. Il offre une variété de types de verrous, dont les verrous partagés (*Share*), exclusifs (*eXclusive*), partagés ou exclusifs de niveau tuple (*Row Share* ou *Row EXclusive* respectivement), etc. Le système choisit *implicitement* le plus petit niveau de verrouillage lors de la rencontre d'une instruction de modification ou de définition de données. Il maintient les verrous pendant toute la durée de la transaction. Des verrous peuvent être explicitement demandés par une transaction par l'émission d'instructions de type *set transaction isolation level*, *lock table* ou *select for update*. On peut également demander des verrous pour la durée d'une session : *alter session set isolation_level*.

Le *déverrouillage* est, quant à lui, opéré à la fin de la transaction quel que soit son issue (annulation ou validation). Dans le cas d'une annulation partielle, c'est-à-dire annulation jusqu'à un point de sauvegarde, seules sont déverrouillées les ressources acquises entre le point de sauvegarde et le point d'annulation.

Bien que, du fait de la fine granularité du verrouillage, les situations d'interblocage (*deadlock*) devraient être rares, Oracle effectue néanmoins leur détection. Si un interblocage survient, le système n'annule que les effets d'une des instructions impliquées dans l'interblocage et il envoie un message à la transaction. On a alors le choix entre annuler toute la transaction ou ré-exécuter ultérieurement l'instruction annulée.

Lors de la validation d'une transaction, il y a écriture des enregistrements du journal et association d'un numéro de modification (*System Change Number*) à la transaction. Il est à noter que les écritures physiques sont réalisées par le processus d'écriture dans la base (*DBWn*) et non par le processus de point de contrôle (*CHKPT*). Ce dernier se charge de mettre à jour des informations de l'en-tête des fichiers de données (*datafiles*) en y inscrivant des informations relatives à la prise des points de contrôle (*checkpoint*). Ces derniers peuvent être explicites ou implicites, c'est-à-dire déclenchés périodiquement par le système. Dans ce cas, la fréquence est essentiellement contrôlée par deux paramètres d'initialisation de la base :

1. *log_checkpoint_interval* : indique la périodicité des *checkpoints* en termes de nombre de blocs physiques du journal, les blocs considérés étant des blocs du système hôte et non des blocs de données Oracle.
2. *log_checkpoint_timeout* : spécifié en nombre de secondes, ce paramètre limite le temps écoulé entre l'enregistrement du *log* le plus récent et le point de contrôle. Ce paramètre peut aussi s'interpréter comme le temps maximum qu'un tampon modifié du cache peut y séjourner avant d'être inscrit sur disque.

### 2.4.5 Les sauvegardes

Les sauvegardes périodiques restent de mise pour assurer d'éventuelles reprises en cas d'incident. Oracle offre un utilitaire de gestion des sauvegardes et des reprises (*Recovery Manager*) fonctionnant en mode ligne de commandes (*rman*) ou à l'aide d'une interface graphique (*Enterprise Manager Backup Manager*). Néanmoins, il est aussi possible de réaliser des sauvegardes "manuellement" en créant des copies des fichiers physiques en utilisant les commandes du système hôte (par exemple, *cp* sous Unix). Ainsi on pourra opérer des sauvegardes partielles ou totales.

*Une sauvegarde totale* d'une base consiste à créer des copies de tous les espaces de données (*datafiles*) et du fichier de contrôle de la base. On notera que si l'archivage automatique n'est pas en fonction sur une base (mode *no archivelog*), la sauvegarde totale est le seul moyen de se prémunir contre une détérioration du support physique de la base.

*Des sauvegardes partielles* peuvent aussi être opérées. Elles peuvent concerner un espace de rangement (*tablespace*), le fichier de contrôle, un ou des fichiers de données particuliers (*datafile*). La sauvegarde du fichier de contrôle d'une base est fortement conseillée après chaque modification de la structure de la base (ajout, suppression d'espaces, modification de leurs caractéristiques, etc.). En outre, le gérant de reprise réalise automatiquement une sauvegarde du fichier de contrôle lors de la sauvegarde du fichier de données (*datafile*) de numéro 1, celui-ci contenant le dictionnaire de la base.

*Une sauvegarde totale et cohérente de la base* peut ainsi être obtenue :

1. En arrêtant l'instance Oracle (*shutdown normal, immediate* ou *transactional*) ;
2. En effectuant, au niveau du système d'exploitation, les copies des fichiers de données, de contrôle et d'initialisation de la base ;
3. En relançant l'instance (*startup . . . open*).

*Des sauvegardes partielles* de tout ou partie des fichiers d'un espace de stockage (*tablespace*) en ligne peuvent aussi être réalisées selon le processus suivant :

1. Indiquer le début de la sauvegarde (*alter tablespace* NomTs *begin backup*, où NomTs est le nom d'une *tablespace*) ;
2. Effectuer la copie au niveau du système d'exploitation ;
3. Indiquer la fin de la sauvegarde (*alter tablespace* NomTs *end backup*).

### 2.4.6 La restauration à partir de sauvegardes sous Oracle

Cette opération consiste à remettre des espaces de rangement dans l'état où ils étaient lors d'une précédente sauvegarde. Comme pour les opérations de sauvegarde, cette opération peut être réalisée à l'aide du gérant de reprise (*rman*) ou à l'aide de commandes de copies de fichiers du système d'exploitation hôte.

Cette opération peut être requise pour effectuer une reprise "à froid", pour changer un support de stockage détérioré, voire pour faire migrer des fichiers d'un support vers un autre. La restauration peut concerner n'importe quel élément de stockage physique : *tablespace*, *datafile*, *redo logs*, *fichier de contrôle*, etc. Concernant un espace de stockage (*tablespace*), Oracle permet de restaurer l'intégralité de ses fichiers comme il permet de n'en restaurer que quelques-uns.

**Exemple :** La ligne notée *(1)* copie dans un nouvel espace les journaux archivés sauvegardés et les lignes *(2)* et *(3)* montrent deux façons de réaliser une reprise en utilisant cette copie de journaux.

(1) *cp* /disk1/LogBackup/*.arc /disk2/tmp/Archives/.
(2) *set logsource* /disk2/tmp/Archives; *recover;*
(3) *alter database recover from* /disk2/tmp/Archives *database;*

### 2.4.7 La reprise en cas d'incident

Sous Oracle, le processus de reprise opère en deux phases comme illustré par la figure 8. *La première phase*, appelée ré-exécution en avant (*rollforward*) ou *cache recovery*, consiste en une ré-exécution des modifications journalisées dans les *redo logs* et en la journalisation de ces modifications dans les segments d'annulation.

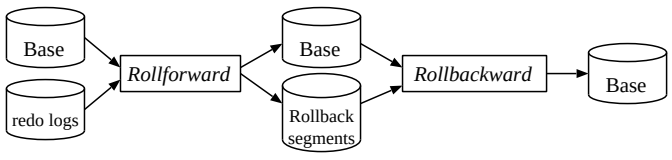


FIG. 8 – *Oracle : le processus de reprise (*recovery*)*

À l'issue de la phase de *rollforward*, la base contient des valeurs de données modifiées par des transactions validées ou pas. *La seconde phase*, appelée ré-exécution en arrière (*rollbackward*) ou *transaction recovery*, consiste à utiliser les segments d'annulation pour défaire les changements opérés par les transactions non validées.

Les journaux (*redo logs*) utilisés dans ce processus dépendent de la nature de l'incident. Dans certains cas, ceux en ligne peuvent suffire alors que dans d'autres cas, il pourra être nécessaire de restaurer les données de la base ainsi qu'un ensemble de journaux à partir de leurs sauvegardes respectives.

Ainsi, *une reprise automatique*, sans restauration et en utilisant généralement les seuls journaux en ligne, est effectuée lors de l'échec d'un processus utilisateur (suite à une déconnexion impromptue, par exemple) ainsi que lors de l'échec d'une instance Oracle (par exemple, suite à un *"shutdown abort"*, à une panne du système hôte ou à l'échec d'un processus Oracle d'arrière-plan).

En cas d'échec de la reprise automatique ou pour pallier la détérioration d'un support physique de stockage, il est nécessaire d'opérer une reprise à partir de sauvegardes : ce type de reprise est appelée reprise sur support ou *media recovery* dans la terminologie Oracle. Elle est réalisée sous SQL*Plus ou à l'aide du gérant de reprise (*rman*).

**Exemple : Restauration et reprise utilisant *rman* :** Restauration et reprise de la *tablespace TabSp1* dans une base ouverte : on notera que *rman* permet d'exécuter des commandes SQL en les introduisant par le mot-clé *sql*.

run { *allocate channel* C1 *type disk*;
*sql "alter tablespace* TabSp1 *offline immediate";*
*restore tablespace* TabSp1;
*recover tablespace* TabSp1;
*sql "alter tablespace* TabSp1 *online";* }

#### 2.4.8 Conclusion

La relative complexité du processus de reprise sous Oracle ainsi que la vaste étendue des possibilités offertes (reprises partielles, reprises sur des base en cours d'exploitation, etc.) ne rendent qu'encore plus cruciale l'adoption d'une organisation et d'une politique rigoureuse de gestion des sauvegardes avec notamment un "étiquetage" minutieux associant sauvegardes de données et journaux correspondants.

# 3 Sécurité par réplication

Ce dispositif permet de synchroniser le contenu d'un espace quelconque de stockage (données, journal, méta-données) avec un réplicat de cet espace. Il a pour effet d'opérer les modifications "en double", c'est-à-dire sur l'espace ou l'*unité primaire* et sur sa copie (que nous appellerons également *espace ou unité secondaire*). De ce fait, en cas de panne ou de détérioration de l'unité primaire, on pourra (presque) assurer une continuité de service en "basculant" sur l'unité secondaire. Il est naturellement plus sécurisant de localiser les unités primaires et secondaires sur des disques physiques différents. En outre, il est recommandé de répliquer au moins le *log* des bases de données "sensibles", même si rien n'empêche de répliquer à la fois les méta-données, les données et les *logs*.

## 3.1 Application au SGBD Sybase

Pour que le mécanisme de réplication, appelé *mirroring*, puisse être utilisé, il doit d'abord être autorisé au niveau du serveur de données. Cette autorisation est indiquée lors de l'installation du serveur. Les opérations de mise en œuvre (voir l'exemple ci-dessous) sont alors :

1. *Activation du miroir d'une unité (disk mirror)* : le contenu de l'unité primaire est copié dans l'unité secondaire et, à partir de cet instant, les écritures sont effectuées sur les deux unités. La commande d'activation spécifie le nom et la localisation physique de l'unité secondaire.
2. *Désactivation temporaire (disk unmirror . . . mode = retain)* : suspension des écritures sur une des deux unités.
3. *Réactivation (disk remirror)* : reprise des écritures sur les deux unités.
4. *Désactivation définitive (disk unmirror . . . mode = remove)* : arrêt définitif des écritures sur les deux unités.

**Exemple** :

//– Activation d'un miroir de *MonJournal* ; /MIR/LaCopie2MonJournal.mir contient le miroir.
*disk mirror name* = "MonJournal", *mirror* = "/MIR/LaCopie2MonJournal.mir"
//– Désactivation temporaire (*mode = retain*) de la copie primaire

*disk unmirror name* = “MonJournal”, *side = “primary”, mode = retain*
//– Réactivation
*disk remirror name* = “MonJournal”
//– Désactivation définitive (*mode = remove*) de la copie secondaire.
*disk unmirror name* = “MonJournal”, *side = “secondary”, mode = remove*

### 3.2 Application à Oracle

Oracle offre la notion de *multiplexage* pour la sécurité par duplication. Cependant, cette notion ne garantissant pas une totale continuité de fonctionnement (une erreur d’entrée-sortie sur une des copies peut nécessiter une relance de l’instance Oracle concernée), il est suggéré d’utiliser conjointement le multiplexage du système Oracle et les services de mise en miroir (*“mirroring”*) d’unités ou de fichiers ([8] est un exemple de ces services) quand de tels services sont offerts par le système d’exploitation hôte.

Le multiplexage doit concerner principalement le fichier de contrôle d’une base (*control file*) et le journal (*redo log*), chaque base devant comporter, au minimum, deux fichiers de contrôle et deux fichiers *redo log*. Pour ce qui concerne le multiplexage des fichiers *redo log actifs* (*actifs* par opposition aux *redo log archivés*), Oracle définit la notion de *groupe* et de *membre* : un membre est un fichier d’un groupe et les membres d’un même groupe sont des copies les uns des autres. Une base doit avoir au minimum deux *groupes* d’au moins un *membre* chacun.

Un processus d’arrière-plan (*LGWR*) assure l’écriture “en double” dans les membres d’un même groupe (par exemple, sur la figure 9, dans *U1/log11.log* et *U2/log12.log* ou dans *U1/log21.log* et *U2/log22.log*).

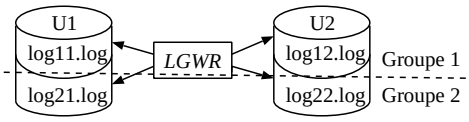


FIG. 9 – *Oracle : multiplexage*

### 3.3 Conclusion

Des outils plus puissants, tels que les serveurs de réplicats (*Replication Server*), peuvent aussi être mis en œuvre pour gérer des copies de bases de données tant centralisées que distribuées [7].

## 4 La surveillance (*audit*)

La surveillance, ou *audit*, permet d’enregistrer des événements liés à des utilisateurs, à des bases de données, ou encore au serveur lui-même. La liste des événements devant faire l’objet d’une surveillance est appelée *options de l’audit* et l’ensemble des enregistrements décrivant la surveillance effective est généralement appelé *audit trail*. La mise en œuvre de ce mécanisme nécessite *(i)* son installation ou sa configuration préalable, *(ii)* la spécification des options de l’audit et *(iii)* la gestion de l’*audit trail*. Ces activités sont exposées successivement sous Sybase puis sous Oracle.

### 4.1 La surveillance sous Sybase

Le système d’audit nécessite une base de données spécifique (*sybsecurity*), une file de messages et quelques paramètres de configuration. La base *sybsecurity* est créée lors de l’installation de l’audit. Elle comporte, entre autres, la table *sysauditoptions*, qui contient les options de l’audit, ainsi que la ou les tables destinées à recevoir les enregistrements de l’audit effectif. Ces dernières (8 au maximum) constituent, en quelque sorte, l’*audit trail* et sont nommées *sysaudits_0i*, *i* prenant ses valeurs dans [1..8]. Cependant, à tout moment il n’y a qu’une seule table *sysaudits_0i active*, c’est-à-dire une seule table dans laquelle sont inscrits les enregistrements d’audit en provenance de la *file de messages*. Seul le responsable sécurité est habilité à installer et à configurer les mécanismes de l’audit puis à consulter, voire modifier, les informations collectées. La configuration du système consiste, entre autres, à fixer la taille de la file des messages, à définir des mécanismes d’archivage des tables *sysaudits*, à choisir une stratégie “d’activation” de ces tables, à indiquer les types d’événements et objets associés à surveiller.

Dans ce qui suit, nous allons successivement présenter *(i)* l’installation et la configuration du système d’audit, *(ii)* le processus d’archivage des tables *sysaudits*, *(iii)* la gestion de la file de messages et *(iv)* les différents types d’événements pouvant faire l’objet d’un audit.

### 4.1.1 Installation et configuration du système d'audit

**Exemple :** Création de *sybsecurity* sur les unités de noms logiques *DataAudit1* pour les données et *LogAudit1* pour le log et lancement du script d'installation :

*create database sybsecurity on* DataAudit1 *log on* LogAudit1
cd $SYBASE/scripts
*isql* -Usa -PMotDePasse -SNomDuServeur < *installsecurity*

Une fois l'installation faite, le serveur de données doit être arrêté et relancé. L'audit pourrait alors à être activé (par *sp_configure "audit", 1*) mais il est recommandé d'ajouter d'autres tables *d'audit*, de préférence sur des unités distinctes, avant de l'activer. Ainsi, dans l'exemple ci-dessous, la procédure *sp_addaudittable* ajoute une table d'audit sur l'unité logique *DataAudit2*. Le système affecte à cette table le premier numéro supérieur au plus grand numéro de table existante). *sp_configure "current audit table", 0, "with truncate"* indique au serveur de rendre active la table audit suivante dans l'ordre des numéros de tables en la vidant de ses tuples, le cas échéant (option *with truncate*).

*use master* ; *alter database sybsecurity on* DataAudit2
*use sybsecurity* ; *sp_addaudittable* DataAudit2
*sp_configure "current audit table", 0, "with truncate"*

### 4.1.2 L'archivage des tables *sysaudits*

L'archivage d'une table d'audit peut se faire "manuellement" en insérant ses tuples dans une table ayant les mêmes colonnes que les tables *sysaudits* (*select ... into*). L'instant de réalisation de cette opération peut être déterminé, par exemple, par une alerte levée par une procédure attachée à un seuil (*threshold*) de remplissage des segments de données de résidence des tables *sysaudits*. On peut également automatiser cet archivage en chargeant la procédure attachée au seuil de le faire.

### 4.1.3 Gestion de la file de messages

Les enregistrements de l'audit transitent par la file de messages avant d'être inscrits dans la base *via* la mémoire cache, les pages de celle-ci étant écrites physiquement dans la base après au plus 20 écritures logiques d'enregistrements d'audit. De ce fait, en cas d'incident (arrêt du serveur ou du système d'exploitation, par exemple), les enregistrements de la file non encore inscrits dans le cache peuvent être perdus. Un responsable sécurité peut modifier la taille par défaut de la file (100 messages) en essayant de trouver un bon compromis entre les performances et le nombre de messages qui risquent d'être perdus en cas d'incident. La configuration de la file (paramètre *statique "audit queue size"*) spécifie le nombre de messages que la file pourra contenir, sachant que la taille d'un enregistrement d'audit varie entre 22 et 424 octets. Par exemple, *sp_configure "audit queue size", 200*, fixe à 200 le nombre de messages et requiert environ 84KO.

### 4.1.4 Les options et les objets de l'audit

Une fois l'installation et la configuration faites, on pourra définir précisément les *options de l'audit*, c'est-à-dire les objets ou les actions concernés par le mécanisme de surveillance. Il existe quatre niveaux de surveillance : *(i)* global ou serveur, *(ii)* bases de données, *(iii)* objets dans une base et *(iv)* utilisateur au sein d'une base.

*(i) Les options de niveau global* portent sur les commandes qui concernent le serveur de données, comme les connexions et déconnexions, les erreurs du serveur, etc.

**Exemple :** Surveiller toute connexion, toute déconnexion, tout appel de procédure distante (*rpc*) entrant ou sortant par tout numéro de compte (*login*). Surveiller également toute exécution, par le numéro de compte *Lui*, d'une commande portant sur les unités (*disk init, mirror,* etc.).

sp_audit "login", "all", "all", "on" ; sp_audit "logout", "all", "all", "on"
sp_audit "rpc", "all", "all", "on" ; sp_audit "disk", "Lui", "all", "on"

*(ii) Les options de niveau base* permettent de surveiller les actions exécutées au sein de la base. Ces actions sont du type allocation/révocation de droits, création d'objets, etc.

**Exemple :** Surveiller toute exécution de *alter database* et *alter table* dans la base *master* ainsi que tout accès (*dbaccess*), création d'objet (*create*) et octroi de droits (*grant*) effectués dans la base *MaBase*.

```
sp_audit "alter", "all", "master", "on" ; sp_audit "dbaccess", "all", "MaBase", "on"
sp_audit "dump","all","MaBase","on" ; sp_audit "create", "all", "MaBase", "pass"
sp_audit "grant", "all", "MaBase", "on"
```

*(iii) Les options de niveau objet* contrôlent les actions (insertion, modification, exécution de procédures, etc.) concernant un objet particulier dans une base et ils sont stockées dans la table *sysobjects* de la base en question.

**Exemple :** Dans la base courante, surveiller tout *select* et tout *insert* exécutés sur la table *MaTable*, tout *delete* sur les futures tables, ainsi que toute exécution de procédure ou de *trigger*.

```
sp_audit "select", "all", "MaTable", "on" ; sp_audit "insert", "all", "MaTable", "on"
sp_audit "exec_procedure", "all", "default procedure", "on"
sp_audit "exec_trigger", "all", "default trigger", "on"
```

*(iv)* Enfin, *les options de niveau utilisateur* s'appliquent à un utilisateur au sein d'une base ou à un rôle système.

**Exemple :** Surveiller tout accès à toute table effectué par l'utilisateur *Elle* ainsi que toute action effectuée par tout utilisateur ayant le rôle *sa* : *sp_audit "table_access","Elle","all","on"; sp_audit "all","sa_role","all","on"*.

## 4.2 La surveillance sous Oracle

L'activation des mécanismes de surveillance est réalisée en mettant à vrai le paramètre *statique audit_trail* d'initialisation de la base. Les événements à surveiller sont spécifiés à l'aide de l'instruction *audit* et les enregistrements de la surveillance (*audit trail*) sont contenus dans la table *sys.aud$* qui fait partie du dictionnaire de chaque base. Ces enregistrements comprennent, entre autres informations, le nom de l'utilisateur concerné, les identifications de la session et du terminal, le nom de l'objet schéma accédé, la date de l'action, le privilège système utilisé, l'opération exécutée ou tentée, etc. En outre, Oracle offre un certain nombre de vues de la table *sys.aud$* qui permettent d'obtenir des informations intelligibles sur les enregistrements de surveillance. Dans le paragraphe 4.2.1 nous introduisons les types d'événements pouvant faire l'objet de l'*audit* et dans le paragraphe 4.2.2 nous discutons la gestion de l'*audit trail*.

### 4.2.1 Les options de surveillance

Les événements ou les actions qui peuvent être l'objet des mécanisme d'*audit* peuvent être classés en trois catégories : *(i)* types d'instructions SQL utilisés, *(ii)* types de commandes autorisées par des privilèges système et *(iii)* types d'actions effectuées sur des objets.

Par ailleurs, la surveillance peut être imposée pour une session (*by session*) ou pour chaque accès (*by access*). Dans le cas d'une surveillance par session, Oracle inscrit un seul enregistrement pour toutes les instructions ou toutes les commandes d'un même type exécutées sur les objets d'un schéma durant une session. En outre, la surveillance peut concerner les tentatives de connexion ou d'exécution d'instructions tant fructueuses (*whenever successful*) qu'infructueuses (*whenever not successful*) et elle peut être particularisée à un ou à plusieurs utilisateurs. Les types des événements à observer, appelés options d'*audit*, sont spécifiés par la commande *audit*, la commande *noaudit* permettant, quant à elle, d'arrêter la surveillance de tout ou partie des événements spécifiés. L'exemple ci-dessous illustre une partie de ces possibilités.

**Exemple :** Dans les formules de la commande *audit* ci-dessous, 1 demande l'*audit* des échecs d'opérations d'interrogation effectuées sur l'objet *U1.MaTable*, 2 demande l'*audit* des opérations d'interrogation et de mises à jour de tables et de vues par les utilisateurs *U1* et *U2*, 3 permet de surveiller les exécutions réussies des instructions relatives aux rôles (*create, alter, drop* et *set role*), 4 demande l'*audit* des commandes associées à tous les privilèges exécutées par *U1*, 5 demande l'*audit* de la table qui contient les enregistrements d'*audit* et 6 met fin à la surveillance de l'interrogation de tables et de vues par les utilisateurs *U1* et *U2*.

1. *audit select on* U1.MaTable *whenever not successful*;
2. *audit select table, update table by* U1, U2;
3. *audit role whenever successful*;
4. *audit all privileges by* U1;
5. *audit insert, update, delete on sys.aud$ by access*;
6. *noaudit select table by U1, U2*;

*Par défaut*, Oracle surveille les opérations de lancement (*startup*) et d'arrêt *shutdown*) d'une instance ainsi que les connexions à une base en utilisant des privilèges d'administrateur.

#### 4.2.2 La gestion des enregistrements de surveillance

La table *sys.aud$*, qui contient les enregistrements d'*audit*, est créée par un script (*catalog.sql*). La taille de *sys.aud$* est déterminée lors de la création de la base. On peut en contrôler le remplissage soit en en archivant périodiquement le contenu (par exemple, à l'aide de l'instruction *"insert into ... select ... from sys.aud$"*) ou en en éliminant des tuples inutiles (*delete from sys.aud$ where ...*). Enfin, la table *sys.aud$* et les diverses vues sont interrogeables comme toute autre table ou vue du dictionnaire de la base.

**Exemple :**

1. *select user_name, audit_option from dba_stmt_audit_opts* permet d'obtenir les options de surveillance pour chaque utilisateur ;
2. *select * from all_def_audit_opts*, sans aucune option mise par défaut, rendra un résultat de la forme :

```
ALT AUD COM DEL GRA IND INS LOC REN SEL UPD ...
--- --- --- --- --- --- --- --- --- --- ---
-/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
```

3. La même instruction exécutée après *"audit alter, grant, insert, update, delete on default"* rendra un résultat de la forme ci-dessous où un tiret ("-") signifie l'absence d'option, un *"S"* qui précède un "/" signifie que l'option est apposée par session et un *"S"* qui suit un "/" signifie que la surveillance doit être faite en cas de succès d'une action :

```
ALT AUD COM DEL GRA IND INS LOC REN SEL UPD ...
--- --- --- --- --- --- --- --- --- --- ---
S/S -/- -/- S/S S/S -/- S/S -/- -/- -/- S/S
```

## 5 Conclusion

Ce papier, sans prétendre à l'exhaustivité sur les capacités offertes par l'un ou l'autre des SGBD utilisés, a voulu montrer les éléments essentiels liés à la sécurité de fonctionnement des bases de données et à la reprise en cas d'incident. Outre la protection et la préservation de la qualité des données, ces mécanismes visent à minimiser les temps de non fonctionnement. On se doit néanmoins de dire que la continuité totale de fonctionnement n'est, à notre connaissance, assurée par aucun système, même si certains, comme Oracle, offrent des moyens de reprise sur des base en cours d'exploitation.

Enfin, on n'oubliera pas que les mécanismes et les outils mis à la disposition d'un administrateur par un SGBD (définition et gestion des vues, gestion des utilisateurs, de leurs droits et de leurs rôles, duplication d'ensembles de données, sauvegardes, surveillance des bases, etc.) peuvent être utilisés en complément de ceux offerts par une plate-forme donnée, tant au niveau système d'exploitation qu'au niveau réseau.

## Références


[1] J. Besancenot and al. *Les systèmes transactionnels: concepts, normes et prouits*. Number ISBN 2-86601-645-9 in Collection informatique. Editions Hermes, Paris, 1997.

[2] N. Boudjlida. *Bases de données et systèmes d'informations. Le modèle relationnel: langages, systèmes et méthodes (Databases and Information Systems. The relational model: Languages, Systems and Design)*. Dunod, Paris, 1999. Cours et exercices corrigés. Collection Sciences Sup. (*in French*).

[3] N. Boudjlida. *Gestion et Administration des Bases de Données: Application à Sybase et Oracle*. Dunod, Paris, 2003.

[4] Silvana Castano, Maria Grazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security.* Addison-Wesley & ACM Press, 1995.

[5] V. Kumar and M. Hsu, editors. *Recovery Mechanisms in Database Systems*. Prentice Hall, 1998.

[6] Oracle-Corp. Oracle 8i, SQL Reference, September 2000. Release 3(8.1.7), Part No A85397-01.

[7] M. Tamer Oszu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall International, December 1998. 2nd edition.

[8] RAID. Redundant Array of Inexpensive Disks, November 1998. *http://linas.org/linux/Software-RAID/Software-RAID.html*.