8. Protection des SGBD

8.1 Introduction

Les SGBD “courent” certains dangers que nous pouvons répertorier en :
- accidents logiciels
- utilisation pernicieuse
- pannes matérielles

Les SGBD doivent faire face et assurer d’une part l’intégrité des données, d’autre part la sécurité et le contrôle des accès.

Dans ce chapitre nous aborderons plusieurs aspects :
- tout d'abord il faut assurer la cohérence des informations stockées. Il s'agit de contraintes d'intégrité que doivent respecter les données.
- si plusieurs utilisateurs sont autorisés à manipuler la même base de données, il faut s'assurer que les actions des uns ne seront pas préjudiciables aux autres. Ce problème est connu sous le nom de contrôle de concurrence. Le contrôle de concurrence dans un SGBD doit rendre le partage des données complètement transparent aux utilisateurs et doit donc contrôler l'exécution simultanée de transactions de sorte à produire les mêmes résultats qu'une exécution séquentielle.
- un SGBD peut fonctionner sur plusieurs machines, or une machine peut tomber en panne entraînant une altération des données de la base. Il faut donc que le SGBD assure le fait que la base restera dans un état cohérent même après une panne : on parle de sûreté de fonctionnement et du mécanisme de reprise (sur panne).
- enfin si plusieurs usagers utilisent la base il s'agit de vérifier que chaque utilisation est autorisée : problèmes des droits.

8.2 Intégrité des données.

Dans une base de données un des problèmes essentiels est d’assurer la cohérence des données (les données doivent rester conformes à la réalité qu'elles représentent).

Une **Contrainte d'Intégrité** par définition est un Prédicat toujours vrai sur une partie ou sur la totalité des données de la base (invariant).

Un schéma de Base de Données est soumis à un ensemble de Contraintes d’Intégrité.

Une instance de ce schéma respectant les Contraintes d'Intégrité est dite Base de Données **cohérente.**

Il existe plusieurs types de Contraintes d'Intégrité comme nous l’avons déjà vu au cours des chapitres précédents : nous pouvons cependant les séparer en deux grandes classes les contraintes **statiques** et les contraintes **dynamiques**.

8.2.1 Contraintes statiques.

Rappel :
- NOT NULL
- PRIMARY KEY
- UNIQUE
- CHECK
- DEFAULT
- FOREIGN KEY
    - ON DELETE CASCADE
    - ON DELETE RESTRICT
    - ON DELETE SET NULL
    - ON UPDATE CASCADE

8.2.2 Contraintes dynamiques

Il s'agit de contraintes que l'on peut souhaiter imposer lors d'un passage d'un état de la base à un autre. Par exemple les n-uplets du nouvel état dépendent des n-uplets de l'état précédent.

Un autre aspect des contraintes dynamiques est celui qui consiste à définir des **actions spontanées** (triggers) qui seront déclenchées automatiquement par le SGBD lorsque certaines opérations se produisent

Quels sont les moyens que le SGBD a de voir si les CI sont respectées ?

- à la compilation si les CI ne dépendent pas des valeurs.
- à l'exécution lorsqu'un traitement est appliqué à une base de données, en général celle-ci passe par des états transitoires durant lesquels certaines CI ne sont plus préservées

Afin d'isoler des unités de traitement respectant la cohérence de la base on introduit la notion de *transaction.* Le contrôle des CI se fera en fin de transaction et peut alors nécessiter une remise en cause de l'exécution de la transaction.

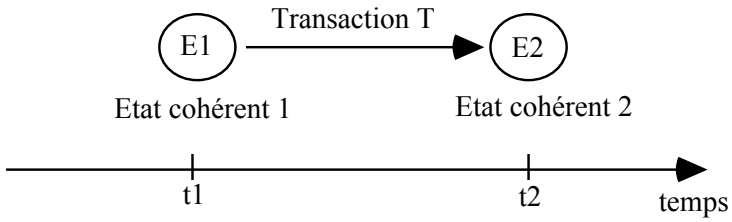
8.3 Concept de Transaction.

8.3.1 Définitions

L'usager manipule la base de données soit au travers du langage de requête, soit au travers de programmes qui font appel au SGBD. L'éxécution d'une requête ou d'un programme fait naître au niveau du SGBD une occurrence de **transaction**.

• Une **transaction** est une unité de traitement séquentiel (séquence d'actions cohérente), exécutée pour le compte d'un usager, qui appliquée à une base de données cohérente restitue une base de données cohérente. (une transaction peut être simplement l'exécution d'une requête SQL, ou bien celle d'un programme en langage hôte dans lequel on a des appels au langage de requête)

Les Contraintes d'Intégrité sont donc toujours des **invariants** pour les transactions ce qui signifie qu'une transaction respecte les Contraintes d'Intégrité à la fin pourvu que celles-ci soient respectées au début. Le respect des contraintes est à la charge du programmeur de transaction.



Les SGBD modernes ne permettent que des **transactions**.

Le mécanisme de gestion des transactions doit assurer :

- que lors d'une exécution d'une transaction toutes ses actions sont exécutées ou bien aucune ne l'est (**atomicité**). De plus les effets d'une transaction qui s'est exécutée correctement doivent survivre à une panne (**permanence**),
- que chaque transaction soit **isolée**, de manière à éviter des **incohérences** lors d'exécutions concurrentes.

6.2.2 Isolation d'une transaction et concurrence.

Même si les transactions sont cohérentes, leur entrelacement peut conduire à des incohérences globales.

Exemple 1 :
T1 et T2 exécutent la suite d'actions ci-dessous.
La variable X à la fin des deux transactions a la valeur 300 alors que sa valeur serait 400 si les deux transactions T1 et T2 s'étaient succédées.

Temps	T1	Etat de la base	T2
t1	lire (X)	(X=100)	———
t2	———		lire (X)
t3	X:=X+100		———
t4	———		X:= X+200
t5	écrire (X)	(X=200)	———
t6	———	(X=300)	écrire (X)

Exemple 2 :

La base est soumise à la contrainte d'intégrité Y=2X (X=5 et Y=10 avant), les deux transactions T1 et T2 s'exécutent, la base est alors dans un état *incohérent* la CI n'étant plus vérifiée. (X=30, Y=20)

Temps	T1	Etat de la base	T2
t1	X:=10	(X=5 , Y=10)	———
t2	écrire (X)	(X=10, Y=10)	———
t3	———		X:=30
t4	———	(X=30, Y=10)	écrire (X)
t5	———		Y:= 60
t6	Y:= 20	(X=30, Y=60)	écrire (Y)
t7	écrire (Y)	(X=30, Y=20)	———

Exemple 3 :

CI : Y=2X (X=5 et Y=10 avant)

La base est cohérente après les deux transactions, mais*T1 a lu des valeurs incohérentes (X=5 et Y=60)*

Temps	T1	Etat de la base	T2
t1	lire (X)	(X=5 , Y=10)	———
t2	———	(X=5, Y=10)	———
t3	———		X:=30
t4	———	(X=30, Y=10)	écrire (X)
t5	———		Y:= 60
t6	———	(X=30, Y=60)	écrire (Y)
t7	lire (Y)		———

Il est donc nécessaire d'une part d'**isoler** les divers éléments d'une transaction et d'autre part de s'assurer de la **cohérence** de la base après exécution d'un ensemble des transactions concurrentes.

a Isolation

La base de données est découpée en éléments appelés *granules* (c'est en général l'administrateur en accord avec le concepteur qui fixe le niveau de granularité : n-uplet, page, table).

L'isolation d'un élément dans une transaction est obtenu par le mécanisme des *verrous* (lock).

Ces verrous sont définis par deux opérations :

• verrouiller(A)
Cette opération oblige toute transaction T à attendre le déverrouillage de l'élément A si elle a besoin de cet élément.

• déverrouiller(A)
La transaction effectuant cette opération libère le verrou qu'elle avait obtenu sur A et permet à une autre transaction candidate en attente d'obtenir le sien.

L'utilisation de verrou entraîne éventuellement la mise en attente de transaction.

Ceci a pour conséquence deux grands types de problèmes :

• la **famine**

Lorsqu'un verrou est relâché sur un élément A, le système choisit parmi les transactions candidates en attente, si les transactions sont liées au niveau de priorité de l'utilisateur certaines transactions attendront longtemps ... La solution pour pallier à ce genre de situation est d'instaurer une file d'attente et de gérer les demandes de verrou en respectant l'ordre d'entrée dans la file.

• L'**interblocage** (deadlock)

Cette situation se présente lorsqu'un ensemble de transactions attendent mutuellement le déverrouillage d'éléments actuellement verrouillés par des transactions de cet ensemble.
Ex Si T1 obtient un verrou sur A, puis T2 obtient un verrou sur B, et qu'enfin T1 demandant un verrou sur B soit mise en attente, la demande ultérieure par T2 d'un verrou sur A entraîne l'interblocage des deux transactions. Les solutions consistent la plupart du temps à faire intervenir le système qui "tue" les transactions.

Le mécanisme de verrouillage permet donc d'isoler un élément pour une transaction, cependant il faut encore que cette utilisation soit faite à bon escient.

Pour que l'exécution d'un ensemble de transactions concurrentes transforme la base depuis un état cohérent jusqu'à un autre état cohérent, on impose, en général, une contrainte forte aux transactions : leur exécution doit être **sérialisable**.

b transactions sérialisables

Définition
Une exécution d'un ensemble de transactions est **sérialisable** ssi elle est équivalente à une exécution séquentielle (ou série) de transactions.

Quand les transactions sont arbitraires, la sérialisabilité est la seule propriété qui assure le bon "entrelacement".

Réalisation

La sérialisabilité peut être obtenue en imposant aux transactions que tous les verrouillages précèdent tous les déverrouillages. Les transactions sont alors dites à **deux phases** : une phase d'acquisition des verrous puis une phase de libération.

8.3.2 Atomicité d'une transaction

La gestion de transaction nécessite la propriété d'**atomicité** d'une transaction c'est à dire d'assurer que les opérations mêmes les plus complexes englobées au sein d'une transaction ne soient perçues que comme une

opération unique. De plus il faut que soit toutes les modifications liées à cette opération soient effectuées, soit aucune ne l'est.

C'est le "système" qui est à même d'assurer l'atomicité des transactions.

Le modèle général de transaction sera le suivant :

BEGIN TRANSACTION ;

REQUETES
INSTRUCTIONS

COMMIT ; ou bien ROLLBACK ;

Comme une transaction peut ne pas se terminer normalement (soit à cause de pannes diverses, soit par intervention du système lors d'un interblocage) on raffine les définitions :

- avant un **point de validation**, une panne entraîne la perte (totale, à cause de l'atomicité) de la transaction.
- après, la transaction sera visible (au bout d'un certain temps) par les autres : **permanence (si panne refaire)**

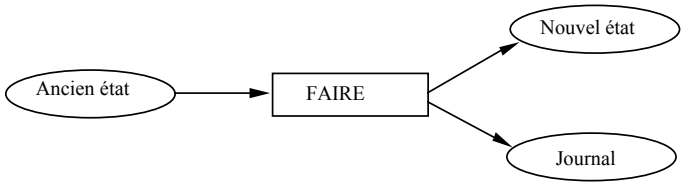
Cette méthode s'appelle la **validation à deux phases**. Elle suppose souvent l'existence d'une **mémoire stable**, dans laquelle, au point de validation, les nouvelle valeurs devront être enregistrées.

Une transaction ayant atteint son point de validation sera dite **validée** (elle ne peut plus être remise en cause). Une transaction n'ayant pas encore atteint son point de validation sera dite **vivante**.

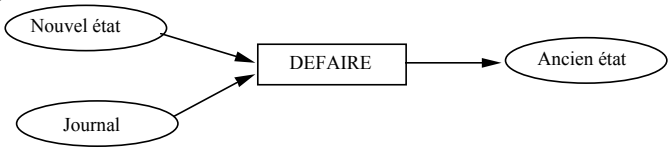
8.3.3 Permanence

Le problème de la permanence est de faire en sorte que lorsqu'une transaction a atteint son point de validation, les effets de la transaction soient conservés sur la base quelles que soient les circonstances.
La solution est obtenue par l'intermédiaire de fichiers **journaux** qui concervent la trace des transactions successives qui ont été effectuées sur la base.

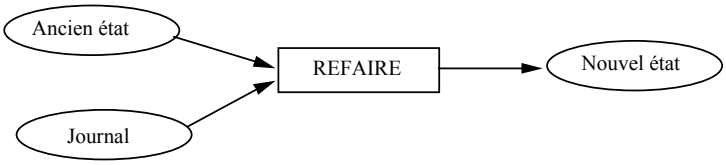
Lors d'une transaction qui effectue une mise à jour sur la base, la base passe d'un ancien état à un nouvel état et on conserve dans le journal, l'identification de la transaction, l'identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur.



Lors d'une panne si la transaction doit être **défaite** on se sert du nouvel état et de l'ancienne valeur se trouvant dans le journal pour reconstituer l'ancien état.



Si la transaction doit être refaite, il faut partir de l'ancien état et du journal pour reconstruire le nouvel état.



8.4 Sécurité

La sécurité dans les SGBD est réalisée à trois niveaux :

- identification
- protection physique
- maintenance et transmission des droits

Identification

Tout usager autorisé possède une identification et un mot de passe gérés par le système (avec période de validité).

Protection physique

L'administrateur ainsi que tout utilisateur peut recourir à la technique de vues en lecture seule pour protéger les données.

Dans certains cas la protection physique peut être réalisée en faisant appel aux mécanismes de protection du système de gestion de fichiers du système d'exploitation (cas de SQL pour UNIX).

Maintenance des droits

Selon les systèmes l'utilisateur peut transmettre (ou faire transmettre via le DBA) ou supprimer aux autres utilisateurs potentiels des droits (lecture, mise à jour ..) sur les objets qu'ils créent (GRANT et REVOKE).