Gestion des transactions

ELMARI ET NAVATHE, 6TH ED.
CHAPITRE 21

Plan

- Introduction
- Problématique
- Modèle transactionnel
- Propriétés recherchées (ACID)
- Récupérabilité (reprise)
- Sérialisabilité
- Gestion transactionnelle sous SQL
- SF, SGBD, SQL et ACID

Introduction

Motivation

- Besoins pratiques devant être assurés par un SGBD
  - desservir simultanément plusieurs clients (plusieurs sessions)
  - fournir un mécanisme de récupération (en cas de défaillance)
  - fournir un mécanisme de restauration (en cas de perte de données)
- Éléments pris en compte
  - EMT : environnement multi-tâches (plusieurs processus concurrents)
  - HM : hiérarchisation des mémoires (plusieurs tampons intermédiaires)
- Corolaires
  - accès concurrents aux variables
  - accès concurrents aux mémoires
  - existence possible d'états transitoires incohérents
- Conséquence
  - complexification du maintien de l'intégrité (préalable à la validité)
- Éléments non pris en compte
  - duplication externe des données
  - répartition des données
  - répartition des traitements

Rappels

- Concurrence : les processus se partagent un même processeur, leurs exécutions s'entrelacent.
- Parallélisme : les processus sont simultanément exécutés sur des processeurs différents.
- En général, les deux en même temps !

Note sur la concurrence et parallélisme

- En général, un modèle adéquat pour la concurrence l'est aussi pour le parallélisme, mais pas l'inverse.
- Les *processus séquentiels concurrents* (CSP) de Hoare demeurent un des modèles les plus pertinents.

Rappels (bis)

- Récupération
  - Ré-exécution de transactions préalablement annulées.
  - Un point de récupération correspond à un état de la BD à partir duquel la récupération des transactions peut être entreprise.
  - Les transactions débutées antérieurement au point de récupération, mais non terminées avant celui-ci doivent être annulées préalablement à la récupération.
- Restauration
  - Remise en service d'une copie de sécurité, suivie de la récupération des transactions intègres intervenues après la prise de la copie de sécurité.
  - Un point de restauration correspond à un état de la BD qui peut faire l'objet d'une copie de sécurité ; l'état devant être intègre.

Références

- ELMASRI, Ramez ; NAVATHE, Shamkant B.; *Fundamentals of database systems ;* Sixth edition, Pearson Addison Wesley, 2011. ISBN 978-0-13-608620-8.
  - chapitres 21 à 23
- George Coulouris, Jean Dollimore, Tim Kindberg ; *Distributed systems : concepts and design* ; Forth edition, Addison-Wesley, 2005. ISBN 0-321-26354-5.
  - pour un approfondissement, le livre au complet!

Problématique

Problèmes et pannes

Accès concurrents

1. Lecture non reproductible
2. Mise à jour perdue
3. Mise à jour transitoire (*dirty read*)
4. Agrégation incorrecte
   - états transitoires
   - fantômes

Pannes et récupération

1. Catastrophe
2. Défaillance matérielle
   - unité de traitement
   - unité de stockage
3. Défaillance logicielle
4. Transaction incorrecte
5. Invariant non respecté
6. Concurrence conflictuelle

Lecture non reproductible

T1	T2
• lire (x) • y := x	
	• x := k • écrire (x)
• action excluant x et y • lire (x) • si x = y alors …	

t

Mise à jour perdue

T1	T2
• lire (x)	
• x := x-n	
	• lire (x)
	• x := x+m
• écrire (x)	
• lire (y)	
	• écrire (x)
• y := y+n	
• écrire (y)	

t

Mise à jour transitoire
(*dirty read*)

T1	T2
• lire (x)	
• x := x-n	
• écrire (x)	
	• lire (x)
	• lire (z)
	• z := x+m
	• écrire (z)
• action quelconque	
• annuler tout!	

t

Agrégation incorrecte – état transitoire

T1	T2
• …	• somme := 0
• lire(x_i)	• lire (x_1)
• x_i := x_i-k	• somme := somme+x_1
• écrire (x_i)	• …
• …	
• lire (x_j)	• lire(x_n)
• x_j := x_j+k	• somme := somme+x_n
• écrire (x_j)	
• …	

t

Agrégation incorrecte - fantôme

T1	T2
• DELETE FROM T WHERE x > k	• SELECT count(*) FROM T

t

La gestion transactionnelle
une solution (partielle) aux problèmes et aux pannes

Coordination des accès concurrents

- coordonner l'accès aux variables partagées
  - a priori
    - optimal
    - assuré
  - a posteriori
    - sous-optimal
    - non assuré, donc
      - possibilité d'échec
      - nécessité de récupération

Mécanisme de récupération

- tenir un journal transactionnel
- calculer les points de récupération
- en cas d'échec
  - restaurer le dernier point de récupération
  - nettoyer les transactions annulées
  - refaire les transactions maintenues

Modèle transactionnel

Qu'est-ce qu'une transaction ?

- Définitions
  - Suite ordonnée d'opérations, cette suite est considérée atomique relativement à l'intégrité de la BD, c'est-à-dire :
    - soit toutes les opérations sont exécutées dans l'ordre,
    - soit aucune n'est exécutée.
  - Une séquence de traitement comprenant une ou plusieurs opérations d'accès à la BD comprises entre un début et une fin de transaction.
  - Quelles opérations ?
    - Lecture [consultation].
    - Écriture [modification, insertion, retrait].

Corolaires

1. Plusieurs instances de plusieurs applications peuvent interagir concurremment avec une même BD.
2. Une même application peut lancer plusieurs transactions concurrentes.
3. L'intégrité d'une BD peut ne pas être vérifiée lorsqu'une transaction est en cours.
4. Le temps d'exécution d'une transaction peut être long.
5. Il peut s'écouler un temps arbitrairement long entre deux moments permettant la vérification de l'intégrité totale de la BD.
6. En pratique, il faut recourir à la vérification partielle de l'intégrité au sein de chaque transaction et garantir l'indépendance des transactions.

Opérations constituant une transaction… et SQL

- Opérations
  - débuter, lire, écrire, terminer (avec confirmation, avec annulation)
  - *begin, read, write, end (with commit, with roll back)*
- « débuter »
  - implicite en SQL-1992
  - peut être explicite depuis SQL-2003
  - devrait être explicite
- « lire »
  - représente toutes les opérations de consultation
  - SQL : SELECT
- « écrire »
  - représente toutes les opérations d'insertion et de suppression
  - SQL : INSERT, DELETE, UPDATE
- « terminer »
  - le plus souvent implicite
  - devrait être explicite

Insertion, suppression… et modification

- Que faut-il distinguer
  - théoriquement ?
  - pratiquement ?
- Pourquoi ?

- Et si les réponses se trouvaient dans la définition même de l'affectation ?
- Voir la fin de la section.

Automate

- voir Elmasri, figure 21.4, page 752

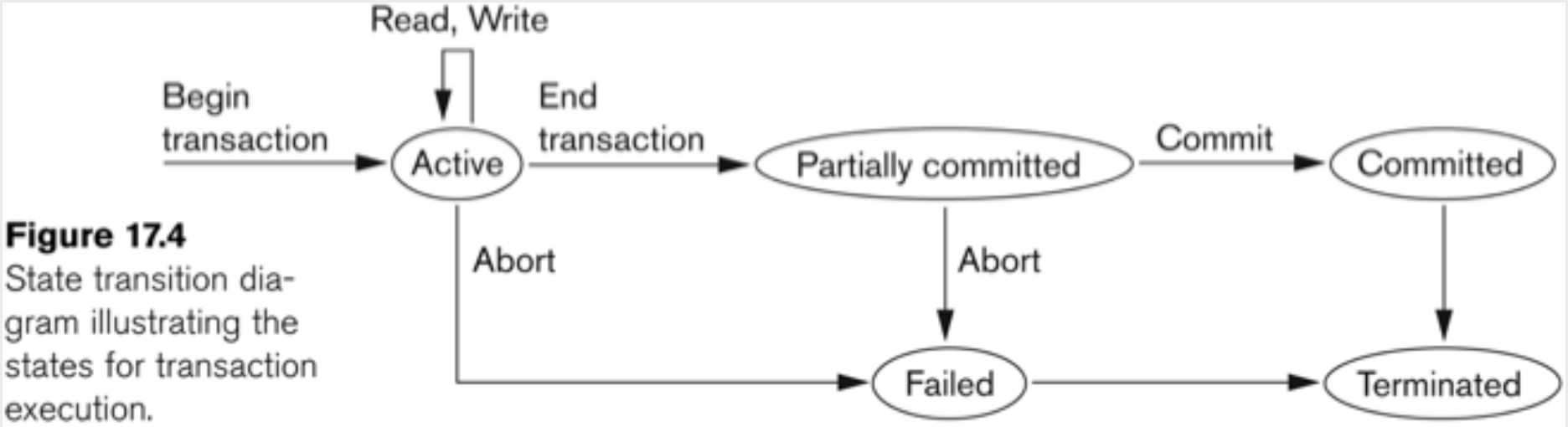


Figure 17.4
State transition diagram illustrating the states for transaction execution.

- *Terminated* -> terminée ?
- Ne manquerait-il pas un état… et des transitions ?

Conflit

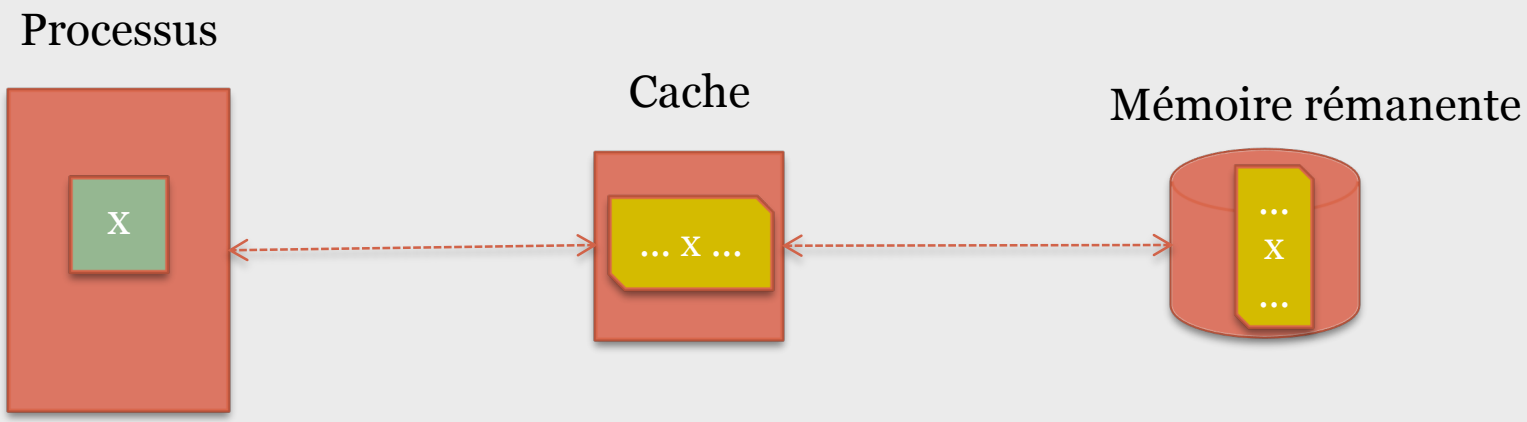
- Deux opérations sont en conflit si et seulement si les trois conditions suivantes sont satisfaites :
  - elles appartiennent à deux transactions distinctes
  - elles accèdent au même élément de la BD
  - l'une d'elles est une écriture

- On remarque que les six problèmes présentés initialement sont la conséquence de l'existence de conflits dans la suite des opérations

Granularité transactionnelle

- Celle de l'entité de donnée accédée, ici appelée « élément » (*item* en anglais)
  - colonne (attribut)
  - ligne (tuple)
  - table (relvar)
  - schéma
- Aux fins de l'exposé, sans perte de généralité, une BD sera un ensemble d'éléments dénotés (un tuple ?)

Modèle de gestion mémoire



- X est un élément dénotable.
- Il est stocké dans un ou plusieurs blocs (du cache comme de la mémoire).
- Il pourrait y avoir plusieurs niveaux de cache.
- En général, les frontières de blocs et d'éléments ne coïncident pas.

Opérations d'accès
(on suppose l'élément contenu en entier dans un seul bloc)

- Lire (x)
  - trouver l'adresse en mémoire rémanente du bloc contenant l'élément dénoté par x
  - prendre une copie en mémoire cache du bloc contenant l'élément x (s'il n'est pas déjà en mémoire cache)
  - copier l'élément de la mémoire cache dans la variable x
- Ecrire (x)
  - trouver l'adresse en mémoire rémanente du bloc contenant l'élément dénoté par x
  - prendre une copie en mémoire cache du bloc contenant l'élément x (s'il n'est pas déjà en mémoire cache)
  - copier le contenu de la variable x dans le bloc à l'endroit approprié
  - recopier (immédiatement ou en différé) le bloc en mémoire rémanente

Opérations d'intendance

- Autres opérations élémentaires
  - débuter
  - terminer
    - confirmer (*commit*)
    - annuler (*roll back*)
- Opérations propres à la récupération
  - défaire (*undo*)
  - refaire (*redo*)

Affectation multiple

- Définition
- De la différence entre transaction et affectation multiple

ACID

Propriétés fondamentales

- Une séquence de transactions T_1, … T_n est ACID ssi les quatre propriétés suivantes sont respectées
  - Atomicité : T_i exécutée entièrement ou pas du tout
  - Cohérence : T_i validée ⇒ $Invariant_{Ti}(T_i(BD))$
  - Isolation : $(T_1 \,||\, \ldots \,||\, T_n) \equiv (T_1 ; \ldots ; T_n)$
  - Durabilité : T_i conservée ⇔T_i validée
- avec
  - $Invariant_{Ti}$ est l'ensemble des prédicats (invariants) susceptibles d'être invalidés par la transaction T_i
  - $T_i(BD)$ est la valeur calculée par T_i appliquée à BD
  - $(T_1 ; \ldots ; T_n)$ est la sérialisation de T_1, …, T_n

Sérialisation

- Une suite d'opérations est une sérialisation de T_1, ..., T_n si elle est formée de la suite des opérations induite par une permutation de T_1, ..., T_n
- Exemple
  - soit
    - T1 = <a11, a12, a13>
    - T2 = <a21, a22>
  - il existe deux sérialisations de T1, T2
    - <a11, a12, a13, a21, a22>
    - <a21, a22, a11, a12, a13>
- En général, il existe *n!* sérialisations de *n* transactions

Récupération et journalisation

Nécessité

- Puisqu'il y a plusieurs transactions concurrentes, nous aurons besoin d'en conserver le journal, aux fins de récupération, et la trace, aux fins d'analyse.

Qu'est-ce qu'un journal ?

- Journal
  - [#, début]
  - [#, lire, x]
  - [#, écrire, x, v, v']
  - [#, fin]
  - [#, confirmer]
  - [#, annuler]
- # représente des informations complémentaires :
  - estampille,
  - identifiant de l'utilisateur,
  - emplacement de l'utilisateur…

- alternative
  - [#, l, x, v]
  - [#, e, x, v']
    est moins performante en général
- omission de [#, f] sans perte de généralité
- omission de [#, l, x] en l'absence de cascades d'annulations
- la valeur antérieure v est inutile si les journaux sont toujours stricts

Qu'est-ce qu'un journal (en) ?

- Journal
  - [#, start]
  - [#, read, x]
  - [#, write, x, v, v']
  - [#, end]
  - [#, commit]
  - [#, rollback]

- Journal
  - [#, début]
  - [#, lire, x]
  - [#, écrire, x, v, v']
  - [#, fin]
  - [#, confirmer]
  - [#, annuler]

Complétude

- Un journal J est complet relativement à T_1, ..., T_n si et seulement si :
  - J contient exactement les opérations de T_1, ..., T_n
  - les opérations d'une même transaction apparaissent au sein du journal dans le même ordre que dans la transaction elle-même.
- Si on admet la possibilité d'une exécution concurrente de la journalisation elle-même, il faut ajouter une troisième condition
  - si deux opérations sont conflictuelles, l'une doit être exécutée entièrement avant l'autre

Complétude et entrelacement

- L'entrelacement est une trace intégrale d'un programme
- L'entrelacement effectif d'une exécution d'un programme est un journal.

Journal, quelques concepts apparentés

en français	en anglais
• journal (récupération) • trace (transaction) • entrelacement • historique (journal) • programme (ensemble de transactions)	• log • trace • interleaving (interlace) • history • schedule

Points de récupération et points de vérification

- Point de récupération
  - coïncide avec la réussite d'une opération de confirmation
  - les opérations appartenant à des transactions non confirmées doivent alors être annulées
  - afin de minimiser les annulations, une opération de confirmation peut ne pas induire de point de récupération…
  - au détriment de la granularité de la récupération
- Point de vérification
  - tout point de récupération déterminant une trace complète

Préalables à la récupération (fr)

- Point de récupération (commit point)
  - Une transaction atteint un point de récupération lorsque toutes ses opérations ont été exécutées et journalisées avec succès.
  - Au-delà de ce point de récupération, la transaction est dite confirmée (comitted).
  - À ce moment seulement, l'entrée [#, f] est inscrite au journal.
- Annulation de transaction (transaction rollback)
  - Toute transaction inscrite au journal par une entrée [#, d] sans entrée de confirmation [#, c] correspondante (et ultérieure) doit être annulée.

Préalables à la récupération (en)

- Definition a Commit Point
  - A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log.
  - Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
  - The transaction then writes an entry [#, commit] into the log.
- Roll Back of transactions
  - Needed for transactions that have a [#, start] entry into the log but no commit entry [#, commit] into the log.

Opérations de récupération (fr)

- Reprise de transactions (Redoing transactions)
  - Toute transaction confirmée doit posséder une trace complète inscrite au journal, le journal contient donc toutes les écritures qui lui sont associées, la transaction est donc récupérable.
  - Note 1 : Le journal doit être stocké sur un support différent de celui de la BD.
  - Note 2 : Une catastrophe (un incident) peut avoir corrompu ou détruit toutes les mémoires dans lesquelles est stockée la BD (sauf le journal). La récupération doit donc être réalisée à partir des seules entrées intègres du journal.
- Écriture forcée
  - Préalablement à l'écriture de l'entrée de confirmation au journal, toute entrée pendante doit y être écrite.

Opérations de récupération (en)

- Redoing transactions
  - Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be redone from the log entries.
  - Notice that the log file must be kept on another media (e.g. disk).
  - At the time of a system crash, only the log entries that have been written back to the log media are considered in the recovery process because the contents of main memory may be lost.
- Force writing a log
  - Before a transaction reaches its commit point, any portion of the log that has not been written to the log media yet must now be written to the log media.
  - This process is called force-writing the log file before committing a transaction.

Récupérabilité

- Un journal est dit « récupérable » si toute transaction confirmée n'est jamais annulée.
- Un journal est dit « sans cascades d'annulations » si l'annulation d'une transaction n'entraine l'annulation d'aucune autre transaction (même non encore confirmée).
- Un journal est dit « strict » si aucun accès à un élément X n'intervient entre une écriture de X et la terminaison de la transaction qui contient cette écriture.

- strict => sans cascades d'annulations => récupérable

Récupérabilité sans cascades d'annulations

- Un journal est sans cascades d'annulations si les lectures d'éléments modifiés sont toujours postérieures à la confirmation de la transaction contenant l'opération d'écriture ayant entrainé la modification

Sérialisabilité

Sérialisabilité

- Un programme T_1, ..., T_n, est sérialisable s'il existe une trace de T_1, ..., T_n équivalente à une sérialisation de T_1, ..., T_n.
- On considère généralement trois types d'équivalences
  - de résultat
  - de conflit
  - de visibilité

Équivalence de résultat
(r-équivalence)

- Deux traces d'un même programme
  $T_1, \ldots, T_n$
  sont équivalentes relativement aux résultats ssi
  - leur exécution détermine le même état de la BD
    (pour un état initial de la BD -- pour tout état initial de la BD)

- Deux états de BD sont égaux si tout élément d'un état a la même valeur que son correspondant dans l'autre état

Équivalence de conflit
(c-équivalence)

- Deux traces d'un même programme

  $T_1, \ldots, T_n$

  sont équivalentes relativement aux conflits ssi
  - l'ordre relatif de deux opérations conflictuelles est toujours le même dans les deux traces.

Équivalence de visibilité
(v-équivalence)

- Deux traces E_1 et E_2 d'un même programme
  $T_1, \ldots, T_n$
  sont équivalentes relativement aux vues ssi
  - pour toute opération Lire(X) de T_i, si la valeur X a été l'objet antérieurement d'une opération Ecrire(X) de T_j dans E_1, la même condition prévaut dans E_2
  - pour toute opération Lire(X) de T_i, si la valeur X n'a été l'objet antérieurement à aucune opération Ecrire(X) de T_j dans E_1, la même condition prévaut dans E_2
  - pour toute opération Ecrire(Y) de T_k qui est la dernière opération modifiant Y dans E_1, il en soit de même dans E_2

Sérialisabilité en pratique

- La r-Sérialisabilité est sujette à l'équivalence accidentelle ; de plus, elle n'est pas calculable en général.
- On préfère utiliser le critère suivant : les opérations appliquées à tout élément modifié par le programme devront l'être dans le même ordre; ceci correspond à la c-Sérialisabilité.
- La v-Sérialisabilité en est une relaxation intéressante et acceptable (elle permet la permutation des lectures antérieures à une écriture).

Sérialisabilité en pratique

- On peut déterminer a priori la c-Sérialisabilité d'un programme (voir Elmasri, Algorithme 21.1, page 764).
- Cela n'est toutefois guère pratique :
  - le programme est rarement connu à l'avance, en particulier, le plus souvent, il n'est pas fermé;
  - l'optimalité d'une exécution concurrente dépend des délais effectifs sur les opérations (qui eux-mêmes dépendent d'éléments non déterministes);
  - certaines opérations peuvent échouer entrainant une annulation non planifiée;
  - etc.

Sérialisabilité en pratique

- Pour ces raisons, on se tournera plutôt vers des solutions dynamiques utilisant
  - les verrous ou
  - les estampilles
- Ce qui n'empêche pas d'avoir recours à certains prétraitements, notamment pour préqualifier les transactions elles-mêmes.
- Par exemple, pour réduire l'incidence des lectures non répétables, on s'assure que toute transaction ne lit un élément qu'une fois (en la transformant, au besoin).

Suite…

- dans le module BD033_Concurrence

Gestion transactionnelle avec SQL

Sources

- … voir Elmasri, section 21.6
- … voir Ullman, section 6.6

La forme la plus simple

- START TRANSACTION
  READ { ONLY | WRITE }
  ISOLATION LEVEL <niveau>
  DIAGNOSTICS SIZE <entier positif>
- … liste de commandes SQL …
- { COMMIT | ROLL BACK }

Niveaux

Niveau	Possibilité de lecture		
	transitoire	non répétable	fantôme
READ UNCOMMITTED	oui	oui	oui
READ COMMITTED	non	oui	oui
REPEATABLE READ	non	non	oui
SERIALIZABLE	**non**	**non**	**non**

Exemple (bis)

- On peut faire beaucoup mieux avec les PSM !

16.1 <start transaction statement>

Rôle

Établir les propriétés d'une transaction puis la démarrer.

Syntaxe

<start transaction statement> ::=

START <transaction characteristics>

<transaction characteristics>

<transaction characteristics> ::=
TRANSACTION <transaction mode> [{ <comma> <transaction mode> }...]
<transaction mode> ::=
<isolation level>
| <transaction access mode>
| <diagnostics size>
<transaction access mode> ::=
READ ONLY
| READ WRITE
<isolation level> ::=
ISOLATION LEVEL <level of isolation>
<level of isolation> ::=
READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE
<diagnostics size> ::=
DIAGNOSTICS SIZE <number of conditions>
<number of conditions> ::=
<simple value specification>

16.2 <set transaction statement>

Rôle

Établir les propriétés de la prochaine transaction à être démarrée (sans la démarrer).

Note 412

La porté de l'instruction est limitée à la prochaine transaction et n'a pas d'effets sur les subséquentes.

Syntaxe

<set transaction statement> ::=

SET [LOCAL] <transaction characteristics>

16.3 <set constraints mode statement>

Rôle

Établir le mode d'exécution des contraintes : si une transaction est cours, de cette transaction; sinon, de la prochaine à être démarrée.

Note 414

La porté de l'instruction est limitée à cette transaction et n'a pas d'effets sur les subséquentes.

Syntaxe

<set constraints mode statement> ::=

SET CONSTRAINTS <constraint name list>
{ DEFERRED | IMMEDIATE }

<constraint name list> ::=

ALL

| <constraint name> [{ <comma> <constraint name> }...]

16.4 <savepoint statement>

Rôle

Établir, puis conserver un point de récupération (*save point*).

Syntaxe

<savepoint statement> ::= SAVEPOINT <savepoint specifier>

<savepoint specifier> ::= <savepoint name>

16.5 <release savepoint statement>

Rôle

Supprimer un point de récupération.

Syntaxe

<release savepoint statement> ::=

RELEASE SAVEPOINT <savepoint specifier>

16.6 <commit statement>

Rôle

Terminer la transaction courante puis amorcer le processus de confirmation (commit).

Syntaxe

<commit statement> ::=
COMMIT [WORK] [AND [NO] CHAIN]

16.7 <rollback statement>

Rôle

Suspendre la transaction courante puis amorcer un des processus de suite suivants :

(a) annuler puis terminer la transaction;

(b) rétablir un point de récupération puis poursuivre la transaction;

(c) annuler la transaction puis enchainer une autre transaction depuis un de ses points de récupération.

Syntaxe

<rollback statement> ::=
ROLLBACK [WORK]
[AND [NO] CHAIN] [<savepoint clause>]

<savepoint clause> ::=
TO SAVEPOINT <savepoint specifier>

Rôle

Éviter l'enchaînement comme la peste!

20.2 <embedded exception declaration>

Rôle

Définir le traitement des exceptions dans la portée courante.

Syntaxe (obsolète et TRÈS variable selon les dialectes)

<embedded exception declaration> ::=
WHENEVER <SQL condition> <condition action>

<SQL condition> ::=
<major category>
| SQLSTATE (<SQLSTATE class value> [, <SQLSTATE subclass value>])
| CONSTRAINT <constraint name>

<major category> ::=
SQLEXCEPTION | SQLWARNING | NOT FOUND

<SQLSTATE class value> ::=
!! See the Syntax Rules.

<SQLSTATE subclass value> ::=
!! See the Syntax Rules.

<condition action> ::=
CONTINUE | <go to>

<go to> ::=
{ GOTO | GO TO } <goto target>

<goto target> ::=
<host label identifier>
| <unsigned integer>

SGF, SGBD, SQL et ACID

SGF : SYSTÈME DE GESTION DE FICHIERS
SGBD : SYSTÈME DE GESTION DE BASES DE DONNÉES

SGBD et SGF

Différences entre un SGBD et un SGF

- La cohérence ne peut pas être assurée par un SGF, puisqu'on ne peut y exprimer de contraintes.
- Corolaire : il est suffisant qu'un SGF soit atomique et intègre (A~~C~~I~~D~~).
- Corolaire : mySQL est un SGF.

SQL et ACID

- Un SGBD non ACID ne peut garantir la validité des données ni celle des résultats.
- SQL n'est *probablement* ACID que si le niveau d'isolation est « sérialisable ».
- Pourquoi les autres niveaux existent-ils ?
  - …
- Pourquoi « *probablement* » ?
  - Parce que certaines instructions, mal utilisées, peuvent invalider ponctuellement les propriétés ACID.
  - Par exemple : la suspension des contraintes, les effets de bord permis par les pointeurs REF…

Fin

PROCHAIN MODULE
LA GESTION DE L'ACCÈS CONCURRENT