Administration des bases de données

Jean-Yves Antoine

http://www.info.univ-tours.fr/~antoine/

Administration des bases de données

IV – SGBD Transactionnels : protection et sécurité des données

OBJECTIFS

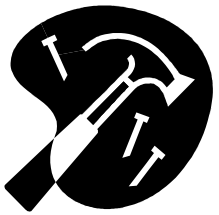
4.1. NOTIONS

4.1.1. SGBD transactionnel

4.1.2 Panne : protection et récupération des données

4.1.3. Gestion des accès concurrents

4.2. PRATIQUES



4.2.1. Gestion de transaction en SQL

4.2.2. Panne : journalisation sous Oracle 10g

4.2.3. Séquences Oracles

PANNES & TRANSACTIONS

Type de panne

- Panne d'action et de transaction : locale à la BD
- Globale au système
  - Panne système (*soft crash*): sans atteinte à la mémoire physique
  - Panne mémoire (*hard crash*) : problème sur la mémoire secondaire

Exemple : gros systèmes

Défaillance	Fréquence	Tps reprise
Locale	~ 10 / minute	instantané
Soft crash	2 à 3 / semaine	qques minutes
Hard crash	1 à 2 / an	qques heures

[Gray 81, Härder, Reuter 83]

Résistance aux pannes : mémoire sûre

Reprise sur panne

- SGBD restreint : pas de support à la reprise
- SGBD complet : reprise basée sur la notion de **transaction**

PANNE LOCALE & TRANSACTIONS

Problème

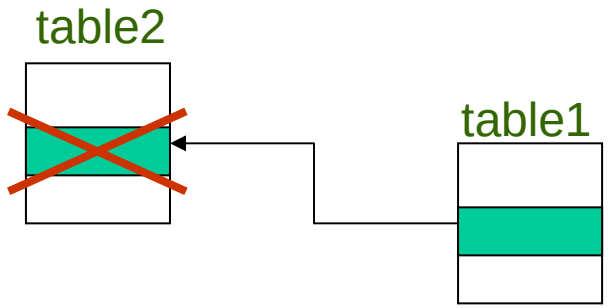
Plusieurs clauses SQL pour une seule action logique

Problème de cohérence si panne lors d'une de ces instructions

Exemple – Suppression manuelle d'un enregistrement en cascade

DELETE FROM table2 WHERE…

INSERT INTO table2 VALUES…



Transaction

- Unité **logique** de traitement correspondant à un ensemble d'actions
- Actions atomiques : validation, annulation et ré-exécution de transaction
- Unité de base pour la gestion des pannes et celle des accès concurrents

TRANSACTION

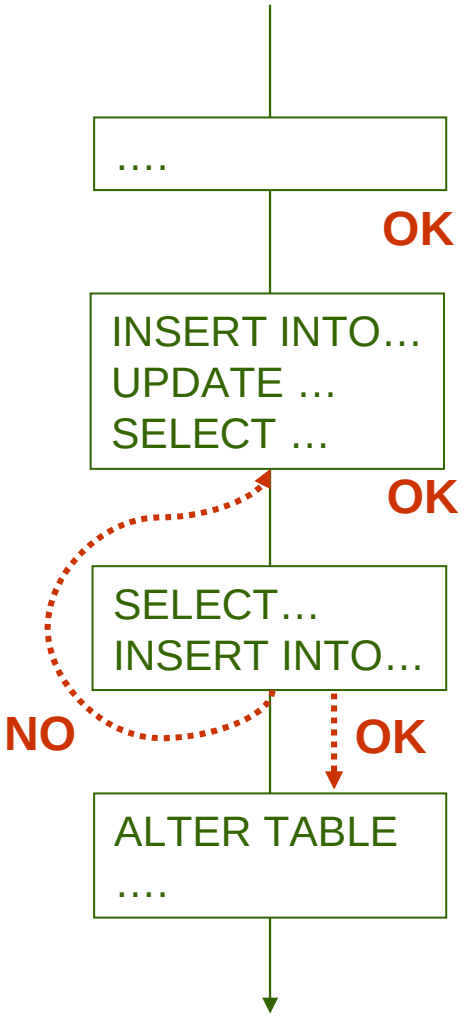
Transaction — Unité logique de traitement formée d'une suite d'actions limitée par une opération de validation ou d'annulation

Validation — Publication définitive des modifications (enregistrement physique)

Annulation / Invalidation — Retour de la BD à l'état en début de transaction

En cours de transaction : mémorisation des actions réalisées dans un **journal** (fichier de *log ou de redo-log*) avant et après

Accès concurrents : verrous en cours de transaction



TRANSACTION

Principes ACID

[Bjork, 1973]

Atomicité	Transactions atomiques (tout ou rien)
Cohérence	Les transactions maintiennent la cohérence de la BD
Isolation	Les Transactions, mêmes concurrentes, sont isolées les unes des autres
Durabilité	Après validation, mise à jour pérenne même si panne

TRANSACTION : SQL

Début de transaction implicite

Début de session ou fin de la transaction précédente

Fin de transaction implicite

Ordre LDD (CREATE, ALTER, DROP etc.), fin de session ou panne

Fin de transaction explicite

Validation COMMIT

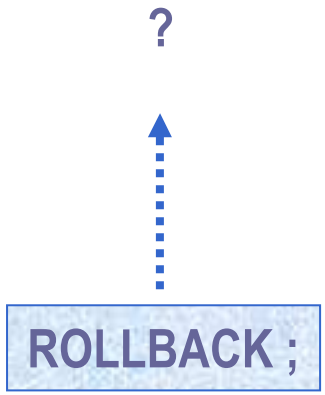
- Termine une transaction par la validation des actions effectuées
- Annulation des verrous et publication définitive des modifications effectuées, qui deviennent alors visibles aux autres utilisateurs

Annulation ROLLBACK

- Termine une transaction en annulant toutes les actions effectuées
- Annulation des verrous et publication définitive des modifications effectuées

TRANSACTION : SQL


```
DELETE FROM personne WHERE .... ;

INSERT INTO TABLE personne VALUES ... ;

COMMIT ;

UPDATE personne ... SET ;

ALTER TABLE societe ADD …. ;

UPDATE personne ... SET ;

DELETE FROM personne WHERE .... ;
```



Contraintes différées

Il est possible de différer la vérification d'une contraintes en fin de transaction

- **Deux statuts** : contrainte différable (DEFERRABLE) ou non
- **Deux états** (contrainte différable) : immédiate ou différée (DEFERRED)

Utilisation d'une contrainte différable

Attribution du caractère différable lors de la création + état initial

```
CREATE TABLE piece
(id NUMBER,
 elt NUMBER,
 CONSTRAINT fk_piece_elt FOREIGN KEY(elt) REFERENCES elt(id)
 DEFERRABLE INITIALLY DEFERRED );
```

INITIALLY IMMEDIATE

Changement d'état dans une transaction ou toute une session

```
SET     { CONSTRAINT | CONSTRAINTS} { ctr1 [, ctr2…] | ALL}
        { IMMEDIATE | DEFERRED }
```

```
ALTER SESSION SET CONSTRAINTS = { IMMEDIATE | DEFERRED }
```

ORACLE : PROPRIETES DES TRANSACTIONS

Associer des propriétés à une transaction

Valable uniquement si aucune autre transaction en cours

```
SET TRANSACTION [mode_acces] [niveau_isolation]
```

Niveau d'isolation cf. partie verrouillage

```
SERIALIZABLE | REPEATABLE READ | READ COMMITED | READ UNCOMMITED
```

Mode d'accès

```
READ ONLY | READ WRITE
```

READ WRITE incompatible avec READ UNCOMMITED

Sous-transaction

- **positionnement d'un point intermédiaire dans la transaction**

```
SAVEPOINT <point_repere> ;
```

- **annulation des actions depuis ce point de repère** (et non depuis le début)

```
ROLLBACK TO <point_repere> ;
```

PANNES ET TRANSACTION : JOURNALISATION

Journalisation

- **Performance** : modifications mémorisées en cache (volatil) et recopie sur disque à intervalles prédéfinis.
- **Sécurité** : cache perdu lors des pannes systèmes
- **Journalisation** : mémorisation des informations qui ont été modifiées en cache mais pas encore reportées sur disque

Journal
Id_transaction Time Fichier / Page Valeur avant Valeur après

Journal des images avant

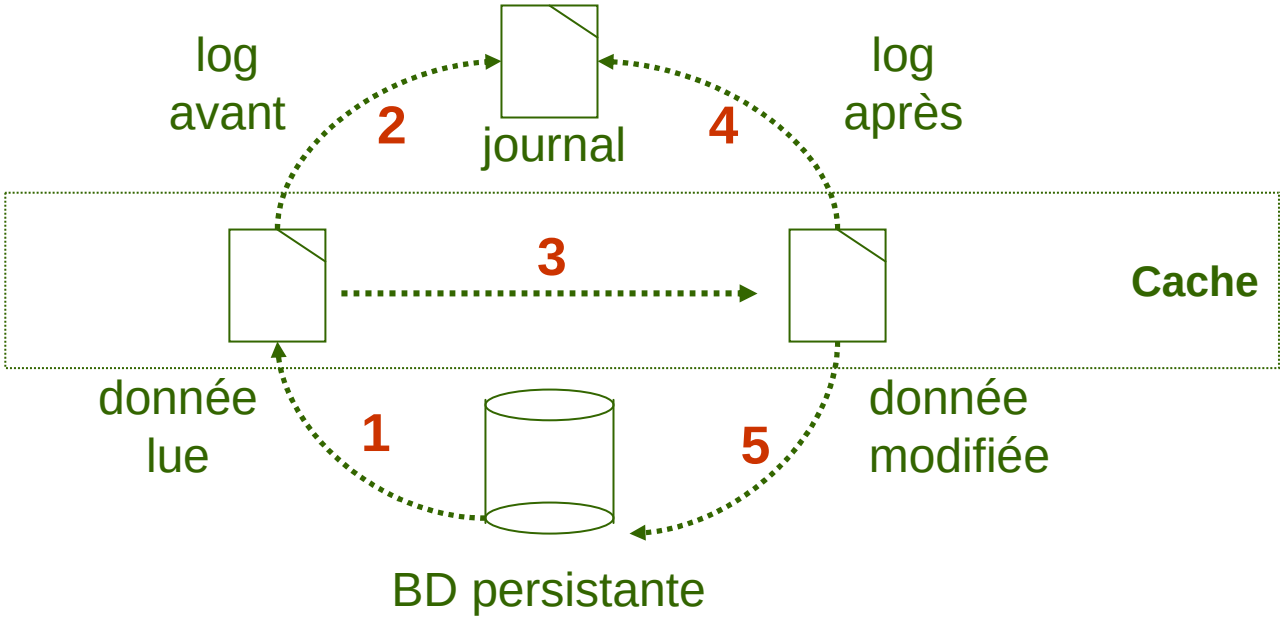
- Valeurs des données mises à jour et pas encore sur écrites sur disque, **avant modification**
- **Protection des données** : utilisation pour défaire une transaction (***undo log***)

Journal des images après

- Valeurs des données mises à jour et pas encore sur écrites sur disque, **après modification**
- **Protection des données** : utilisation pour rejouer une transaction (***redo log***)

PANNES ET TRANSACTION : JOURNALISATION

⇨ **Protection des données** : journalisation **avant** action



⇨ **Performance** : journal également en mémoire volatile (cache)

⇨ **Protection des données**

validation ≠ sauvegarde automatique sur DD

validation après écriture

Fichiers de journalisation

- **Fichiers de redo-log** (`.rdo` ou `.log`) : journalisation avant et après

  Peuvent être multiplexés pour limiter les risques sur panne mémoire

- **Dimensionnement du journal** : tablespace UNDO
- **Restauration de la base** : suppression ou archivage (ARCHIVELOG MODE) du fichier une fois utilisé pour rejouer les transactions
- **Manipulation** : outil *Log Miner*

Vue	Rôle
V$LOGFILE	Description des fichiers de journalisation
V$LOG_HISTORY	Informations sur l'historique de tous les redos
V$LOG	Information sur chaque groupe de log

Gestion des tablespaces UNDO

- **Tablespace** : partition logique de la mémoire
- **Tablespace UNDO** : partition spécifiquement attribuée au journal.
- **Dimensionnement** : importance du dimensionnement du journal : étude des vues dynamiques du dictionnaire de données + outil *Undo Advisor*

Vue	Rôle
DBA_TABLESPACES	Liste des TABLESPACES. Champ CONTENT : type (UNDO…)
DBA_UNDO_EXTENTS	Caractéristiques des UNDO segments
V$UNDOSTATS	% d'utilisation du TABLESPACE

REPRISE APRES PANNE

Panne locale (problème de cohérence de la base)

Gestion immédiate : ROLLBACK explicite ou implicite

Reprise à froid : *hard crash*

- support mémoire physique endommagé
- recopie d'une sauvegarde antérieure (*dump*)
- si copie physique du journal non atteinte : on rejoue les actions du journal

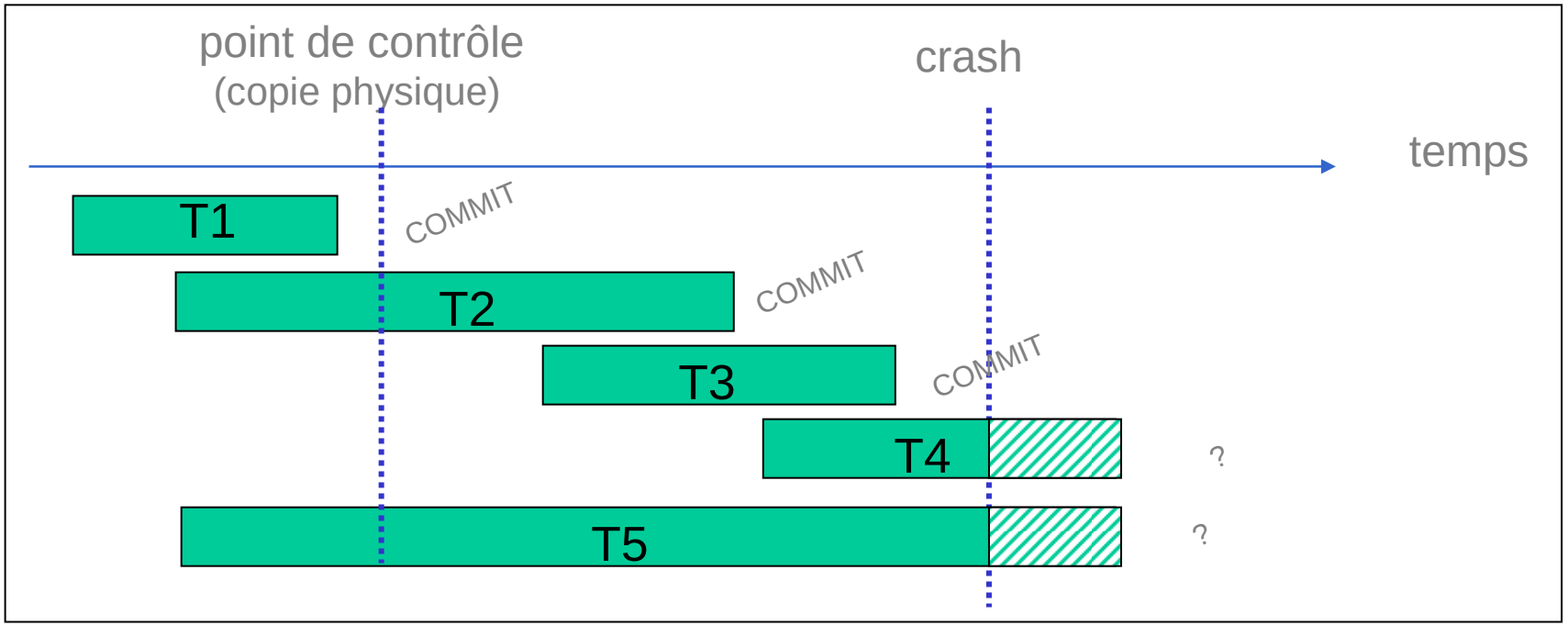
Reprise à chaud : *soft crash*

1. Recherche dans le journal du dernier point de reprise système (copie physique)
2. Ré-exécution des transactions mémorisées après le point de reprise (*redo log*)
3. Annulation des transactions journalisées mais non encore validées (*undo log*)

Oracle : gestion automatique à l'aide des segments d'annulation (ROLLBACK segments / UNDO segments)

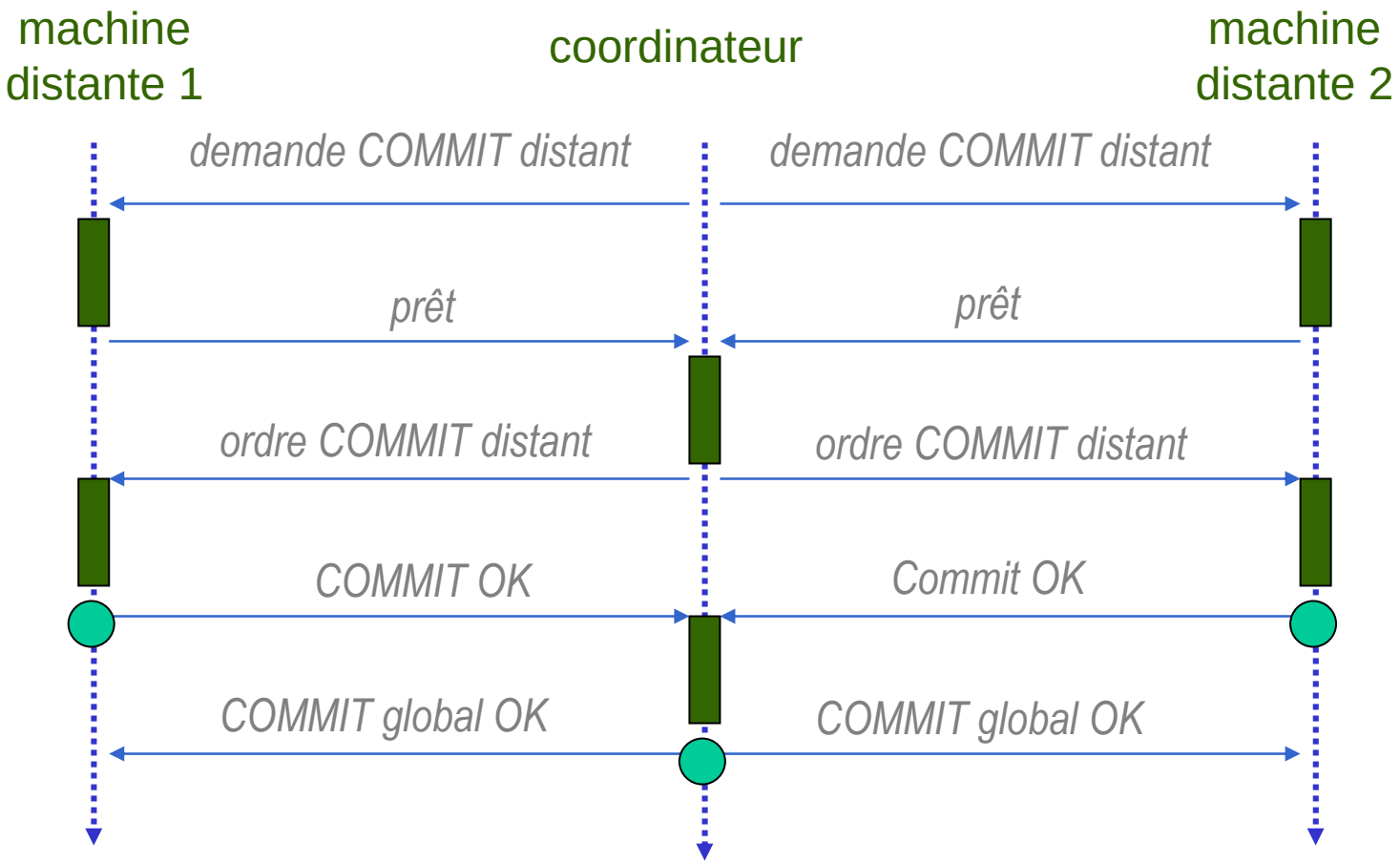
REPRISE A CHAUD : SOFT CRASH

Validation logique et sauvegarde physique

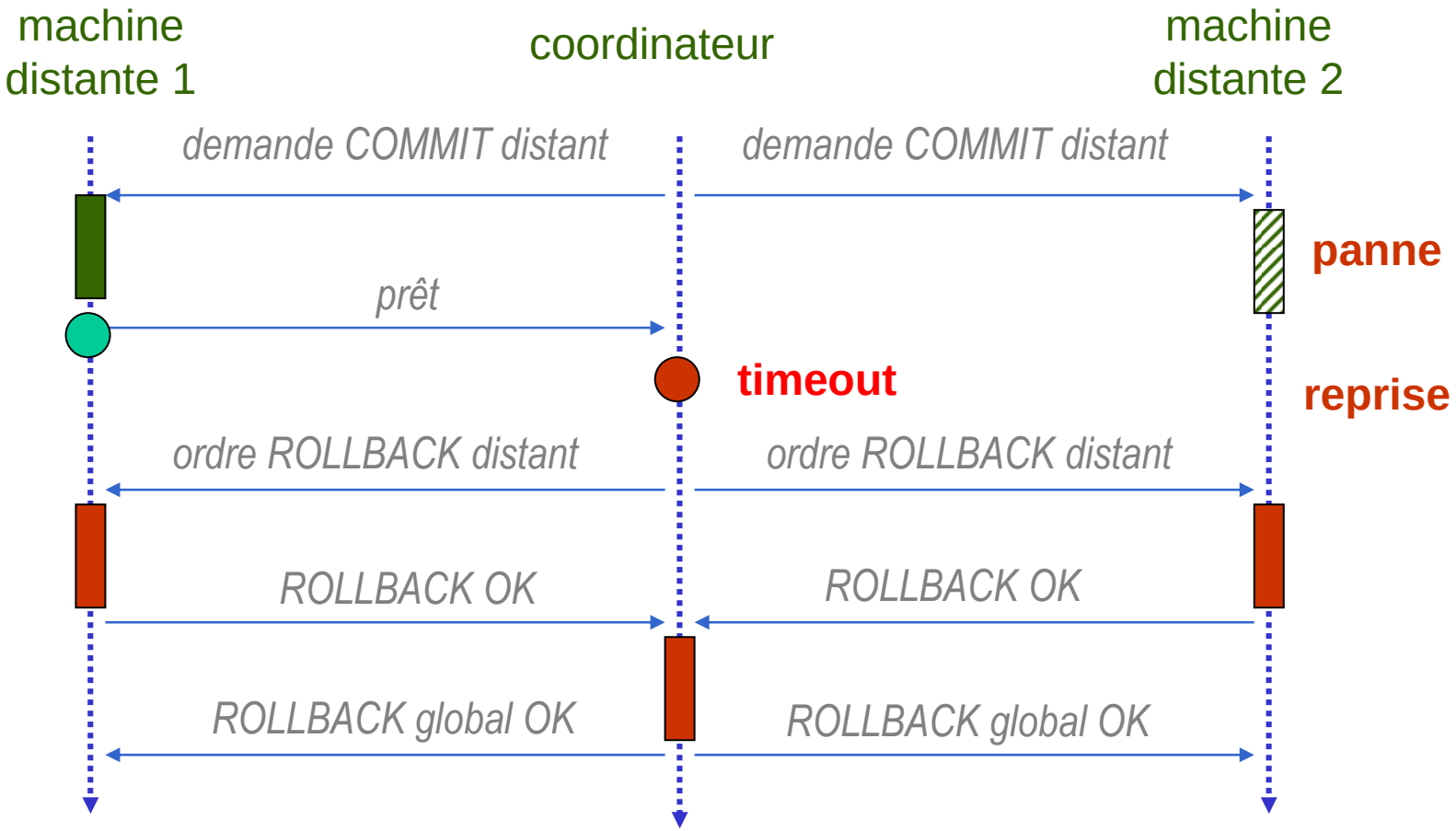


Action de restauration	Transaction
Aucune	T1
Annulation (Rollback)	T4, T5
Rejouer le transaction	T2, T3

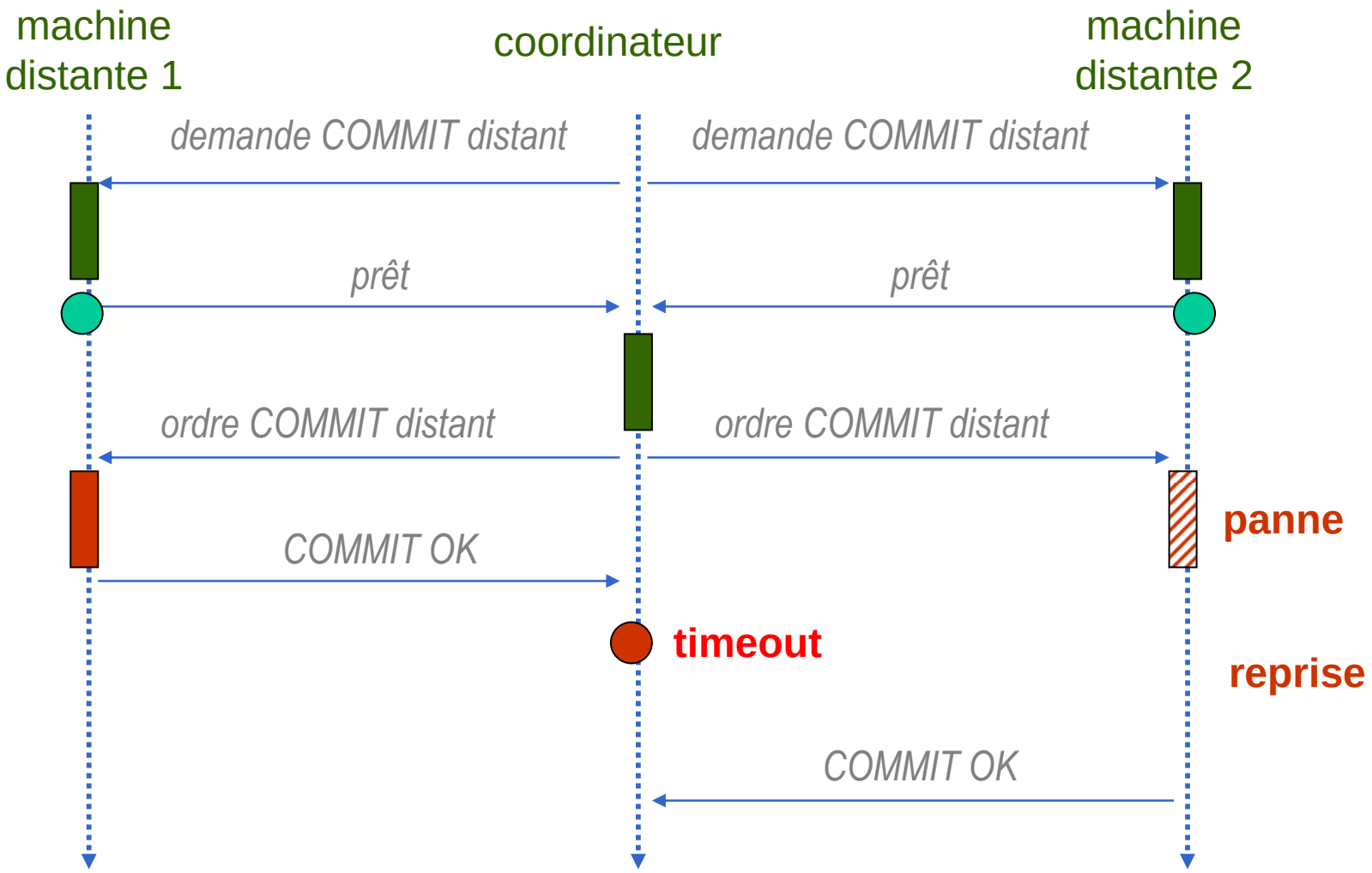
Validation en deux étapes Double COMMIT



Panne sur demande de préparation



Panne sur validation distante

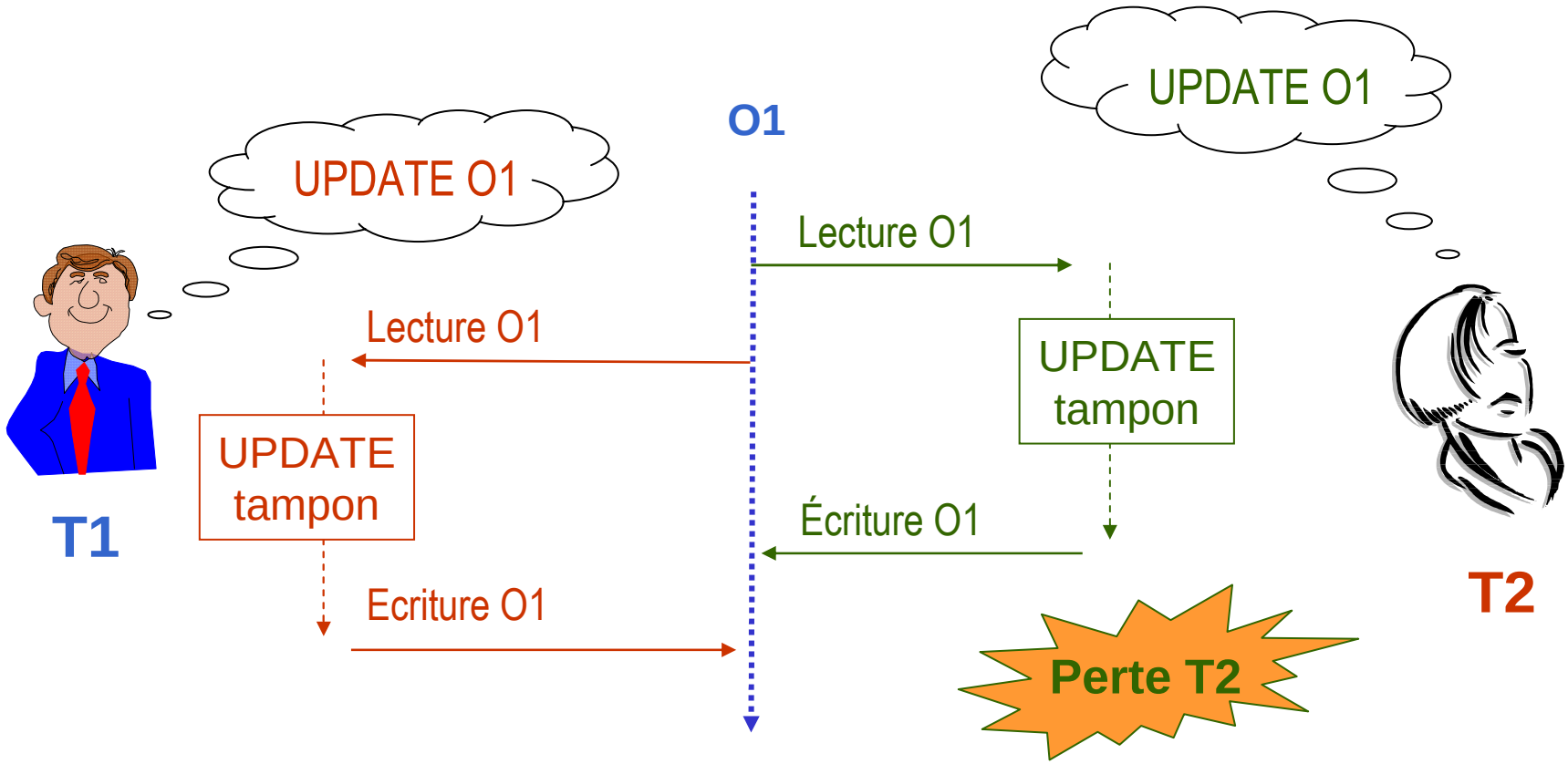


CONCURRENCE : PROBLEME

Accès concurrents

- Plusieurs utilisateurs ou applications accèdent/modifient aux mêmes données.
- Problèmes sur les valeurs de lectures et sur le maintien de la cohérence.

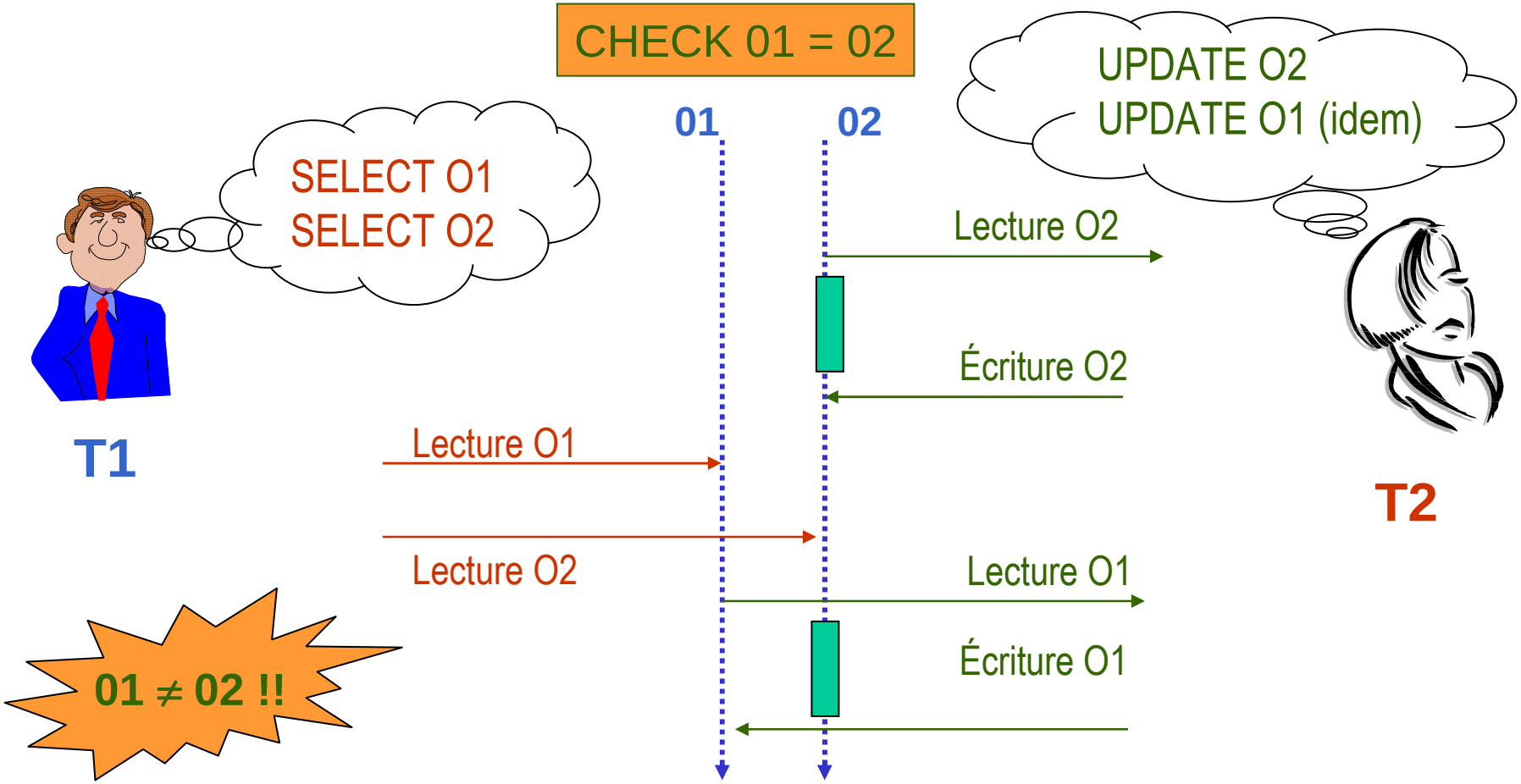
Perte d'opération



CONCURRENCE : PROBLEME

Lecture fantôme

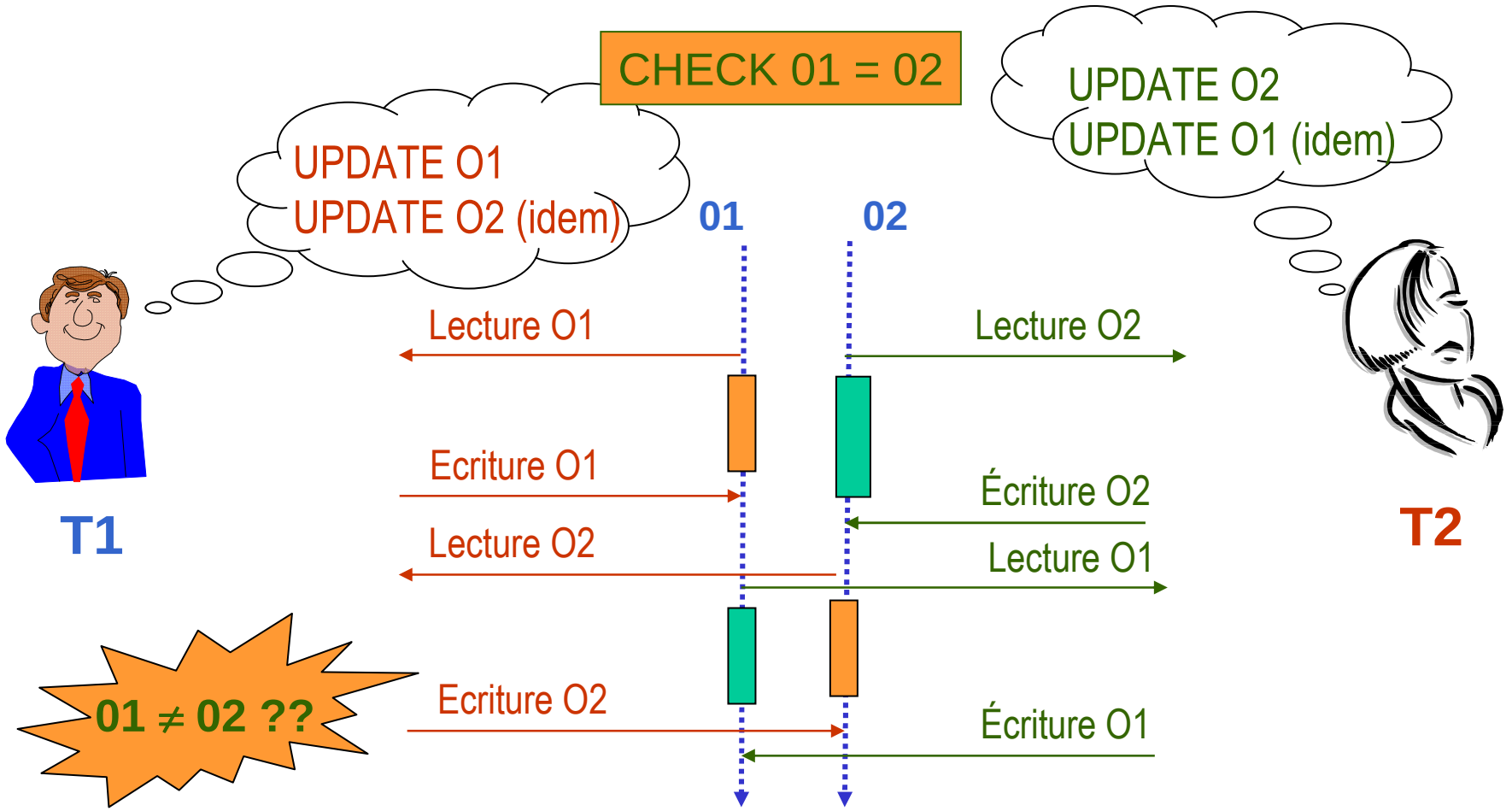
Incohérence apparente : laisse croire au non respect des contraintes



CONCURRENCE : PROBLEME

Incohérence de la base

Lecture fantôme conduisant à des inconsistances permanentes



CONCURRENCE : SECURISATION

Principes

- Contrôle des accès concurrents sur les données en cours de m.a.j.
- **Transaction** : grain temporel minimal de gestion des accès (A**C**ID)
- **Granularité** : Unité de base dont l'accès peut-être contrôlé variable suivant la transaction : champ, tuple, relation…
- **Objectif** : **sérialisation** de l'exécution des transactions.

Sérialisation

- **Schéma sériel**: schéma d'exécution où les transactions sont exécutées à la suite les unes des autres : pas de concurrence.
- **Schéma sérialisable** : un schéma de transactions concurrentes $T_1,\ldots,T_n$ est sérialisable si il existe un schéma sériel de transactions qui produit le même résultat quelque soit l'état initial de la base
- **Schéma sérialisable par permutation** : schéma sérialisable où on peut permuter les actions deux à deux pour avoir un schéma sériel

CONCURRENCE : SECURISATION

Opérations

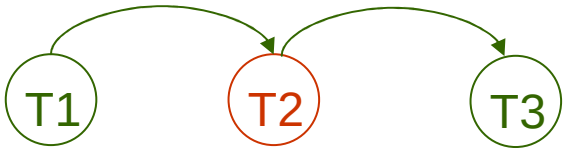
Compatibles: exécutables en parallèle sans problème

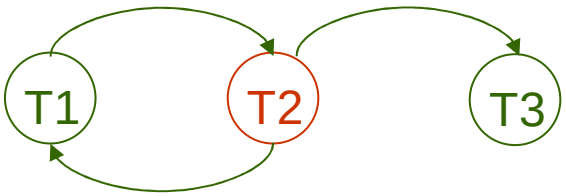
Permutables: ordre d'exécution sans influence sur le résultat

Opérations de T1 et T2	compatible	permutable
Lecture/Ecriture A et Lecture/Ecriture B	oui	oui
Lecture A et Ecriture A	non	non
Ecriture A et Ecriture A	non	non

Graphe de précédence d'un schéma d'exécution

Définition: il y a précédence d'une transaction T1 sur une transaction T2 si il existe au moins une opération O1 de T1 non permutable avec une opération O2 de T2 et qui est avant O2 dans le schéma d'exécution





Condition suffisante de sérialisation: un schéma d'exécution est sérialisable si le graphe de précédence qui lui est associé est sans circuit.

GESITON DE CONCURRENCE PAR ESTAMPILLAGE

Sérialisation

- approche optimiste : contrôle par **estampillage** ou **validation**
- approche prudente : **verrouillage**

Estampillage

– Numéro d'ordre attribué aux transactions

– Traces sur les éléments de la BD lors de toute lecture / écriture :

- LL(E) numéro de la dernière transaction à avoir lu l'élément E
- LE(E) numéro de la dernière transaction à avoir écrit / modifié E

– **Algorithme d'ordonnancement**: T_i veut opérer sur une donnée E

- si i ≥ LL(E) (resp. LE(E)) alors on peut exécuter i (transaction bien postérieure)
- sinon, on annule et reprend la transaction d'estampille LL(E) (resp. LE(E)).

– **Approche optimiste** : on suppose que tout se passe bien a priori. Si ordonnancement non sérialisable, il faut alors différer ou défaire les transactions

GESITON DE CONCURRENCE PAR VEROUILLAGE

Verrouillage : principes

- Une transaction peut poser et libérer des verrous sur les données sur lesquelles elle travaille (lecture, mise à jour, écriture…).
- Une transaction ne peut réaliser une opération sur une donnée que s'il n'y a pas de verrou posé dessus, où si la transaction possède le verrou en question. Sinon, elle est mise en attente au niveau de l'opération en cours
- **Approche prudente** : pas d'annulation de transaction en cours, sauf si verrou mortel

Stratégies de verrouillage

- **Verrous uniques stricts** : verrou sur une donnée dès qu'on l'utilise
- **Verrous distincts** non nécessairement exclusifs : verrous différents suivant l'action
  - lecture (verrou partagé) / écriture (exclusif)
  - lecture, mise à jour protégés ou non, exclusive ou non , etc…

↳ Niveaux d'isolation des transactions

↳ Matrice de compatibilité des opérations

lecture / écriture

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Accès concurrents : verrouillage

Algorithme de verrouillage (cas général : verrous distincts)

- Avant chaque opération, la transaction vérifie si elle peut poser le verrou (exclusif ou non) qui correspond à cette dernière. Si non, attente.
- En pratique :
  - ⇒ Vecteur O(i,e) des opérations de la transaction T_i en cours sur l'élément e
  - ⇒ Demande LOCK : vecteur booléen $M_j(e)$ des modes de verrouillage demandés
  - ⇒Demande UNLOCK: remise à zéro du mode correspondant dans O(i,e)
  - ⇒ Verrouillage autorisé si :

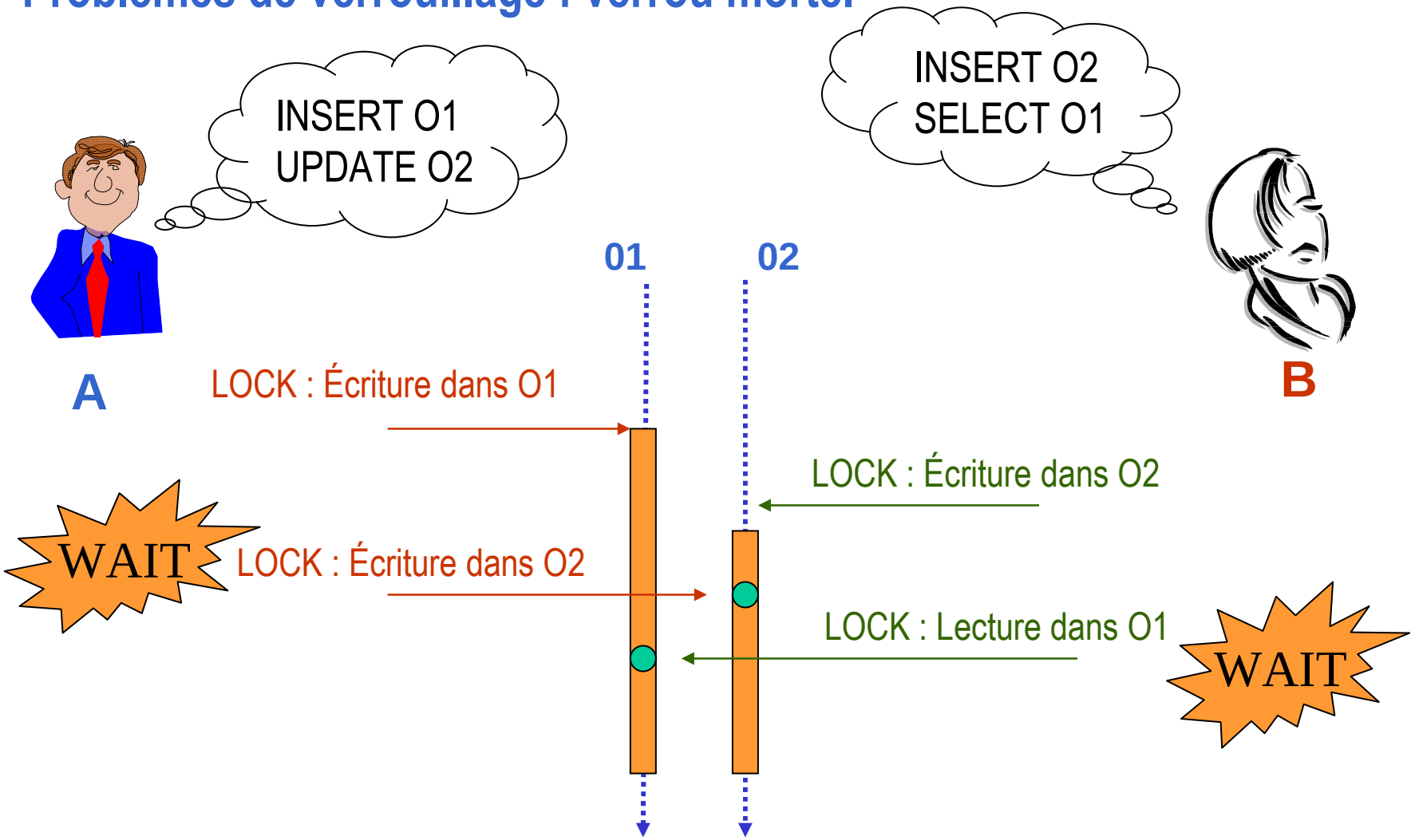
$$M_j(e) \subset \neg(\neg C * \Sigma_{Boole}^{i \neq j} O(i,e))$$

Protocole de verrouillage en deux phases (2PL)

- **2PL** : dans la transaction, tous les verrouillages précèdent les libérations de verrou.
- **Intérêt** : condition suffisante pour assure des schémas d'exécution **sérialisables**.

Accès concurrents : verrouillage

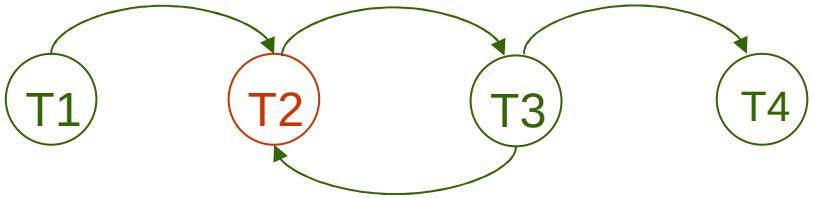
Problèmes de verrouillage : verrou mortel



Accès concurrents : verrouillage

Détection des interblocages: graphe des attentes

- **Graphe des attentes**: une transaction T_i attend une transaction T_j si T_i attend de pouvoir verrouiller une donnée *d* alors que le verrou correspondant appartient à T_j.
- **Interblocage (verrou mortel)** ssi le graphe des attentes possède un circuit



Résolution des interblocages

- **Prévention**: pré-ordonnancement des transactions
- **Correction**: algorithme de déblocage implicite

Trop lourd

  - **Stratégie DIE_WAIT** : T_i attend T_j uniquement si T_i est plus vieille que T_j . Sinon, T_i est annulée (reprise).
  - **Stratégie WOUND_WAIT** : T_i attend T_j uniquement si T_i est plus jeune que T_j . Sinon, T_i tue T_j (reprise provoquée).
  - **Stratégies plus souples** : on blesse (ultimatum d'une certaine durée) avant de tuer

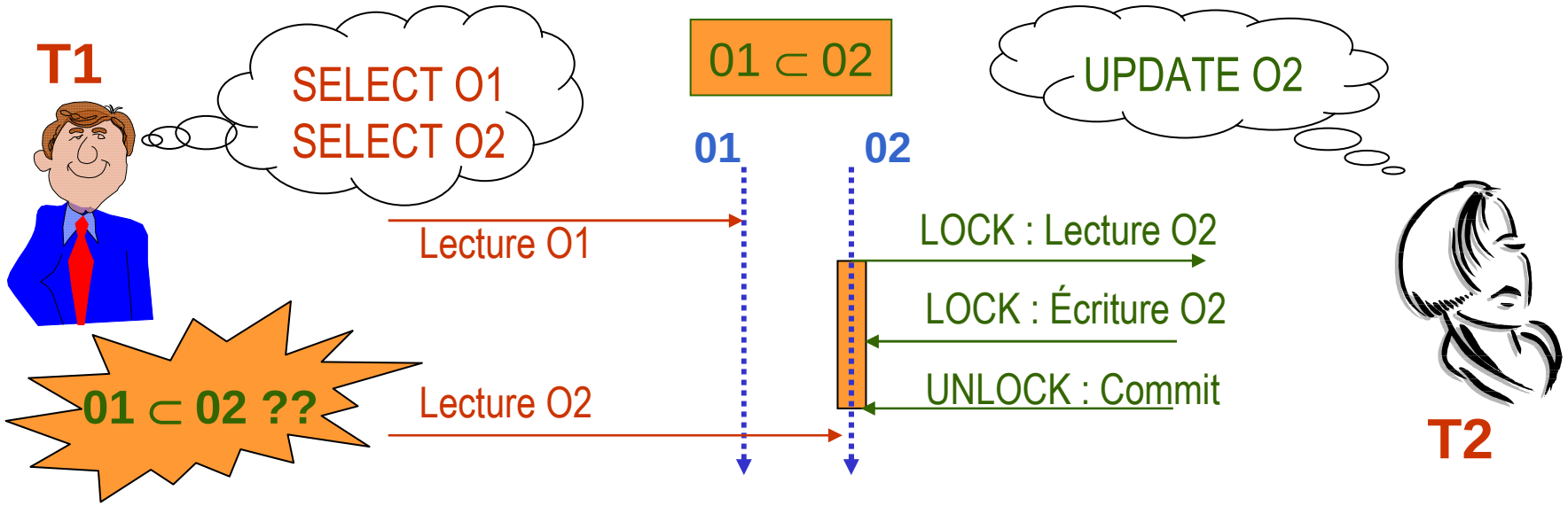
Accès concurrents: verrouillage

Problèmes de verrouillage : blocage permanent

- **Problème**: ensemble de transactions effectuant des actions compatibles (pas de verrou mortel) mais bloquant *de facto* les autres transactions.
- **Solutions** - file d'attente des demandes de verrouillage
  - stratégies DIE_WAIT / WOUND_WAIT: pas de blocage permanent

Problèmes de verrouillage : lecture fantôme

- Toujours possible : difficulté de définir des granules logiques communs

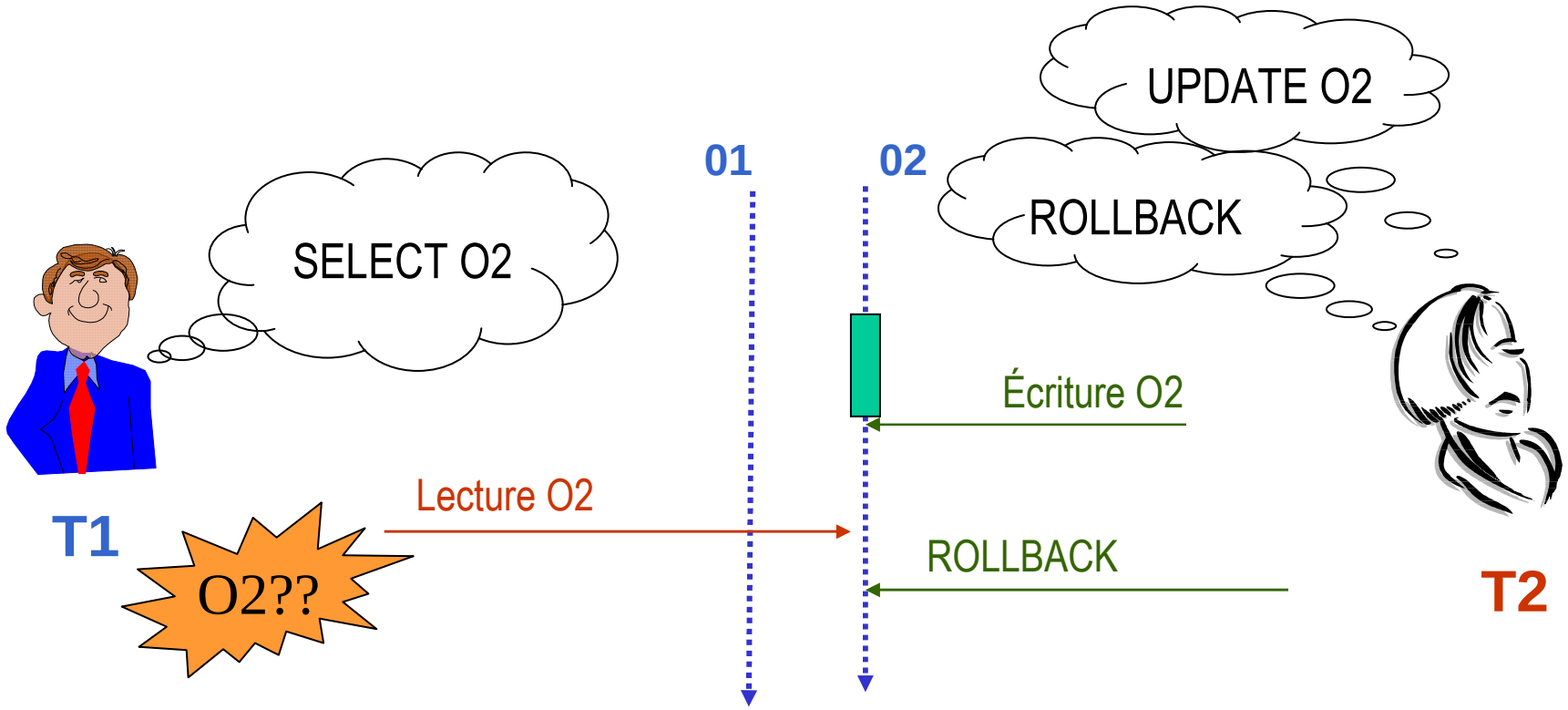


Accès concurrents : verrouillage

- **Niveaux d'isolations variables**
  - ➔ 2PL : assure la sériabilité mais restreint les exécutions concurrentes
  - ➔ Protocoles plus permissifs … plus de concurrence mais dangers associés
  - ➔ Mode d'opérations autorisés et **durée des verrous** :
    - verrou court — libéré après l'exécution de l'opération concernée
    - verrou long — libéré en fin de transaction (2PL maximal)
- **SQL2 : Niveaux d'isolation normalisés**
  - **Niveau 0** verrou exclusif court en écriture uniquement
    Garantie : mises à jour assurées uniquement
  - **Niveau 1** verrou exclusif long en écriture uniquement
    Garantie : non perte et cohérence des mises à jour
  - **Niveau 2** Niveau 1 + verrous courts partagés en lecture
    Garantie : niveau 1 + cohérence des lectures
  - **Niveau 3** **2PL** : verrous longs écriture exclusive / lecture partagée
    Niveau 2 + lecture renouvelable

Accès concurrents : verrouillage

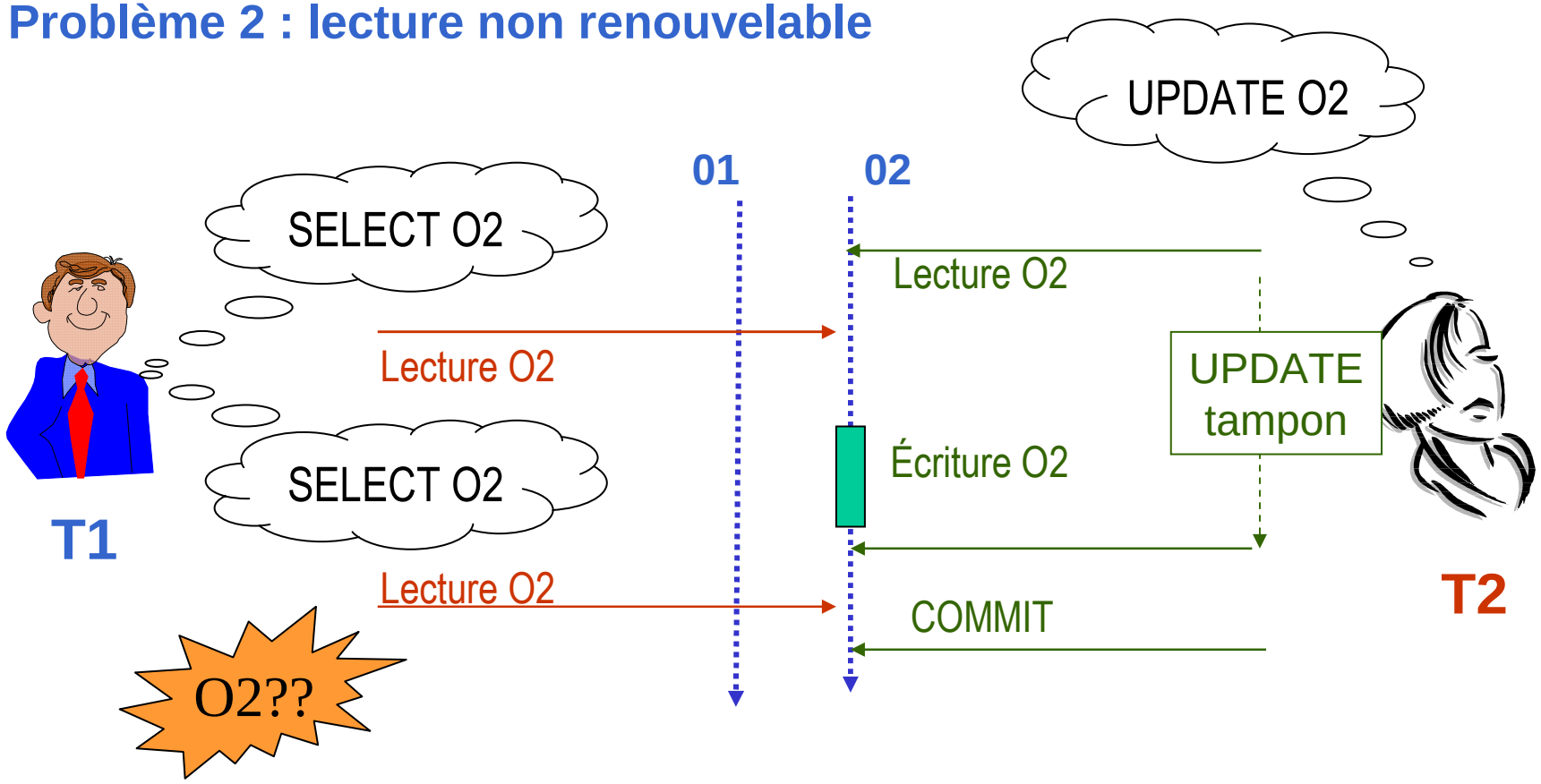
- **Niveaux d'isolations variables** ➔ problèmes éventuels
- **Problème 1 : lecture salissante**



Verrou court : UNLOCK avant COMMIT

Accès concurrents : verrouillage

Problème 2 : lecture non renouvelable

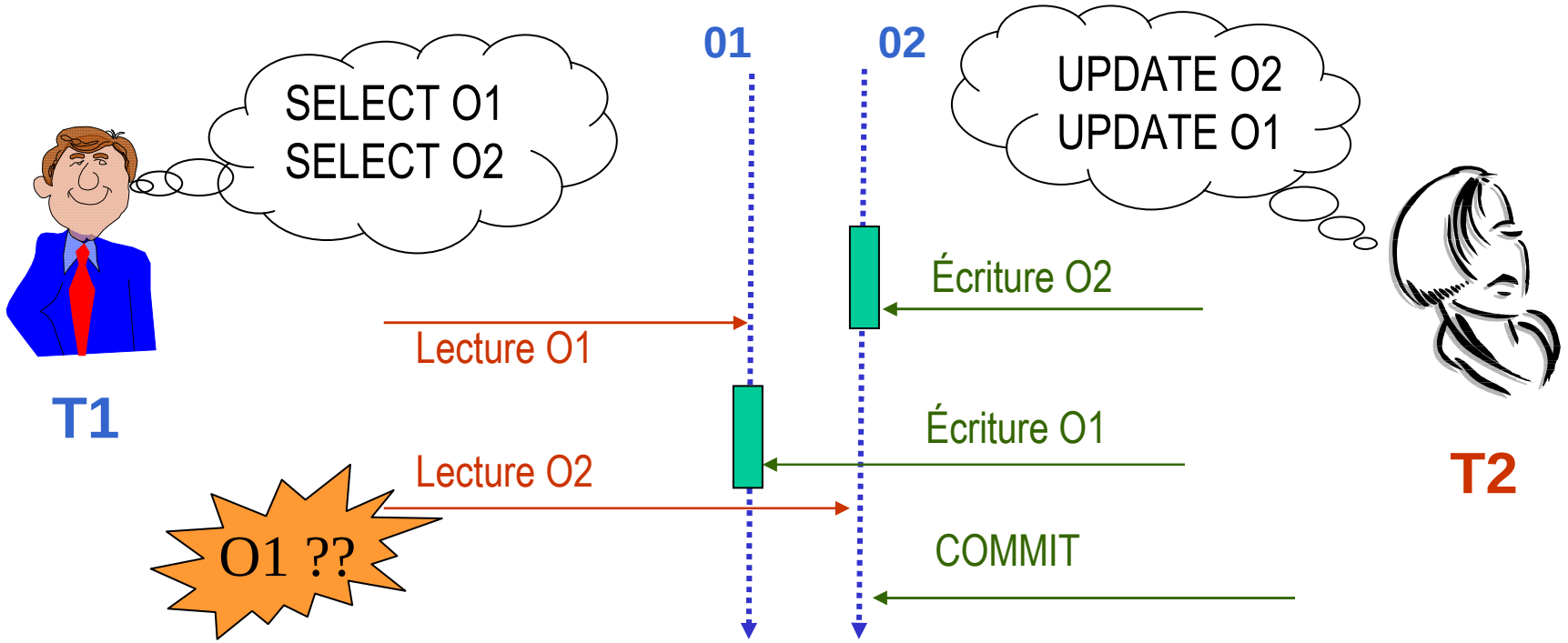


Verrou court : UNLOCK avant COMMIT

Accès concurrents : verrouillage

Problème 3 : lecture fantôme

Rappel : au mieux, on ne peut que limiter les situations de lectures fantômes



Verrou court lecture / écriture

Accès concurrents : Oracle

Verrouillage implicite des données

- Gestion transparente: automatique, suivant le niveau d'isolation choisi
- Plusieurs niveaux d'isolation : contrôle plus ou moins strict de la concurrence
  SERIALIZABLE | REPEATABLE READ | READ COMMITED | READ UNCOMMITED
- **Protocole 2PL** : déverrouillage final sur COMMIT (si niveau SERIALIZABLE)
- **Granularité fine** : on peut ne verrouiller qu'un tuple, un attribut
- Dictionnaire Oracle : **V$LOCK**

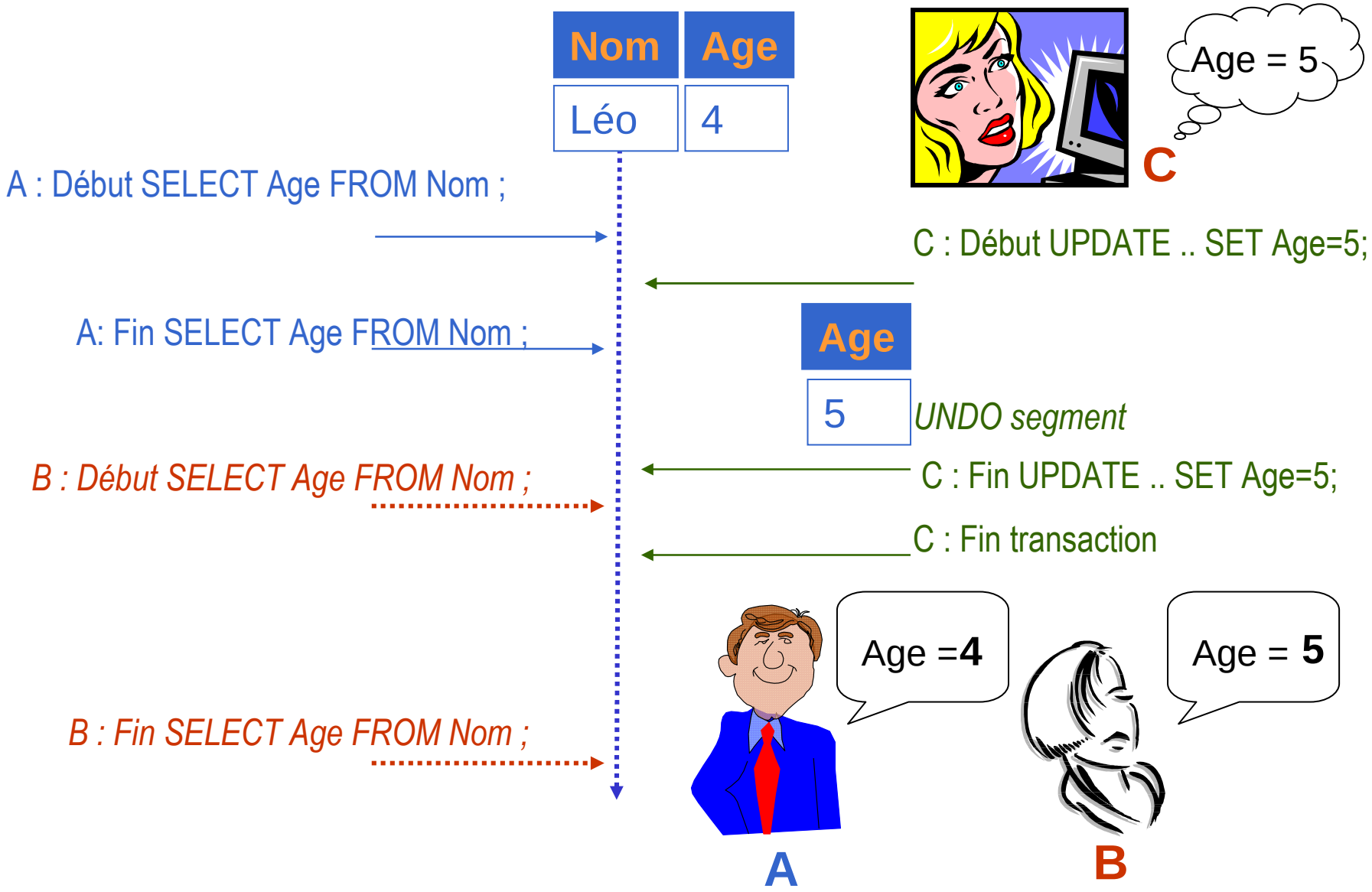
Mise à jour des données (INSERT, UPDATE et DELETE)

- Niveau SERIALIZABLE: verrou sur le tuple (ROW) de la table concernée.
- Création d'un *ROLLBACK segment* : modifications visibles du seul utilisateur responsable de la transaction jusqu'à la fin de transaction.

Consultation de données (SELECT)

- Niveau SERIALIZABLE : verrou lecture protégée.
- Ne peut pas voir les mises à jour non encore publiées par les autres transactions.

Accès concurrents : Oracle



Accès concurrents : Oracle

Niveaux d'isolation

Niveau d'isolation	Lecture salissante	Lecture non renouvelable	Fantôme
Lecture non validée	☹	☹	☹
Lecture validée	✓	☹	☹
Lecture renouvelable	✓	✓	☹
Sérialisable	✓	✓	

SQL : Définir un niveau d'isolation

- SERIALIZABLE par défaut
- Définition par transaction **SET TRANSACTION** [niveau_isolation]

SERIALIZABLE | REPEATABLE READ | READ COMMITED | READ UNCOMMITED

Accès Concurrents : Oracle

Verrouillage explicite

sur table ou enregistrements

- **Verrouillage de table**

```
LOCK TABLE <nom_table> IN {EXCLUSIVE | SHARED } MODE
```

  - **Mode exclusif** (X) : table totalement verrouillée sauf en consultation (SELECT).
  - **Mode partagé** (S) : d'autres transactions peuvent mettre un verrou partagé sur la table. Si plusieurs verrous S, impossibilité de modifier les données par aucune des transactions.

- **Verrouillage de tuple** (ROW)

```
LOCK TABLE <nom_table>  IN {ROW EXCLUSIVE | ROW
SHARED | SHARE ROW EXCLUSIVE} MODE
```

  - Si verrou partagé uniquement sur les enregistrements (RS), modifications parallèles possible sur enregistrements différents.

Séquences

Principe: définition de valeurs unique et accès concurrents

- Objet nommé dans la base de données qui sert à générer des valeurs (entiers) uniques dans un environnement d 'utilisation multi-utilisateurs, sans conflit et sans risque de verrous mortels
- Une séquence est un objet spécifique qui peut être utilisé dans plusieurs tables et par plusieurs utilisateurs.
- Un pas d'incrément permet de définir le calcul de la valeur courante à partir de la valeur précédente. Les valeurs sont comprises entre des valeurs maximales et minimales (-10^{26} et 10^{27} par défaut).
- Une fois l'ensemble des valeurs parcouru, on peut autoriser le retour à la valeur initiale pour poursuivre la génération à l'aide de l'option CYCLE

Exemple d'utilisation : clé primaire

Générer les valeurs de la clé sur une importation si aucun champ de données ne joue ce rôle (équivalent à `NumeroAuto` en ACCESS ou `Auto Increment` en MySQL)

Séquences

Création

CREATE SEQUENCE [<schema>.]<nom_seq>
[INCREMENT BY <entier>]
[START WITH <entier>]
[MAXVALUE entier] [MINVALUE entier]
[{CYCLE | NOCYCLE}]
[{CACHE <entier> | NOCACHE}] ;

Modification

Généralement pour changer l'amplitude de variation ou le pas d'incrément

ALTER SEQUENCE [<schema>.]<nom_seq> + un des éléments de création

Suppression

DROP SEQUENCE <nom_sequence> ;

Séquences

- **Utilisation**
  - Utilisation dans toute commande SQL de manipulation des données (SELECT, INSERT, UPDATE) avec quelques restrictions
  - **<nom_sequence>.NEXTVAL** : directive (pseudo-attribut) générant la première ou la prochaine valeur de la séquence et retourne la valeur de celle-ci (lecture et écriture)
  - **<nom_sequence>.CURVAL** : valeur courante de la séquence (lecture seule)
- **Exemple**

```
CREATE SEQUENCE seq_exple NOCYCLE ;
SELECT seq_exple.NEXTVAL FROM seq_exple ;        /* 1ère utilisation */
...
SELECT seq_exple.CURVAL FROM seq_exple ;         /* lecture 1ère valeur */
...
SELECT seq_exple.CURVAL FROM seq_exple ;
```

Séquences : Oracle

Visualisation de l'état d'une séquence

Lecture (SELECT) du pseudo-attribut `<nom_seq>.CURVAL` dans la pseudo-table **DUAL**

Dictionnaire Oracle : vue ALL_SEQUENCES

Nom	NULL ?	Type
SEQUENCE_OWNER	NOT NULL	VARCHAR2(30)
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

Séquences

Utilisation explicite de séquences avec SQL Loader

1. Créer une séquence

```
CREATE SEQUENCE seq_exple START WITH 1 INCREMENT 1 BY NOCYCLE ;
```

2. Utiliser la séquence dans le fichier de contrôle

```
LOAD DATA INFILE 'donnees.don'
INTO TABLE Table_exemple
FIELDS TERMINATED BY ';'
(pk_att "seq_exple.nextval",
NOM ….
```

Utilisation implicite de séquences avec SQL Loader

```
LOAD DATA INFILE 'donnees.don' INTO TABLE Table_exemple
FIELDS TERMINATED BY ';'
(pk_att SEQUENCE(Valeur_Init, Increment),
…
```

Bibliographie

Ouvrages de références

- DATE C. J. (2000) *Introduction aux bases de données* (7° édition), Vuibert, Paris, ISBN 2-7117-8664-1. Partie IV : gestion de la concurrence, pp. 443-496.
- GRAY J., ANDREAS R. (1993) Transaction processing : concepts and techniques. Morgan Kaufmann, San Mateo, CA.

Travaux cités

- GRAY J.N. *et al.* (1981) The recovery manager of the system R Database Manager. *ACM Computing Surveys*, 13(2).
- HÄRDER T., REUTER A. (1983) Principles of transaction-oriented database recovery. *ACM Comput. Surv*. 15(4).
- MOHON C. et al. (1992) A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM TODS* 17(1).