BASES DE DONNÉES
COURS 4
TRANSACTIONS / SÉCURITÉ

Mickaël Coustaty

Jean-Loup Guillaume

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

OBJECTIFS DU COURS 4

Comprendre la gestion des droits et la gestion des accès concurrents

- Rôles
- Transactions
- Verrous

PRÉREQUIS ET LIENS AVEC D'AUTRES COURS

Prérequis :

- Requêtes de base

Liens avec d'autres futurs cours :

- Indexation
- Optimisation
- Bases de données réparties

SOMMAIRE

Contrôle des données et des accès

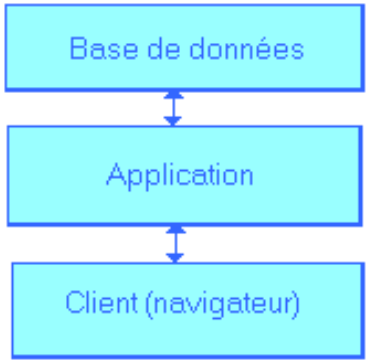
- Définition de rôles et gestion des droits

- Transactions
  - Concurrence
  - Verrous
  - Estampillage

CONSIDÉRATIONS GÉNÉRALES

SGBD

- Souvent au centre des systèmes d'informations
  - Ressources partagées
  - Multi-utilisateurs



Devoir du SGBD

- Maintenir la structure et l'intégrité des données
- Garantir l'accès aux données en un temps minimum
- Gérer l'espace occupé de manière optimale
- Protéger les données des effets des accidents de toute nature
- Autoriser les accès concurrents et les modifications parallèles
- Contrôler les accès selon les autorisations

MODÈLES DE CONTRÔLE D'ACCÈS

Une politique de contrôle d'accès sont des directives (règles) qui spécifient qui a la permission d'exercer quoi sur quelle donnée

Les trois entités fondamentales d'une politique de contrôle d'accès sont

- Sujet : entité active qui accède aux données du système (utilisateur, application, @IP ...)
- Objet : entité passive qui représente les données à protéger (fichier, table relationnelle, classe ...)
- Action : représente l'action à traiter par le sujet sur l'objet (lire, écrire, exécuter)

MATRICES DE CONTRÔLE D'ACCÈS (ACM)

Modèle le plus traditionnel et encore souvent utilisé

Utilise simplement des listes de sujets avec indication de tous les permissions de contrôle d'accès

- Lignes de la matrice = listes de permissions ou capacités (capabilities)
- Pour chaque sujet est indiqué ses permissions
- Les colonnes de la matrice sont les listes de contrôle d'accès

Les permissions sont donnés et modifiés par les administrateurs de la sécurité de l'organisation, suivant la politique de l'organisation

	Fichier Salaires	Fichier Impôts	Programmes impôts	Imprimante P1
Alice	Lire, Écrire		Exécuter	Écrire
Bob		Lire		
Jean	Lire			Écrire

CRITIQUES D'ACM

\+ Permet de spécifier un contrôle d'accès très détaillé

– Difficile à gérer, car les usagers et les objets doivent être considérés individuellement

- Surtout dans les grandes organisations
  - Milliers de sujets et d'objets

Exemple : Chaque fois qu'un étudiant arrive à l'Université, il faut s'assurer de lui enlever et ajouter individuellement toute une série de permissions d'accès à des cours

- Lesquelles exactement?

– Problème de garder cohérentes les listes des permissions et d'accès

MODÈLE DE CONTRÔLE D'ACCÈS DISCRÉTIONNAIRE (DAC)

Extension du modèle ACM

- Utilise les matrices de contrôle d'accès

Chaque objet a un « propriétaire »

- Propriétaire détermine qui a quelles permissions sur « ses objets »
- Créateur d'un objet en est le propriétaire

Possibilité de transférer les droits d'accès

- Ou même la propriété

CRITIQUES DE DAC

\+ DAC est très flexible du point de vue des propriétaires des données

– Mais il ne les protège pas par rapport à des transferts imprévus

- Je transfère à Alice qui transfère à Ben, qui transfère à …
- Ignore le fait que souvent dans une organisation le propriétaire d'une information n'est pas celui qui l'a créée

– Complexe à gérer dans des organisations où il y a beaucoup de propriétaires et beaucoup d'objets à protéger

MODÈLE DE CONTRÔLE D'ACCÈS OBLIGATOIRE (MAC)

Mandatory Access Control (MAC)

Les sujets ne sont pas propriétaires des objets

L'accès aux objets est défini par des règles fixes

- les sujets ne peuvent pas modifier ces règles

On suppose des classifications fixes des sujets et objets

Les permissions sont fixées par des règles liées aux classifications

Exemples :

Classer les usagers et les objets dans des classes de confidentialité

(Top secret, Secret, Confidentiel, …)

Un sujet à un certain niveau ne peut pas lire un objet à un niveau supérieur

(soldat au niveau 'Confidentiel' ne peut pas lire des informations 'Secret')

CRITIQUES DE MAC

+ Très approprié dans les contextes où les classifications fixes par niveaux de sécurité sont possibles

+ S'occupe non seulement du contrôle d'accès, mais aussi du contrôle de flux

– Rigide, appropriée pour les organisations où le besoin de la sécurité est très stricte

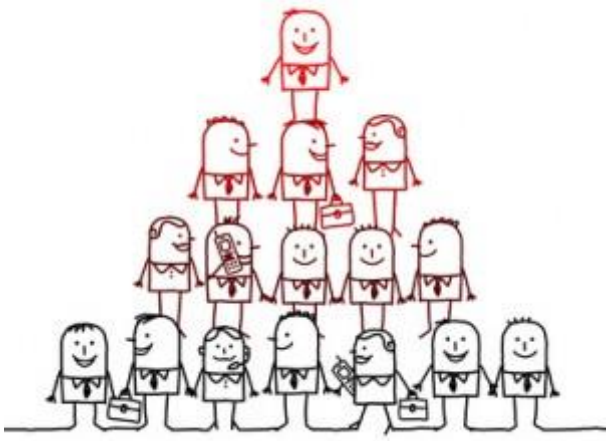
– Dans la plupart des organisations, nous avons besoin de classifications plus souples

MODÈLE À BASE DE RÔLES (RBAC)

Dans la plupart des organisations, les usagers sont classés par rôles

Par exemple, rôles à l'Université de La Rochelle

- Président
- Vice-présidents
- Doyens
- Directeurs de services
- Responsables de formation
- Professeur
- Etudiant
- …



Les usagers sont affectés à des rôles et les permissions des usagers sont déterminés par le(s) rôles auxquels ils appartiennent

CRITIQUES DE RBAC

\+ Approprié aux organisations de structure assez fixe et hiérarchiques

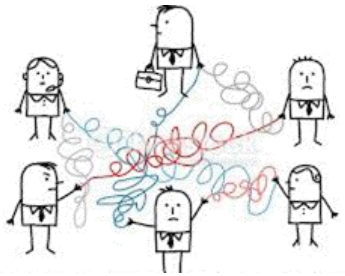
Banques, finances, gouvernement, administrations, …

\+ Probablement la méthode la plus largement utilisée dans les grandes entreprises

\+ Peut simuler DAC, MAC (mais avec difficulté …)

± Nombreuses variations, pour l'adapter à différents besoins

– Peu approprié pour les organisations très dynamiques ou qui suivent d'autres modèles (par exemple réseau)



CONTRÔLE D'ACCÈS BASÉ SUR LES ATTRIBUTS (ABAC)

Dans ABAC, chaque sujet et objet a des attributs

Des attributs d'environnement peuvent être considérés

- l'heure
- le placement dans l'espace
- …

Les décisions de contrôle d'accès sont prises en fonction de ces attributs

CRITIQUES DE ABAC

+ Très flexible

– Difficile de déterminer en principe qui a droit à quoi

– Difficile de bien saisir les implications dérivant des changements d'attributs

- Si un des attributs d'un usager change, quels seront ses nouveaux permissions, quelles permissions perdra-t-il?
- Difficile à gérer pour l'administrateur de la sécurité

MODÈLES HYBRIDES

Il est parfois possible de combiner les caractéristiques de différents modèles

Les modèles qui font ceci sont appelés modèles hybrides

Cependant, en faisant ceci, il est important d'éviter que les modèles se 'cassent' l'un l'autre

Problème d'interaction de fonctionnalités

ET AVEC POSTGRESQL ?

Système de type RBAC

- Multiversion Concurrency Control : MVCC

MVCC

- Isolation des transactions pour chaque session
- Chaque requête SQL voit une image des données telle qu'elles étaient quelque temps auparavant
- Evite que les requêtes puissent voir des données non cohérentes produites par des transactions concurrentes effectuant des mises à jour sur les mêmes lignes de données

Principal avantage

- Lire ne bloque jamais l'écriture
- Ecrire ne bloque jamais la lecture

CONTRÔLE DES ACCÈS

Base de données = Partage de données

- Communauté d'utilisateurs
- Communauté de bases

Le partage impose la protection contre des atteintes à

- La confidentialité (divulgations d'information non autorisées)
- L'intégrité (modifications non autorisées)
- La disponibilité (déni de service)

De nombreux modèles de contrôle d'accès

- Matrices de contrôle d'accès (ACM), Contrôle d'accès discrétionnaire (DAC), Contrôle d'accès obligatoire (MAC), Contrôle d'accès basé sur les rôles (RBAC), Contrôle d'accès basé sur les attributs (ABAC)

Chacun de ces modèles connaît nombreuses variations, adaptations et implémentations

DÉFINITIONS

Utilisateur = permet d'identifier une personne qui accède à la BD

Les groupes permettent de partager des droits entre utilisateurs

Privilège = autorisation accordée à un utilisateur pour effectuer une opération sur un objet

Rôles = classe générale d'utilisateurs

- Depuis postgresql 8.1, « users » et « groups » sont des rôles
- « user » : role qui peut se logger
- « group » : role qui ne peut pas se logger

Les privilèges sont attribués à des rôles

On attribue un (ou plusieurs) rôle(s) à un utilisateur

COMMANDES ÉLÉMENTAIRES

Créer un utilisateur

- `CREATE ROLE user_name LOGIN <ATTRIBUTES>;`

Créer un groupe

- `CREATE ROLE group_name NOLOGIN <ATTRIBUTES>;`

Ajouter un rôle à un utilisateur

- `GRANT ROLE group_name TO user_name;`

COMMANDES ÉLÉMENTAIRES

Modifier un rôle

- `ALTER ROLE user_name WITH options;`

Retirer des accès à un rôle

- `REVOKE action FROM table;`
- `REVOKE role FROM user;`

Supprimer un rôle

- `DROP ROLE name;`

EXEMPLE D'AJOUT

On créé le rôle **CONSULTANT,** on lui attribue un ensemble de privilèges sur la table **CLIENT**, et on l'assigne à l'utilisateur OMER

```
create role CONSULTANT;

grant select on CLIENT to CONSULTANT;

grant update (ADRESSE,LOCALITE) on CLIENT to CONSULTANT;

grant CONSULTANT to OMER;
```

EXEMPLE DE RÉDUCTION

On supprime le droit de sélectionner l'attribut LOCALITE

On retire le rôle CONSULTANT de l'utilisateur OMER

On supprime le rôle CONSULTANT

```
revoke select (LOCALITE) from CONSULTANT;

revoke CONSULTANT from OMER;

drop role CONSULTANT;
```

RÔLES PARTICULIERS

Superusers

Le rôle PUBLIC

Héritage

RÔLES PARTICULIERS

Superusers

- Par default « postgres » sans mot de passe (!)
- Superuser est le dieu du cluster
- Peut être associé à n'importe quel rôle :

```
CREATE ROLE role_name SUPERUSER;

ALTER ROLE role_name NOSUPERUSER;
```

Le rôle PUBLIC

Héritage

RÔLES PARTICULIERS

Superusers

Le rôle PUBLIC

- Groupe implicite auquel tous les utilisateurs appartiennent
- Assigne des droits par défaut
  - Pas d'accès aux tables, colonnes, schémas et tablespaces

Héritage

RÔLES PARTICULIERS

Superusers

Le rôle PUBLIC

Héritage

- Permet à un rôle d'obtenir les droits d'autres rôles
- Commande SET ROLE pour obtenir les droits des autres rôles
- Protège du « rôle qui tous les droits tout le temps »

EXEMPLE D'HÉRITAGE

Que fait la série d'action ci-dessous ?

```
CREATE ROLE admins NOLOGIN NOINHERIT;
CREATE ROLE one_admin LOGIN PASSWORD 'foobar';
GRANT admins TO one_admin;
GRANT postgres TO admins;
```

Créer un groupe « admins » avec héritage

Créer un compte admin sans les droits superuser

Ajoute le compte one_admin dans le groupe « admins »

Ajoute le rôle « postgres » au groupe « admins »

Délègue les privilèges de Superuser sans donner le mot de passe de postgres aux utilisateurs

SOMMAIRE

Contrôle des données et des accès

- Définition de rôles et gestion des droits

- Transactions
  - Concurrence
  - Verrous
  - Estampillage

PROBLÈME DES ACCÈS CONCURRENTS

Deux internautes veulent réserver une place pour un spectacle

Problème : il ne reste qu'une seule place

Les deux internautes interrogent la base de données à une seconde d'intervalle, il reste une place. Comment faire pour que :

La place soit vendue à l'un des deux internautes…

… et à un seul !

Une solution est la décomposition en transaction et l'isolation de transactions

TRANSACTIONS

Séquence d'instructions indissociables (unité logique de travail)

Se termine de 2 manières possibles

- Par validation : COMMIT (utilisateur) – terminaison normale. Mises-à-jour deviennent persistantes et les lectures sont correctes
- Par annulation : ABORT (utilisateur ou système) – terminaison anormale. Mises-à-jour annulées et les valeurs lues ne sont pas sûres

SCHÉMA DE TRANSACTION SIMPLE

Fin avec succès ou échec

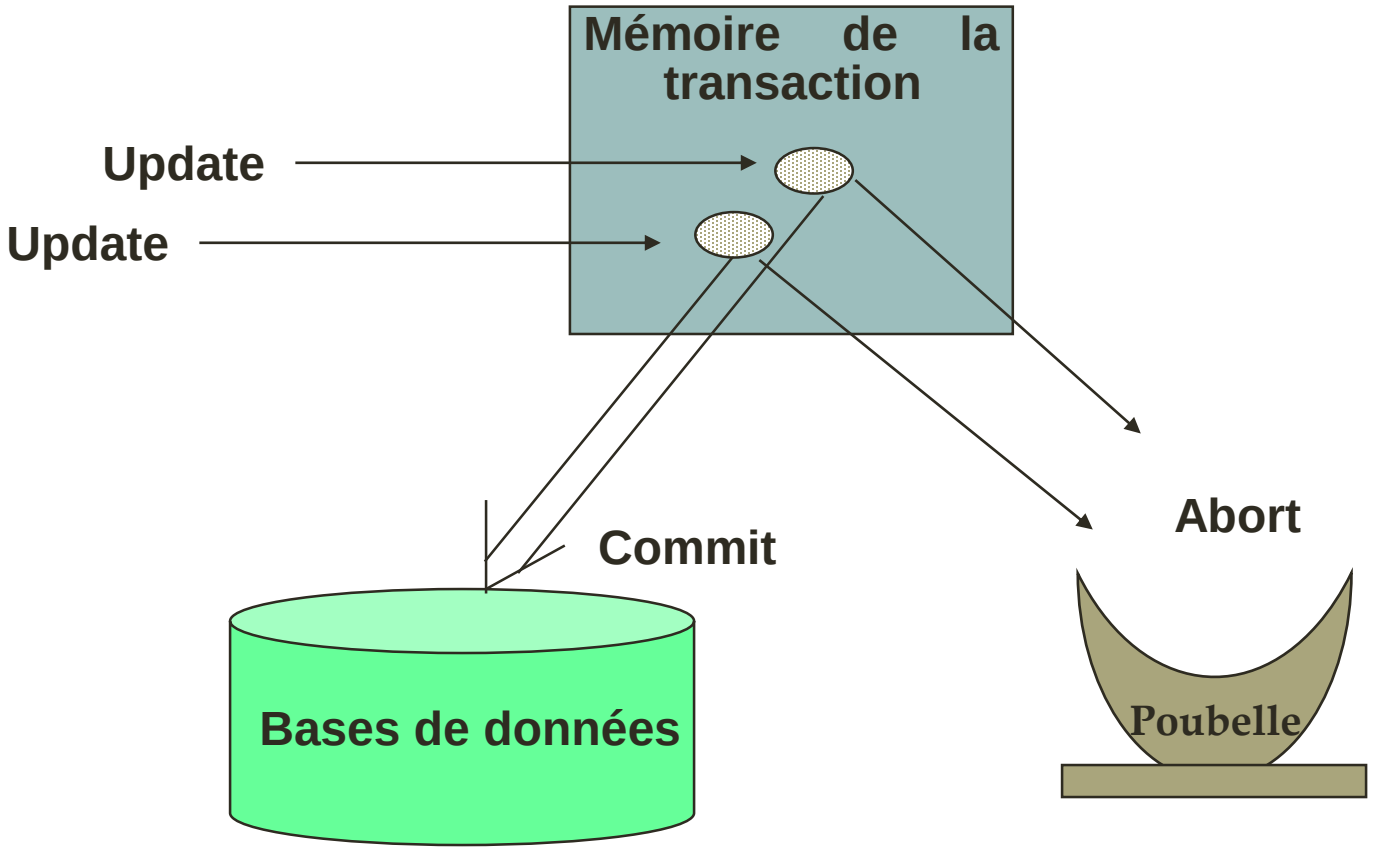
Begin_Transaction

- Update
- Update
- .....
- Commit ou Abort

- Provoque l'intégration réelle des mises à jour dans la base
- Relâche les verrous

- Provoque l'annulation des mises à jour
- Relâche les verrous
- Reprend la transaction

EFFET LOGIQUE



PROPRIÉTÉ D'UNE TRANSACTION

Propriétés ACID

- **Atomicité** : les modifications doivent être totalement réalisées ou pas du tout
- **Cohérence** : les modifications apportées à la base doivent être valides
- **Isolation** : les transactions lancées au même moment ne doivent jamais interférer entre elles
- **Durabilité** : toutes les transactions sont lancées de manière définitive
- **Légalité** : respecter les privilèges liés aux rôles

PHÉNOMÈNES PROBLÉMATIQUES

Phénomènes liés aux interactions entre des transactions concurrentes

- **Dirty read** : Une transaction lit des données écrites par une transaction concurrente non validée

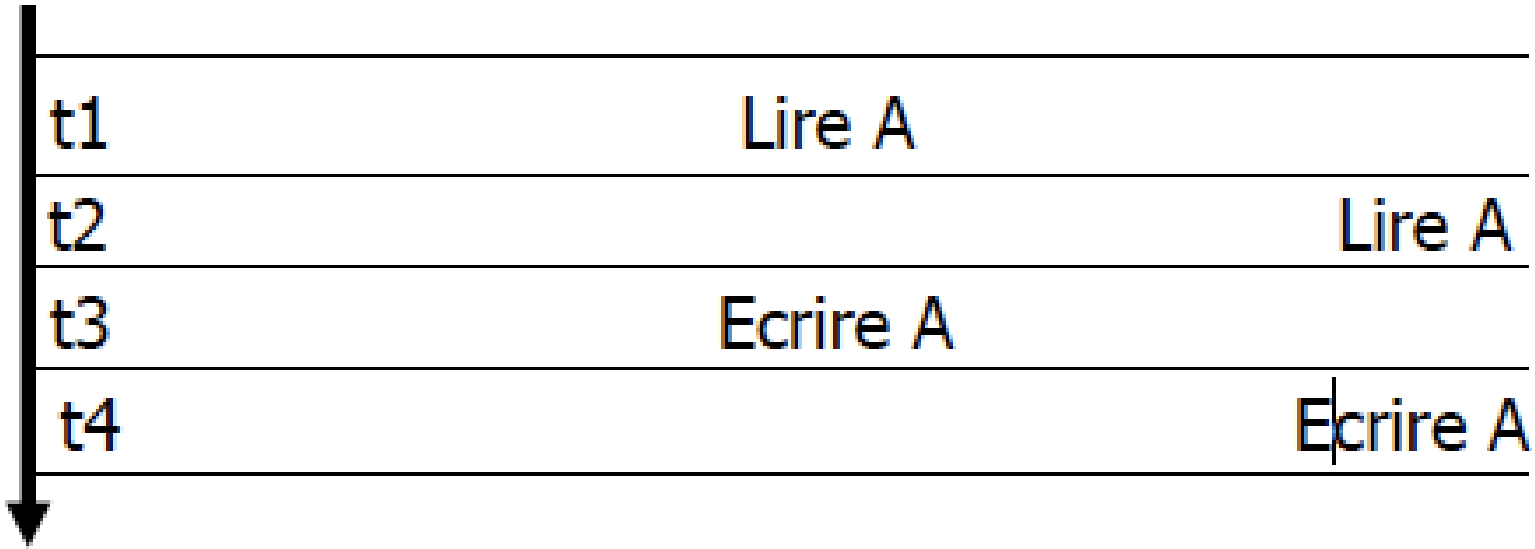
- **Phantom read** : Une transaction ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé du fait d'une autre transaction récemment validée

- **Non repeatable read** : Une transaction relit des données qu'elle a lu précédemment et trouve que les données ont été modifiées par une autre transaction (validée depuis la lecture initiale)

PROBLÈMES À ÉVITER (1)

(1) Perte de mise à jour

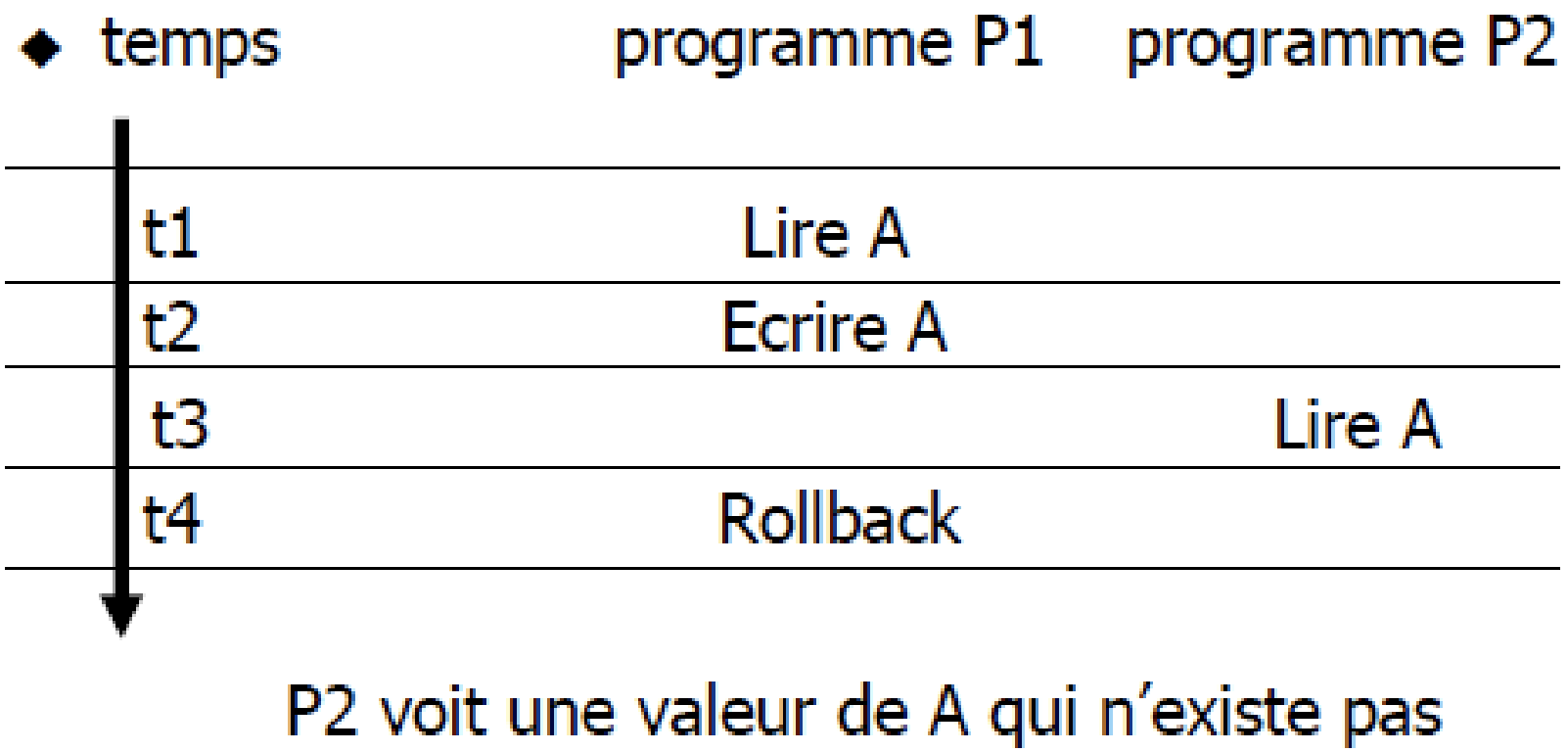
- temps programme P1 programme P2



La mise à jour faite par P1 est perdue

PROBLÈMES À ÉVITER (2)

(2) Données fantômes



P2 voit une valeur de A qui n'existe pas

PROBLÈMES À ÉVITER (3)

3) analyse incohérente

- temps programme P1 programme P2

t1	Lire A	
t2	sum:=A	
t3	Lire B	
t4	sum:=sum+B	
t5		Lire C
t6		Lire A
t7		Transférer 10 de C vers A
t8		COMMIT
t9	Lire C	
t10	sum:=sum+C	

!! Sum aurait dû être = à 120

t1 Sum 0 A 40 B 50 C 30

t2 Sum 40 A 40 B 50 C 30

t4 Sum 90 A 40 B 50 C 30

t7 Sum 90 A 50 B 50 C 20

t10 Sum 110 A 50 B 50 C 20

ISOLATION DE TRANSACTIONS

SQL définit quatre niveaux d'isolation de transaction

Niveau d'isolation	Lecture sale	Lecture non reproductible	Lecture fantôme
Uncommited Read « Lecture de données non validées »	Possible	Possible	Possible
Commited Read « Lecture de données validées »	Impossible	Possible	Possible
Repeatable Read « Lecture répétée »	Impossible	Impossible	Possible
Serializable « Sérialisable »	Impossible	Impossible	Impossible

En SQL : `SET TRANSACTION mode_transaction`

- `SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED`

NIVEAU D'ISOLATION

REPEATABLE READ – « lecture répétable »

- Niveau par défaut
- Plusieurs requêtes de sélection (non-verrouillantes) de suite donneront toujours le même résultat, quels que soient les changements effectués par d'autres sessions

READ COMMITTED

- Chaque requête SELECT (non-verrouillante) va reprendre une "photo" à jour de la base de données (même si plusieurs SELECT se font dans la même transaction)
- un SELECT verra toujours les derniers changements commités, même s'ils ont été faits dans une autre session, après le début de la transaction

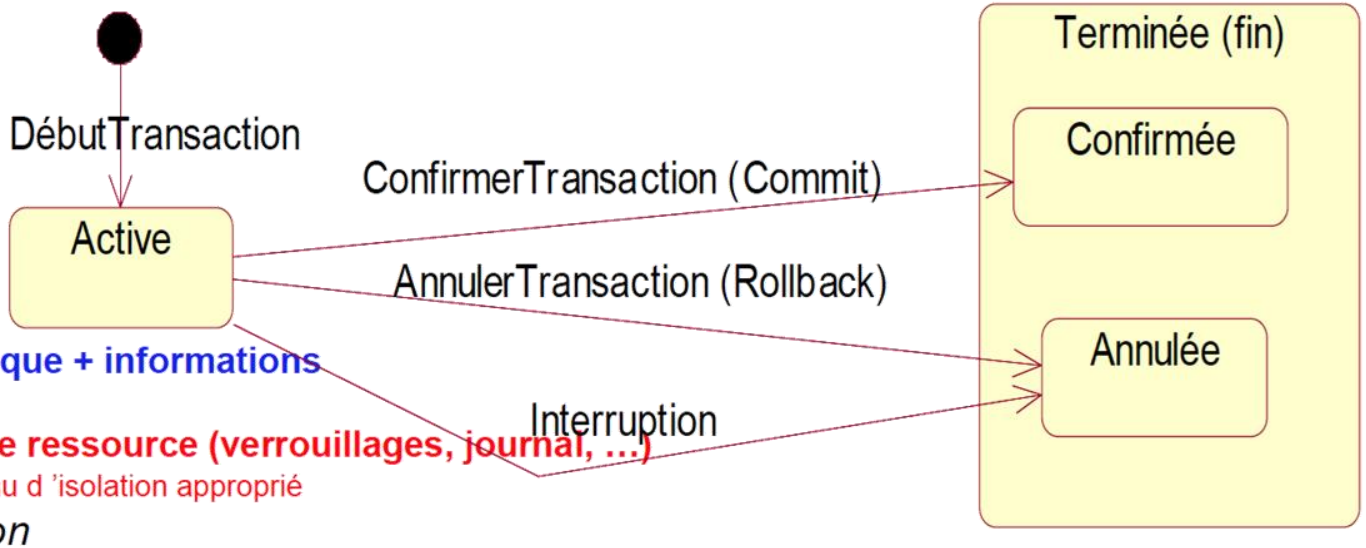
READ UNCOMMITTED

- Fonctionne comme READ COMMITTED
  - Sauf qu'il autorise la « lecture sale »
  - Une session sera capable de lire des changements encore non commités par d'autres sessions

SERIALIZABLE

- Se comporte comme REPEATABLE READ
- Tous les SELECT simples sont implicitement convertis en SELECT ... LOCK IN SHARE MODE

ETAT D'UNE TRANSACTION



- *DébutTransaction*
  - **identificateur unique + informations**
  - **état *actif***
  - **consommation de ressource (verrouillages, journal, …)**
    - choisir le niveau d 'isolation approprié
- *ConfirmerTransaction*
  - **état *confirmé***
  - **point de confirmation**
    - enregistrement persistant (journal)
  - **libération de ressources**
    - confirmer le plus tôt possible pour éviter de monopoliser les ressources
- *AnnulerTransaction ou* interruption suite à une erreur (faute)
  - **état *annulé***
  - **effets doivent être « défaits »**
- *Fin = confirmé ou annulé*

PRINCIPALES UTILISATIONS DES TRANSACTIONS

Traitement des opérations sémantiquement liées

- Doit garantir le « tout ou rien »
  - Soit toutes les opérations sont validées, soit annulées
- Exemple : débit-crédit

Gestion des concurrences

- Acquisition de verrous sur les enregistrements traités, empêchant une utilisation malencontreuse des données

Reprise sur pannes

- Utilisation du système transactionnel pour la reconstitution d'un état cohérent de la base au redémarrage d'un système après une panne, quel que soit le type de panne

DÉMARRER UNE TRANSACTION

Standard SQL :

- `START TRANSACTION [ mode_transaction [, ...] ]`

En Postgresql :

- `BEGIN [ mode_transaction [, ...] ]`
- `BEGIN TRANSACTION;`

où mode_transaction peut être :

- `ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED } READ WRITE | READ ONLY`

Par défaut, chaque commande SQL est traitée comme une transaction autonome (avec Postgresql)

SAUVER LES CHANGEMENTS

COMMIT sert à sauver les changements invoqués dans la transaction

COMMIT sauve toutes les transactions dans la base de données depuis le dernier COMMIT ou ROLLBACK

Syntaxe :

```
COMMIT;

ou

END TRANSACTION;
```

Les résultats persistent jusqu'au prochain COMMIT ou ROLLBACK

La transaction finira par un ROLLBACK si une erreur apparaît

ANNULATION / ERREUR

La commande ROLLBACK permet d'annuler les actions d'une transaction qui n'ont pas encore été sauvées

ROLLBACK annulera les actions depuis le dernier COMMIT ou ROLLBACK

Syntaxe :

```
ROLLBACK;
```

EXEMPLE DE TRANSACTION

```
 id | name   | age | address    | salary
----+--------+-----+------------+--------
  1 | Paul   |  32 | California |  20000
  2 | Allen  |  25 | Texas      |  15000
  3 | Teddy  |  23 | Norway     |  20000
  4 | Mark   |  25 | Rich-Mond  |  65000
  5 | David  |  27 | Texas      |  85000
  6 | Kim    |  22 | South-Hall |  45000
  7 | James  |  24 | Houston    |  10000
```

```
BEGIN;
DELETE FROM COMPANY WHERE AGE = 25;
COMMIT;
```

```
 id | name   | age | address    | salary
----+--------+-----+------------+--------
  1 | Paul   |  32 | California |  20000
  3 | Teddy  |  23 | Norway     |  20000
  5 | David  |  27 | Texas      |  85000
  6 | Kim    |  22 | South-Hall |  45000
  7 | James  |  24 | Houston    |  10000
(5 rows)
```

EXEMPLE DE TRANSACTION

id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000
6	Kim	22	South-Hall	45000
7	James	24	Houston	10000

```
BEGIN;
DELETE FROM company WHERE age = 25;
ROLLBACK;
```

id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000
6	Kim	22	South-Hall	45000
7	James	24	Houston	10000

GESTION FINE DES TRANSACTIONS

Les transactions longues peuvent être problématiques

- Volonté de revenir à un point particulier
- Volonté de valider une partie seulement

SAVEPOINT permet de renvoyer la transaction à une certain point

- Permet de choisir les actions à annuler
- Les actions précédentes sont conservées

Syntaxe pour créer le point de sauvegarde :

```
SAVEPOINT SAVEPOINT_NAME;
```

Syntaxe pour annuler les actions qui suivent un SAVEPOINT :

```
ROLLBACK TO SAVEPOINT_NAME;
```

PARTICULARITÉS DES SAVEPOINTS

Après avoir executé un ROLLINGBACK TO, celui-ci reste définit

- Possibilité de faire plusieurs appels au même SAVEPOINT

Les SAVEPOINT sont définis dans un block transaction

- Invisible aux autres sessions de la base de données

COMMIT = les actions deviennent visibles aux autres sessions

ROLLBACK = les actions ne seront jamais visibles

Possibilité de relâcher un SAVEPOINT (suppression)

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

GESTION PLUS FINE

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SAVEPOINT SP1;
        Savepoint created.

DELETE FROM customers WHERE id=1;
        1 row deleted.

SAVEPOINT SP2;
        Savepoint created.

DELETE FROM customers WHERE id=2;
        1 row deleted.

SAVEPOINT SP3;
        Savepoint created.

DELETE FROM customers WHERE id=3;
        1 row deleted.
```

GESTION PLUS FINE

```
ROLLBACK TO SP2;
  Rollback complete.
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

VERROUS

Les transactions ne permettent pas de verrouiller l'accès aux données
- Problème lié aux accès concurrents en écriture
- Nécessité de verrouiller l'accès à ces données

Verrous préviennent la modification par un utilisateur
- Locks | Exclusive Locks | Write Locks
- S'appliquent sur une ligne ou une table complète
- UPDATE et DELETE sont des verrous exclusifs pour la durée de la transaction
- Valables jusqu'au COMMIT ou ROOLBACK

Verrous consistent à faire patienter un utilisateur
- Attend un autre utilisateur
- Pas de verrou (d'attente) pour les requêtes SELECT

DEADLOCK
- Quand 2 transactions s'attendent mutuellement
- PostgreSQL peut les détecter et les terminer avec un ROLLBACK
- Faire attention aux objets que vous verrouillez

VERROUS - SYNTAXE

```
LOCK [ TABLE ]
name
 IN
lock_mode
```

Name :

- Nom de la table à vérrouiller

lock_mode :

- Spécifie le type de verrou à appliquer
  - ACCESS EXCLUSIVE (par defaut)
  - ACCESS SHARE, ROW SHARE , ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE

Un verrou est valable pendant toute la transaction

Il n'y a pas de commande UNLOCK TABLE

Les verrous sont automatiquement relâchés à la fin de la transaction

TYPES DE VERROUS

ACCESS SHARE
- Conflit avec ACCESS EXCLUSIVE
- Acquis par la commande SELECT pour une table
- En général, pour une requête qui ne fait que lire (sans modification)

ROW EXCLUSIVE
- Conflit avec SHARE, ACCESS EXCLUSIVE
- Acquis par les commandes UPDATE, DELETE, et INSERT pour une table
- En général, acquis par toute commande qui modifie des données

SHARE
- Conflit avec ROW EXCLUSIVE, ACCESS EXCLUSIVE
- Protège contre les changements concurrents de données
- Acquis par CREATE INDEX

ACCESS EXCLUSIVE
- Conflit avec tous les autres verrous
- Garantie que le propriétaire est la seule transaction à accéder à la table
- Acquis par ALTER TABLE, DROP TABLE, TRUNCATE, …
- Verrou par défaut si rien de spécifié explicitement !

Liste exhaustive : https://docs.postgresql.fr/13/explicit-locking.html

VERROUS - EXEMPLE

```
testdb# select * from COMPANY;
 id | name   | age | address    | salary
----+--------+-----+------------+--------
  1 | Paul   |  32 | California |  20000
  2 | Allen  |  25 | Texas      |  15000
  3 | Teddy  |  23 | Norway     |  20000
  4 | Mark   |  25 | Rich-Mond  |  65000
  5 | David  |  27 | Texas      |  85000
  6 | Kim    |  22 | South-Hall |  45000
  7 | James  |  24 | Houston    |  10000
(7 rows)
```

```
BEGIN;
LOCK TABLE company1 IN ACCESS
EXCLUSIVE MODE;
```

Résultat  LOCK TABLE

BASES DE DONNÉES
COURS 4
TRANSACTIONS / SÉCURITÉ

Mickaël Coustaty

Jean-Loup Guillaume

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/