Cours 1
Transactions

SQL dans les langages de programmation

R3.07

Qu'est-ce qu'une BD ?

- Une base de données est un ensemble d'informations :
  - Construit pour répondre à une besoin
  - Stocké sur un support physique
  - Organisé pour être facilement accessible, géré et mis à jour
- Dans une base de données relationnelle, les données sont stockées sous forme de relations (**tables**)

Adherent (numadh, nom, prenom, fonction, adresse, telephone, skipper, anneeadh)
Bateau (numbat, nombat, taille, typebat, nbplaces)
Activite (numact, typeact, depart, arrivee, datedebut, datefin)
Proprietaire (numadh, numbat)
Chefdebord (numact, numadh, numbat)
…

Qu'est-ce qu'un SGBD ?

- Un SGBD doit permettre de
  - **Stocker**, **sauvegarder** (en cas de panne), et **interroger** les données
  - Contrôler la **redondance** des données
  - Autoriser les **accès simultanés** (par différents utilisateurs) aux données
  - **Protéger** les données (restreindre les accès non autorisés)

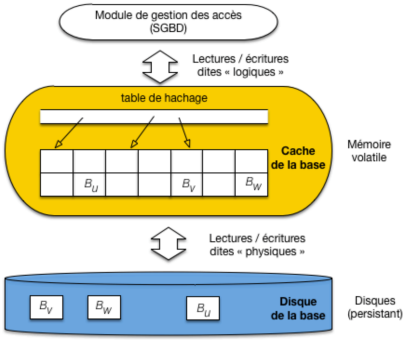


Image issue de http://sys.bdpedia.fr/stock.html

Gestion de la sécurité

- Il faut assurer
  - **La sûreté de fonctionnement** : assurer la cohérence des données en dépit des pannes matérielles qui peuvent se produire
  - **Les accès concurrents** : autoriser les accès simultanés à la BD par plusieurs utilisateurs

La notion de « Transaction » existe pour répondre à ces 2 besoins

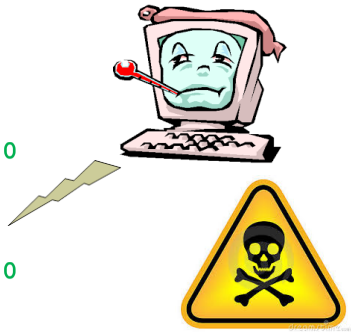
Exemple

- Soit la relation
  - COMPTE (numc, client, solde)

- Effectuer en PostgreSQL un virement de 100 euros du compte d'Alice vers celui de Bob

```
UPDATE compte
      SET solde = solde - 100.00
      WHERE nom = 'Alice';

UPDATE compte
      SET solde = solde + 100.00
      WHERE nom = 'Bob';
```



Transactions

- Une **transaction** est une **séquence d'actions** à réaliser sur la BD, qui a les propriétés suivantes (**ACID**)
  - **Atomicité** : tout est exécuté ou rien
  - **Cohérence** : une transaction fait passer la BD d'un état cohérent à un autre état cohérent
    - Exemple 1 : la somme des comptes d'Alice et de Bob après la transaction doit être égale à celle avant la transaction
    - Exemple 2 : le salaire ne doit pas passer un seuil
  - **Isolation** : les mises à jour faites par une transaction ne sont pas visibles « de l'extérieur » tant que la transaction n'est pas terminée
  - **Durabilité** : les actions effectuées par une transaction terminée sont définitives

Composition d'une transaction

- Une **transaction** est constituée d'**une suite finie d'actions portant sur des objets**
  - Début-transaction (BEGIN)
  - Fin-transaction (COMMIT, ROLLBACK)
  - Lire un objet (SELECT)
  - Écrire sur un objet (INSERT, UPDATE, DELETE)

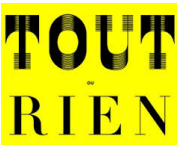
Exemple de transaction validée

```
BEGIN;

   UPDATE compte
      SET solde = solde - 100.00
      WHERE nom = 'Alice';

   UPDATE compte
      SET solde = solde + 100.00
      WHERE nom = 'Bob';

COMMIT;
```



SOIT le solde du compte d'Alice est débité de 100 ET le solde du compte de Bon est crédité de 100
SOIT aucun des comptes n'est modifié

Vie d'une transaction

- Une vie sans histoire (COMMIT)
  - Tout se passe bien, la transaction atteint son point de confirmation (ou validation), les valeurs modifiées sont confirmées
- Un assassinat (ROLLBACK implicite)
  - Un événement extérieur vient interrompre l'exécution de la transaction : panne, décision du SGBD (inter-blocage)
  - Il faut revenir en arrière et défaire les mises à jour effectuées
- Une annulation (ROLLBACK explicite)
  - Si le langage dans lequel est écrit la transaction autorise à programmer une annulation, la transaction peut se supprimer en effaçant toute trace de son passage

Exemple de transaction annulée

```
BEGIN;

   UPDATE compte
      SET solde = solde - 100.00
      WHERE nom = 'Alice';

   UPDATE compte
      SET solde = solde + 100.00
      WHERE nom = 'Bob';

ROLLBACK;
```

La transaction est annulée
-> aucun des comptes n'est modifié

Propriétés à assurer

- Un SGBD doit assurer que
  - Lors de l'exécution d'une transaction, soit toutes ses actions sont exécutées, soit aucune ne l'est (**Atomicité**)
  - Chaque transaction doit être isolée de façon qu'une exécution concurrente de plusieurs transactions n'introduise pas d'incohérence (**Cohérence + Isolation**)
  - Les effets d'une transaction qui s'est exécutée correctement survivent à une panne (**Durabilité**)

Gestionnaire de transactions

- Il doit
  - **Initialiser** chaque transaction T et en **contrôler l'exécution**. Si celle-ci se passe bien il doit la **confirmer**, sinon, il doit **annuler** tout ce qu'elle a fait
  - **Contrôler les accès concurrents** en synchronisant les transactions en conflit
  - Assurer la **reprise après panne**
    - **Refaire** le travail des transactions qui ont atteint leur point de confirmation avant la panne, mais aussi
    - **Défaire** le travail de celles qui n'avaient pas atteint leur point de confirmation

Gestion de la concurrence

- Pour des raisons d'**efficacité**, il est nécessaire d'autoriser plusieurs transactions à être exécutées concurremment tout en contrôlant que ces transactions n'entrent pas en **conflit** et ne mettent pas la base dans un état **incohérent**

- Les conflits se produisent lorsque deux transactions s'intéressent à un **même objet A** : ces transactions sont alors dites concurrentes

Gestion de la concurrence

- Des incohérences peuvent apparaître

T1	T2	Risque
Lecture	Lecture	Aucun
Ecriture	Ecriture	Perte de mise à jour
Ecriture	Lecture	Lecture impropre
Lecture	Ecriture	Lecture non reproductible

Perte de mise à jour

- Ecriture-Ecriture
  - Il y a perte de mise à jour lorsque T2 vient écraser l'écriture faite par T1

Temps	Transaction T1	Etat de base	Transaction T2
t1	lire(A)	(A=10)	—
t2	—		lire(A)
t3	A:=A+10		—
t4	—		—
t5	—		A:=A+50
t6	écrire(A)	(A=20)	—
t7	—	(A=60)	écrire(A)

En cas d'exécution séquentielle (T1 puis T2)
La valeur finale de 'A' doit être 70

Lecture impropre (1)

- Ecriture-Lecture (écriture → lecture)
  - Lorsque T2 lit une valeur modifiée par T1 et que T1 est annulée. Tout doit se passer comme si T1 n'avait jamais existé, donc la valeur lue par T2 est impropre car elle n'est pas confirmée

Temps	Transaction T1	Etat de base	Transaction T2
t1	lire(A)	(A=10)	—
t2	A:=A+20		—
t3	écrire(A)	(A=30)	—
t4	—		lire(A)
t5	*annulation*		—
t6	—		—

Pour T2, la valeur de 'A' doit être 10 à la place de 30

Lecture impropre (2)

- Ecriture-Lecture (lecture → écriture)
  - Utilisation par T2 d'une valeur non encore mise à jour par T1

Temps	Transaction T1	Etat de base	Transaction T2
t1	—	(C1=1000) (C2=1000)	—
t2	lire(C1)		—
t3	C1:=C1-500		—
t4	écrire(C1)	(C1=500)	—
t5	—		lire(C1)
t6	—		lire(C2)
t7	—		afficher(C1+C2)
t8	lire(C2)		—
t9	C2:=C2+500		—
t10	écrire(C2)	(C2=1500)	—

En cas d'exécution séquentielle (T1 puis T2) : T2 affiche 2000
D'après l'exécution ci-dessus : T2 affiche 1500

Lecture non reproductible

- Lecture-Ecriture (lecture → écriture → lecture)
  - T1 lit plusieurs fois une valeur de A et s'attend à obtenir chaque fois la même valeur. Or entre deux lectures, la valeur de A est modifiée

Temps	Transaction T1	Etat de base	Transaction T2
t1	—	(A=10)	lire(A)
t2	lire(A)		—
t3	—		A:=A+10
t4	—	(A=20)	écrire(A)
t5	—		—
t6	lire(A)		—

Au t2 la transaction T1 lit A=10, et au t6 la transaction T1 lit A=20, même si rien se passe entre t2 et t6 dans la transaction T1

Exécution sérialisable

- Une exécution d'un ensemble de transactions est dite **sérialisable** si elle donne le même résultat qu'une exécution en série (exécution séquentielle).
- Pour cela, 3 niveaux de cohérence :
  - Une transaction T ne doit pas écrire sur un objet dont la valeur a été modifiée par une autre transaction T' qui n'a pas atteint son point de confirmation (éviter Ecriture-Ecriture)
  - Une transaction T ne doit pas lire des valeurs non confirmées, manipulées par d'autres transactions (éviter Ecriture-Lecture)
  - Aucune transaction ne doit modifier une valeur lue par une autre transaction avant que cette dernière ne soit confirmée (éviter Lecture-Ecriture)

Exclusion mutuelle

- Allocation exclusive des objets par des verrous
- Un **verrou** est une variable associée à un objet indiquant son état par rapport aux lectures/écritures.
- Verrou binaire
  - **verrouiller(x)** : acquérir un contrôle exclusif de x
  - **libérer(x)** : libérer l'objet x
  - **Règle 1** : aucune transaction ne peut effectuer une màj ou une lecture d'un objet si elle n'en a pas acquis l'exclusivité par l'action verrouiller
  - **Règle 2** : si une transaction T2 ne peut acquérir l'exclusivité d'un objet x, elle attend jusqu'à ce que x soit libéré
- Ce protocole peut conduire à l'**interblocage**

Exclusion mutuelle

- Exemple

temps	T1	état de la base	T2
t1	verrouiller(A)	(A=10)	—
t2	lire(A)		—
t3	—		verrouiller(A)
t4	A:=A+10		(attente)
t5	écrire(A)	(A=20)	(attente)
t6	libérer(A)		(attente)
t7	—		lire(A)
t8	—		A:=A+50
t9	—	(A=70)	écrire(A)
t10	—		libérer(A)

Interblocage (deadlock)

- Il y a interblocage lorsque 2 transactions s'attendent mutuellement
- Supposons que T1 a l'exclusivité de A et T2 celle de B, si T1 demande B et T2 demande A il y a interblocage

temps	T1	T2
t1	verrouiller(A)	—
t2	—	verrouiller(B)
t3	verrouiller(B)	—
t4	(attente)	verrouiller(A)
t5	—	(attente)
t6	—	—

- Solution BD :
  - en cas d'interblocage, arrêter l'une des transactions pour l'exécuter plus tard

Protocole à deux phases

- Une transaction est dite **à 2 phases** si elle n'effectue aucun verrouillage après avoir effectué un déverrouillage
  - une phase d'expansion
  - une phase de libération
- Le protocole à 2 phases garantit la sérialisabilité mais n'empêche pas les interblocages.

Les transactions sous Postgres

Objectifs du TP sur les transactions :

- Découvrir le fonctionnement des transactions Postgres en utilisant les commandes : BEGIN et COMMIT / ROLLBACK
- Comprendre les différences entre les 3 niveaux d'isolation de Postgres

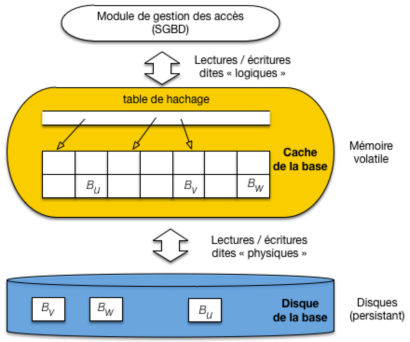
Niveaux d'isolation des transactions Postgres

- **Read Commited** : Le niveau le moins strict, qui est celui par défaut d'une transaction Postgres.
- **Repeatable Read**
- **Serializable** : Le plus strict des niveaux d'isolation (moins de risque d'anomalie, mais des performances possiblement moins bonnes)

Par contre, les verrous étant automatiquement gérés par le SGBD Postgres, vous n'aurez pas à en définir vous-même.

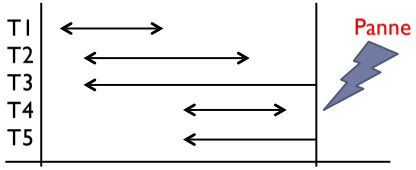
Reprise après panne

- Un SGBD doit assurer la cohérence des données en dépit des pannes matérielles et logicielles qui peuvent se produire
  - Défaillance de la machine
    - × destruction de la mémoire centrale
    - ✓ contenu des disques ou bandes conservé même si certaines écritures se sont incomplètement déroulées
  - Défaillance d'un périphérique
    - × destruction totale ou partielle du contenu du disque
    - ✓ avoir un état de la base à partir duquel il sera possible de repartir
  - Défaillance logicielle
    - × redémarrage logiciel
    - ✓ mémoire centrale perdue, disques dans un état plus ou moins cohérent
- La notion de cohérence est liée à la notion de transaction qui se termine normalement et est donc confirmée, ou bien qui ne se termine pas normalement et dont les effets doivent disparaître



Reprise après panne (Exemple)

- T1, T2, T4 se sont terminées correctement, et T3 & T5 étaient en cours d'exécution
  - Les effets de T1, T2, T4 doivent survire à la panne mais ceux de T3 & T5 doivent être éliminés
  - Il faut **défaire** le travail de T3 & T5
  - Si la panne détruit les effets de T1, T2, T4 alors il faut **refaire** le travail de ces transactions
- En cas de panne, il faudra soit défaire soit refaire des modifications



Méthodes de résistance aux pannes

- Mécanisme personnel (**backup**) : Copier, à intervalle régulier, l'état de la base sur un support stocké en lieu sûr (feu, eau, …)
  - Au pire, on revient à l'état de la base au jour de la dernière sauvegarde
- Mécanisme interne au SGBD : le journal (**log file**)
  - Tenir un journal (log) de toutes les opérations effectuées sur la base depuis la dernière sauvegarde. Les opérations enregistrées dans le journal sont :
    - Ti écrit un objet : l'ancienne et la nouvelle valeur.
    - Ti fait un commit/rollback : cette action doit être enregistrée.

Journal

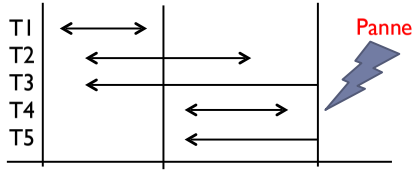
- Le journal est **un fichier texte** dans lequel le SGBD
  - inscrit dans l'ordre toutes les actions effectuées depuis la dernière sauvegarde de la BD
    - début de transaction
    - Ti écrit un objet : l'ancienne et la nouvelle valeur
    - fin de transaction avec validation ou échec
  - rajoute des **points de contrôle** à intervalles réguliers
    - Un point de contrôle est une sauvegarde des modifications de la BD sur disque

Redémarrage du SGBD

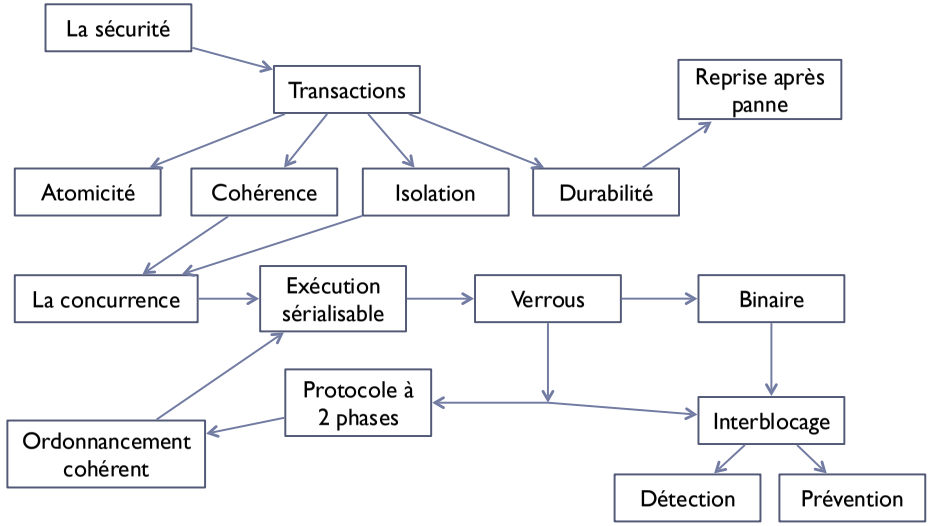
- Quand le système redémarre, il détermine en fonction du contenu du journal
  - Les transactions gagnantes
    - Confirmées avant la panne
  - Les transactions perdantes
    - Actives au moment de la panne
- Le SGBD doit
  - Refaire le travail des transactions gagnantes
  - Défaire le travail des transactions perdantes

Redémarrage du SGBD

- Exemple
  - T1 n'est pas à refaire : toutes les modifs figurent dans la sauvegarde
  - Aucune modification de T5 n'apparaît dans la sauvegarde donc T5 est déjà défait
  - Refaire T2 & T4 nécessite de parcourir le journal vers l'avant à partir du point de sauvegarde. Puisque le journal contient les nouvelles valeurs, il suffit de refaire chacune des modifications
  - Défaire T3 nécessite un parcours arrière du journal. Pour chaque modification faite, il faut revenir à l'ancienne valeur et ceci jusqu'à ce que l'on rencontre le début-transaction de T3



Résumé



En complément

- Vous pouvez regarder une vidéo sur
  - la notion de transaction:

https://www.youtube.com/watch?v=nsODZRHZ6S0

  - la notion de verrou

https://www.youtube.com/watch?v=o7Yjg8Ct4Bs