BD : partage de données

Garantir la cohérence des données lors de manipulations simultanées par différents utilisateurs - Gros volumes de données - Accès distribués

Notions importantes :

1. *transaction* : séquence cohérente d'actions
2. *accès concurrent* : accès simultanés à une ressource (bd, table, tuple) qui peuvent aboutir à des conflits
3. *session* : période délimitée dans le temps pendant laquelle un client entre en communication avec un serveur de données (vue ici comme une collection de transactions)
4. *connexion* : gestion de l'ouverture de la session - souvent associée à un mécanisme d'identification

Schéma illustratif

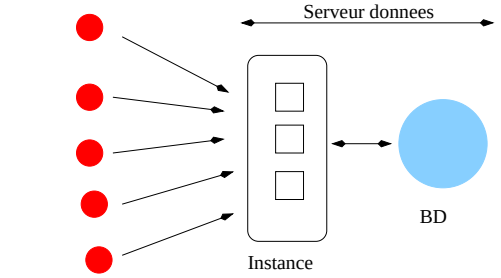

Figure: illustration accès concurrents

Domaines cibles

Exemples de domaines pour lesquels ces notions vont revêtir une importance capitale

1. *Systèmes bancaires : transferts monétaires*
2. *Systèmes de réservation : train, avion, hôtel, . . .*
3. *Centrales d'achats*
4. *Systèmes de santé*
5. *. . .*

Concept de transaction

Unité de traitement séquentiel (séquence cohérente d'actions), exécutée pour le compte d'un usager, qui appliquée à une bd cohérente, restitue une bd cohérente

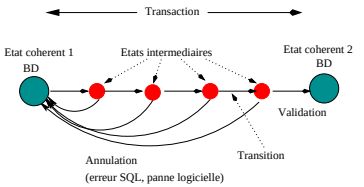

Figure: Illustration transaction

Transaction $T_i :< a_i^1, a_i^2, a_i^3, a_i^4, a_i^5, ...a_i^n >$

Voir une transaction comme un objet

Les opérations de la transaction doivent être soit exécutées en bloc, soit annulées en bloc (tout ou rien) $\Longrightarrow$ se doter d'un début et d'une fin de transaction

Marquer le début (implicite ou explicite)

- au premier ordre SQL (ouverture de la session)
- après la fin d'une transaction (validation ou annulation)
- ordre : `start transaction`, `begin transaction` ou `set transaction` ....

Fin de transaction

Marquer la fin (implicite ou explicite)

- fin explicite d'une transaction à l'aide des commandes COMMIT (validation des actions) et ROLLBACK (annulation des actions)
- fin implicite d'une transaction
  - exécution d'une commande LDD (CREATE, ALTER, RENAME et DROP) : actions exécutées depuis le début de la transaction sont validées
  - fin normale d'une session ou d'un programme avec déconnexion : la transaction est validée
  - fin anormale d'un programme ou d'une session (sortie sans déconnexion) : la transaction est annulée

Comptez les transactions

```
CREATE TABLE Compte (numC integer primary key,
typeC varchar(10),  solde float);

INSERT INTO Compte VALUES (2, 'courant', -200);
INSERT INTO Compte VALUES (3, 'courant', 500);
INSERT INTO Compte VALUES (4, 'courant', 100.50);
COMMIT;

UPDATE Compte SET solde = solde + 100 WHERE numC=3;
ROLLBACK;

DELETE FROM Compte WHERE numC=3;
ALTER TABLE Compte ADD numAg integer;
```

Détails sur l'illustration

Actions : read (select), write (update, insert, delete), commit, rollback

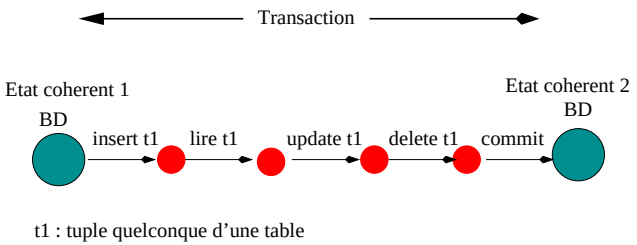

Figure: Illustration transaction

Propriétés d'une transaction

Gérer les transactions : s'assurer qu'elles possèdent les propriétés *ACID*

- **Atomicité** : lors d'une exécution d'une transaction, toutes ses actions sont exécutées ou bien aucune ne l'est.
- **Cohérence** : les modifications apportées à la b.d lors d'une transaction doivent être valides cad respecter les contraintes d'intégrité.
- **Isolation** : chaque transaction est isolée, pour s'affranchir des incohérences lors d'exécutions concurrentes
- **Durabilité ou Permanence** : les effets d'une transaction, qui s'est exécutée correctement, doivent survivre à une panne

Atomicité

Assurer que les actions même les plus complexes, englobées au sein d'une transaction, soient perçues comme une opération unique. Les usagers doivent connaître, en toute circonstance, l'état des données

Modèle général

- Tdébut
- Actions isolation, atomicité (panne $\Longrightarrow$ défaire)
- Tfin
- Validation : calcul de la validité de la transaction - certification
- Point de validation (commit)
- Permanence (panne $\Longrightarrow$ refaire éventuellement)
- Vrai fin de transaction

Validation à deux phases

La **validation à deux phases** suppose l'existence d'une mémoire stable, dans laquelle au point de validation, les nouvelles valeurs seront enregistrées.

- importance du point de validation : calculs pour accepter ou rejeter la transaction finissante (certification). segments d'annulation, mémoire redo-log
- après le point de validation, la transaction sera visible (au bout d'un certain temps) par les autres.
- transaction vivante : avant le point de validation
- transaction validée : après le point de validation

Transactions sérialisables

Une exécution d'un ensemble de transactions est sérialisable ssi elle est équivalente à une exécution séquentielle (ou en série) de transactions. Quand les transactions sont arbitraires, la sériabilité est la seule à pouvoir assurer un entrelacement correct.

$< T_p(1),, T_p(2), T_p(3), T_p(4), ..., T_p(n) >$ avec p une permutation de 1, 2, 3, ..., n

Actions conflictuelles : portent sur le même objet, et une action au moins, sur les deux, est en écriture. Actions commutables A et B si l'exécution de A suivie de B est identique à l'exécution de B suivie de A

Isoler au moyen de verrous

Isoler un élément dans une transaction en le verrouillant (lock). Les verrous sont définis par deux opérations : Verrouillage à deux phases (2PL)

- verrouiller(A) (lock(A)): cette opération oblige toute transaction à attendre le déverrouillage de l'élément A si elle a besoin de cet élément
- déverrouiller(A) (unlock(A)): la transaction effectuant cette opération libère le verrou qu'elle avait obtenu sur A et permet à une autre transaction candidate, en attente, de poser, à son tour, un verrou.

Inconvénients des verrous

Situation d'attente pour d'autres transactions

- **la famine** : lorqu'un verrou est relâché sur A, le système choisit parmi les transactions candidates en attente : ordre d'entrée dans une file d'attente
- **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.