IFT3030
Base de données

Chapitre 8
Fonctions avancées

Plan du cours

- Introduction
- Architecture
- Modèles de données
- Modèle relationnel
- Algèbre relationnelle
- SQL
- Conception
- **Fonctions avancées**
- Concepts avancés
- Modèle des objets
- BD à objets

Introduction

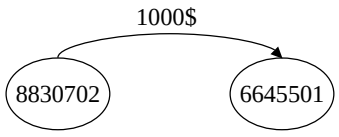
- En plus d'offrir la possibilité de définir et d'interroger une base de données, un SGBD relationnel, offre également des fonctionnalités avancées
- Fonctions avancées
  - Reprise après panne
  - Gestion de la concurrence
  - Sécurité
  - Intégrité

3

Reprise après panne

- Transaction
  - Une unité logique de travail
  - Exemple

```
EXEC SQL WHENEVER ERROR GOTO toto;
EXEC SQL UPDATE COMPTE SET solde = solde - 1000 WHERE num = 8830702;
EXEC SQL UPDATE COMPTE SET solde = solde + 1000 WHERE num = 6645501;
EXEC SQL COMMIT WORK;
return;
toto:
EXEC SQL ROLLBACK WORK;
return;
```



4

Reprise après panne

- Transaction
  - Propriétés
    - *Atomicité*

      Tout ou rien
    - *Cohérence*

      Au début et à la fin d'une transaction, mais pas forcément à l'intérieur
    - *Isolation*

      Les effets d'une transaction ne sont visible par les autres transaction qu'à sa fin
    - *Durabilité*

      Un fois terminée, ses effets sont durables

5

Introduction
Architecture
Modèles de données
Modèle relationnel
Algèbre relationnelle
SQL
Conception
Fonctions avancées
Concepts avancés
Modèle des objets
BD à objets

Reprise après panne

- Journal (log)
  - Fichier disque (ou bande) dans lequel sont écrites les modifications de la BD (valeurs concernées avant et après chaque requête élémentaire)
  - Utiliser pour défaire une transaction
  - Problème: Comment garantir la cohérence de la BD à l'intérieur d'une requête élémentaire ?

6

Reprise après panne

- Point de commit (syncpoint)
  - Point du programme dans lequel la BD est dans un état cohérent
  - Début du programme et à la fin de chaque transaction
  - Implications
    - Toutes les mises à jour sont rendues permanentes
    - Toutes les variables de positionnement sont détruites et tous les verrous sur les n-uplets effacés (cette implication s'applique également au ROLLBACK)

7

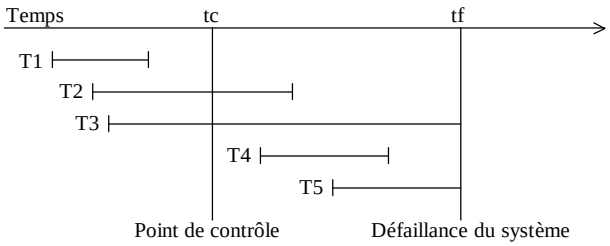
Reprise après panne

- Défaillance système (soft crash)
  - Affecte toutes les transactions courantes sans endommager la BD physiquement
  - Si la mémoire centrale est effacé, lors du redémarrage,
    - Les transactions courantes sont annulées (ROLLBACK)
    - Certaines transactions terminées doivent être rejouées (mémoires tampons non transférées sur disque)
    - Comment le système sait quelles transactions annuler et quelles transactions rejouer ?

8

Reprise après panne

- Défaillance système (soft crash)
  - Point de contrôle
    - A intervalle régulière, le système transfert le contenu des mémoires tampons dans la bases de données (physique) et produit un compte rendu dans le journal (physique)



Reprise après panne

- Défaillance des supports (hard crash)
  - Destruction physique d'une partie de la base de données.
  - Reprise après panne se fait en deux étapes :
    - Restaurer la dernière sauvegarde de la base
    - Rejouer toutes les transactions terminées avec succès depuis cette dernière sauvegarde

Gestion de la concurrence

- Constat
  - Des transactions concurrentes peuvent travailler sur les mêmes parties d'une base de données
  - Principe de l'isolation
  - Principe de la cohérence
- Conséquences
  - Problème de la perte d'une mise à jour
  - Problème des dépendances non validées
  - problème de l'analyse incohérente

Gestion de la concurrence

- Quelques définitions
  - Un ordonnancement est un scénario particulier d'exécution d'un ensemble de transactions
  - Il est séquentiel si les transactions sont exécutées les unes après les autres (non entrelacées)
  - Il est sérialisable s'il est équivalent à un ordonnancement séquentiel (pas forcément à tous les ordonnancement séquentiels possibles)
  - Deux opérations de lecture ou d'écriture dans deux transactions différentes sont dites non permutables si elles portent sur la même donnée et qu'au moins une des deux est une écriture
  - Un ordonnancement est sérialisable par permutation si les opérations non permutables sont effectuées dans le même ordre relatif que dans un ordonnancement séquentiel

Gestion de la concurrence

- Principales stratégies de contrôle
  - Verrouillage
    - Utilise un mécanisme de verrous
  - Estampillage
    - Se base sur l'identification unique des transactions et la gestion de l'ordonnancement
  - Certification
    - Retarde les écritures et la vérification des conflits à la fin des transactions (approche optimiste)
  - Multiversion
    - Utilise des versions différentes d'un même objets pour éviter les blocages

Gestion de la concurrence

- Verrouillage
  - Mécanisme qui permet à une transaction de bloquer l'accès à un objet dont elle a besoin (habituellement un n-uplet) pour qu'il ne soit modifié de manière imprévisible par les autres transactions
  - Deux types de verrous
    - Verrous exclusifs (X locks)
      - Verrous d'écriture
    - Verrous partagés (S locks)
      - Verrous de lecture

	X	S	-
X	R	R	A
S	R	A	A
-	A	A	A

Gestion de la concurrence

- Verrouillage
  - Problème de la perte de mise à jour
    - problème résolu mais blocage
  - Problème des dépendances non validées
    - problème résolu (dans les deux cas)
  - Problème de l'analyse incohérente
    - problème résolu mais blocage

Gestion de la concurrence

- Blocage
  - Situation où deux ou plusieurs transactions sont mutuellement bloquées en attente d'un verrou
- Solutions
  - Technique de l'examen du graphe d'attente
  - Technique du mécanisme de délai
- Variantes
  - Rejouer la transaction
  - Signaler à l'application

Sécurité

- Attribution des privilèges

  GRANT {liste_privilèges | ALL [PRIVILEGES]} [liste_attribut] ON objet TO {liste_utilisateurs | PUBLIC}] [WITH GRANT OPTION]

- privilèges
  - ALTER, DELETE, INSERT,SELECT, UPDATE, REFERENCES
- objets
  - table, vue, ...
- WITH GRANT OPTION
  - Permet à "l'autorisé" d'autoriser

17

Sécurité

- Retrait des privilèges

  REVOKE {liste_privilèges | ALL [PRIVILEGES]} ON objet FROM {liste_utilisateurs | PUBLIC}] [CASCADE CONSTRAINTS]

- privilèges et objets
  - idem
- CASCADE CONSTRAINTS
  - Supprime en les contraintes introduites par l'utilisateur (fonctionne avec REFERENCES et ALL PRIVILEGES)

18

Intégrité

- ***Contrainte d'intégrité statique***
  - respectée pour chacun des états de la BD
  - mécanismes déclaratifs
    - PRIMARY KEY, UNIQUE, NOT NULL, DOMAIN, FOREIGN KEY, CHECK, ASSERTION
  - procédural
    - TRIGGER (SQL3)
- ***Contrainte d'intégrité dynamique***
  - contrainte sur changements d'états
  - référence aux états successifs
  - TRIGGER

Intégrité

- CHECK intra-ligne
  - Plusieurs colonnes de la même ligne
  - Les *Articles* dont le *noArticle* est supérieur à 90 ont un prix supérieur à $15.00

```
CREATE TABLE Article
(noArticle    INTEGER            NOT NULL,
 description  VARCHAR(20),
 prixUnitaire       DECIMAL(10,2)NOT NULL,
 quantitéEnStock    INTEGER      NOT       NULL
  DEFAULT 0,
 PRIMARY KEY (noArticle),
 CHECK (noArticle <=90 OR prixUnitaire > 15.0)
)
```

Intégrité

Check inter-lignes d'une même table

- Concerne plusieurs lignes
- Le *prixUnitaire* d'un *Article* ne peut dépasser le prix moyen de plus de $40.00

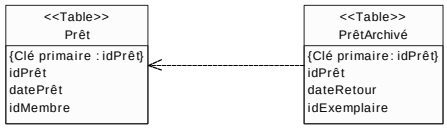
```
CREATE TABLE Article
(noArticle    INTEGER             NOT NULL,
 description  VARCHAR(20),
 prixUnitaire       DECIMAL(10,2) NOT NULL,
 quantitéEnStock    INTEGER       NOT         NULL
  DEFAULT 0,
 PRIMARY KEY (noArticle),
 CHECK (prixUnitaire-20 <= (SELECT
  AVG(prixUnitaire)
                            FROM Article))
)
```

Non support par Oracle

21

Intégrité

Check inter-tables




```
CREATE TABLE PrêtArchivé
(idPrêt      INTEGER             NOT NULL,
 dateRetour  DATE                NOT NULL,
 idExemplaire INTEGER            NOT NULL,
 PRIMARY KEY (idPrêt),
 FOREIGN KEY (idPrêt) REFERENCES Prêt,
 CHECK (dateRetour >= SELECT datePrêt
                      FROM Prêt
                      WHERE Prêt.idPrêt =
              PrêtArchivé.idPrêt)
      )
```

Non support par Oracle

22

Intégrité

- Assertions
  - Le *prixUnitaire* moyen d'un *Article* ne peut dépasser $25.00

```
CREATE ASSERTION assertionPrixMoyenMaximal
CHECK (25 >= SELECT AVG(prixUnitaire)
             FROM Article)
```

- Toujours valide par opposition au CHECK
- Non supporté par Oracle

Intégrité

- Assertion inter-tables
  - La somme des *quantitéLivrées* pour une *LigneCommande* ne peut dépasser la *quantité* commandée

```
CREATE ASSERTION assertionQuantitéLivrée
(NOT EXISTS(
   SELECT *
   FROM LigneCommande L,
       DétailLivraison D
   WHERE      L.noArticle = D.noArticle
       AND L.noCommande = D.noCommande
   GROUP BY L.noCommande,
       L.noArticle, quantité
       HAVING quantité < SUM(quantitéLivrée)))
```

Intégrité

- Trigger
  - Opération (procédurale)
  - Déclenchée par événement pré-déterminé
  - Exécutée au niveau serveur de BD
  - S'applique aussi bien au contraintes statiques que dynamiques
  - Exemple
    - Lorsqu'une augmentation du *prixUnitaire* d'un *Article* est tentée, il faut limiter l'augmentation à 10% du prix en cours

Intégrité

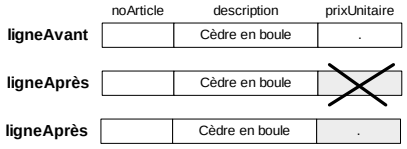
- Trigger

```
CREATE TRIGGER BUArticleBornerAugmentationPrix
BEFORE UPDATE OF prixUnitaire ON Article
REFERENCING
   OLD ROW AS ligneAvant
   NEW ROW AS ligneAprès
FOR EACH ROW
WHEN (ligneAprès.prixUnitaire >
   ligneAvant.prixUnitaire*1.1)
BEGIN
   ligneAprès.prixUnitaire =
   ligneAvant.prixUnitaire*1.1;
END
```

Intégrité

- Trigger

```
UPDATE      Article
SET         prixUnitaire = 15.99
WHERE  noArticle = 10
```



27

Intégrité

- Trigger

  Empêcher une augmentation du *prixUnitaire* d'un *Article* au delà de 10% du prix en cours

```
CREATE TRIGGER EmpêcherAugmentationPrixTropElevée
BEFORE UPDATE OF prixUnitaire ON Article
REFERENCING
    OLD ROW AS ligneAvant
    NEW ROW AS ligneAprès
FOR EACH ROW
WHEN (ligneAprès.prixUnitaire >
    ligneAvant.prixUnitaire*1.1)
BEGIN
     RAISE_APPLICATION_ERROR;
END
```

28