Chapitre 5. Implémentation des SGBDO

1- architectures client/serveur
2 - concurrence, pannes
3 - Quelques SGBDRO
4 - Quelques SGBDOO

Introduction

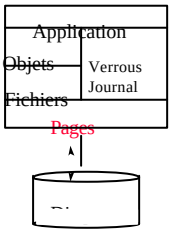
- années 80 :
  - serveur : BD
  - client : applications et langages
  - appel au serveur via SQL
- adaptable aux BDO
  mais client = station de travail

↳ déporter des fonctionnalités serveur ⇒ client

Architecture fonctionnelle

- couche 1 : outils de développement
  - outils **graphiques** de développement
  - permettent **conception interactive**
  - génération d'application avec **interface graphique**
- couche 2 : langage
  - langage de définition de données (schéma)
  - langage de requêtes (interrogation)
  - langage de programmation (méthodes et applications)
  - interfaces avec d'autres systèmes
- couche 3 : gestion d'objets
  - mécanismes d'accès & manipulation d'objets

Architecture opérationnelle



La couche Pages :
gestion mémoire virtuelle

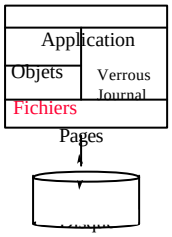
- page = unité de transfert disque / MC
- taille classique = 4Ko

- cache de pages
  - pages les + utiles aux appli

↳ éviter les accès disque

- si page manquante : accès disque
- repose sur système d'accès disque
  - allocation espace disque
  - lecture de blocs
  - écriture de pages

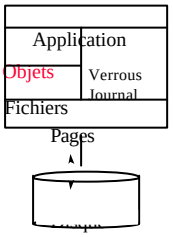
La couche Fichiers



manipulation de fichiers d'index et fichiers d'enregistrements
fichier = regroupe physiquement objets à caractéristiques communes (même classe)

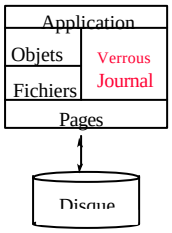
↳ objets grande taille
↳ connexions inter-objets

La couche Objets



- utilisée par une application
- gère:
  - création & manipulation d'objets persistants
  - espace adressage des objets

Mécanismes de contrôle de concurrence

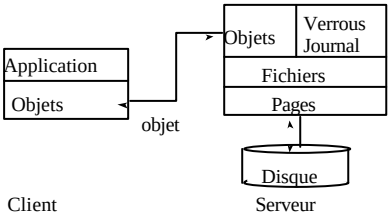


gestion transactions multi-utilisateur
- verrouillage (multi-utilisateur)
- journalisation (reprise après panne)

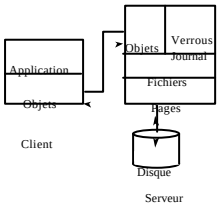
- fonctions partagées par couches Objets et Fichiers
- s'appuient sur couche Pages

Architecture serveur d'objets

Unité de transfert = objet ou groupe d'objets
utilisée dans tous les **SGBDRO**
et dans les SGBDOO **Orion, Ontos, Versant**



Architecture serveur d'objets



Client : exécute application
Serveur : gère BD sur Disque
Client et Serveur : même couche Objets

cache d'objets
- client : réduire accès serveur
- serveur : réduire accès disque

Architecture serveur d'objets (2)

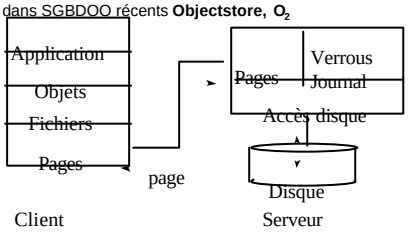
- Interface entre Client et Serveur
  - créer, détruire un objet
  - lire ou modifier un objet
  - envoyer un message à un objet
- serveur :
  - les fonctions BD
  - maintenance intégrité (acyclicité graphe d'objets)
- client : gestion interfaces et outils L4G

Bilan Architecture serveur d'objets

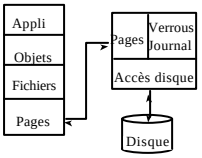
- Avantages
  - verrouillage et journalisation niveau objet : accès concurrent à 2 objets d'1 même page
  - serveur gère BD : sélection avant transfert
    ex : select sur grande collection
- Inconvénients
  - accès par objet au serveur (évitable grâce au cache)
  - opérations utilisateurs (requêtes) sur le serveur : menace sécurité et fiabilité
  - excessive centralisation : ralentissement client

Architecture serveur de pages

majeur partie fonctions BD sur Client
unité transfert = page
utilisée dans SGBDOO récents **Objectstore, O_2**



Architecture serveur de pages (2)



Client : gère Objets et Fichiers
Serveur : gère accès Disque
Client et Serveur : même couche Pages

cache de pages : verrouillage et journalisation au niveau page

Interface Client-Serveur :
- allouer, désallouer des pages
- lire ou écrire des pages

Bilan architecture serveur de pages

- Avantages :
  - gestion fichiers et objets : consommatrice en calcul ⇒client
  - + gd nombre de clients servis
  - serveur isolé des pannes
- Inconvénients :
  - requêtes ensemblistes : transfert toute la collection
  - verrouillage et journalisation niveau page : ↯ concurrence multi-utilisateur
- serveur de page ou serveur d'objets
  - grande mémoire principale et forte localité de référence : serveur de page > serveur d'objets

Architecture multi-serveur

- solution hybride :
  - certaines fonctions de couche objet duppliquées sur serveur de pages
  - ↯ verrouillage niveau objet
- chaque machine utilisable
  - en client
  - ou en serveur
  - ou les deux
- unité de transfert : page ou objet

Architecture multi-serveur : la couche «objets répartis»

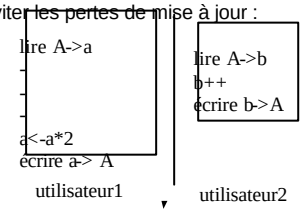
- transforme objets non locaux en appel aux serveurs concernés
- basée sur catalogue = info sur répartitions des objets
- catalogue dupliqué sur chaque serveur
- en général :
  - unité localisation= base d'objets (et pas l'objet)
  - ex: GemStone de Ontos un objet ne peut pas référencer un objet non local

Chapitre 5 : Implémentations des SGBDO

2- gestion des transactions : concurrence et pannes

Pertes de mise à jour

- éviter les pertes de mise à jour :



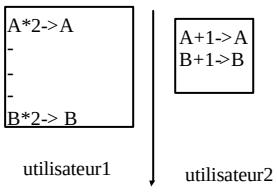
↯ la mise à jour utilisateur2 est perdue

Incohérence

- mises à jour intercalées de données

↳ violation de contrainte d'intégrité

CI : A=B



↳ CI est violée !

Gestion des transactions

- verrouillage deux phases :
  - objets verrouillés avant d'être fournis à l'application et relâchés en fin de transaction
  - technique la plus utilisée
- journalisation avant écriture
- contrôle de concurrence optimiste
  - accès aux objets sans contrôle, validation en fin de transaction (annulation des transactions accèdant à un objet dans un mode incompatible)
- GemStone : choix entre verrouillage 2 phases et contrôle de concurrence optimiste

O_2Technology

Les transactions chez O2

- transaction = unité de modification de la base
  - validée : modifications répercutées sur disque.
  - avortée : l'état de la base n'est pas modifié.

  - 2 personnes ne doivent pas être capables de **modifier** le même objet simultanément

  - si une personne utilise un objet, en **consultation**, il ne faut pas que quelqu'un d'autre puisse alors le **modifier**.

- le système O_2 gère des **verrous en lecture** et des **verrous en écriture.**

O_2Technology

Mode de verrouillage O2

- Plusieurs utilisateurs peuvent **lire** en même temps la même donnée (plusieurs verrous en **lecture** peuvent y être posés simultanément).
- Un seul utilisateur peut la **modifier** (il doit être seul à l'avoir verrouillée en **écriture**, aucun autre verrou ne pourra être posé, même en lecture)
- La pose de verrou est faite automatiquement par le système:

```
transaction;
str= John->name;          -> verrou en lecture R de John
. . .
John->age++;              -> verrou en écriture W de John

Validation ou Abort       -> verrous relâchés
```


O_2Technology

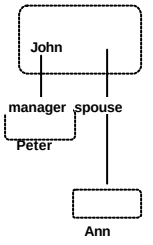
O2 : Evolution du mode de verrouillage

- donnée **lue** en transaction reste verrouillée en **lecture** jusqu'à :
  - une **écriture** dans la même transaction (elle devient alors verrouillée en écriture)
  - **la fin de la transaction**

- donnée **modifiée** en transaction reste verrouillée en **écriture** jusqu'à :
  - **la fin de la transaction**

O_2Technology

O2 : Exemple

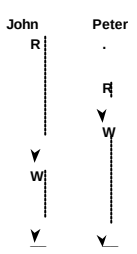



```
transaction;
Peter= John-> manager;

str=Peter -> name;

Peter ->age++;

John->salary+= 100;
Ann= new Person;
John->spouse= Ann;
validate;
```



O2 : Conflit d'acces attente

- Si un ou plusieurs utilisateurs lisent une donnée, l'utilisateur qui veut la modifier est mis en attente sur ressource ; il doit attendre que les autres accès en lecture soient terminés (fin des autres transactions).

USER1	USER2
en lecture sur A	en lecture sur A
veut modifier A	en lecture sur A
ATTENTE	
	VALIDATION
modifie A	

- Cette attente déclenche un signal qui peut être traité par l'application.

O2 : Conflit d'acces verrou mortel

- Si plusieurs utilisateurs sont en ATTENTE mutuelle de ressources, le système de verrous est alors bloqué. Le système est dit en "DEADLOCK".

USER1	USER2
en lecture sur A	en lecture sur B
veut modifier B	
ATTENTE	veut modifier A
	ATTENTE-> DEADLOCK
	ABORT
peut modifier B	

- DEADLOCK : un cycle d'attente.
- Le signal émis par le système peut être récupéré par l'application. Par défaut, c'est la transaction ayant créé le cycle qui sera avortée.

Transaction O2

- Une transaction commence par une instruction transaction;
- Elle se termine par une instruction de validation (validate ou commit) ou par une instruction d'annulation des modifications faites (abort).

validate:	enregistre les modifications du schéma et de la base, libère les verrous
commit:	fait un validate; libère la mémoire pour les objets temporaires O_2.
abort:	renonce à toutes le modifications faites depuis le début de la transaction; libère les verrous; libère la mémoire pour les objets temporaires O_2.

- Ces instructions peuvent être utilisées dans des programmes C++, C, O2C, O2API ou dans le langage de commandes.

Mode hors transaction O2

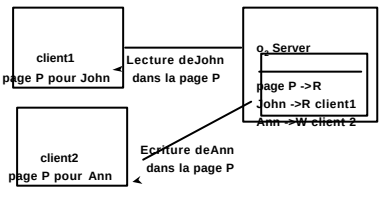
- Le mode hors transaction (ou mode programme) :
  - possibilité de lire les objets de la base, interdiction de les modifier.
  - les objets ne sont pas vérouillés ⇨ si des transactions les modifient, l'utilisateur n'en est pas informé.
- Le mode hors transaction est le mode par défaut des applications O_2.
- L'instruction **transaction** passe en mode **transactionnel**,
- La première instruction de validation (**validate** ou **commit**) ou **abort** remet en mode **hors transaction**.

Gestion du cache d'objets O2

O_2	Cache Client	Cache Serveur
		Ann_0
transaction;		
Ann= John->spouse;	**Ann_0**	Ann_0
Ann->age++;	**Ann_1**	Ann_0
validate;	Ann_1	**Ann_1**
	Ann_1	**Ann_2**
age= Ann->age;	Ann_1 **!!!!!**	Ann_2
transaction;		
name= Ann-> name;	**Ann_2**	Ann_2
Ann->salary+= 100;	**Ann_3**	Ann_2
commit;		**Ann_3**

Verrouillage page/objet O2

- Par défaut, demande d'un objet par client : verrou sur la page (le client garde en mémoire le lien verrou-objet)
- En cas de conflit, le serveur descend le verrouillage au niveau de l'objet.



3. Quelques SGBDRO

- Odaptor (HP, 1992, C)
- Postgres (Université Berkeley, 1992, C)
- Illustra (Informix)
- UniSQL

Quelques SGBDOO

- GemStone
- ObjectStore
- Orion
- Ontos
- Ode
- Versant

Bilan

- Domaines visés => architecture SGBDO
- serveur de pages :
  - meilleur exploitation puissance postes de travail
  - ↳ applications coopératives comme l'ingéniérie
- serveur d'objets :
  - verouillage au niveau objet
- architecture idéale : multiserveur de pages avec verrouillage niveau objet

Bilan (2)

- Extensions des SGBDR : nécessaire
  objets volumineux, ramasse miettes, transactions longues, optimisation requêtes
- SGBDRO : évolution SGBDR vers objets
  SGBDR : outil déjà bien adaptés
- SGBDOO : meilleurs pour applications complexes
- des SGBDOO veulent s'intégrer SGBDR
  - Informix et Illustra
  - Unidata et O2
  - ↳ intérêt de cumuler avantages des 2 approches
- ↳ standards :
  SGBDRO convergent vers SQL3, SGBDOO vers ODMG

FIN