Pratique d’un SGBD relationnel

1. Introduction aux différentes architectures des SI

Trois tâches importantes

- le stockage des données,
- la logique applicative,
- la présentation.

Parties indépendantes les unes des autres.

Toutefois, une couche est utilisée dans celle d'au-dessus : la conception de la logique applicative se base sur le modèle de données et la conception de la présentation dépend de la logique applicative.

Stockage et accès aux données

- Il a pour but de conserver les données de façon structurée en permettant le partage des données via un réseau.
- La méthode d'accès à ces données dépend du type d'organisation de ces données. Dans le cas d'une base de données relationnelle, l'accès peut se faire par des API qui dépendent du langage et de l'environnement.
- Quelle que soit l'API, le langage SQL est utilisé.

Logique applicative

- Traitements nécessaires sur les données afin de les rendre exploitables par chaque utilisateur (besoins variés et évolutifs).
- Nécessité de permettre l'évolution du système sans pour autant devoir tout reconstruire.
- Utilise les données pour les présenter de façon exploitable par l'utilisateur.
- Bien identifier les besoins des utilisateurs afin de réaliser une logique applicative utile tout en structurant les données utilisées.

Présentation

- Partie visible pour l'utilisateur.
- L'ergonomie d'un site Intranet HTML n'est pas toujours idéale pour tous les types d'applications. Il peut être intéressant de proposer plusieurs types d'interface pour une seule logique applicative (par ex: client en html, admin en applet, …).

Architecture client-serveur

- Constitué de 2 parties : un client gèrant la présentation et la logique applicative, un serveur stockant les données et gèrant une partie de la logique applicative.
- Dans cette architecture à 2 niveaux (architecture 2-tier), le client demande une ressource et le serveur la lui fournit sans faire appel à une autre application.
- L'interface entre ces 2 parties est le langage SQL.
- Indépendance du client par rapport au serveur : la programmation du client effectue sans se préoccuper de la base de données (taille des disques, répartition des données, optimisation des accès, sauvegarde, etc.).

Architecure c-s : avantages

- Qualité basée sur 2 technologies éprouvées : BDR et SQL.
- SGBD relationnels interfacés par un langage unique : SQL.
- SQL permet la gestion de transactions. 4 propriétés garantissent l'intégrité des données dans un environnement multi-utilisateurs (ACID) :
  - Atomicité : comportement indivisible ; soit toutes les modifications sur les données d'une transaction sont effectives, soit aucune. Garantit un état cohérent de la base.
  - Cohérence des données de la base.
  - Isolation : les modifications effectuées au cours d'une transaction visibles que par l'utilisateur qui effectue cette transaction. Ces modifications seront visibles par tous si la transaction est correcte.
  - Durabilité : stabilité de l'effet d'une transaction dans le temps (même en cas de perte d'un disque).

Architecure c-s : inconvénients

2 inconvénients principaux :

- difficulté à bien gérer les questions de sécurité : sécurité gérée au niveau du SGBDR contrôlant l'accès aux données par des autorisations aux utilisateurs.
  - Il faut accorder à chaque utilisateur des droits sur un grand nombre de tables → laborieux.
  - Ne créer qu'un utilisateur avec lequel tous les utilisateurs se connectent → aucune gestion de la sécurité (login + mot de passe pour accéder aux données sans restriction)
- durées et coûts de déploiement : installation et configuration sur chaque poste utilisateur.
  Mise à jour nécessite un nouveau déploiement.

Architecture 3-tiers

- Séparer la réalisation des trois parties :
  - un SGBDR pour le stockage des données,
  - un serveur applicatif pour la logique applicative,
  - un navigateur web pour la présentation.

- Architecture partagée entre :
  - le client qui est le demandeur de ressources,
  - le serveur d'application (middleware) chargé de fournir la ressource en faisant appel à un autre serveur,
  - le serveur secondaire (serveur de base de données), fournissant un service au premier serveur.

Architecture 3-tiers (suite)

- Essentiel du développement implanté au niveau du serveur applicatif.
- SGBDR nécessite un travail d'administration car quantité de données importante.
- Conception de la base de données est la pierre angulaire du système.
- Navigateur web nécessite un code spécifique permettant de gérer l'affichage. Code placé sur le serveur applicatif pour permettre une mise à jour sans nouveaux déploiements.

Architecture 3-tiers : avantages

- Architecture développée au sein des entreprises.
- Système basé sur des technologies éprouvées : aspect relationnel et transaction.
- Logique applicative déplacée au niveau du serveur d'application (maintien du SQL).
- Deux facteurs de qualité : choix du SGBDR et des programmes de conception et de gestion de la base.

Architecture 3-tiers : avantages

- Facilité de déploiement
  - application déployée que sur la partie serveur (serveur applicatif et serveur de base de données).
  - installation et configuration minimes pour le client. Navigateur web compatible avec l'application. Evolution régulière du système.
- Amélioration de la sécurité
  - dans un système client-serveur tous les clients accédent à la base de données → vulnérable.
  - avec une architecture multi-tiers l'accès à la base n'est effectué que par le serveur applicatif.
  - gérer la sécurité au niveau de ce serveur applicatif : liste des utilisateurs avec leurs mots de passe et leurs droits d'accès.

Architecture n-tiers

- Architecture distribuée découpe l'application sur différents serveurs : la gestion des données, la gestion de la logique applicative et le client pour la présentation.
- Permettre l'évolutivité du système : quantité de données stockée, disponibilité du serveur, nombre d'utilisateurs, etc.
- Système indépendant du serveur sur lequel il s'exécute.
- 2 types de répartition dans une architecture distribuée : répartir les données et répartir la logique applicative.

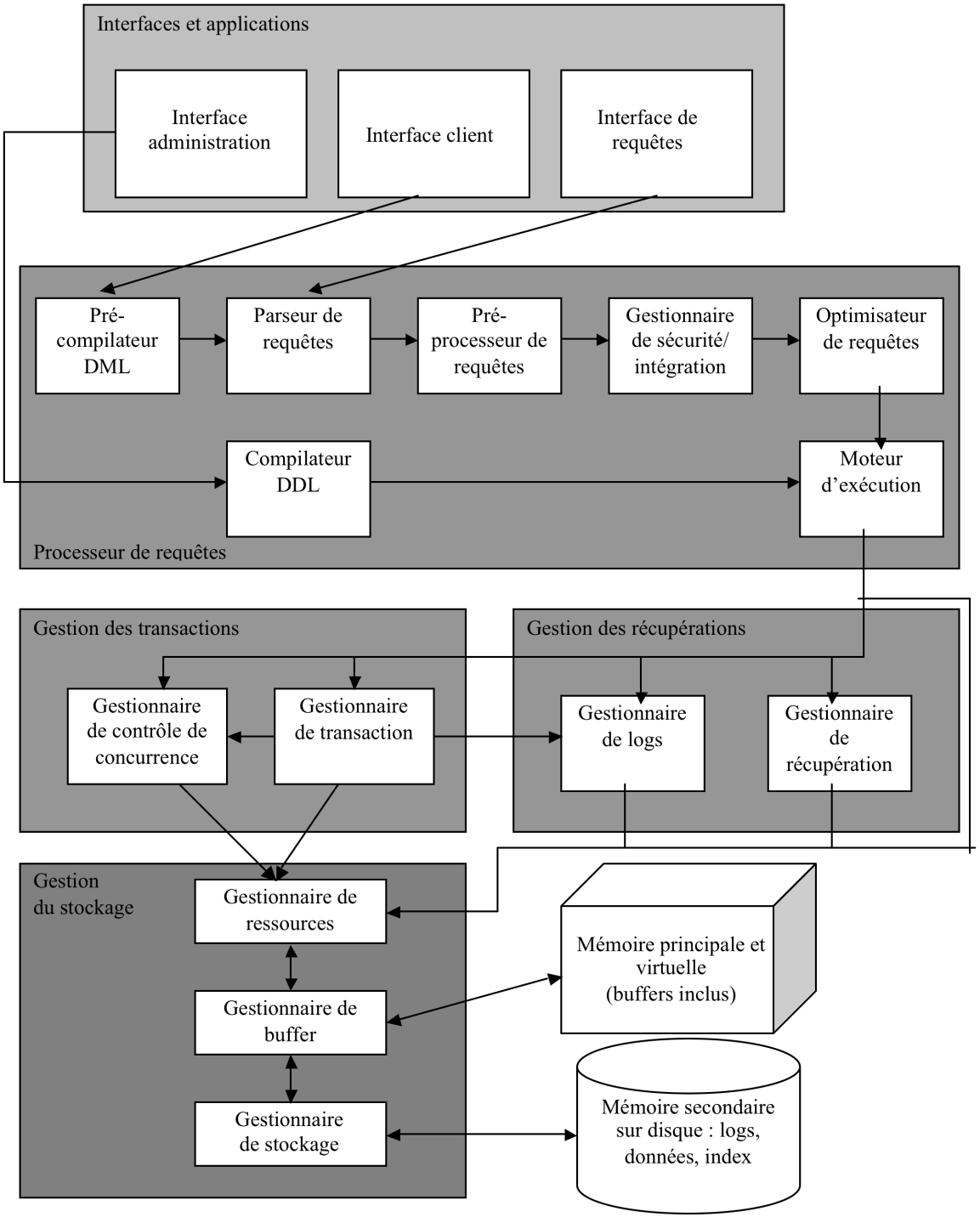
Architecture n-tiers : répartir les données

- Plusieurs sources de données dans une même application avec différentes structures et gestions.
- Interface de programmation commune aux sources.
- Si relations entre les données des différents serveurs, alors nécessité de gérer des transactions distribuées.
- Base de données distribuées
  - performance du système : augmenter la disponibilité des données. Eviter de démultiplier les transactions distribuées.
  - réutilisation des systèmes existants dans une même application afin de restructurer le SI sans repartir de zéro mais nécessitant diverses technologies.

Architecture n-tiers : répartir la logique applicative

- Distribuer les traitements sur différentes machines.
- Programmation orientée objets : composants instanciés puis utilisés à travers le réseau.
- Communication entre les différents éléments de l'application (RMI).
- Déploiement et évolution du système : système distribué (EJB).
- Les buts recherchés : la performance, l'évolutivité et la maintenabilité.

2. Architecture MySQL


Interfaces et applications
Interface administration
Interface client
Interface de requêtes
Pré-compilateur DML
Parseur de requêtes
Pré-processeur de requêtes
Gestionnaire de sécurité/ intégration
Optimisateur de requêtes
Compilateur DDL
Moteur d'exécution
Processeur de requêtes
Gestion des transactions
Gestionnaire de contrôle de concurrence
Gestionnaire de transaction
Gestion des récupérations
Gestionnaire de logs
Gestionnaire de récupération
Gestion du stockage
Gestionnaire de ressources
Gestionnaire de buffer
Gestionnaire de stockage
Mémoire principale et virtuelle (buffers inclus)
Mémoire secondaire sur disque : logs, données, index

Architecture MySQL
Couche applicative

Echanges avec le client

3 utilisateurs possibles :

- les administrateurs utilisant les utilitaires administratifs
  - *mysqladmin* (fermer un serveur, créer ou détruire des bases)
  - *isamchk* et *myisamchk* (réparations de tables)
  - et *mysqldump* (sauvegarder des bases ou en copier à partir d'autres serveurs)
- les clients communiquant avec le SGBDR par des APIs (C, Perl, PHP, Java, Python, C++) ;
- et les utilisateurs SQL interagissant avec le SGBDR grâce à *mysql* (requêtes SQL et visualiser les résultats)

Couche logique

Le processeur de requêtes (effectue et optimise les requêtes)

- le pré-compilateur DML (Data Manipulation Language) qui extrait la requête d'une commande de l'API
- le compilateur DDL (Data Definition Language) qui compile les commandes des administrateurs écrites en SQL
- le parseur qui transforme la requête en plan d'exécution
- le pré-processeur qui effectue une analyse syntaxique et sémantique du plan (arbre)
- le gestionnaire de sécurité qui contrôle les droits d'accès du client à la bas
- l'optimisateur de requêtes qui tente d'optimiser la requête
- le moteur d'exécution réalise la requête en accédant à la couche physique.

Couche logique

La gestion de transactions

- le gestionnaire de transactions (unité élémentaire de travail comportant une ou plusieurs commandes) gère la continuité du traitement et fait les réparations nécessaires en revenant au dernier état stable de la base (commandes COMMIT et ROLLBACK).
- le gestionnaire de concurrence contrôle l'indépendance des transactions. Il utilise des serrures sur les tables en cours de modification pendant une transaction.

Couche logique

La gestion des récupérations

- le gestionnaire de logs enregistre toutes les opérations effectuées sur la base. Si le système plante, on peut réeffectuer l'ensemble des commandes sur la dernière version stable.
- le gestionnaire de récupération a pour but de remettre la base dans sa dernière version stable en utilisant le log.

Couche logique

La gestion du stockage (grâce aux buffers)

- Le gestionnaire de stockage est au plus bas niveau. Il véhicule la requête du gestionnaire du buffer vers la mémoire secondaire.
- Le gestionnaire de buffer alloue la mémoire pour l'utilisation et la visualisation des données : taille allouée par buffer et combien de buffers par requêtes.
- Le gestionnaire de ressources accepte les requêtes du moteur et les envoie au gestionnaire de buffer. Il reçoit les références à des données en mémoire et retourne ces données au niveau supérieur.