Pratique d’un SGBD relationnel

1. Différentes architectures des SI

Trois tâches importantes

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

Parties indépendantes les unes des autres.

Couche utilisée dans celle d'au-dessus : conception de la logique applicative basée sur le modèle de données et conception de la présentation dépend de la logique applicative.

Stockage des données

- BUT : conserver les données de façon structurée en permettant le partage des données via un réseau.
- Méthode d'accès dépend du type d'organisation de ces données. Pour BDR, 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

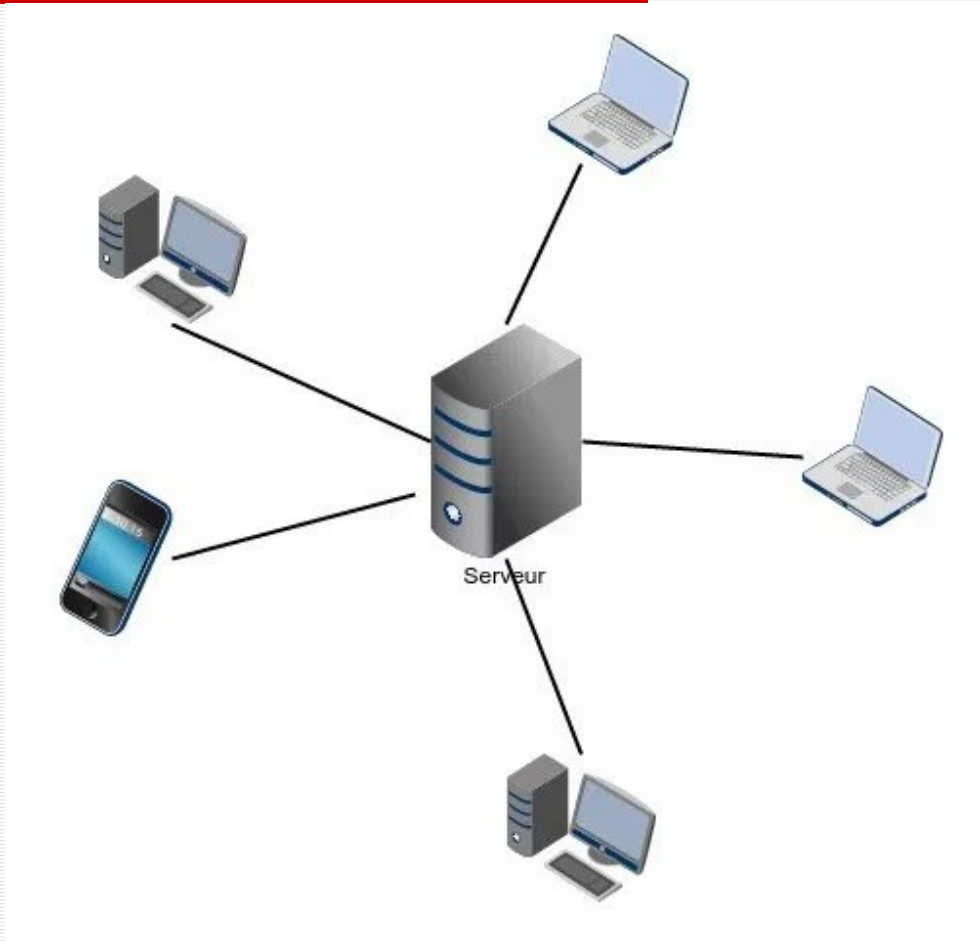
- Traitements nécessaires sur les données afin de les rendre exploitables par chaque utilisateur (besoins variés et évolutifs).
- Permet l'évolution du système sans pour autant devoir tout reconstruire.
- Utilise les données pour les présenter de façon exploitable par l'utilisateur.

→ Bien identifier les besoins utilisateurs pour réaliser une logique applicative utile tout en structurant les données.

Présentation

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

Architecture client-serveur (2-tiers)



Architecture client-serveur

- 2 parties
  - un **client** gère la présentation et une partie de la logique applicative,
  - un **serveur** stocke les données et gère une autre partie de la logique applicative.
- Client demande une ressource et le serveur la fournit sans faire appel à une autre application.
- Interface entre ces 2 parties est le langage SQL.
- Indépendance du client par rapport au serveur : programmation du client sans se préoccuper de la BD (taille des disques, répartition des données, sauvegarde, etc).

Architecure c-s : avantages

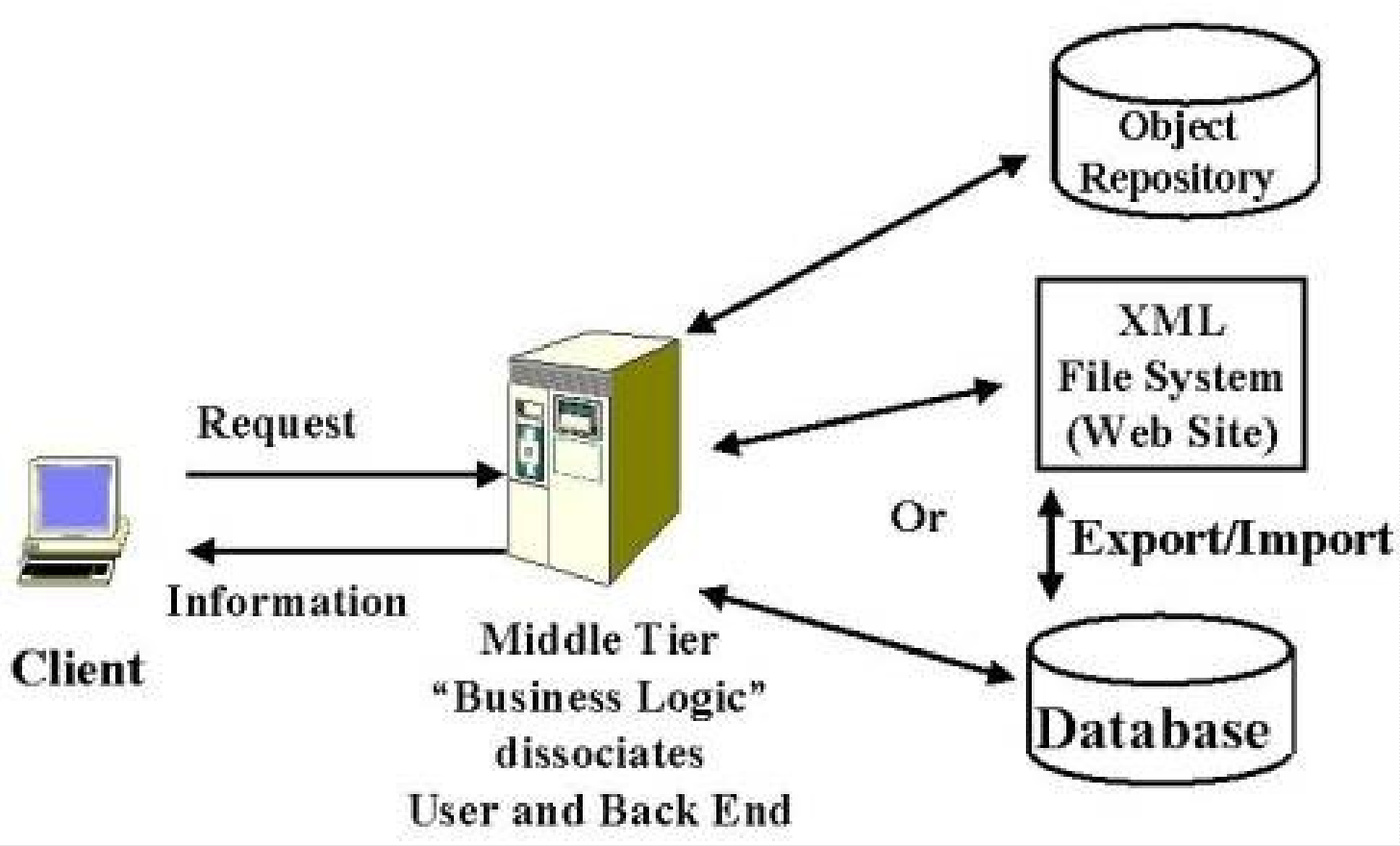
- SGBD relationnels interfacés par un langage unique : SQL
- Permet la gestion de transactions.

  ACID : 4 propriétés pour l'intégrité des données dans un environnement multi-utilisateurs
  - Atomicité : comportements indivisibles, cad soit toutes les modifications d'une transaction sont effectives, soit aucune.
  - Cohérence des données de la base.
  - Isolation : modifications effectuées au cours d'une transaction visibles que par celui qui l'effectue. Ces modifications 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

- Difficulté à bien gérer les questions de sécurité : accès aux données par des autorisations aux utilisateurs.
  - 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. Chaque mise à jour nécessite un nouveau déploiement.

Architecture 3-tiers



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.

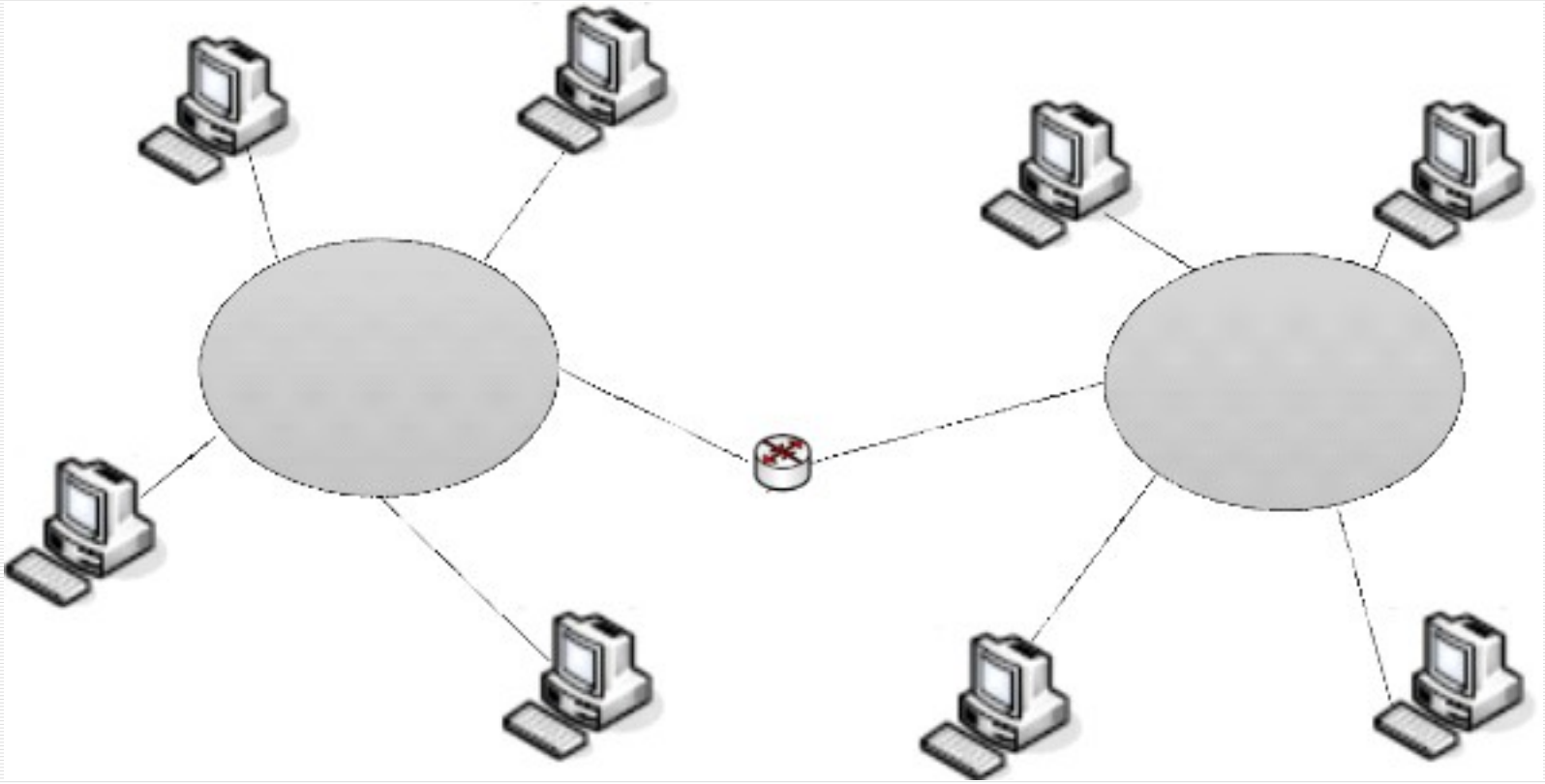
Architecture 3-tiers : avantages

- 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 (distribuée/répartie)



Architecture n-tiers

- Application sur différents serveurs : gestion des données, gestion de la logique applicative et client pour la présentation.
- Evolutivité du système : quantité de données stockée, disponibilité du serveur, nombre d'utilisateurs, etc.
- Système indépendant du serveur sur lequel il s'exécute.
- 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 : systèmes distribués (EJB).
- Buts recherchés

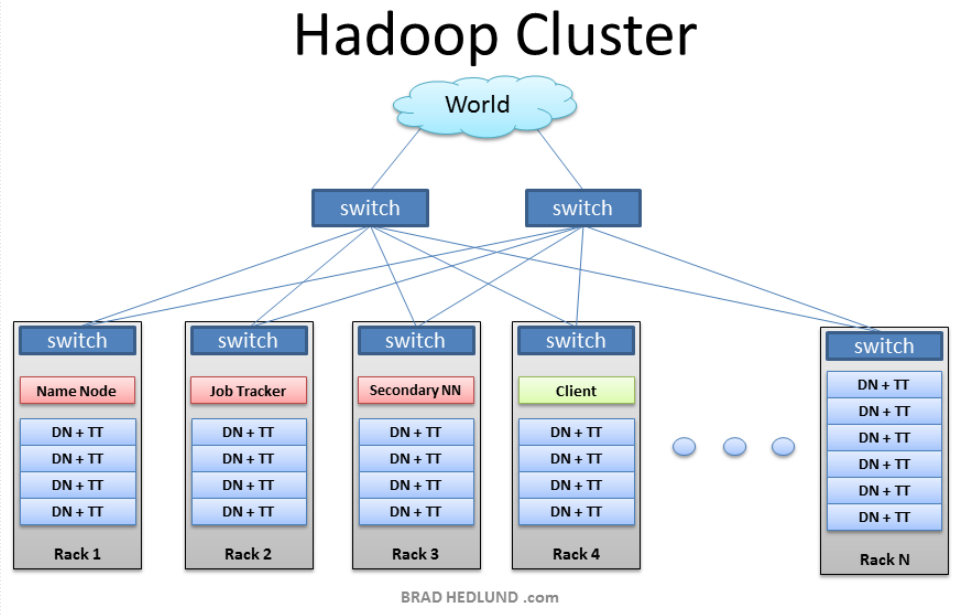
  performance, évolutivité et maintenabilité.

Architecture n-tiers : exemple Hadoop (1)

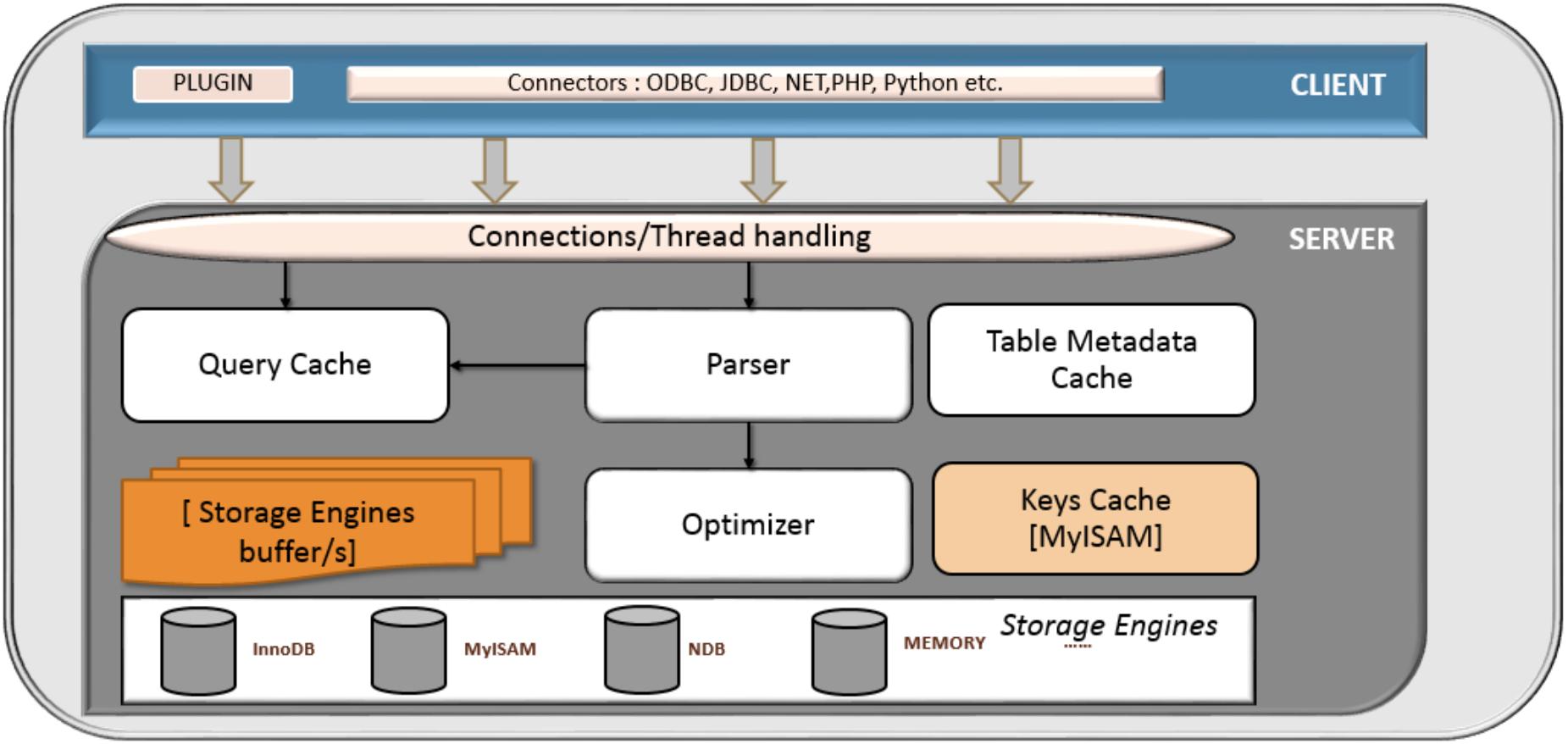
- *High-Availability Distributed Object-Oriented Platform*, framework de référence, libre et open source
- Créé par Doug Cutting en 2002, projet indépendant de la fondation Apache depuis 2008
- Données de très grande taille gérées par de très nombreuses machines (data centers).
- Depuis 2013, Facebook détient le record du plus large cluster Hadoop. Parmi les autres clusters d'envergure : Google, IBM.

Architecture n-tiers : exemple Hadoop (2)

- Hadoop fractionne les fichiers en gros blocs distribués à travers les nœuds du cluster.
- NameNode et JobTracker sont les nœuds maîtres. Les autres serveurs font à la fois office de DataNode et de TaskTracker (nœuds esclaves).



2. Architecture MySQL (client/serveur)



Architecture MySQL : le client

Echanges avec le client, 3 utilisateurs possibles :

- les **administrateurs** utilisant les utilitaires
  - *mysqladmin* (fermeture d'un serveur, création ou destruction des bases)
  - *isamchk* et *myisamchk* (réparations de tables)
  - *mysqldump* (sauvegardes ou copies des bases)
- les **clients** communiquant avec le SGBDR par des APIs (C, Perl, PHP, Java, Python, C++)
- les **utilisateurs** SQL (requêtes MySQL)

Architecture MySQL : processeur de requêtes (serveur)

Effectue et optimise les requêtes

- Gestion des sessions : authentification du client, et extraction de la requête de l'API, si nécessaire
- Parseur : effectue une analyse syntaxique et sémantique de la requête
- Optimisateur transforme la requête en plan d'exécution et tente de l'optimise . Contrôle aussi les droits d'accès du client à la base.
- Le moteur d'exécution réalise la requête en accédant à la couche physique (storage engine).

Architecture MySQL : gestion du stockage (serveur)

- Au plus bas niveau, véhicule la requête des buffers 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 envoie les requêtes au gestionnaire de buffer, reçoit les références à des données en mémoire et retourne ces données au niveau supérieur.