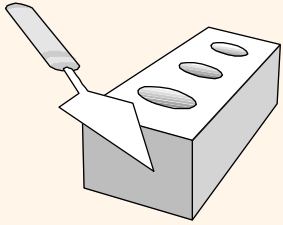


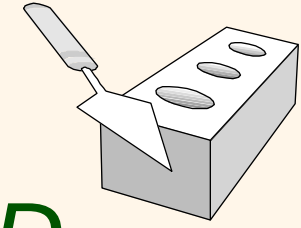
Bases de Données Avancées

Ioana Ileana

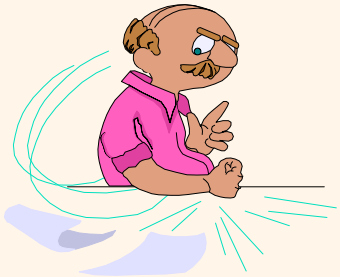
Université Paris Descartes

Cours basé sur le livre (et les diapositives de):
Database Management Systems 3ed, R. Ramakrishnan et J. Gehrke

LIPADE
Laboratoire d'Informatique PAris DEscartes

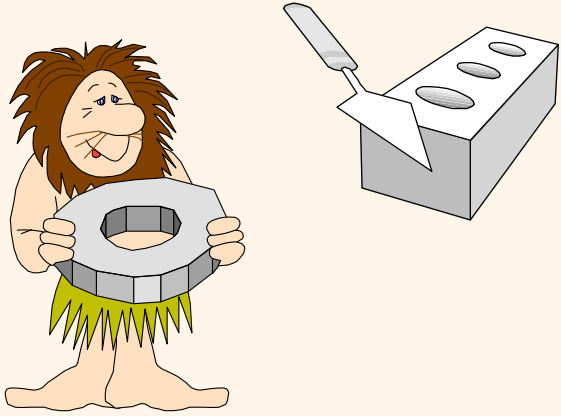


Cours 1: Intro / rappels SGBD

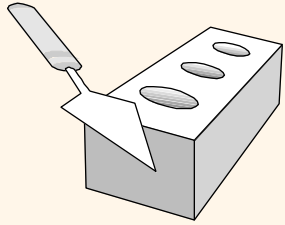


- Diapos traduites et adaptées du matériel fourni en complément du livre Database Management Systems 3ed, par Ramakrishnan et Gehrke ; un grand merci aux auteurs pour la réalisation et la disponibilité de ce matériel !
- Les diapos originales (en anglais) sont disponibles ici :
  http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- Plus particulièrement, ce cours touche aux éléments dans le Chapitres 1 et 8 du livre ci-dessus

Qu'est-ce qu'un SGBD?



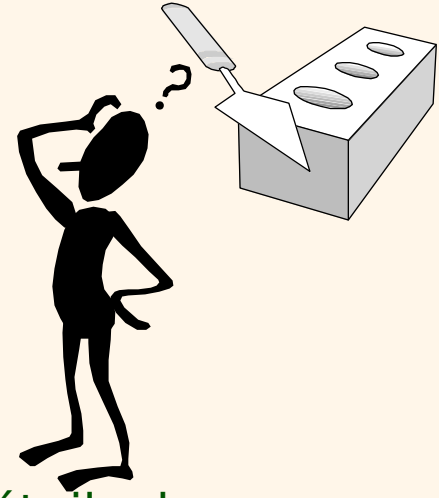
- Base de données (BD): une grande collection de données « intégrées » / « reliées »
  - Qui modélise souvent des entreprises et organisations
    - Entités (étudiants, cours, produits, clients)
    - Associations (un étudiant s'inscrit dans un cours, un client achète un produit...)
- *Un Système de Gestion de Bases de Données (SGBD), en anglais DBMS (Database Management System)* = ensemble logiciel conçu pour stocker et gérer les bases de données



Pourquoi pas des fichiers tout simplement?

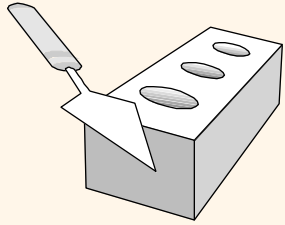
- Une application concernée par la gestion des données devrait assurer / comporter:
  - Le déplacement de préférence efficace ;) de gros volumes de données entre la RAM et le disque
  - Du code spécifique pour différents types de requêtes (en: queries) sur les données
  - La protection de la consistance / cohérence des données en cas d'accès concurrent / utilisateurs multiples
  - La récupération / le rétablissement des données en cas de pannes / incidents / « plantages » (en: crashes)
  - La sécurité et le contrôle d'accès aux données suivant le spécifique et la granularité de ces données
- → Les fonctionnalités fournies par le système de fichiers = souvent inadaptées ou insuffisantes!

Avantages d'un SGBD



- Indépendance des données
  - Les applications ne sont pas concernées par les détails de la représentation / du stockage des données
- Accès efficace
  - Tout un tas de techniques spécialisées / sophistiquées..
- Intégrité et sécurité des données
- Administration uniforme des données
- Gestion de l'accès concurrent et de la reprise / récupération après panne / incident (crash recovery).
- Moins de temps de développement applicatif
  - Et moins de temps passé à réinventer la roue ;)

☛*Désavantages / limitations ?*



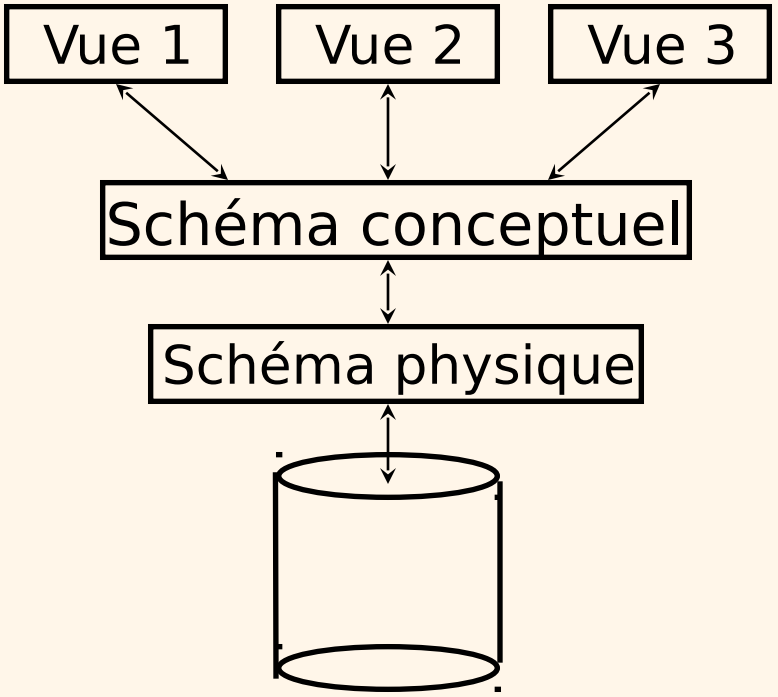
Modèles de Données

- Un *modèle de données (en: data model)* est un ensemble de concepts utilisés pour décrire des données
  - Censé cacher beaucoup des détails « bas-niveau » du stockage
- Un *schéma (en: schema)* est une description d'un ensemble (jeu) de données spécifique, qui s'appuie sur un modèle de données
- Le *modèle relationnel (en: relational model)* reste le modèle de données le plus utilisé de nos jours
  - Concept principal: la *Relation*
    - Table avec des lignes et des colonnes
    - Les lignes sont appelées aussi tuples, ou enregistrements (records) ; les colonnes sont aussi appelées champs
    - Chaque relation a un *schéma*, comprenant son nom et la description de ses colonnes (nom, type)
  - **Ce cours se focalisera sur le modèle relationnel et les SGBD relationnels (SGBDR ; en: RDBMS)**

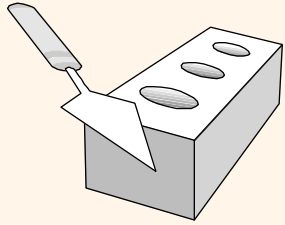
Cours basé sur le livre (et les diapositives de):
Database Management Systems 3ed, R. Ramakrishnan et J. Gehrke

Les 3 niveaux d'abstraction d'un SGBD(R)

- Plusieurs *vues (groupées dans des "schémas externes")*, un *schéma conceptuel (logique)* *et un* *schéma physique*.
  - Les vues décrivent comment les utilisateurs voient les données.
  - Le schéma conceptuel définit la structure logique (les relations ds la BD ...).
  - Le schéma physique définit les détails du stockage - les fichiers et indexes utilisés...

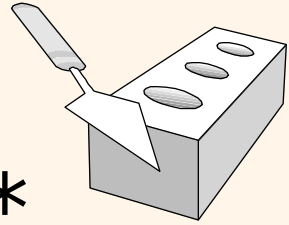


- *Les schémas sont définies avec un DDL*
- *Les données sont modifiées / intérrogées avec un DML*.



Exemple: La BD (relationnelle) d'une université

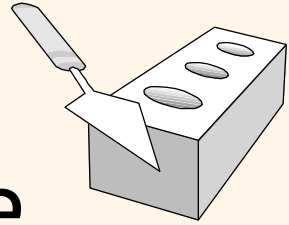
- Schéma conceptuel / logique :
  - Etudiants *(eid: string, enom: string, login: string, age: integer, moyenne: real)*
  - Cours *(cid: string, cnom: string, credits: integer)*
  - Inscrits *(eid: string, cid: string, note: string)*
- Schéma physique :
  - Relations stockées en tant que fichiers non-triés
  - Index sur la première colonne de *Etudiants*
- Schéma externe: vue Info_cours
  - Info_cours *(cid:string, nb_inscrits:integer)*



L'indépendance des données *

- Les applications sont "isolées", séparées de la manière dont les données sont structurées et stockées
  - Et des changements dans cette structuration !
- *Indépendance logique des données*: protection par rapport aux changements dans la structure *logique* des données (ex. rajout d'une colonne)
  - À l'aide de l'abstraction offerte par les schémas externes
- *Indépendance physique des données*: Protection par rapport aux changements dans la structure *physique* des données (ex. trier le fichier)
  - Le rôle du schéma logique !

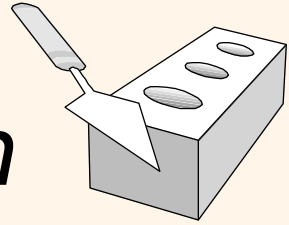
**Un des plus importants avantages dans l'utilisation d'un SGBD!*



Le contrôle de la concurrence

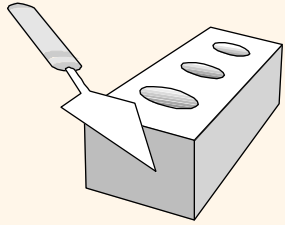
- L'exécution concurrente / "parallèle" des programmes utilisateur est essentielle pour la performance d'un SGBD
  - Les accès disque sont fréquents et relativement lents, il est donc important de garder le CPU actif en faisant tourner plusieurs programmes utilisateurs de manière concurrente!
- Mais "entrelacer" les actions de plusieurs programmes utilisateurs peut causer de l'inconsistance ! (ex. chèque encaissé pendant que le solde du compte est en train d'être calculé...)
- Le SGBD garantit que de tels problèmes d'inconsistance ne surviennent pas!
  - Chaque utilisateur peut considérer qu' « il est seul à interagir avec le SGBD » - pas besoin de « se synchroniser en amont » ou de prendre en compte qui d'autre modifie les données !

La transaction: l'exécution d'un programme BD

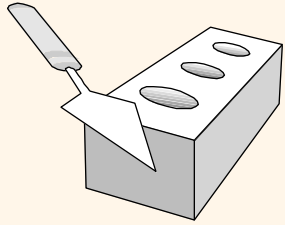


- Concept clé: *la transaction* = une séquence *atomique* d'actions sur la BD: lectures (reads) / écritures (writes).
- Chaque transaction, à la fin de son exécution, doit laisser la BD dans un *état consistant* - si la BD était consistante au début de la transaction.
  - Les utilisateurs peuvent spécifier des *contraintes d'intégrité* sur les données, et le SGBD garantira le respect de ces contraintes
  - Outre cela, le SGBD ne saura pas "comprendre" la sémantique des données (ex. comment les intérêts sur un compte sont calculés...)
  - → S'assurer qu'une transaction (exécutée toute seule) préserve la consistance est en fin de compte la responsabilité de *l'utilisateur*!

Gestion / ordonnancement des transactions concurrentes



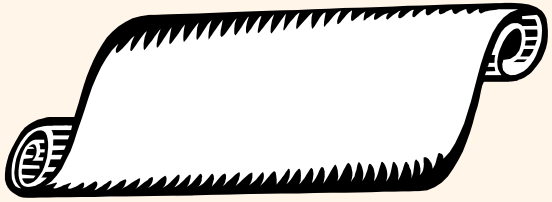
- Le SGBD garantit que l'exécution concurrente de {T1... Tn} est équivalente à une exécution *sériale* T1' ... Tn' en utilisant des protocoles de verrouillage (locking protocols)
- Esquisse du fonctionnement d'un protocole de verrouillage :
  - Avant de lire / écrire un objet, une transaction demande un verrou (lock) sur l'objet, et attend que le SGBD lui donne le verrou. Tous les verrous sont libérés / rendus (released) à la fin de la transaction.
  - L'idée derrière: Si une action de Ti (ex: écrire X) affecte Tj (qui lit X), une des deux transactions - par ex Ti, aura d'abord le verrou de X, forçant ainsi Tj à attendre la fin de Ti; ceci induit un ordonnancement des transactions.
  - Mais que se passe-t-il si Tj a déjà verrouillé Y et Ti demande plus tard un verrou sur Y? : (Deadlock!) Ti ou Tj sera annulée (aborted) et redémarrée (restarted)!

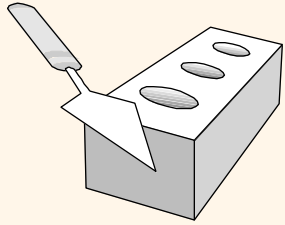


Atomicité des transactions

- Le SGBD garantit *l'atomicité* ("tout-ou-rien") d'une transaction même si crash au milieu de la transaction
  - Dans ce dernier cas, l'état de la BD sera retabli
- Idée: Garder un *journal (log)* (historique) de toutes les actions / modifications:
  - Avant qu'un changement soit fait dans la BD, l'entrée correspondante dans le journal est « sauvegardée » dans un endroit « sûr » (*WAL : Write Ahead Log*)
  - Après un crash, les effets des transactions qui n'ont été que partiellement exécutées sont *défaits / annulés* en utilisant le journal (grâce à WAL, si l'entrée correspondante du journal n'a pas été sauvée avant le crash, c'est que l'action n'a pas été faite sur la BD!)

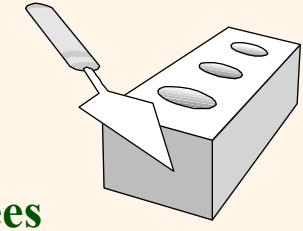
Le journal (log)





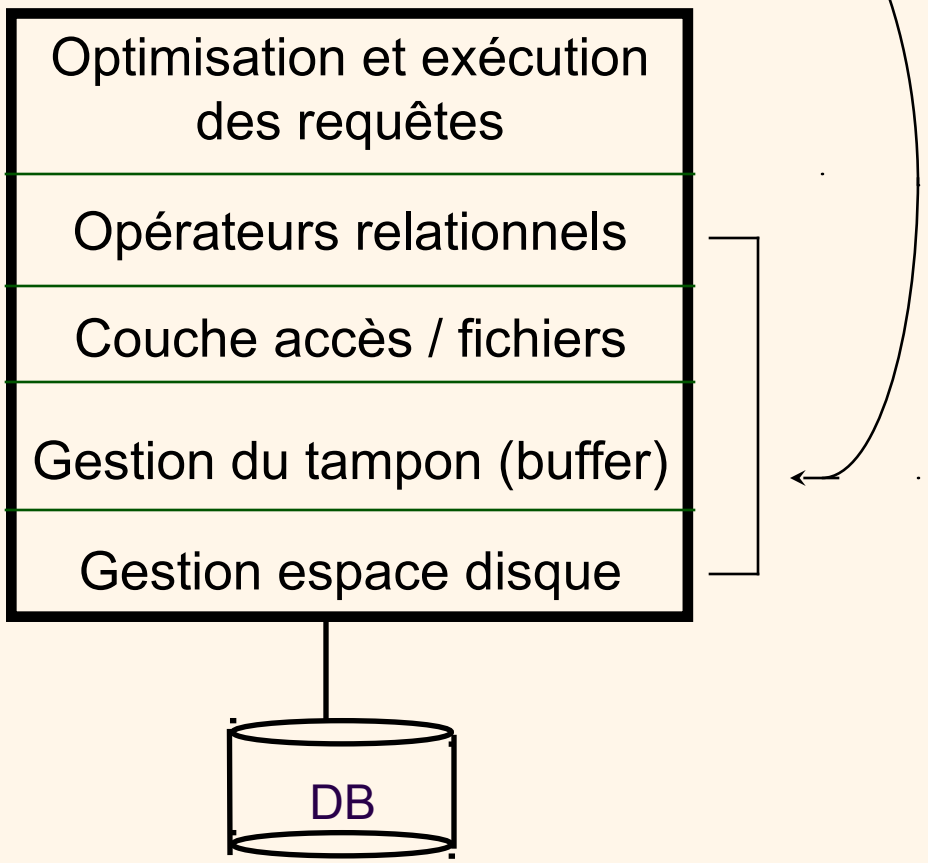
- Exemples d'actions / détails enregistrés dans le journal:
  - *Ti écrit un objet*: L'ancienne valeur et la nouvelle valeur
  - *Ti finit (commit) / est annulée (abort)*: Une entrée de journal correspondant à l'action
- Les entrées de journal comprennent l'ID de la transaction, ce permet d'annuler les effets d'une transaction (y compris en cours, dans le cas d'un deadlock !)
- Toutes les actions liées au journal (et au contrôle de la concurrence, comme le verrouillage / déverrouillage) sont gérées de manière transparente par le SGBD!

Structure d'un SGBD



Ces couches sont concernées par le contrôle de la concurrence et la reprise après crash

- Un SGBD(R) typique a une architecture “en couches” (niveaux, en: layers).
- La figure n'illustre pas les éléments de gestion de la concurrence et de reprise après crash
- Aussi, c’est une des architectures possibles; chaque système aura ses propres spécificités et variations!



Structure d'un SGBD

- La requête utilisateur, après parsing, est présentée à l'optimiseur, qui produit un *plan d'exécution de la requête* (arbre d'opérateurs relationnels)
- Les opérateurs relationnels sont « des briques d'exécution » qui font à leur tour appel à la couche accès / fichiers
- Plus bas niveau encore, on retrouve le gestionnaire du tampon (buffer) et le gestionnaire de l'espace disque
- Nous allons examiner tous ces aspects dans les cours suivants, en commençant par les trois couches « en bas de la pile »