Bases de Données Avancées : TP1

Pendant ce TP, vous allez démarrer en douceur votre projet : vous allez lire beaucoup de consignes, définir la structure globale de votre application… et coder un peu.

Dans ce descriptif TP ainsi que les suivants, les « choses à faire » sont classées en trois catégories :

- **Information** (à lire attentivement – pour agir conformément! ;))
- **Documentation** (chercher comment faire pour la suite ; pas de code dans l'immédiat) et
- **Code** (…).

A. Information : but et fonctionnalités de votre projet

Comme vous devez déjà le savoir, votre projet concernera l'implémentation d'un « MiniSGBDR », autrement dit un SGBDR (Système de Gestion de Bases de Données Relationnelles) très simplifié : mono-utilisateur, pas de gestion de la concurrence, pas de transactions, pas de droits d'accès, pas de crash recovery…

Les commandes que votre SGBD devra traiter seront dans un langage souvent plus simple que SQL, tout en correspondant à des « vraies commandes SQL » : insertion, sélection, etc. Il y aura de plus des commandes de type « debug », pour avoir des informations supplémentaires sur les données.

Vous allez progressivement travailler sur ces commandes au fil des TPs, après avoir codé les couches « bas-niveau » du SGBD. Tout ceci va suivre, autant que possible, le rythme du cours (nous allons incorporer progressivement des éléments vus en cours).

Ayez toujours en tête le « but final » de votre SGBD : celui de savoir gérer les commandes de création de tables, sélection, jointure, etc. Il est facile des fois de s'y perdre lorsqu'on travaille sur les couches bas-niveau !

B. Information : structure de votre projet, dossiers, fichiers, scripts etc.

Vous allez placer tout le code de votre projet dans un dossier principal appelé PROJET_BDDA. Ce dossier peut également contenir des projets correspondant aux diverses IDE (VS Code, Eclipse, IntelliJ…) que vous utilisez, et il est libre à vous de structurer son contenu comme vous le souhaitez.

Il est en revanche *impératif* que votre dossier contienne à la racine un ou plusieurs scripts qui permettent de compiler (si besoin, en fonction du langage que vous utilisez) puis de lancer votre SGBD depuis la ligne de commande. Par exemple, pour un projet Java, les scripts contiendront une invocation des commandes *javac* (pour compiler) et *java* (pour éxécuter).

Même si lors des TPs vous testez votre code en le lançant depuis votre IDE, l'absence de script impliquera une note 0 lors de l'évaluation finale.

C. Information : terminologie POO

Les descriptifs TP feront toujours référence à des classes / méthodes.
Si vous utilisez un langage qui n'est pas orienté objet, il faut adapter ces descriptifs au langage que vous utilisez: par exemple, en C, vous allez créer une struct à la place de la classe, et des fonctions qui prennent en argument supplémentaire un pointeur sur la struct à la place des méthodes…
Rappel : utilisez pour votre projet un langage que vous maîtrisez bien !

D. Information : application console et commandes

Votre projet sera une application console, sans interface graphique.

Le déroulement typique d'une utilisation de votre application sera : lancement dans le terminal via le script, puis série de commandes qu'on tape dans la console et auxquelles l'application réagit, ceci jusqu'à ce qu'on lui donne la commande **EXIT** (tout en majuscules).

Si vous souhaitez (mais ce n'est pas du tout obligatoire), vous pouvez afficher un « command prompt » (par exemple la phrase : « tapez votre commande (svp)») lorsque votre application est en attente d'une commande. Vous pouvez également rajouter (pour votre plaisir, pour la gloire, ou simplement pour apprendre plus de choses sur les GUI) une interface graphique en complément de l'utilisation console ; gardez toutefois à l'esprit que *cette GUI ne sera pas utilisée lors de l'éval.*

E. Documentation : git / GitHub / GitLab

Sans que cela soit obligatoire, il est vivement conseillé d'utiliser pour le travail sur votre projet un système de gestion de version comme Git, et une plateforme comme GitHub ou GitLab. Des liens vers des ressources de documentation Git sont disponible sur la page Moodle du cours, mais libre à vous d'utiliser la source de votre choix pour apprendre l'usage basique de ce gestionnaire de version et de GitHub / GitLab.

F. Code : DBConfig

Pour faciliter leur usage, les divers paramètres de configuration de votre SGBD seront regroupés dans une classe appelée DBConfig. Une instance de cette classe pourra être créée « directement » via le code, mais également construite en exploitant le contenu d'un fichier texte (donc « lisible par les humains »).

Créez une classe **DBConfig** qui comportera pour l'instant une seule variable membre, *dbpath*, de type chaine de caractères, correspondant au chemin vers le dossier de données de votre SGBD (nous en parlerons plus en détail lors des prochains TPs).
Rajoutez dans cette classe (au moins) les méthodes suivantes :

- un constructeur qui prend en argument une chaine de caractères correspondant au chemin *dbpath*.

- une méthode statique *config* **LoadDBConfig** (*fichier_config*), avec *config* une instance de **DBConfig**, et fichier_config un fichier texte.

  Vous avez le choix sur le format du fichier et la manière de lire son contenu (avec ou sans bibliothèques externes). Il est simplement demandé qu'en lisant le contenu de ce fichier vous soyez capable de construire / remplir un objet **DBConfig** correspondant à ce contenu. Par exemple, pour l'état courant de **DBConfig** tel que décrit ci-dessus, un fichier texte basique peut contenir simplement la ligne dbpath = '../DB'

  Un format choisi par vos soins convient très bien. Pour des choix plus standard, n'hésitez pas à considérer JSON ou TOML (https://toml.io/fr/v1.0.0).

  Nous allons progressivement enrichir le contenu de la classe **DBConfig** avec d'autres paramètres du SGBD ; ces paramètres devront être rajoutés dans le constructeur ainsi que dans les fichiers texte décrivant les configurations.

G. Code : Tests de DBConfig

D’abord un conseil à visée large: prenez le temps de bien tester chaque classe que vous codez !

Le code qui correspond à vos tests fait partie de votre projet ! Il ne doit pas être effacé ou mis en commentaire, et la bonne approche c’est de refaire exécuter tous les tests d’une « brique » du SGBD à chaque fois que vous modifiez quelque chose dans la brique en question ! Vous allez ainsi détecter très vite les *régressions* (= « *ça marchait avant mais je viens d’y introduire un bug* » ;))

Vous allez apprendre plein de choses sur les tests (et en particulier les tests unitaires) dans votre cours de Programmation Avancée. Pour les tests de votre projet BDDA nous ne vous demandons pas de respecter un quelconque framework ou formalisme ; vous êtes libres de faire vos tests comme vous le voulez – juste ne les zappez pas ! :)

Dans les tests de **DBConfig**, essayer de construire une instance en mémoire, ainsi que via un fichier texte, et testez également les cas d’erreur (par exemple, chemins incorrects etc.)