# Bases de Données Avancées : TP4

Au menu : début de la gestion du stockage d'une relation. Le code demandé lors de ce TP est entièrement « disjoint » du code des TPs 2 et 3, facilitant ainsi le travail « en parallèle ».

## A. Information : gestion d'une relation

Vous allez commencer à coder lors de ce TP des classes pour gérer une relation (=une table) et les records / tuples, permettant ainsi le rajout, la suppression et la consultation des tuples dans une relation. Pour cela, on utilisera deux classes : **Relation** et **Record**.

La classe **Relation** contiendra entre autres le *schéma* d'une relation, c'est à dire le nom de la table ainsi que le nom et le type de ses colonnes.

**IL Y A QUATRE TYPES DE COLONNES POSSIBLES DANS CE PROJET :**

- **INT** correspondant à un type entier sur 4 octets
- **REAL** correspondant à un type float sur 4 octets
- **CHAR(T)** correspondant à une chaîne de caractères de taille (nombre de caractères) *exactement* T
- **VARCHAR(T)** correspondant à une chaîne de caractères de taille variable, mais dont la taille *maximale* (nombre maximal de caractères) est T

Un tuple (record) sera simplement une liste de valeurs correspondant à une ligne dans la table.

## B. Code : la classe Record

Créez une classe appelée **Record** qui correspond à un record. Votre classe aura comme variable membre une liste de valeurs correspondant aux valeurs du tuple.

L'option la plus simple pour stocker l'ensemble de ces valeurs est d'utiliser une liste (ou tableau) de chaines de caractères.

Toutefois, si vous le souhaitez, il est possible d'utiliser pour les valeurs des types plus complexes, y inclus Object en Java, des classes faites maison, des enums etc. Le but de ces types c'est de pouvoir également consulter, directement sur un tuple, le type de ses valeurs, sans utiliser la relation à laquelle il appartient. Attention toutefois à la compatibilité de types et d'arité entre le record et la relation dans laquelle il sera rajouté !

## C. Code : la classe Relation

Créez une classe appelée **Relation** qui gardera un ensemble d'informations spécifiques à une relation. Nous allons enrichir le contenu de cette classe au cours des TPs suivants.

Initialement, cette classe devra stocker les informations de schéma de la relation, c'est à dire :

- le nom de la relation
- le nombre de colonnes
- les noms et les types des colonnes

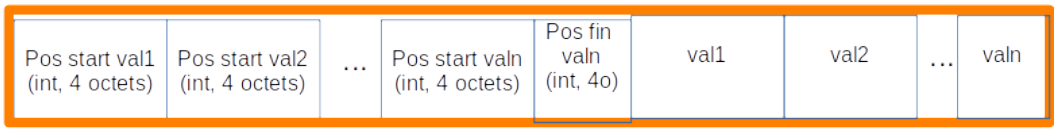
Vous pouvez stocker les noms des colonnes en tant que liste ou tableau de chaînes de caractères, et faire de même pour les types des colonnes.

*Si vous le souhaitez, vous pouvez aussi créer une classe supplémentaire appelée par exemple* ***ColInfo****, et mettre dans cette classe le nom et le type de la colonne ; puis stocker, dans la classe* ***Relation****, une liste ou un tableau de* ***ColInfo****.*

## D. Information : stockage des records dans les pages / buffers

Pour stocker les records dans les pages / buffers, nous allons utiliser des formats vus en cours, plus particulièrement :

- un format de record à taille fixe, où les valeurs sont simplement écrites les unes après les autres, si la relation ne contient pas de colonne de type **VARCHAR(T).**
- autrement, un format pour les records à taille variable: le offset directory. Pour ce format, nous allons donc utiliser une représentation comme ci-dessous :




Avant les valeurs elles-mêmes, nous stockons le *offset directory* : (n+1) entiers pour un record qui a n valeurs. Les n premiers entiers du *offset directory* correspondent aux positions ***relatives*** de début des valeurs du record. Le dernier correspond à la position (toujours relative) de fin de la dernière valeur.

## E. Code : écriture d'un record dans un buffer

Rajoutez à la classe **Relation** la méthode suivante :

int **writeRecordToBuffer**(*record, buff, pos*)

avec *record* un **Record** (dont les valeurs sont remplies correctement par l'appelant), *buff* un buffer et *pos* un entier correspondant à une position dans le buffer.

Cette méthode doit écrire les valeurs du **Record** dans le buffer en partant de la position *pos,* en utilisant le format à taille fixe ou variable tel que décrit ci-dessus.

Cette méthode doit rendre comme résultat la taille totale (=le nombre d'octets) écrits dans le buffer.

Pour savoir comment écrire les valeurs, il faudra potentiellement regarder les *types de colonnes* de la relation. Notamment, pour les colonnes de type INT ou FLOAT il faudra *convertir si besoin en int respectivement float* les éléments correspondants de valeurs du record (si vous les avez sous forme de chaîne de caractères). Pour la valeur « 123 » il faudra écrire l'entier 123 et non pas le caractère '1', suivi du caractère '2', suivi du caractère '3'.
Pour les vraies chaînes de caractères, en revanche, vous pouvez écrire caractère par caractère.

Pour écrire des valeurs à un endroit donné dans un buffer, ainsi que positionner ou consulter le « curseur » d'écriture / lecture, jetez un coup d'œil aux méthodes disponibles sur la classe **ByteBuffer**.

A partir du moment où vous utilisez des méthodes « correspondantes » de lecture et écriture (voir ci-dessous), vous n'êtes en principe pas censés avoir besoin de savoir le nombre d'octets occupés par int, float ou char).

### F. Code : lecture d'un record depuis un buffer

Rajoutez à la classe **Relation** la méthode suivante :

int **readFromBuffer**(*record, buff, pos*)

avec *record* un **Record** dont la liste de valeurs est vide et sera remplie par cette méthode, *buff* un buffer (alloué par l'appelant) et *pos* un entier correspondant à une position dans le buffer.

Cette méthode devra lire les valeurs du Record depuis le buffer à partir de *pos*, en supposant que le Record a été écrit avec **writeToBuffer.** C'est en quelque sorte « l'inverse » de la méthode précédente.

Pour lire des valeurs depuis un buffer, regardez les méthodes du **ByteBuffer**.

Cette méthode doit rendre comme résultat la taille totale (=le nombre d'octets) lus depuis le buffer.

*Testez vos méthodes en vérifiant que si vous lisez un record après une écriture à une position donnée, vous retrouvez bien les valeurs écrites ! En particulier, vérifiez la bonne écriture / lecture des chaînes de caractères.*