BUT Informatique – 2022-2023

**R1.05 P2**

# Exercice de Lecture (cours 1) : Intérêt des SGBD

Beaucoup d'applications informatique manipulent d'importantes masses de données. Le stockage optimisé sur support physique (Disque, RAM, mémoire cache), le contrôle des accès à l'information, le partage des données entre utilisateurs et applications, la manipulation de ces données (accès, création, mise-à-jour, suppression) gagnent à être réalisés par un logiciel ad hoc qui va garantir une haute performance, l'intégrité et la sécurité des données. Ces logiciels dédiés sont appelés « ***systèmes de gestion de base de données »*** (SGBD). Un SGBD gère une ou plusieurs « ***bases de données »***. Une base des données est un magasin de données composé physiquement de plusieurs fichiers manipulés exclusivement par le SGBD. Elle peut contenir par exemple les données persistantes d'une application web (par exemple celles du site marchand de la FNAC) ou d'une application de gestion comptable.

Un SGBD cache la complexité de manipulation des structures de la base de données et plus encore la structure réelle physique (le format des fichiers sur les disques durs et leur répartition) utilisée pour son stockage en mettant à disposition une vue synthétique du contenu aux utilisateurs et aux applications. La manipulation des données se fait : i) soit directement par un utilisateur depuis le SGBD à l'aide d'un éditeur supportant un langage de haut niveau (par exemple le langage SQL), soit depuis une application au travers d'une « ***API*** » (Application Programming Interface comme par exemple JDBC pour Java). Une API est un ensemble normalisé de classes, de méthodes et de constantes qui servent de guichet par lequel un logiciel (ici un SGBD) offre des services à d'autres logiciels. C'est une bibliothèque logicielle ou un service web, le plus souvent accompagné d'une documentation précise qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur. L'usage d'un SGBD délivre le développeur de nombreuses tâches fastidieuses et complexes, augmente sa productivité et garantit des temps d'exécution optimaux.

Les SGBD peuvent s'appuyer sur différents paradigmes de représentation de l'information pour la vue synthétique. Le plus utilisé est le « ***modèle relationnel »***. Nous présentons dans R1.05 ce modèle. Le modèle relationnel a été introduit par Codd, en 1970 au laboratoire de recherche d'IBM de San José. Il s'agit d'un modèle simple et puissant reposant sur une théorie mathématique solide. Les SGBD supportant ce modèle sont appelés « ***systèmes de gestion de bases de données relationnelles »*** (SGBDR).

## *Q1. Stockage en RAM des données*

La taille de la mémoire vive (RAM) d'un bon serveur PC est actuellement de 16Go. La bibliothèque de l'université de Bretagne Sud gère un fond documentaire composé d'environ 60.000 unités et environ 10.000 utilisateurs (étudiants et personnels). Chaque document nécessite en moyenne 500Ko de données à stocker (photo de couverture, scan, résumé, description, etc.). Un utilisateur, lui, impose le stockage en moyenne de 40Ko (photo, informations diverses).

a) Sachant qu'environ 5Go de la RAM du serveur de la bibliothèque sont pris par les applications et processus divers peut-on espérer gérer cette masse de données de la bibliothèque en RAM ?
b) Quel problème majeur pose de toute manière ce mode de stockage en cas d'arrêt de l'application ?

## *Q2. Stockage par fichiers sur disque*

On vient de le voir il est impérativement nécessaire pour des raisons de taille et de pérennité des données de réaliser un stockage complet sur disque des données persistantes d'une application. Non seulement on conserve les données même en cas d'arrêt d'une application mais on dispose alors de possibilités de stockage énormes : on peut en effet multiplier à loisir les disques durs ou SSD internes ou externes de plusieurs To. On va supposer ici qu'une application se contente d'un unique fichier texte sur le disque dur pour réaliser ce stockage.

Tous les langages de programmation généralistes (c'est le cas de Java) offrent des librairies pour manipuler des fichiers sur disque de données. Cette manipulation peut se faire sous la forme de flux d'octets, de caractères, d'objets, etc. Pour simplifier travaillons en mode flux de caractères : l'écriture et la lecture dans un fichier se fait via des caractères ou des chaînes de caractères. Les chaines lues dans un fichier peuvent ensuite être manipulées par les opérateurs classiques du langage de programmation notamment ceux permettant de connaître la taille d'une chaîne ou de la parcourir caractère par caractère.

On suppose ici que l'on dispose des méthodes qui suivent :

- Ouvrir un fichier : `Open(file) -> descripteur`
- Lire la prochaine ligne dans un fichier : `ReadLine(descripteur) -> String` (retourne « null » si la fin de fichier est atteinte)
- Fermer un fichier : `Close(descripteur)`
- Connaître la taille d'une chaîne : `Length(string) -> int`
- Accéder au caractère à la place spécifiée d'une chaîne de caractères : `ChartAt(string, int) -> char`

On s'appuie sur un ensemble de données très simple portant sur une bibliothèque. Il ne comporte que du texte et des informations chiffrées stockés dans un unique fichier de nom `Bibliotheque.txt`.

Ce fichier use d'un format « ligne » indiquant sur chacune d'entre elles plusieurs champs séparés par le caractère « ; » : `le numéro d'adhérent, la date de l'emprunt, le titre de l'ouvrage emprunté`.

On suppose enfin que les données ont été enregistrées dans ce fichier par date croissante. Voici une vue partielle de quelques lignes de ce fichier.

…

```
123;01/08/2021;Le messie de dune
110;05/08/2021;Dune
12;25/08/2021;L'empereur dieu de dune
123;01/09/2021;La maison des mères
3;01/09/2021;Dune
110;01/10/2021;Dune
```

…

a) Donnez de manière très informelle la trame de code à écrire en usant de la librairie ci-dessus pour déterminer depuis ce fichier : `la liste des adhérents ayant emprunté le livre « Dune » depuis le 01/06/2021`.

Vous observez que même une requête en apparence très simple nécessite l'écriture d'un code important. Tout cela n'est pas très intéressant… Le stockage des données persistantes d'une application dans des fichiers sur disque dont la gestion est laissée à la charge de cette même application est le plus souvent une très mauvaise idée : particulièrement si les données sont volumineuses, structurées, fréquemment manipulées ou nécessitant des requêtes à forte valeur ajoutée sémantique. Les SGBD offrent des langages de haut niveau et puissant pour réaliser ce type de recherche. Le parangon est le langage SQL que vous étudierez. Vous verrez par la suite que cette requête dans un SGBDR avec SQL s'écrit en une seule instruction de trois lignes de manière très simple !

b) L'algorithme de la question précédente impose de parcourir le fichier depuis son début jusqu'à la première ligne contenant une date supérieure ou égale à la date du 01/06/2021 avant de réellement étudier les lignes d'intérêt. Or le gestionnaire de la bibliothèque réalise ce type de bilan fréquemment sur un fichier qui comporte chaque jour de plus en plus de lignes (actuellement plusieurs dizaines de millier). Que pourrait-on faire pour limiter cet inconvénient ? Le gain de performance peut-il être important ?

Les SGBD offrent la possibilité de créer des « ***index »*** sur certaines informations qui servent fréquemment comme critère dans les recherches (par exemple ici la *date* ou le *titre de l'ouvrage*). Un index permet d'associer à chaque valeur stockée du domaine d'une donnée sa position dans le lieu de stockage. Ces index vont permettre d'accélérer considérablement les recherches. Cela rajoute cependant de la redondance pour mémoriser l'association valeur/lieu. Ce mécanisme est totalement transparent pour le développeur qui n'en voit que les bénéfices (l'accélération massive du temps de réponse à ses requêtes !). Un index posé sur la date permettrait ici au SGBD « d'associer » à chaque date une « place » dans le lieu physique de stockage. Lors d'une requête, le SGBD commencerait sa recherche sur les lignes répondant au critère de date sans perdre du temps à parcourir toutes les lignes correspondants aux dates antérieures. Le gain de temps peut ainsi être considérable !

## *Q3. Gestions multi-utilisateurs depuis un fichier*

On suppose que plusieurs applications accèdent un même fichier qui contient des données médicales sur des patients hospitalisés. Ce fichier contient en particulier pour chaque patient : son nom, prénom, sa date de naissance, son numéro de sécurité sociale, sa mutuelle, la date d'entrée, le service d'affectation, le numéro de chambre, la pathologie concernée et les traitements qui sont mis en œuvre.

a) Ce fichier est partagé par deux applications : l'application de gestion de l'hôpital (utilisée par le personnel administratif) et l'application de suivi des malades dont l'accès est strictement réservé au personnel soignant. Comment pourrait-on garantir que toutes les informations médicales ne sont pas récupérables par l'application de gestion ou que le fichier ne soit pas consultable ou modifiable « à la main » par des personnels non habilités ? Avec la solution que vous proposez que se passe-t-il si l'on introduit de nouvelles contraintes d'accès à l'information par exemple en distinguant des droits propres aux médecins et d'autres pour le reste du personnel soignant.

Les SGBD permettent d'associer des « ***droits d'accès »*** très fins en lecture, insertion, suppression, mise à jour à chaque information (par exemple ici celles liées à la pathologie et aux traitements en cours). La nécessité d'interopérer nécessairement avec le SGBD pour accéder aux données garantit donc le bon respect des privilèges paramétrés aussi bien pour les applications que pour les utilisateurs. Il est également très facile de modifier à tout moment ces droits d'accès sans pour autant toucher à la structure de la base et donc remettre en cause le code des applications qui accèdent une base.

b) Imaginons qu'un membre du personnel administratif décide de réaliser le changement de service et donc de chambre d'un patient à l'aide de son application. Cette opération nécessite (sans que l'utilisateur ne le sache) 2 écritures au niveau du fichier ; la première écriture pour changer le service puis la seconde pour changer le numéro de chambre. Que se passe-t-il si une panne de l'application intervient entre le changement de service et de chambre ? Comment éviter ce problème ?

c) Imaginons que 2 administratifs A1 et A2 accèdent via leur application web et de manière concurrente au fichier des données. Supposons que A1 constate via une lecture qu'une place est disponible en chirurgie et lance l'affectation d'un patient dans ce service. Mais avant même que l'application web de A1 n'ait pu finaliser par une écriture dans le fichier cette affectation, celle de A2 a elle aussi constatée cette disponibilité par lecture et lance, elle aussi, une affectation. L'affectation de A1 se termine la première mais est écrasée par l'écriture de l'application de A2. Quel problème cela pose-t-il ? Comment l'éviter ?

Quand on développe un programme P accédant à une base de données, on effectue en général plus ou moins explicitement deux hypothèses : i) P s'exécutera indépendamment de tout autre programme ou utilisateur et ii) l'exécution de P se déroulera toujours intégralement sans « plantage ». Il est clair que ces deux hypothèses ne se vérifient pas toujours !

D'une part les bases de données constituent des ressources accessibles simultanément à plusieurs utilisateurs qui peuvent y rechercher, créer, modifier ou détruire des informations : les accès simultanés à une même ressource sont dits concurrents et l'absence de contrôle de cette concurrence peut entraîner de graves problèmes de cohérence dus aux interactions des opérations effectuées par les différents utilisateurs.

D'autre part on peut envisager beaucoup de raisons pour qu'un programme ne s'exécute pas jusqu'à son terme. Une interruption de l'exécution peut alors laisser la base dans un état transitoire incohérent, ce qui nécessite une opération de réparation consistant à ramener la base au dernier état cohérent connu avant l'interruption.

Il est périlleux pour un développeur de tenter de gérer par lui-même ces 2 types de problème. Les SGBD assurent, par différents mécanismes, un partage concurrent fiable des données et une gestion des interruptions (reprise sur pannes). Le concept de « ***transaction*** » offert par un SGBD est en particulier central : il s'agit de garantir qu'un ensemble d'actions est conduit dans un mode « tout ou rien ». Soit la transaction échoue ou est interrompue et dans ce cas on l'annule complètement soit elle est réalisée complètement. Une transaction constitue, pour un SGBD, une unité d'exécution. Toutes les opérations de la transaction doivent être validées ou annulées solidairement. Cet apport des SGBD est fondamental en termes de sécurité et simplifie grandement la vie des programmeurs.