Projet de langage

Le projet de langage se fait par groupe de deux élèves.

Chaque groupe déposera au secrétariat de l'IUP un rapport accompagné d'une disquette au plus tard le 3 juillet. Chaque jour de retard sera sanctionné d'un point en moins.
La disquette contiendra le code source intégral (fichiers C, `flex` et `bison`) de l'interpréteur développé lors du projet ainsi que quelques fichiers d'exemples.
Le rapport devra discuter les choix d'implémentation (grammaires, structures de données et algorithmes) et documenter l'utilisation de l'interpréteur. Le contenu de la section 2 devra obligatoirement être implémenté par l'interpréteur, mais il n'est pas interdit d'y ajouter des améliorations de votre cru, ou de traiter des parties de la section 3.

Lors d'une soutenance orale ultérieure (date à déterminer), chaque groupe fera la démonstration sur ordinateur de l'interpréteur et répondra aux questions des examinateurs. Une note pour l'ensemble du projet (rapport, disquette et soutenance) sera donnée à chaque élève.

1 Introduction

Le but du projet est d'écrire un interpréteur de commandes pour un langage simple de requêtes dans une base de données semblable à `SQL`.

L'interpréteur lit des requêtes entrées au clavier par l'utilisateur. Ces requêtes référencent des bases de données stockées dans des fichiers textes. À chaque fois que l'utilisateur entre une commande valide, l'interpréteur exécute la commande, affiche le résultat sur l'écran puis rend la main à l'utilisateur pour qu'il entre une nouvelle commande, et ainsi de suite, jusqu'à ce que la commande `quit` soit entrée.

2 Langages

2.1 Bases de données

Chaque base de données est une *table* stockée séparément dans un fichier au format texte. Il est ainsi facile de créer et modifier une base à l'aide d'un éditeur de texte !

Une base de données commence par lister, sur la première ligne, les *noms d'attributs* qu'auront chacune des *entrées* de la base. Ces noms d'attributs sont des mots contenant des lettres majuscules ou minuscules ; ils sont séparés par un *point-virgule* `;`.
À partir de la deuxième ligne commencent les entrées. Chaque entrée occupe exactement une ligne et est composée d'autant de *valeurs d'attributs* qu'il y a de noms d'attributs définis sur la première ligne. Ces valeurs peuvent être numériques (nombre à virgule) ou textuelles, et sont séparés par un point-virgule `;`.

Les espaces et tabulations ne sont pas significatifs dans la base de données. Si une valeur texte doit contenir des espaces, tabulations, retours à la ligne ou points-virgules, elle doit être entrée entre guillemets doubles `"`, en utilisant les conventions des chaînes de caractères `C`.

Exemple. Pour gérer une classe, on utilise deux bases de données. La première base, ÉLÈVES, donne la liste des élèves. La deuxième, NOTES, donne la liste des notes. La correspondance entre les deux bases se fait par un identificateur unique ID (on ne peut utiliser le prénom ou le nom de famille car la classe peut contenir des homonymes).

ÉLÈVES

PRÉNOM	NOM	ID
Truc	Muche	1
Antoine	Miné	45
Bill H.	Gates	69

NOTES

ID	EXAMEN	NOTE
1	partiel	12
45	partiel	20
69	partiel	2
1	final	7
45	final	19.5

Ces bases sont représentées par les fichiers `eleves.txt` et `notes.txt` suivants :

```
eleves.txt               notes.txt

Prénom ;Nom ;Id          Id ;Examen ;Note
Truc ;Muche ;1           1 ;partiel ;12
Antoine ;Miné ;45        45 ;partiel ;20
"Bill H." ;Gates ;69     69 ;partiel ;2
                         1 ;final ;7
                         45 ;final ;19.5
```

2.2 Langage de requêtes

Toutes les requêtes commencent par le mot-clef `SELECT` et se terminent par un point-virgule `;`. Les retours à la ligne, espaces, tabulations servent de séparateur et ne sont pas significatifs.

Requêtes simples. `SELECT` *attributs* `FROM` *table* `;` affichera toutes les entrées de la base *table*, contenue dans le fichier `table.txt`, mais, pour chaque entrée, seuls les attributs dont le nom est dans la liste de noms d'attributs séparés par des virgules *attributs* seront affichés. De plus, les valeurs d'attribut seront affichés dans l'ordre indiqué par la liste *attributs*. On essayera de présenter le tableau aussi joliment que possible (valeurs des attributs alignés en colonnes, en-tête indiquant le nom de chaque attribut, etc.).

Exemple :

```
> SELECT Nom,Prénom FROM eleves ;
 Nom    Prénom
 ---    ------
 Muche  Truc
 Miné   Antoine
 Gates  Bill H.
```

Requêtes conditionnelles. On peut ajouter, avant le point-virgule, une ou plusieurs conditions sur la valeur des attributs de la table : `SELECT` *attributs* `FROM` *table* `WHERE` *cond* `AND` *cond* $\cdots$ `AND` *cond* `;`. Chaque condition *cond* est de la forme *valeur opérateur valeur* où *valeur* est soit un nom d'attribut, soit une valeur numérique ou textuelle, et *opérateur* est un opérateur parmi `=`, `<`, `<=`, `>`, `>=`, `<>`. Les valeurs textuelles seront entrées entre guillemets `"` pour éviter de les confondre avec un nom d'attribut. La relation d'ordre utilisée sur les valeurs textuelles est l'ordre du dictionnaire. Seules les entrées de la table dont les valeurs des attributs vérifient toutes les conditions données seront affichées.

Requêtes relationnelles. Dans une requête relationnelle, on spécifie plusieurs tables `SELECT` *attributs* `FROM` *tables* `WHERE` *conds* `;` où *tables* est une liste de noms de bases de données distinctes, séparés par des virgules : *table1*`,`*table2*`,`...`,`*tablen* correspondant aux fichiers `table1.txt`, `table2.txt`, ..., `tablen.txt`. Les noms d'attributs de toutes les tables de *tables* peuvent être utilisés dans *attributs* et *conds*. Si les deux bases ont des noms d'attributs commun, on les différenciera en préfixant le nom d'attribut par le nom de la table et le séparateur `.` (point).
`SELECT` examine tous les n-uplets construits en choisissant une entrée de chaque table *table1* à *tablen* et sélectionne ceux qui vérifient toutes les conditions *conds*. Pour chaque n-uplet d'entrées sélectionné, seuls les attributs précisés dans la liste *attributs* seront affichés.

Exemple :

```
> SELECT Nom,Note FROM eleves,notes WHERE eleves.Id=notes.Id AND Examen="partiel" ;
 Nom    Note
 ---    ----
 Muche  12
 Miné   20
 Gates  2
```

Fonctions d'agrégation. Une fonction d'agrégation est de la forme `AVG(`*attribut*`)` (moyenne), `SUM(`*attribut*`)` (somme), `COUNT(`*attribut*`)` (nombre d'éléments), `MIN(`*attribut*`)` (minimum), ou `MAX(`*attribut*`)` (maximum), où *attribut* est un nom d'attribut.
On peut utiliser une fonction d'agrégation juste après le mot-clef `SELECT`, à la place d'une liste d'attributs. Plutôt que d'afficher une liste de valeurs d'attributs, la requête calcule la moyenne, la somme, le cardinal,

le minimum ou le maximum des valeurs d'un attribut des entrées vérifiant toutes les conditions. Le résultat affiché est dont une unique valeur numérique et non une table.
Il est également possible d'utiliser dans une condition, à la place d'une valeur ou d'un nom d'attribut, une commande de la forme `(SELECT` *f(attribut)* `FROM ... WHERE ... )` où *f* est une fonction d'agrégation. (Attention toutefois de bien savoir quel nom d'attribut correspond à quelle table dans le cas où on utilise deux fois la même table).

Exemples :

```
> SELECT MOY(Note) FROM notes WHERE Id=45;
19.75
> SELECT Nom FROM eleves WHERE (SELECT MOY(Note) FROM notes WHERE Id=eleves.Id)>=10;
 Nom
 ---
 Miné
```

3 Extensions

Cette partie propose quelques extensions que vous pouvez ajouter si vous avez le temps. Vous pouvez traiter une ou plusieurs extension(s) (pas forcément dans l'ordre proposé ici) mais elles ne sont pas obligatoires. Vous pouvez également proposer une ou des extension(s) de votre cru !
Il vaut mieux se limiter à un interpréteur sans extension mais qui marche bien que d'essayer de tout faire... mal !

3.1 Génération de bases de données

On remarque que, quand on n'utilise pas de fonction d'agrégation dans la partie *attributs* d'une requête `SELECT`, le résultat est une table, affichée dans un format lisible par l'utilisateur.
On propose donc une nouvelle commande `CREATE VIEW` *table* `(` *attributs* `) AS SELECT` ...qui crée dans le fichier `table.txt` une nouvelle base de données dont les noms d'attributs sont dans liste de nom séparés par des virgules *attributs* et dont le contenu est le résultat d'une commande `SELECT`. La commande `SELECT` doit sélectionner autant de noms d'attributs que précisés dans la liste *attributs* suivant la commande `CREATE VIEW`.
La nouvelle table peut alors être immédiatement utilisée.

Exemple :

```
> CREATE VIEW notes2 (Nom,Examen,Note)
AS SELECT Nom,Examen,Note FROM eleves, notes WHERE eleves.Id=notes.Id;
view notes2 generated
> SELECT Nom,Note FROM notes2 WHERE Examen="partiel";
 Nom    Note
 ---    ----
 Muche  12
 Miné   20
 Gates  2
```

3.2 Utilisation de conditions complexes

Dans cette extension, on se propose d'autoriser des expressions arbitraires dans les conditions *conds*. Chaque condition *cond* est de la forme *exp opérateur exp* où *exp* est une expression et *opérateur* est un opérateur de comparaison (`=`, `<`, `<=`, `>`, `>=`, `<>`). Les expressions *exp* peuvent faire appel à des valeurs textuelles ou numériques, à des noms d'attributs, et à des opérateurs tels que $+$, $-$, $\times$, max, sqrt, etc. On pourra ainsi taper `SELECT x FROM truc WHERE x+y>=z AND z>=0;`.

Comme au moment du *parsing* de l'expression, les arguments ne sont pas connus, l'interpréteur devra construire une représentation intermédiaire facilement exécutable de ces expressions. Chaque expression sera compilée en un arbre dont les nœux internes contiennent des opérateurs, et les feuilles contiennent des valeurs ou des noms d'attributs.

Une amélioration de cette extension consiste à autoriser, dans la liste *attributs* ou dans les fonctions d'agrégations, des noms de variables dont la valeur est définie par des contraintes d'égalité dans *conds*. Par exemple, on pourra indiquer `SELECT Id,x FROM notes WHERE x=2*Notes ;` où `SELECT sqrt(AVG(X)) FROM notes WHERE X=note*note ;`.