UNIVERSITE DON BOSCO DE LUBUMBASHI

République Démocratique Du Congo

Province du Haut – Katanga

Lubumbashi

« projet langage c : mini SGBDR »

JUILLET 2024

LISTES DES MEMBRES ET LEURS TACHES

1. NATHAN : module table (table.c et table.h)
2. Leonard : module champ(champs.c et champ.h)
3. Dan : module db (dnfn.c et data-type.h)
4. Guerschom : module commander (commander.c et commander.h)
5. Gentil : module explode (explode.c et explode.h)
6. Emma : module priimitive (primitive.c et primitive.h)

Rapport sur la Modélisation Logicielle et les Fonctionnalités

Introduction

Ce rapport présente la modélisation logicielle d'un système de gestion de bases de données, en spécifiant les classes, leurs relations et les fonctionnalités associées. Le système est conçu pour gérer des bases de données, des tables, des champs et des données, avec des fonctionnalités pour créer, manipuler et interroger ces éléments.

Structure du Code

Le code est structuré autour de plusieurs fichiers d'en-tête (header files) et de leurs définitions. Chaque fichier d'en-tête définit une ou plusieurs classes, leurs attributs, et leurs méthodes associées. Les principales classes et leurs fonctionnalités sont décrites ci-dessous.

Classes et Fonctionnalités

1. Classe Champ

Fichier: champ.h

La classe Champ représente un champ dans une table de la base de données. Chaque champ a un nom, un type, et une liste de données associées.

```
struct Champ {
    char nom[20];
    char type[8];
    struct Data* listeData;
    struct Champ* suivant;
};
typedef struct Champ CH;
```

Méthodes:

- CH* initListe(): Initialise une liste de champs.
- CH* newChamp(CH* Head, char* nom, char* type): Ajoute un nouveau champ à la liste.
- CH* getChamp(CH* head, char* nom): Récupère un champ par son nom.
- CH* removeChamp(CH* head, char* nom): Supprime un champ par son nom.

- int getNPlace(char* nom, CH* liste): Obtient la position d'un champ dans la liste.
- void PrintListe(CH* head): Affiche la liste des champs.
- int countChamp(CH* head): Compte le nombre de champs dans la liste.

2. Classe Data

- **Fichier**: data.h
- La classe Data représente les données associées à un champ.

```
struct Data {
    char value[30];
    int nLine;
    struct Data* suivant;
};
typedef struct Data DT;
```

Méthodes:

- DT* initListeData(): Initialise une liste de données.
- DT* newData(DT* Head, char* value): Ajoute une nouvelle donnée à la liste.
- DT* getData(DT* head, char* value): Récupère une donnée par sa valeur.
- DT* removeData(DT* head, char* value): Supprime une donnée par sa valeur.
- void PrintListeData(DT* head): Affiche la liste des données.
- int countData(DT* head): Compte le nombre de données dans la liste.
- char* getNvalue(int n, DT* head): Récupère la n-ième valeur dans la liste.

3. Classe Table

Fichier: table.h

La classe Table représente une table dans la base de données. Chaque table a un nom et une liste de champs.

```
struct Table {
    char nom[20];
    CH* ListeChamp;
    struct Table* suivant;
};
typedef struct Table TB;
```

Méthodes:

- TB* initListeTable(): Initialise une liste de tables.
- TB* newTable(TB* Head, char* nom): Ajoute une nouvelle table à la liste.
- TB* getTable(TB* head, char* nom): Récupère une table par son nom.
- TB* removeTable(TB* head, char* nom): Supprime une table par son nom.
- int tableExist(char* name, TB* tabList): Vérifie si une table existe.
- void PrintListeTable(char* tabName, TB* head): Affiche la liste des tables.

4. Classe Process

Fichier: commander.h

La classe Process gère le traitement des commandes et la gestion des bases de données.

```
struct Process {
    int nbrDB;
    int DBcursor;
    int sortie;
    TB* DBTables;
    Pdb _Ref;
};
typedef struct Process PXR;
```

Méthodes:

- PXR traite(char* cmd, PXR pxr): Traite une commande.
- void toLower(char* ch): Convertit une chaîne en minuscules.

5. Classe db

Fichier: data_type.h

La classe db représente une base de données et gère les opérations associées.

```
struct db {
    db_name nom;
    db_ref_name ref_;
};
typedef struct db stock[4];
typedef struct db *Pdb;
```

Méthodes:

- RE_ initDBInfos(Pdb stock, Ip pos): Initialise les informations de la base de données.
- void showDB(Pdb stock, int plafond): Affiche les bases de données.
- void useDB(db_name name, Pdb stock, int plafond, Ip use): Sélectionne une base de données.
- State dbExist(db_name nom, Pdb stock, int plafond): Vérifie si une base de données existe.
- char* createRef(): Crée une référence unique pour la base de données.
- State createDirFiles(char *nom): Crée les fichiers de la base de données.
- State addDbToList(db_name nom, db_ref_name ref_, Pdb stock, int pos): Ajoute une base de données à la liste.
- State createDB(db_name nom, Pdb stock, Ip pos): Crée une nouvelle base de données.
- TB* getDBDatas(Pdb Stock, int ref_use): Récupère les données de la base de données.

6. Classe Explode

Fichier: explode.h

La classe Explode gère la séparation des chaînes de caractères en parties.

```
struct Explode {
    char value[100];
    struct Explode* suivant;
};
typedef struct Explode ExplodeR;
```

Méthodes:

- ExplodeR* initExplodeListe(): Initialise une liste de chaînes éclatées.
- ExplodeR* append(ExplodeR* Head, char* value): Ajoute une chaîne à la liste.
- void PExplodeListe(ExplodeR* head): Affiche la liste des chaînes éclatées.
- ExplodeR* explode(char* chaine, char sep): Sépare une chaîne en parties.
- int CountResult(ExplodeR* head): Compte le nombre de parties éclatées.
- char* EgetNvalue(int n, ExplodeR* head): Récupère la n-ième valeur éclatée.

7. Classe Primitives

Fichier: primitives.h

La classe Primitives contient des fonctions utilitaires et des types définis.

```c
typedef char db_name[20];
typedef char db_ref_name[10];
typedef char Le;
typedef int *Ip;
typedef enum {
    OK,
    UNOK
} State;
typedef enum {
    INT,
    STR,
    FLT
} DATA_TYPE;
```

Méthodes:

- char* type2str(DATA_TYPE type): Convertit un type de données en chaîne de caractères.
- int getSeparator(char *chaine, char separator): Obtient l'index d'un séparateur dans une chaîne.
- void attribPart2(char *dest, char *source, int sep, int part): Attribue une partie de la source à la destination.
- char* concate(char *chaine1, char *chaine2): Concatène deux chaînes.
- char int2char(int x): Convertit un entier en caractère.
- char* toStr(int* tab): Convertit un tableau d'entiers en chaîne de caractères.
- int* getArrayOfNumber(int x): Récupère un tableau de nombres.
- char* delN(char* ch): Supprime les nouvelles lignes d'une chaîne.

8. Classe Reponse_

Fichier: response.h

La classe Reponse_ gère les réponses et les messages du système.

```c
struct Reponse_ {
    char action_titre[20];
    char msg[80];
    int etat;
};
typedef struct Reponse_ RE_;
```

Méthodes:

- void setReponse(RE_* r, char* action, char* msg, int etat): Définit une réponse.
- void printReponse(RE_ r): Affiche une réponse.

Relations entre les Classes

Les relations principales entre les classes sont les suivantes :

- **Composition** :
  - Champ a une relation de composition avec Data.
  - Table a une relation de composition avec Champ.
  - db a une relation de composition avec Table.
- **Dépendance** :
  - Process utilise db.
  - Process utilise Explode.
  - Process génère des réponses via Reponse_.

Conclusion

Ce système de gestion de bases de données est bien structuré avec des classes représentant différents aspects des bases de données, des tables, des champs et des données. Les fonctionnalités incluent la création, la manipulation et l'interrogation des éléments de la base de données. Les relations entre les classes sont clairement définies, facilitant ainsi l'extension et la maintenance du système.