Rapport sur le projet B-Tree SQL Database

Projet éducatif en cybersécurité - ESGI 3SI

Auteur : Mohamed Mokdad, ESGI 3SI

Date : 4 mai 2025

Avertissement : Ce projet est destiné à des fins éducatives uniquement. Les tests doivent être effectués dans un environnement contrôlé et isolé pour éviter tout préjudice.

Table des matières

1 Introduction

Ce rapport présente le développement d'une base de données basée sur un arbre B (B-Tree) pour le cours de cybersécurité à l'ESGI 3SI (TLP : AMBER+STRICT, 2024-2025). Le projet, réalisé dans un cadre éducatif, implémente une base de données capable de traiter des commandes SQL simplifiées telles que `INSERT`, `SELECT`, `DELETE`, `IMPORT`, et `PRINT`. La base de données utilise une structure d'arbre B pour organiser les données et assure la persistance sur disque via des fichiers binaires et CSV.

L'objectif est de démontrer une compréhension approfondie des structures de données, de la gestion de la mémoire, et du développement modulaire en C. Ce document détaille l'architecture du projet, le processus de compilation, les tests effectués, et les mesures de qualité du code, conformément aux exigences de l'examen. Tous les tests ont été réalisés dans un environnement isolé (machine virtuelle Kali Linux) sans utilisation d'IDE, avec des commits réguliers pour refléter un travail continu.

2 Architecture et implémentation

Le projet est structuré de manière modulaire pour éviter un fichier C unique, respectant les exigences de l'examen. Voici les principaux composants :

- **`btree.h`, `btree.c`** : Définit et implémente l'arbre B, avec des fonctions pour créer (`create_node`), insérer (`insert_node`), rechercher (`search_node`), supprimer (`delete_node`), et afficher (`print_btree`) des nœuds. La structure `BTreeNode` stocke jusqu'à 3 clés (`MAX_KEYS`) et 4 enfants (`MAX_CHILDREN`).
- **`db.h`, `db.c`** : Gère la persistance sur disque avec `save_to_file` (écriture binaire), `load_from_file` (lecture binaire), et `load_csv_into_table` (importation de CSV).
- **`migration.h`, `migration.c`** : Traite les commandes SQL (`handle_insert`, `handle_select`, `handle_delete`, `handle_import`, `handle_print`) en analysant les entrées utilisateur.
- **`main.c`** : Point d'entrée du programme, gère l'interface en ligne de commande et appelle les fonctions appropriées.
- **`test_btree.c`** : Contient des tests unitaires pour valider l'insertion, la recherche, la suppression, et le fractionnement des nœuds.

2.1 Arbre B

L'arbre B est une structure auto-équilibrée qui maintient les clés triées et permet des opérations efficaces. Chaque nœud contient :

— Un indicateur `is_leaf` (feuille ou nœud interne).
— Un nombre de clés (`num_keys`).
— Des clés (`keys`) et valeurs (`values`, type `Row`) pour stocker `id`, `name`, et `age`.
— Des pointeurs vers les enfants (`children`).

Les opérations clés incluent le fractionnement des nœuds (`split_child`) pour maintenir l'équilibre lors des insertions.

2.2 Persistance

Les données sont sauvegardées dans un fichier binaire (`data.db`) via `save_to_file`, qui écrit récursivement les nœuds. `load_from_file` reconstruit l'arbre à partir du fichier binaire, et `load_csv_into_table` importe des données CSV au format `id,name,age`.

2.3 Commandes SQL

Les commandes sont analysées avec `sscanf` pour extraire les paramètres. Par exemple, `INSERT INTO table VALUES (1, 'Alice', 20)` insère une ligne si l'`id` est unique, et `SELECT * FROM table WHERE id=1` recherche une ligne spécifique.

3 Compilation et maintenance

Le projet est compilé avec un `Makefile` pour automatiser le processus de construction. Voici les détails :

3.1 Compilateur et drapeaux

- **Compilateur** : GCC version 12.2.0 (version standard sur Kali Linux 2024).
- **Drapeaux** :
  — `-Wall` : Active tous les avertissements pour détecter les erreurs potentielles.
  — `-Wextra` : Ajoute des avertissements supplémentaires pour des constructions risquées.
  — `-Werror` : Traite les avertissements comme des erreurs pour garantir un code propre.
  — `-g` : Inclut les informations de débogage pour Valgrind et GDB.
  — `-O2` : Optimise le code pour de meilleures performances sans compromettre la lisibilité.

3.2 Makefile

Le `Makefile` suivant compile le projet :

```
CC = gcc
CFLAGS = -Wall -Wextra -Werror -g -O2
SRC = src/btree.c src/db.c src/migration.c src/main.c
OBJ = $(SRC:.c=.o)
TEST_SRC = src/btree.c src/db.c src/migration.c test/test_btree.c
TEST_OBJ = $(TEST_SRC:.c=.o)
EXEC = btree_db
TEST_EXEC = test_btree

all: $(EXEC)

$(EXEC): $(OBJ)
        $(CC) $(OBJ) -o $(EXEC)

test: $(TEST_EXEC)

$(TEST_EXEC): $(TEST_OBJ)
        $(CC) $(TEST_OBJ) -o $(TEST_EXEC)

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

clean:
        rm -f $(OBJ) $(TEST_OBJ) $(EXEC) $(TEST_EXEC) data.db

.PHONY: all test clean
```

3.3 Instructions de construction

Pour compiler :

```
make
```

Pour exécuter :

```
./btree_db
```

Pour tester :

```
make test
./test_btree
```

Pour nettoyer :

```
make clean
```

4 Tests

Le fichier `test_btree.c` contient des tests unitaires utilisant `assert` pour valider les fonctionnalités de l'arbre B :

- **`test_insert_into_empty_tree`** : Vérifie l'insertion dans un arbre vide.
- **`test_split_root_node`** : Teste le fractionnement du nœud racine après insertion de 4 lignes.
- **`test_search_node`** : Vérifie la recherche de clés existantes et non existantes.
- **`test_delete_leaf_node`** : Valide la suppression d'une clé dans un nœud feuille.

Les tests sont exécutés avec :

```
make test
./test_btree
```

Tous les tests passent sans erreurs, confirmant la robustesse de l'implémentation.

4.1 Exemples d'utilisation

Voici quelques exemples d'utilisation de la base de données B-Tree :

Séquence de commandes SQL :

```
INSERT INTO table VALUES (1, 'Mohamed Mokdad', 23);
INSERT INTO table VALUES (2, 'Alice Dupont', 22);
SELECT * FROM table WHERE id=1;
DELETE FROM table WHERE id=2;
```

Extrait simplifié de la fonction d'insertion dans `btree.c` :

```
void insert_non_full(BTreeNode *node, int key) {
    int i = node->num_keys - 1;
    if (node->is_leaf) {
        while (i >= 0 && node->keys[i] > key) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->num_keys++;
    }
}
```

5 Qualité du code

Pour garantir la qualité du code, nous avons utilisé des outils d'analyse dynamique conformément aux recommandations de l'examen :

- **Valgrind** : Utilisé pour détecter les fuites de mémoire et les accès invalides. Commande :

```
1        valgrind --leak-check=full ./btree_db
```

  Résultat : Aucune fuite détectée après correction d'une allocation non libérée dans `create_node`.
- **AddressSanitizer (ASan)** : Ajouté avec `-fsanitize=address` pour détecter les erreurs de mémoire. Une erreur de débordement de tableau dans `insert_non_full` a été corrigée.
- **Fuzzing** : Utilisation d'AFL++ pour tester les entrées SQL. Des entrées malformées ont révélé un crash dans `sscanf`, corrigé par une validation stricte des chaînes.

Les corrections ont été validées par des commits réguliers, reflétant un développement continu.

6 Conclusion

Ce projet a permis de développer une base de données basée sur un arbre B, capable de traiter des commandes SQL simplifiées avec persistance sur disque. L'architecture modulaire, les tests unitaires, et l'utilisation d'outils comme Valgrind et AFL++ garantissent un code robuste et de haute qualité. Le respect des exigences, notamment l'absence d'IDE, les commits réguliers, et les drapeaux de compilation stricts, témoigne d'une approche rigoureuse.

Avertissement final : Ce projet est éducatif. Toute utilisation en dehors d'un environnement contrôlé est interdite.