UFUG2601 - C++ Programming
Course Project

Oct. 2024

1 Project Overview

In computer science, a "**database**" is an organized collection of data that allows for easy access, management, and updating. It's essentially a storage system where you can keep large amounts of information, such as student records, product inventories, or customer information. The most popular type of databases is *relational*, where the data is organized as tables.

Your Task. In this project, you will implement a mini-database management system (i.e., `miniDB`) that supports creating tables, updating records, querying tables, and more. Your system should be able to process commands in **our simplified version of SQL (i.e., miniSQL)** and output the query results.

What is SQL? SQL (Structured Query Language) is a standard language for managing and manipulating relational databases, enabling users to create, read, update, and delete data, as well as define and modify database structures. Here, you'll work with a simplified version of SQL designed specifically for our miniDB.

How a table in our miniDB looks like? Table 1 shows a table used in our mini database. Each row is a "record," and each column is a "field." Your database needs to manage multiple tables.

Table 1: A table showing student information

ID	Name	GPA	Major
1000	Jay Chou	3.0	Microelectronics
1001	Taylor Swift	3.2	Data Science
1002	Bob Dylan	3.5	Financial Technology

Usage of `minidb` Compile your program into a single executable named `minidb`, which accepts two command-line arguments: the first is the input SQL file, and the second is the output file. The basic usage format of the program is as follows:

```
./minidb input.sql output.txt
```

2 Syntax of miniSQL

2.1 Create Database and Use Database

A **database** contains multiple **tables**. Before operating on specific table, the user has to specify the database they want to use.
Your mini-database system will support basic database management with "CREATE DATABASE" and "USE DATABASE" commands.

Create Database The user can create a database using the following syntax:

```
CREATE DATABASE database_name;
```

Your `minidb` should be able to create a database. Before exiting `minidb`, your program should store all the current table content in files to disks.
Before doing anything regarding to the tables (query, update, delete, etc.), the user must specifiy which database they want to operate on. This step is done by "USE DATABASE" statement:

Use Database The "USE DATABASE" command is used to switch to a specific database, so any subsequent commands (like creating tables, inserting data, or querying) will apply to the selected database.

```
USE DATABASE database_name;
```

2.2 Create Tables

In this project, `minidb` will support a simplified `CREATE TABLE` command with three data types: `FLOAT`, `TEXT`, and `INTEGER`. This command allows users to define a table by specifying its name and columns. Each column must have a unique name and one of the three supported data types.

Syntax

```
CREATE TABLE table_name (
    column1_name column1_type,
    column2_name column2_type,
    column3_name column3_type,
    ...
);
```

This example creates a table named `table_name` with three columns: `column1_name`, which holds floating-point numbers; `column2_name`, which holds text values; and `column3_name`, which holds integer values.

Examples if the user wants to create a table the same as Table 1, the user needs to input the following create table command:

```
CREATE TABLE student (
    ID INTEGER,
    Name TEXT,
    GPA FLOAT,
    Major TEXT
);
```

2.3 Drop Tables

In this project, `minidb` will also support a simplified `DROP TABLE` command, which allows users to delete a table and all its associated data from the database. Once a table is dropped, all data in that table is permanently removed, and any queries referencing the deleted table will fail. This command is useful for freeing up database space or removing outdated or unneeded tables.

Syntax

```
DROP TABLE table_name;
```

This example removes the table named `table_name` from the database, along with all data stored within it.

Examples If the user wants to delete a table named `student`, the user would input the following `DROP TABLE` command:

```
DROP TABLE student;
```

This command permanently removes the `student` table and all its data, ensuring it can no longer be accessed or queried in `minidb`.

2.4 Data Insertion

After creating a table with the `CREATE TABLE` command, you can insert a record into it using the `INSERT INTO` command. This command specifies the table name, followed by a list of column values in the same order as they were defined in the table schema.

Syntax

```
INSERT INTO table_name VALUES (value1, value2, ...);
```

Examples To insert the data shown in the `student` table created above, you would use:

```
INSERT INTO student VALUES (1000, 'Jay Chou', 3.0, 'Microelectronics');
INSERT INTO student VALUES (1001, 'Taylor Swift', 3.2, 'Data Science');
INSERT INTO student VALUES (1002, 'Bob Dylan', 3.5, 'Financial Technology');
```

Each `INSERT INTO` command adds one row of data to the table, with values corresponding to the columns defined in `CREATE TABLE`.

2.5 Data Query: Basics

Suppose you have a `student` table with the following columns and data:

ID	Name	GPA	Major
1000	Jay Chou	3.0	Microelectronics
1001	Taylor Swift	3.2	Data Science
1002	Bob Dylan	3.5	Financial Technology

Syntax

```
SELECT column1, column2, ... FROM table_name;
```

Examples Suppose you have the student table like Table 1.
1. **Selecting Specific Columns**: If you want to retrieve only the `Name` and `GPA` columns of all students, you would use:

```
SELECT Name, GPA FROM student;
```

Result:

Name	GPA
Jay Chou	3.0
Taylor Swift	3.2
Bob Dylan	3.5

2. **Selecting All Columns**: To retrieve all columns in the table, you can use the `*` symbol:

```
SELECT * FROM student;
```

Result:

ID	Name	GPA	Major
1000	Jay Chou	3.0	Microelectronics
1001	Taylor Swift	3.2	Data Science
1002	Bob Dylan	3.5	Financial Technology

This simple selection retrieves rows from the `student` table based on the specified columns, showing just the data you need.
Any order of the rows is acceptable.

2.6 Data Query: "Where" Clause

The `WHERE` clause in `minidb` allows you to filter records based on specific conditions, using basic comparison operators and logical connectors. In this simplified version, the `WHERE` clause supports only the following conditions:

- Comparisons: `column > value`, `column < value`, `column = value`
- Logical Connectors: `AND` and `OR` to combine multiple conditions

Syntax for `WHERE`

```
SELECT column1, column2 FROM table_name WHERE condition1 AND/OR condition2;
```

Examples Suppose you have the `student` table:
1. **Single Condition**: To retrieve names of students with a GPA greater than 3.0, you would use:

```
   SELECT Name FROM student WHERE GPA > 3.0;
```

ID	Name	GPA	Major
1000	Jay Chou	3.0	Microelectronics
1001	Taylor Swift	3.2	Data Science
1002	Bob Dylan	3.5	Financial Technology

Result:

Name
Taylor Swift
Bob Dylan

2. **Multiple Conditions with `AND`**: To find students with a GPA greater than 3.0 who are also majoring in `Data Science`, use:

```
SELECT Name FROM student WHERE GPA > 3.0 AND Major = 'Data Science';
```

Result:

Name
Taylor Swift

3. **Multiple Conditions with `OR`**: To retrieve students with a GPA less than 3.1 or majoring in `Financial Technology`, use:

```
SELECT Name FROM student WHERE GPA < 3.1 OR Major = 'Financial Technology';
```

Result:

Name
Jay Chou
Bob Dylan

By using `WHERE` with supported comparison operators and logical connectors, you can retrieve data based on specific conditions, making your queries both flexible and efficient.

2.7 Data Query: "Inner Join" Clause

An `INNER JOIN` connects records from two tables based on a specified condition. *When we `INNER JOIN` Table A's `X` column with Table B's `Y` column, each record in Table A is matched with a record in Table B if the value in Table A's `X` column is equal to the value in Table B's `Y` column.*

Only pairs of records that meet this condition are included in the result, forming a combined view of related data from both tables.

In other words, the `INNER JOIN` condition effectively links records from Table A and Table B wherever there is a match between Table A's `X` and Table B's `Y`.

Syntax

```
SELECT table1.column1, table2.column1
FROM table1
INNER JOIN table2
ON table1.X = table2.Y;
```

The `INNER JOIN` clause connects records from two tables based on a matching condition between columns. When we use `INNER JOIN` on Table A's `X` column and Table B's `Y` column, each record in Table A is paired with records in Table B where values in `X` and `Y` are equal. Only records that satisfy this condition are included in the result, which provides a combined view of related data from both tables.

When using `INNER JOIN` to combine data from multiple tables, it's essential to specify which table each column belongs to by using the table name followed by a dot, and then the column name. This is called a *fully qualified column name* and helps avoid ambiguity, especially when both tables have columns with the same name.
For example, if both `student` and `course_enrollment` tables have a column named `StudentID`, using `student.StudentID` and `course_enrollment.StudentID` makes it clear which table each column comes from.

Examples Suppose you have two tables, `student` and `course_enrollment`, as follows:

StudentID	Name
1	Jay Chou
2	Taylor Swift
3	Bob Dylan
4	Omnipotent Youth Society

Table 2: student table

To retrieve each student's name along with the courses they are enrolled in, we can `INNER JOIN` the `student` and `course_enrollment` tables on the `StudentID` column:

```
SELECT student.Name, course_enrollment.Course
FROM student
INNER JOIN course_enrollment ON student.StudentID = course_enrollment.StudentID;
```

StudentID	Course
1	Microelectronics
2	Data Science
2	Machine Learning
3	Financial Technology
4	Mathematics

Table 3: course_enrollment table

Name	Course
Jay Chou	Microelectronics
Taylor Swift	Data Science
Taylor Swift	Machine Learning
Bob Dylan	Financial Technology
Omnipotent Youth Society	Mathematics

Result:
In this example, the result includes every matched pair where `StudentID` values are equal in both tables, showing each student's name alongside their enrolled courses.
With `INNER JOIN`, each match between `student` and `course_enrollment` based on `StudentID` results in a row in the final output, showing how `INNER JOIN` combines related data across tables.

2.8 Data Update

The `UPDATE` command in SQL is used to modify existing records in a table. In `minidb`, the `UPDATE` command allows you to change values in specific columns based on a condition specified with the `WHERE` clause. This helps ensure that only certain rows are updated, rather than modifying every row in the table.

Syntax for `UPDATE` The general syntax for using `UPDATE` is as follows:

```
UPDATE table_name
SET column1 = new_value1, column2 = new_value2, ...
WHERE condition;
```

- `table_name`: The name of the table where records will be updated.
- `SET`: Specifies the columns to update and their new values.
- WHERE: (Optional) Specifies a condition to determine which rows to update. Without `WHERE`, all rows in the table will be updated.

Examples Suppose you have a `student` table:
1. **Updating a Specific Row**: If you want to update `Jay Chou`'s GPA to 3.6, you can specify a `WHERE` condition to target only his record:

ID	Name	GPA
1000	Jay Chou	3.0
1001	Taylor Swift	3.2
1002	Bob Dylan	3.5

Table 4: student table

```
UPDATE student
SET GPA = 3.6
WHERE Name = 'Jay Chou';
```

Result:

ID	Name	GPA
1000	Jay Chou	3.6
1001	Taylor Swift	3.2
1002	Bob Dylan	3.5

2. **Updating Multiple Rows**: To increase the GPA of all students with a GPA less than 3.5 by 0.1, use the following command:

```
UPDATE student
SET GPA = GPA + 0.1
WHERE GPA < 3.5;
```

Result:

ID	Name	GPA
1000	Jay Chou	3.6
1001	Taylor Swift	3.3
1002	Bob Dylan	3.5

In this example, only records meeting the `WHERE` condition are updated, which makes the `UPDATE` command powerful for targeted changes. Without a `WHERE` clause, all rows in the table would be updated with the new values specified.

2.9 Data Deletion

The `DELETE` command in SQL is used to remove records from a table. Similar to `UPDATE`, the `DELETE` command can use the `WHERE` clause to specify conditions, allowing you to delete only certain rows rather than all rows in the table.

Syntax for `DELETE` The general syntax for using `DELETE` is as follows:

```
DELETE FROM table_name
WHERE condition;
```

- `table_name`: The name of the table from which records will be deleted.
- `WHERE`: (Optional) Specifies a condition to determine which rows to delete. Without `WHERE`, all rows in the table will be deleted.

Example of `DELETE` with `WHERE` Suppose you want to delete records of students with a GPA less than 3.0 in a `student` table:

```
DELETE FROM student
WHERE GPA < 3.0;
```

Only rows meeting the `WHERE` condition will be deleted. Without a `WHERE` clause, all rows in the `student` table would be removed.

3 Format of the Output

Each query (`SELECT` statement) result will be printed in a CSV format with the following rules: 1. **Output Format**: Each result is printed in CSV format, where fields are separated by commas. 2. **Header Row**: The first row of the output will contain the column names, also separated by commas. 3. **Text Fields**: Text fields are enclosed in double quotes (`""`). There will be no text fields containing double quotes in the data, so no further formatting is necessary. 4. **Numeric Fields**:

- **Integer** fields are printed as-is.
- **Float** fields are printed with exactly two decimal places, rounded if necessary.

`minidb` executes the following `SELECT` query:

```
SELECT ID, Name, GPA FROM student;
```

and returns the following data:

ID	Name	GPA
1000	Jay Chou	3.00
1001	Taylor Swift	3.20
1002	Bob Dylan	3.50

The output in CSV format would be:

```
ID,Name,GPA
1000,"Jay Chou",3.00
1001,"Taylor Swift",3.20
1002,"Bob Dylan",3.50
```

3.1 Example 1

Here's an example `input.sql` and the corresponding `output.csv` based on the simplified miniSQL format.

`input.sql`

```
CREATE DATABASE db_university;

USE DATABASE db_university;

CREATE TABLE student (
    ID INTEGER,
    Name TEXT,
    GPA FLOAT
);

INSERT INTO student VALUES (1000, 'Jay Chou', 3.0);
INSERT INTO student VALUES (1001, 'Taylor Swift', 3.2);
INSERT INTO student VALUES (1002, 'Bob Dylan', 3.5);

SELECT ID, Name, GPA FROM student;

SELECT ID, Name, GPA FROM student WHERE GPA > 3.1;
```

`output.csv`

```
ID,Name,GPA
1000,"Jay Chou",3.00
1001,"Taylor Swift",3.20
1002,"Bob Dylan",3.50
---
ID,Name,GPA
1001,"Taylor Swift",3.20
1002,"Bob Dylan",3.50
```

In this `output.csv`:

- Each query result is separated by a line (`---`) to clearly differentiate outputs from consecutive queries.
- Text fields are surrounded by double quotes, while integers and floats are printed directly, with floats formatted to two decimal places.

3.2 Example 2

After executing Example 1, `minidb` should serialize the information of `db_unversity` in disks. Now, when `minidb` reads a SQL statement "use database", it needs to load previous database content.

Here's an example `input2.sql` and the corresponding `output2.csv` based on the simplified miniSQL format.

`input2.sql`

```
USE DATABASE db_university;

SELECT ID, Name, GPA FROM student WHERE GPA > 3.1;
```

`output2.csv`

```
ID,Name,GPA
1001,"Taylor Swift",3.20
1002,"Bob Dylan",3.50
```

4 Error Report

`minidb` should be able to check the syntax of miniSQL, and report the line number where the error happens.