教育用データベース管理システムの調査

Survey of database management systems for educational purposes

追川 修一 [1*]

Shuichi Oikawa[1*]

[1] 東京都立産業技術大学院大学 Advanced Institute of Industrial Technology
[*]Corresponding author: Shuichi Oikawa, oikawa-shuichi@aiit.ac.jp

Abstract This paper examines database management systems (DBMS) designed for educational purposes. It first compares several DBMS implementations developed for academic use, including SimpleDB (Boston College), BusTub (Carnegie Mellon University), Go-DB (MIT), and RookieDB (UC Berkeley). These DBMS differ in programming language, system architecture, and features. SimpleDB, written in Java, provides a basic relational database with SQL support, while others, such as BusTub (C++) and Go-DB (Go), offer different functionalities and learning approaches. To explore whether implementing a DBMS in Python enhances understanding for learners, the study attempts to reimplement SimpleDB in Python. The conversion process highlighted differences in syntax, type handling, and memory management between Java and Python. While Python simplifies some aspects, such as eliminating explicit type declarations, it also introduces challenges in debugging and maintaining readability. Key differences include Python's lack of built-in tools equivalent to Java's StreamTokenizer and ByteBuffer, requiring alternative implementations. The findings suggest that merely rewriting a DBMS in Python does not inherently improve understanding. Instead, leveraging Python's existing libraries, modularizing components, and incorporating visualization tools could make DBMS concepts more accessible. The paper concludes by identifying future research directions, including the integration of Python's standard libraries and visualization techniques to enhance learning efficiency. Overall, the study contributes to database education by evaluating the effectiveness of Python in DBMS implementation and proposing improvements for educational DBMS tools.

Keywords systems software; database management systems; education

1 はじめに

コンピュータの主要な利用目的がデータ処理であるため、データベースは常に必要とされている。データベースを効率的に利用するためには、データベースを構築し管理するデータベース管理システム (DBMS) に対する理解の重要性は非常に高い。DBMS は、利用目的に合わせ、大規模なデータベースを対象とするものから、組み込みシステムでも使用できるものまで、様々な種類が開発されている。その中でも、組み込みシステムで使用できる SQLite [1] は、非常に手軽に利用できる環境が整っていることから、学習用途としても使いやすい。一方で、DBMS を構築するための技術を学ぶには、実用的な性能を提供するために、SQLite でもその実装は複雑であり、データベースを学ぶ初学者が理解することは非常に困難である。

この状況は、システムの根幹となるソフトウェアであるオペレーティングシステム (OS) カーネルや、ハードウェアではプロセッサでも同様である。OS カーネルやプロセッサは、コンピュータの実行基盤や実行機能そのものを提供する基盤システムとして、効率的なシステムの開発および運用のために、その理解の重要性は非常に高い一方で、非常に複雑なシステムである。OS カーネルを構築するための技術を学ぶには、実用的に用いられている Linux 等の OS カーネルの実装を初学者が理解することは非常に困難である。プロセッサについては、基本的に商用でプロプライエタリであることから、実用的に用いられているプロセッサの実装を学ぶことはできない。そのため、OS カーネルやプロセッサの実装を学ぶための教材としての実装が、これまで数多く開発されてきた。

OS カーネルの実装を学ぶための教材としては、現在では xv6 [2] が最も有名である。Xv6 の前には、教育を主目的として XINU, MINIX, Nachos 等が開発されてきたが、それぞれ異なる難点があり、xv6 ほど普及しなかった。Xv6 は、単純ではあるが概念的には現在でも通用する UNIX V6 のシステムコールインタフェースの提供、モノリシックデザインによる理解しやすさ、平易な C 言語による単純な実装、プロセッサの特権モードを用いて実際に動作する OS カーネルであり、誰でも簡単にコンパイルし、プロセッサのシステムエミュレータで実行できる手軽さを提供し、修正や機能追加を通して、OS カーネルの動作原理を理解することができるようになっていることから、広く普及したと考えられる。

DBMS にも、その実装を学ぶために、大学の講義における教材として開発されたものとして、Boston College で開発された SimpleDB [3] があり、解説書 [4] も出版されていることから、広く知られている。その他、CMU では BusTub [5], MIT では Go-DB [6], UCB では RookieDB [7] 等が、データベースの講義のプロジェクト用教材として開発されている。それぞれの実装に用いられているプログラミング言語は、SimpleDB は Java, BusTub は C++, Go-DB は Go, RookieDB は Java であり、実用的なシステムを実装するためには適しているプログラミング言語で実装されている。

本論文では、教材として開発された教育用途の DBMS の比較を行い、その差異を明らかにする。さらに、教育用途の DBMS の 1 つである SimpleDB を Python で実装し、Python で DBMS を実装することで、データベースを学ぶ初学者が DBMS の実装を理解しやすくなるかを検証する。

以下、2 章では教材として開発された DBMS の比較を行う。3 章では SimpleDB を Python で実装した経験について述べ、考察する。4 章では今後の課題を述べ、5 章で本論文をまとめる。

2 教育用 DBMS の比較

本章では、大学の講義における教材として開発された教育用 DBMS の比較を行う。講義の教材として提供されているソースコードまたはレポジトリがあるものとして SimpleDB [3, 4], BusTub [5], Go-DB [6], RookieDB [7] を選択した。まず、それぞれの DBMS の特徴について述べ、その後に、特徴をふまえて比較を行う。

Received: 2025-10-31; Published online: 2026-02-28

SimpleDB

SimpleDB [3, 4] は Boston College で開発された DBMS である。Java で記述されており、テストプログラムも含めて、12,649 行のソースコードから構成されている。

構文解析器は、文字列をトークン化する際には、Java が標準で提供する StreamTokenizer を用いた、独自の実装である。構文解析器を単純化するために、実行可能な SQL はかなり制限されているが、SELECT FROM WHERE からなる基本的な SELECT 構文をはじめ、CREATE TABLE, CREATE INDEX などの CREATE 構文、INSERT INTO などの INSERT 構文が実行できる。一方、HAVING, GROUP BY, SORT など、機能を実装したソースコードは提供されいるものの、構文解析器には含まれておらず、SQL では実行できない機能もある。

対象とする値は整数と文字列だけである。レコードは固定長に限定されているものの、機能面では、基本的な機能は網羅している。リレーショナル代数の演算としては、直積、射影、選択を提供している。選択条件は等号だけである。物理的データ格納方式は、固定長レコードのヒープファイルである。固定長レコードであるため、ヒープファイル内は、固定オフセットを用いた順次アクセスを行う。インデックスは、ハッシュと B 木を提供している。問合せ処理においては、結合はインデックスを用いた結合とマージ結合、ソートはマージソートを提供している。また、問合せ最適化は、選択は先に実行するというようなヒューリスティックスとコスト見積りに基づき行う。トランザクションは、ロックを用いた同時実行制御により実現している。そして、UNDO/NO-REDO 方式に基づいたログを用いた障害回復を行う。

以上のように、SimpleDB は機能を絞り込んだ実装となっているものの、基本的な機能は網羅した実装となっていることがわかる。従って、一般的なデータベースの教科書に述べられている機能の実現方法について、ソースコードから学ぶことができるものとなっている。

BusTub

BusTub [5] は Carnegie Mellon University (CMU) で開発された DBMS である。C++で記述されており、テストプログラムも含めて、bustub 本体の 17,541 行のソースコードに加え、付加機能を提供するサードパーティからの 117,802 行のソースコードと共に提供されている。Linux (Ubuntu) および MacOS でコンパイル、実行可能とされているが、Linux が開発環境として推奨されている。構築に必要なパッケージをインストールするためのシェルスクリプトも用意されており、レポジトリに記載の手順に従って、ソースコードから DBMS およびテストを構築可能であることを、Linux を用いて実際に確認した。

講義 [8] の資料は、スライドが公開されているだけでなく、授業動画も YouTube で提供されており、誰でも視聴できるようになっている。プロジェクトに関する動画は提供されていないが、開発内容および開発方法について詳細な情報が掲載されているため、授業動画を視聴し、プロジェクトに取り組むことで、DBMS の実装についての理解を深めることができる。

構文解析器は、元々は PostgreSQL で開発されたものを DuckDB 用に改変されたものを用いており、ソースコードの構成としては、サードパーティのソースコードに含まれる。実用的な DBMS で用いられている構文解析器であるため、構文解析器だけでソースコードは 41,212 行あるが、bustub で対象とする値は整数と文字列だけである。

プロジェクトとしては、Buffer Pool Manager, Index, Query Execution, Concurrency Control を開発することとなっている。レポジトリが提供するソースコードから構築した bustub は、テーブルを参照しない単純な SQL 文の実行は可能であるが、テーブルを参照する SQL を実行できるようにするには、これらのプロジェクトに取り組み、ソースコードを完成させる必要がある。

Go-DB

Go-DB [6] は Massachusetts Institute of Technology (MIT) で開発された DBMS である。講義 [9] の資料は、スライドのみが公開されている。Go-DB は、その名のとおり、Go で記述されており、テストプログラムも含めて、DBMS 本体は 7,820 行のソースコードから構成されている。プロジェクトを実施するにあたり、特に開発環境の指定は無いが、レポジトリのプロジェクトページに記載の手順に従って、ソースコードからテストを実行可能であることを、Linux を用いて実際に確認した。

構文解析器は、Go のレポジトリに登録されている sqlparser を用いた、独自の実装である。SELECT, INSERT, DELETE 構文の他、トランザクションのための BEGIN, COMMIT, ROLLBACK 構文も処理できるようになっている。SELECT 構文では、WHERE, GROUP BY, ORDER BY, HAVING 句が処理可能であり、WHERE, HAVING における選択条件として、等号だけでなく大小関係も用いることができる。また、副問合せも処理できる。対象とする値は整数と文字列だけであり、レコードは固定長に限定されている。

プロジェクトとして、Storage Access, Operators, Transactions を開発することとなっている。開発内容および開発方法について情報をもとに、プロジェクトに取り組むことが可能である。Go-DB で SQL を実行できるようにするには、これらのプロジェクトに取り組み、ソースコードを完成させる必要がある。

RookieDB

RookieDB [7] は University of California, Berkeley (UCB) で開発された DBMS である。講義 [10] の資料は公開されていないが、プロジェクトの開発内容および開発方法について情報 [11] は公開されている。RookieDB は、Java で記述されており、テストプログラムも含めて、DBMS 本体は 43,604 行のソースコードから構成されている。プロジェクトを実施するにあたり、特に開発環境の指定は無いが、説明は IntelliJ IDE の使用を前提としている。CLI でのソースコードからテストを実行可能する方法についての記載は無いが、Java の構築ツールである Maven の設定ファイルが提供されているため、標準的な方法で実行できることを、Linux を用いて実際に確認した。

構文解析器は、JavaCC および JJTree を用いて、BNF により定義された文法から生成するようになっており、基本的な構文および句が処理できるようになっている。プロジェクトとして、B+ Trees, Joins and Query Optimization, Concurrency, Recovery を開発することとなっており、これらに取り組むことは可能である。

表 1　プロジェクト課題で実装する機能

DBMS	課題
bustub	Buffer Pool Manager, Index, Query Execution, Concurrency Control
Go-DB	Storage Access, Operators, Transactions
RookieDB	B+ Trees, Joins and Query Optimization, Concurrency, Recovery

比較

これまで述べた DBMS における、プログラミング言語、構文解析器、プロジェクト課題について、比較を行う。

プログラミング言語は、Java, C++, Go が用いられている。4 つの DBMS のうち、SimpleDB および RookieDB の 2 つが Java で記述されている。MIT では Go-DB が開発される前は、Java で記述された DBMS が用いられていたことから、教育用システムの記述に Java が良く用いられていることがわかる。

構文解析器の実装方法は、既存のソフトウェアから流用している bustub のような方法から、BNF から生成している RookieDB, ライブラリで基本的なトークン化を行った後の構文解析のソースコードは独自に実装する SimpleDB, Go-DB まで、様々である。当然、既存のソフトウェアから流用は、処理できる SQL 文に制限は無くなる一方で、複雑である。独自の実装は、単純で理解しやすい一方で、処理できる SQL 文には制限があり、一長一短である。

表 1 にプロジェクト課題で実装する機能をまとめる。 内容としては、ストレージアクセスやバッファ管理、インデックス、問合せ、トランザクション処理または同時実行制御、障害回復と、類似した課題が出されていることから、これらが実装を理解すべき内容であり、特に問合せおよびトランザクション処理は全てに含まれることから、特に重要であることがわかる。

3　Python による SimpleDB の実装

SimpleDB の実装の理解、および Python を用いることで DBMS 実装の理解が容易になるのかを確認することを目的として、SimpleDB を Python で実装した経験について述べる。

実装手順

実装は、機能モジュールごとにテストプログラムがあるため、そのテストを実行でき、正しい結果が得られるようにするテストファーストによる開発を行った。まずテストプログラムについて、次に述べるように、ソースコードの Python への書き換えを行う。次に、テストプログラムが直接参照するクラスのソースコードの書き換えを行う、というように、参照関係に沿ってソースコードの書き換えを進めた。

SimpleDB が提供するテストは、機能テストのレベルであり、網羅性も低い。また、1 つ 1 つのメソッドをテストするユニットテストは提供されていない。そのため、テストとしては不十分であり、例えば、機能 A に提供されているテスト A が実行できたとしても、機能 A を使用する別の機能 B のテストを実行した際に、機能 A の問題が生じることが頻発した。

ソースコードの書き換え

現時点では、SimpleDB の Java プログラムを、Python で書き換えただけである。Java プログラムを Python に書き換える作業は、全般的には単純である。まず、文法を Python に合わせる変更を行う。これには、変数の型の指定の削除、文末の区切り文字 `;` や複合文の中括弧 `{}` の削除、クラスからインスタンスを生成する `new` の削除、インスタンス変数を指定するために `this` から `self` への変更、print 文の名称変更、コメントを指定する文字の変更などが、どのソースコードファイルでも共通に必要となる作業であった。

また、データ型としては、整数や文字列等のプリミティブ型以外に、データの集合を管理するデータ型として、Java の ArrayList と Map が使用されている。ArrayList は、Python ではリスト (list) 型に置き換えた。また、Map は、Python では辞書 (dict) 型に置き換えた。リスト型、辞書型ともに、必要な機能が提供されており、それぞれのデータ型の操作命令を書き換えるだけで十分であった。

書き換えにあたり、Java と Python の文法上の違いから注意を要する点として、インデント、インスタンス変数の指定方法、オブジェクトの比較がある。Python は、インデントにより文のブロック（複合文）を構成する方法を採用している。一方、Java では中括弧 `{}` で囲むことでブロックを構成する。従って、Java ではインデントは単にソースコードの読みやすさだけに関係するが、Python ではインデントによりプログラムの意味が変わってしまうことになる。現在のエディタはプログラムの構文から正しくインデントを付けてくれるため、エディタが付けてくれたインデントを維持するようにすれば基本的には問題はないが、Java のソースコードにはある中括弧を削除する際などに、インデントを変更することが無いように、気を付ける必要はある。

書き換え時には、インスタンス変数の指定方法の違いにも注意をはらう必要がある。Java では、`this.` を変数名の前に付けることでインスタンス変数であることを指定することができるが、同じ変数名がローカル変数として使用されていない場合は、`this.` を省略可能である。一方、Python では、メソッドのローカル変数とインスタンス変数を明確に区別できるようにとの意図から、必ず `self.` をインスタンス変数名の前に付ける必要がある。ここで問題となるのは、Python では変数を宣言する必要がないという点である。つまり、インスタンス変数であるにも関わらず `self.` を付け忘れると、メソッドのローカル変数として扱われてしまうだけでなく、実行時にエラーも発生しないことになり、発見が困難な書き換えの間違いとなってしまう。

その他、Java と Python で概念的には同じであるが、詳細は異なっている点として、比較演算子における等価性と同一性の扱いがある。Java では、等価性の比較には `equals` メソッド、同一性の比較には `==` 演算子を用いるものとしている。Python では、等価性の比較には `==` 演算子、同一性の比較には `is` 演算子を用いるものとしている。しかしながら、Python では、特にメソッドが定義されていなければ、`==` 演算子で同一性の比較を行い、実行時に

エラーが発生しない。そのため、発見が困難な間違いとなってしまうことがある。

不足機能の実装

単純なソースコードの書き換えでは対応できず、Java にはあり Python には無かったため、不足した機能がいくつかあった。それらは、構文解析器における StreamTokenizer、バイト単位のデータを格納する ByteBuffer、ファイルアクセスのメソッドである。

StreamTokenizer は、Java が標準で提供するクラスであり、文字列をトークン化する汎用的な機能を提供している。Python もトークン化する機能を tokenize モジュールを標準で提供しているが、これは Python ソースコードに特化しているため、使用できない。Python には様々なモジュールが開発されており、それらを組み合わせて使用するのが Python の特徴ともなっている。SQL の構文解析器として sqlparse も提供されているが、StreamTokenizer との差異が大きいため、必要な機能のみを持つトークン化を行うクラスを実装した。

ByteBuffer は、バイト単位のデータを格納することを目的としたクラスであり、Java 処理系が管理するヒープの外にバッファ領域を確保することも可能である。Python では、処理系が管理するヒープの外にバッファ領域を確保するには、例えば C 言語の関数を呼び出すなど、Python の処理系の外での処理を行う必要がある。DBMS の実装としては、データ領域を DBMS の管理下に置くという点で、ヒープの外にバッファ領域を確保する意味があるが、Python で記述する DBMS ではそこまでする必要は無いと判断した。ByteBuffer が提供するバイト単位のデータの操作については、Python の `bytearray` を用いた。`bytearray` 内のデータはスライスを用いることでバイト単位でアクセス可能である。整数 (int) 型とバイト表現の変換のためには `from_bytes`, `to_bytes` 関数を用いて行うことができる。

ファイルアクセスのメソッドも、Java と Python では大きく異なっており、単純な書き替えでは対応できず、Python で実装し直す必要があった。ファイルをアクセスする際には、アクセスのモードを指定してファイルを open した後に読み書き、即ち read, write を行うという処理の流れとなるが、Python におけるファイルを open する際のモードの指定方法について理解不足があり、正しい動作のためのモード指定を把握するのに時間がかかってしまった。

具体的には、書き込みの対象となるファイルが存在しない場合、ファイルを作成してから書き込みを行うことになるが、ファイルが存在する場合と分けて処理する必要があった。ファイルを作成するためには、モードとして w を指定する。しかしながら、ファイルが存在する場合は、モードとして w を指定すると、既存の内容は消されて (truncate) しまう。また、モード a を指定すると、既に書かれているデータの後に追加するため、既存の内容が消されることはないが、既に書かれているデータは変更できない。ファイルが存在する場合としない場合で場合分けをすることで、ファイルが存在する場合はモードとして r+ を指定することで、書き込まれた内容はそのままに、書き込みが可能になり、またファイルが存在しない場合はモードとして w を指定することで、ファイルを作成できる。

考察

SimpleDB の Python による実装について考察する。まず、SimpleDB の Python による実装の目的とした、Python を用いることで DBMS 実装の理解が容易になるのか、について考察する。さらに、ソースコードの書き換えおよびテストについて考察する。

Python を用いることで DBMS 実装の理解が容易になるのか、であるが、Python を用いるだけではならない、と言える。実際に、単に Java から Python に書き換えを行っただけの SimpleDB の Python による実装の、クラス構成や処理の流れは Java のソースコードと何ら変わりないという現状では、違いはプログラミング言語が Java と Python で異なっているだけであり、わかりやすさに影響するのは、プログラミング言語への習熟度だけである。次の、書き換えについての考察でも述べるが、Java のソースコードでは明示的に変数の型が宣言されていることから、ソースコードの一部分のみを見た場合には、どのデータ型に対する処理を行っているのかがわかりやすい一方で、Python のソースコードでは型情報を削除してしまったために、デバッグ時にはデータ型を調べるところから始める必要があることも多かった。Python のソースコードでは、それなりの記述量になる変数の型宣言がないことで、処理の流れに集中できるという利点はあるものの、SimpleDB では public 宣言されているテスト以外のクラスは 93 あり、これらの数多くのクラスを把握するには、変数の型宣言は有用であると考えられる。以上の考察からは、Python を用いるだけでは DBMS 実装の理解が容易になることはなく、理解を容易にするためには、単に Python を用いるだけでなく、Python の利点を活用した実装を行う必要があると言える。

次に、Java から Python へのソースコードの書き換えについて考察する。前述した点として、Python では変数や関数の型宣言が不要であるため、書き換えでは全て削除してしまったことで、どのデータ型に対する処理を行っているのかが、わかりにくくなってしまったのは、望ましくない点である。Python では、変数や関数に型アノテーションを付けることができる。処理系は、プログラムの実行にあたり、変数や関数の型アノテーション情報を用いることは無いが、開発者にはヒントになり、また処理系とは別の静的解析ツールによる型検査を行うことができる。Java から Python へ書き換える際には、元々の Java のソースコードには型宣言があるため、書き換え作業が多少煩雑にはなるが、Python の型ヒントへの書き換えは単純作業で可能である。型アノテーションを付けることで、デバッグ時にはデータ型を調べる必要がなくなり、また静的解析ツールによる型検査が行えるようになることは利点である。

変数の型情報に関しては、本研究では教育目的での開発であることから処理性能を重要視しないため、実行時に積極的に型を検査する利点は大きいと考えられる。Python では、`type` 関数および `isinstance` 関数により実行時に型情報を取得できる。デバッグ時には `assert` 文（言語によっては関数）により、変数の値の範囲等を検査するのと同様に、型を検査することが考えられる。

さらに、Python において、型に基づき適切な処理を行うための洗練された方法として、抽象基底クラスやプロトコル、データクラス (dataclass) がある。抽象基底クラスおよびプロトコルは、引

数として受け取ったオブジェクトがあるメソッドをサポートしているか、確認できるようにするための仕組みである。抽象基底クラスは実行時に、プロトコルは型検査器による確認を可能にするが、プロトコルは実行時に確認することも可能である。単純に `type` 関数および `isinstance` 関数を用いるだけでは、型に応じた処理を行うには、対象とする型を列挙する必要があるが、抽象基底クラスおよびプロトコルを用いることで、メソッドに注目したかたちで型に応じた処理を抽象化することが可能である。

データクラスは、複数のインスタンス変数を保持する機能に特化したクラスであり、コンストラクタや比較演算のための特殊メソッドを自動生成する機能を提供する。演算子が用いられた時に呼ばれる特殊メソッドが自動生成されることから、例えば、等価性の比較が容易に行えるようになる。

これら、型アノテーション、抽象基底クラス、プロトコル、データクラスは、比較的新しい機能である。標準化されているプログラミング言語と異なり、コミュニティで開発され、広く用いられているプログラミング言語では、段階的に毎年のようにアップデートされていくため、定期的に知識をアップデートする必要性を実感した。

最後に、テストについて考察する。実装手順の節で述べたが、SimpleDB が提供するテストは、機能テストのみであり、ユニットテストは提供されていない。機能テストの提供目的は、機能の使い方の例示であるとも言える。そのため、ユニットテストが提供されていないことも含めて、網羅性については考慮されていないと考えられる。SimpleDB の開発目的からすれば、簡単な機能テストのみの提供となるのは自然な成り行きであるが、Python への書き換え時にはユニットテストがあれば、デバッグが容易になり、より効率良く書き換えができたと考えられる。また、Python に適した処理に変更する、即ちリファクタリングを行うには、ユニットテストおよび網羅性の高い機能テストが必要となる。

4 今後の課題

本研究の目的は、データベースを学ぶ初学者が理解しやすい実装の DBMS を開発することである。その目的を達成するための課題として、既存モジュールの活用、モジュール化、可視化がある。

既存モジュールの活用

Python で実装することを前提とすると、既存モジュールを活用するという選択肢が有力になる。Python の高い生産性の理由の 1 つとしてあげられるのが、数多くの既存モジュールが提供されていることである。Python でプログラミングをするにあたり、デファクトスタンダードとなっているモジュールを用いない理由は無いと言っても良い。例えば、データ処理では、数値計算のための NumPy、データ分析のための Pandas を用いることが当然であり、NumPy, Pandas を用いて実現できることを、独自に実装することに意味は無い。

一方、SimpleDB は Java 標準のライブラリ以外を使用することなく、実装されている。Java にもライブラリのレポジトリは存在し、デファクトスタンダードとなっているライブラリはあるものの、SimpleDB は外部のライブラリには依存せずに、必要な機能を実装している。そのため、主要なクラスの部品となるクラスが数多く定義、実装されることになり、ソースコード全体としては見通しの悪いものとなっている。

従って、既存モジュールの活用という点では、SimpleDB が実装している機能のうち、Python のデファクトスタンダードとなっているモジュールが提供している機能については、それらのモジュールに置き換えることで、抽象度が上がり、理解しやすい実装となると考えられる。

モジュール化

Python でデファクトスタンダードとなっているモジュールが活用されている理由として、組み合わせて使用することが非常に容易であることがある。例えば、Pandas は NumPy を基盤に構築されている。NumPy が基本的な配列のデータ構造とその操作、計算処理を提供し、Pandas は NumPy の上のレイヤのモジュールとして、応用的なデータ操作を提供している。さらにその上のレイヤとして、機械学習処理を提供する Scikit-Learn があり、入出力に Pandas のデータ形式を用いている。NumPy, Pandas, Scikit-Learn は、それぞれ機能的に異なる、独立に開発されているモジュールであるにも関わらず、それぞれの責務が明確であり、機能独立性が高く、関連性がわかりやすいことが、容易に組み合わせ可能であることにつながっている。

このようなモジュール間の関係性で言えば、NumPy, Pandas を基盤として、その上のレイヤとしてデータベースの機能を実現するモジュールを構築する方法が考えられる。Pandas で可能なことは Pandas に任せ、Pandas が提供しないデータベース機能を、その上のレイヤのモジュールとして実現することで、抽象度が上がり、理解しやすい実装となると考えられる。

可視化

既存モジュールの活用、モジュール化は、理解しやすいソースコードを実装するための課題として述べた。理解しやすくするということでは、実装したソースコードが実現している処理の流れを理解しやすくすることも必要であり、そのための手段としては可視化がある。

DBMS は、選択、結合、ソート、検索といったデータ処理に関わる処理の他、同時実行制御や障害回復などの実行制御に関わる処理を行い、それぞれ様々なアルゴリズムが実装される。アルゴリズムの説明を行う際には、例となるデータセットについて実行の流れを図示することが多い。説明では少ない固定のデータセットを用いるが、実際に動作する DBMS がその処理を可視化することができれば、様々なデータセットについての処理を確認することができ、アルゴリズムの理解度が上がると考えられる。

5 おわりに

本論文では、教材として開発された教育用途の DBMS の比較を行い、その差異を明らかにした。さらに、教育用途の DBMS の 1 つである SimpleDB を Python で実装し、Python で DBMS を実装することで、データベースを学ぶ初学者が DBMS の実装を理解しやすくなるかを検証した。単に Java から Python に書き換えただけの DBMS の実装では、DBMS 実装の理解は容易にはならないことがわかった。Java から Python に書き換えた DBMS の実装で

は、クラス構成や処理の流れは Java のソースコードと何ら変わりないためである。

そこで今後の課題として、既存モジュールの活用、モジュール化、可視化をあげた。まずは、既存モジュールの活用、モジュール化に取り組み、データベースを学ぶ初学者が理解しやすい実装の DBMS の開発を進める。

参考文献

1. SQLite. Available: https://www.sqlite.org
2. Xv6, a simple Unix-like teaching operating system. Available: http://pdos.csail.mit.edu/6.828/xv6
3. The SimpleDB database system. Available: https://cs.bc.edu/ sciore/simpledb/
4. Sciore E. Database Design and Implementation, 2nd ed. Springer; 2020.
5. Cmu-db/bustub. Available: https://github.com/cmu-db/bustub
6. MIT-DB-Class/go-db-2024. Available: https://github.com/MIT-DB-Class/go-db-2024/
7. Berkeley-cs186/sp25-rookiedb. Available: https://github.com/berkeley-cs186/sp25-rookiedb/
8. CMU 15-445/645 database systems. Available: https://dsg.csail.mit.edu/6.5830/
9. MIT 6.5830/6.5831: Database systems. Available: https://dsg.csail.mit.edu/6.5830/
10. UCB CS186 introduction to database systems. Available: https://cs186berkeley.net
11. UCB CS186 projects. Available: https://cs186.gitbook.io/project