Vol.2018-CE-143 No.9
2018/2/17

プログラミング学習環境Bit Arrowにおける JavaScriptによるデータベースAPIの設計と実装

岩田 麻暉[1]　長島 和平[1]　長 慎也[2]　兼宗 進[3]　並木 美太郎[1]

概要：データベースの概念は情報科学の分野において重要な学習項目の一つである．近年では SQL を用いた問い合わせだけでなく，Web 表示やデータ解析で後処理にプログラミングも必要である．しかし，データベースを授業で扱い，複数の生徒に配布するなどの環境を整えることは，通常の RDBMS では容易ではない．そこでプログラミング学習環境における JavaScript によるデータベース API を提案する．オンラインプログラミング学習環境「Bit Arrow」に組み込むことで，容易にデータベースの配布や共有が可能となり，より効率的にデータベースの概念を学習することができる．プログラミング学習環境「Bit Arrow」では複数の言語を扱うことが可能であり，複数の言語を JavaScript にトランスパイルするため，ブラウザ上での動作ができる．つまり，JavaScript で API を作成することで他の言語でも操作可能となる．

キーワード：JavaScript, データベース API, Bit Arrow

1. はじめに

データベースの概念は情報科学の分野において重要な学習項目の一つである．データベースの学習において代表的な概念がリレーショナルデータベース (RDB) である．RDB の教育では一般的にその仕組みを理解するためにデータ操作の演習を行われる．データベースを行う演習において使われる演習用ツールはデータベース管理システム（DBMS）や表計算ソフトが見られる [1]．しかし，DBMS は一般的に実用向けのツールであるため，環境設定が困難である．また，DBMS は業務用であるため，初心者には難しい操作が多く，混乱を招きやすい．DBMS に接続するライブラリ（PDO[2], node.js を使った Sequlize[3] など）もいくつかあるがそのライブラリも同様である．また，授業の演習でこのようなツールを使う際には，通常教育機関内の演習室を使って行われることが多いが，その演習端末に導入することも容易ではない．近年ではデータベースの操作だけでなく，そのデータベースを使った解析や Web 表示ができるプログラミングの能力も重要視されている．また，SQL を簡単に学べるソフトウェアが増えてきているが，SQL などを用いた問い合わせが中心で，データ解析や後処理までを含むプログラミング環境は少ない．

1　東京農工大学
2　明星大学
3　大阪電気通信大学

2. 課題

2.1 関連研究

2013 年にデータベース操作の学習が可能なオンライン学習教材の設計と実装についての研究 [1] が行われている．彼らの研究では sAccess という学習環境で RDB の基本概念を，SQL エディタで SQL を，PHP エディタでプログラム中での SQL 実行と HTML の関係を学べるツールを開発した．オンラインでデータベースを SQL の命令を使って操作できる学習教材である．従来のデータベースの教育現場で使われてきた Access や Excel よりも実習に適している機能が多く，従来の授業でデータベースにもちいられてきたツールと比較することで sAccess の利点について議論している．データベース教育において調査した結果，学習環境の提案に意味があると判断し，sAccess の設計と実装を行った．しかし，[1] の研究ではデータベースの問い合わせが主な内容であり，プログラミングは主な内容ではない．本研究では問い合わせだけでなく，データの解析や後処理のためのプログラミングもできるように設計し，実装する．

また，RDBMS に関しては従来のものでは環境を整えるのが困難であり，データベースの配布や共有が容易にできない．既存の RDBMS[6] はほとんどが商業用であるため，大人数のユーザ登録が困難で，教育に適していないことが多い．教育現場では演習を行うためにデータの配布やデー

ⓒ 2018 Information Processing Society of Japan

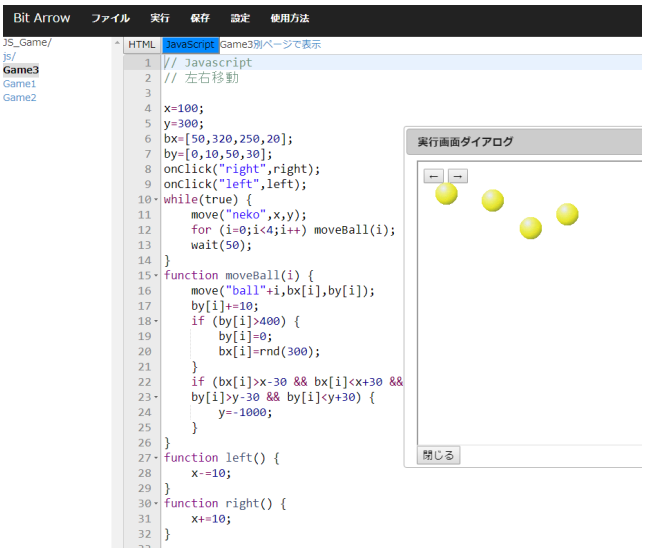

図 1　BitArrow の実行例

タベースの共有が容易にできた方が望ましい．

2.2　Bit Arrow

Bit Arrow[4] は JavaScript，C やドリトルなど複数の言語に対応している初心者向けのプログラミング学習環境のことである．コーディングと実行を Web ブラウザで行い，コードをクラウド上で保存することができる．Bit Arrow は複数の言語に対応しているが，その仕組みは言語 C やドリトルのプログラムを JavaScript にトランスパイルして実行している．授業支援環境として，生徒が作成したプログラムが自動的に保存され，教員がその状況を確認したり，課題を評価したりする機能が実装されている．BitArrow でのプロミングの実行例 [5] を次の図 1 に示す

2.3　本研究の目標

2.1 で述べた課題の元，次のような研究方針とした．

1). プログラミング学習環境として RDB の概念が学習できる教育支援機能を構築する．RDB が学習でき，かつプログラムが書ける学習環境はまだ少ない．近年では，RDB の概念を理解するだけでなく, そのデータを解析したり，分析したりするプログラミングの技術が必要とされている．その技術が学習しやすいような教育支援機能を構築する．プログラミング学習環境としては，2.2 で述べた，ブラウザだけあれば学習できる「Bit Arrow」に組込む．

2). SQL を用いた問い合わせだけでなく，データ処理や解析も行えるプログラミングの学習環境として使えるようにする．そのために RDB 操作ができる API を提案する．

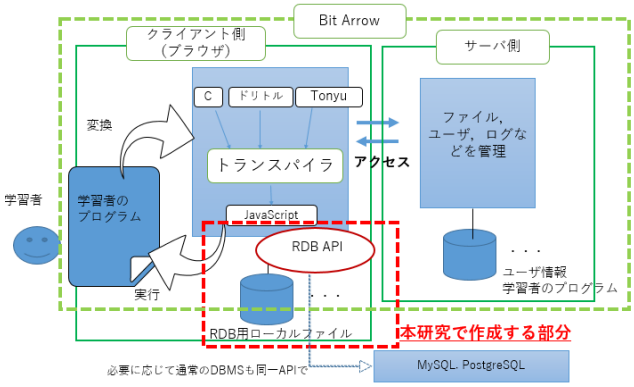

図 2　全体構成図

3.　設計

3.1　設計方針

JavaScript による RDB の操作ができる API（以下，JS-RDB API と呼ぶ）を設計するにあたって SQL との対応は重要な要素の一つとなる．そこでまず，SQL の命令を文字列で発行する方式を取ることにした．しかし，文字列であると当然，文字列を多用しなくてはならないため，プログラムが読みづらくなってしまう．そこで JavaScript で作成するという特徴を活かして，オブジェクトとして設計することにした．また，メソッドの名前などもなるべく SQL に近い形とした．その理由としては，本研究では実装は行わないが，JS-RDB API が DBMS との接続を行えるようにメソッドを対応させることができるようにするためである．

3.2　RDB 操作の主なオブジェクトとメソッド

設計するオブジェクトとメソッドについて以下に述べる．代表的なメソッドとは本研究において設計するメソッドの主要なメソッドのみについて説明する．まず，TableObject である．TableObject はリレーショナルデータベースの表オブジェクトである．以降では仕様の説明のため，tabObj は返り値がテーブルオブジェクトのものを表す．().method のように括弧の部分が大文字となっているのはコンストラクタの型を示している. つまり，TABOBJ の場合はテーブルオブジェクトのコンストラクタを表す．LIST とは複数指定できるという意味で使用する．

3.2.1　表生成

```
tabObj = new TableObject(TABNAME);
tabObj = new TableObject(TABNAME, ARRAY);
tabObj = new TableObject(TABNAME, JSON);
```

文字列 DBNAME の名前を持つ表を生成する．最初はデータがないものとする．文字列 ARRAY を引数に指定すると，その属性を持つ表を生成する．

このような記述方法にした理由としては，JavaScript の特性であるプロトタイプチェーンを利用するためである．

```
student = new TableObject("Student",
    "id", "name", "dept_id", "gender");
student = new TableObject("Student",
    ["id", "name", "dept_id", "gender"]);
student = new TableObject("Student",
  {"id":"int", "name":"char",
    "dept_id":"int", "gender":"char"});
```

図 3 ソースコード:テーブル作成

```
tabObj = student.select("id", "name");
tabObj = student.select(
  [["id", "学籍番号"], ["name", "氏名"]]);
tabObj = student.select({id:"学籍番号", name:"氏名"});
```

図 4 ソースコード:射影

表を扱うメソッドをオブジェクトとして定義することで，表を操作するメソッドをそのコンストラクタに連結して記述できるようにするためである．

SQL では表を作成するときデータのスキーマを決めるが，今回は JavaScript での実装のため，データの型の指定がなくてもプログラムは動作するように設計する．ただし，後に SQL での変換も関わってくるため，データの型を定義できるようにも設計しておくことにする．データの型を定義する場合は JSON 形式で宣言を行う."属性名" : "データの型", ... というように宣言する．以下の例では変数 student には表オブジェクトが格納されているとする．図 3 の例では属性 id, name, department を持つ空の表を生成する．

3.2.2 射影

表の射影を返す．結果は新しい表である．

```
tabObj = TABLEOBJ.select(ATTRLIST);
```

ATTRLIST には文字列で属性を渡す．ATTRLIST で射影する属性の文字列を指定する．図 4 の例は先ほどの表から id と name の属性だけにした表に射影する．引数に属性名を並べて書いた時は，その属性を新しい表にする．射影については，集約関数とも深く関係するので，後の章で再度示す．select では，名前を配列に，またはオブジェクトにすることで as としても使える．

3.2.3 選択

条件を指定してその条件を満たすレコードを検索して新しい表として返す．

```
tabObj = TABLEOBJ.where(ConditionalFunction,
ARGLIST);
```

ARGLIST は条件関数で使うための変数のリストのことである．ARGLIST で検索する値を指定する．ConditionalFunction（条件関数）は SQL の where の条件式に該当し，無名関数を渡す．この関数の実体は，true/flase を返す return の一文で構成される述語と考える．ConditionalFunction の引数は指定する要素と record オブジェクトを渡す．第一引数を rec とする．それ以降の引数に選択したい要素の変数を指定する．以降の ConditionalFunction も第一引数はレコードのための変数とする．選択は SELECT にまとめることも考慮したが，SQL と同様に where で条件付けした方が理解しやすいという理由でこのような形式の設計とした．選択した結果を結果を射影する場合は，カスケーディングで tabObj = student.where(...).select(...) とする．その例を図 5 に示す．

```
tabObj = student.where(
  function(student, name) {
    return student.name == name; }, "Bob"
).select("id", "name");
```

図 5 ソースコード:選択と射影

3.2.4 結合と from メソッド

選択は一つの表オブジェクトに対する操作であるが，結合 (join) は複数の表を指定する必要がある．複数の表の結合にはテーブルオブジェクトを引数に渡して結合するという方法を考慮したが，その記述方法だと引数が長くなってしまい，判別が困難であると判断したため，SQL と同様に from を使ったメソッドとした．SQL では FROM を使うが，本設計では，それを JavaScript で記述するために新たに TABLEGROUPOBJECT を導入する．TABLEGROUPOBJECT は，複数の表からなるオブジェクトである．複数の表については，from メソッドで指定した順に意味は持つ．from メソッドは，TABLEOBJECT のメソッドであり，

```
tabGroupObj = TABLEOBJ.from(TABLIST);
```

の形式をとる．TABLIST とはテーブルのリストのことである．TABLIST にテーブルオブジェクトのコンストラクタを複数渡すことができる．複数渡す場合はコンストラクタごとにカンマで区切る．TABLEGROUPOBJECT は，where メソッドにもある．形式は 3.2.3 と同じである．

```
tabObj = TABLEGROUPOBJ.where(
  ConditionalFunction);
```

ただし引数は，一つでなく，from で指定した順に引数として与える．ConditionalFunction は先ほどのべた条件関数のことである．

例えば，次のような学生の所属学科を表示する SQL を考える．

```
SELECT id, name, dept_name
FROM Student, Department
WHERE Student.dept_id = Department.dept_id
```

この SQL については，from メソッド，where メソッド，select メソッドを用いて SQL との対応は明確である．その対応例を図 6 に示す．

```
student.from(department)
.where(function(student,department) {
   return student.dept_id == department.dept_id;})
.select("id", "name", "dept_name");
```

図 6　ソースコード:結合

```
tabObj = student.orderby("name", false, "id", true);
tabObj = student.orderby(["name", false, "id", true]);
```

図 7　ソースコード:ソート

3.2.5　ソート（order by）

表のレコードの並べ替えを行う order by に相当する．並べ替えのキーとなる属性を順に書く．順序については，ASC(true) または DESC(false) で指定する．

tabObj = TABLEOBJ.orderby(ORDERARGLIST);
tabObj = TABLEOBJ.orderby(ORDERARRAY);

ORDERARGLIST とは並び替える属性とその順序を示したものであり，ORDERARRAY はその配列である．ソートの例を図 7 に示す．

3.2.6　集約関数と GroupBy

集約のメソッド (SQL では集約関数) は，表オブジェクトと表グループオブジェクトが使用できる．集約関数には COUNT（レコードの数を数える），SUM（総和），AVG（平均），MAX（最大値），MIN（最小値）がある．SQL では，集約関数は select 句に記述する．ここまで示した書き方に準じて，select は単に属性だけではなく，集約関数を書けるように考える．しかし，単に関数だけを書いて

select(ave(”salary”))

とは書けない．名前のスコープから，グローバルを探しに行くためである．工夫が必要である．そこで，まず Aggregate オブジェクトを導入する．Aggregate オブジェクトは，これらの集約関数をメソッドに含んでいるとする．その上で，

Aggregate.AGGRFUNC(PROP)

の形式をとるとする．AGGFUNC とは集約メソッドのことを指し，PROP は表の属性を指す．この employee 表で例えると，人数，総和，平均は，

employee.select(Aggregate.count(),
Aggregate.sum(”salary”),
Aggregate.sum(”salary”));

とすればよい．ただし，この記法ではファイルで実装するときにレコードオブジェクトの渡し方がよくわからない．そこで

[Aggregate.FUNCNAME, PROP]
[Aggregate.FUNCNAME, PROP, NEWPROP]

の形式とすることで select 内で評価する．FUNCNAME は集約メソッドの名前を指し，PROP は現在の属性，NEW-PROP は新しく設定する属性名を指す．

```
employee.groupby(["dept"])
.select("dept", [Aggregate.count, null, "部署ごとの社員数"],
  [Aggregate.sum, "salary", "部署ごとの給料総支払い"],
  [Aggregate.ave, "salary", "部署ごとの平均給料"]);
```

図 8　ソースコード:集約関数と Group by

次に GroupBy の設計について述べる．SQL の group by では，表の属性値が同じものを集めて集約関数を適用する．SQL では，属性名で集約関数を指定する．そこで，まず，先ほどの表グループの TABLEGROUPOBJECT オブジェクトを利用する．groupby に該当するものとして，TABLEOBJECT に対して groupby メソッドを設ける．groupby メソッドでは，グループにするキーとして属性を指定し，TABLEGROUPOBJECT を返す．from メソッドでは属性の異なる表をまとめたが，groupby ではキーが同値なレコードでまとめた異なる表の集合であることが異なる．groupby メソッドは，

tabgObj = TABLEOBJ.groupby(KEYATTRARRAY);

引数にグループ化する属性のリストを指定する．この表グループオブジェクトに対して，couut，sum，avg，min，max のメソッド (集約関数) を実行することで，個々のグループに対する値を求めることができる．先ほどの集約関数をそのまま用いることができる．違いは，引数に新しい属性を設定することである．

3.2.7　DB オブジェクト

SQL ではデータベースを宣言してからテーブルを作成する．そこで，データベースを管理する DB オブジェクトを設計する．DB オブジェクトは，内部に表を持つ．

dbObj = new DBObject(DBNAME);
dbObj = new DBObject(DBNAME, TABLELIST);

一つ目は，DBNAME(文字列) の名前を持つ DB を作成する．DBNAME のファイルがすでに存在する場合は，その DB をサーバーから読み込む．

3.2.8　DB の格納

Bit Arrow のファイル処理は Key-valueStore 形式であるため，その形式に合わせてファイル処理のメソッドを設計する．前提知識として，Bit Arrow ではプログラムを作成する際にクラス名，ユーザ名，プロジェクト名を決めて作成することができる．Bit Arrow に実装されているファイルの扱いのメソッドは getFromServer() と putToServer() である．

DBOBJ.save();

データベースにあるすべてのテーブルをサーバーに格納するメソッドである．このメソッドはファイルを閉じるメソッドと似たような動作のため，close としようとしたが，close は JavaScript で予約語となっているため，save とした．現在あるテーブルオブジェクトを Bit Arrow に実

ⓒ 2018 Information Processing Society of Japan

装されている putToServer を使ってライトバックさせる．このときに書き戻す key 文字列は（クラス名＋ユーザ名＋プロジェクト名＋ DB 名＋テーブル名）で，書き戻すデータの文字列は tabobj.data（二次元配列データ）を CSV 形式の文字列に変換させたものである．このようにキーを設定することでテーブルやデータベース同士の名前の衝突を回避できる．現在 DBObject が持つテーブルの名前のリストを文字列に変換させ，キーをクラス名+ユーザ名+DB 名+”___LISTFILES”としてサーバー上に保存する．

4. 実装

4.1 DB 実装の基本方針

まず，JS-RDB API ではファイルでデータを管理する．一つのデータベースについてはディレクトリと考え，一つの表を一つのファイルで管理する．また，本 API は，学習用として大規模なデータは想定せず，データをオンメモリで扱う．次に JavaScript はデータの型がないので，型について必要であれば別途ファイルで管理する．テーブルオブジェクトのデータをサーバーに保存する場合，データをすべて文字列に変換するが, データをサーバーから取得する際にはもう一度，データを文字列から元のデータの型に直す必要がある．そこでデータの型の指定（データベースでのスキーマ）を設定しておき，データの型指定がある場合は，データを文字列に変換させた文字列の先頭に “__type”＋“属性 1 の型．属性 2．・・・”という文字列を追加させることで区別することにした．JavaScript の場合，データの型は数値か文字列しかないため，int 型であるかどうかの判断をすればよいことになる．つまり，サーバーからデータを読み込む際に，先頭に__type があれば，データの型を決め，データの型の中に int 型があれば，文字列を数値に変換させる処理を加えるように実装した．

4.2 Bit Arrow での実装

本研究では作成したプログラムを Bit Arrow 上で動かすことが目的の一つであるため，Bit Arrow 上での動作確認を行った．Bit Arrow では，JavaScript でプログラミングを行う場合，HTML 用のファイルと JavaScript 用のファイルを切り替えてプログラミングを行っていくことができる．本研究で作成した RDBAPI は，サーバーのファイルから読み込む形で Bit Arrow でも作成できるように実装した．Bit Arrow から RDBAPI を使用する場合は，HTML でソースを読み込み，JavaScript 側で native RDB と宣言することで，RDBAPI のメソッドを使用することができる．native の後にオブジェクトを宣言することで BitArrow でそのオブジェクトがある外部ファイルが使えるようになる．

また，実装の動作確認のためにテーブルを出力させるメソッドを実装した．このメソッドは RDB.m2.print() と Bit Arrow 上で記述することで利用することができる．テーブルオブジェクト.data と HTML 側のテーブル id タブの divname を引数として渡すことで，指定したテーブルタブに表示させることができる．表示形式は CSS を使って自由に変更できるように実装した．

```
var sdb = new RDB.m1.DBObject("soccerDB");
var p1 = sdb.table("Players1");
var p2 = sdb.table("Players2");
RDB.m2.print(p1.data, "table1");

// 射影・選択
var result1 = p1.where(function(p1, name){
  return p1.name == name;
}, "カピノ").select("id", "name", "position");
RDB.m2.print(result1.data, "result1");

// 結合
var result2 = p1.from(p2).where(function(p1, p2){
  return p1.id == p2.id;
});
RDB.m2.print(result2.data, "result2");
```

図 9 ソースコード:Bit Arrow 実行例

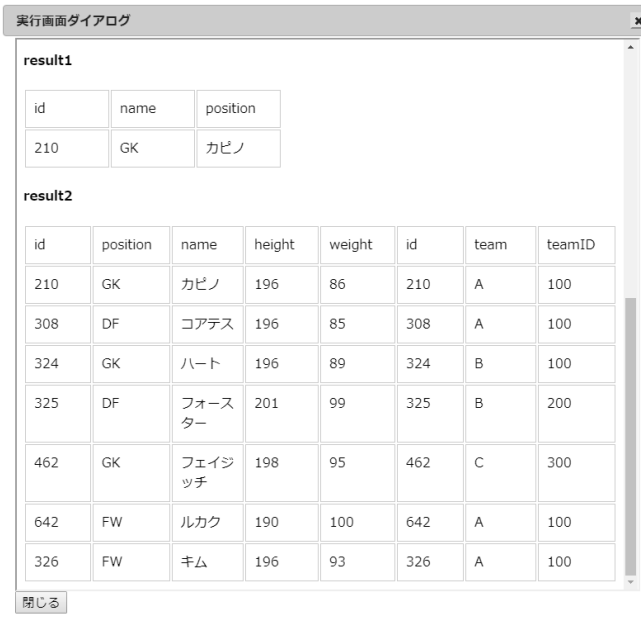

図 10 Bit Arrow 上の実行結果

実際に Bit Arrow で使えること示した Bit Arrow の実行画面ダイアログの図を図 10 に示す．Bit Arrow では実行ボタンを押すことで作成したプログラムが実行画面ダイアログに表示される．

図 9 のソースコードではあらかじめ,「soccerDB」という名前の DBObject をサーバーに保存してあるものとする．そのデータベースには「Players1」と「Players2」の二つの表が格納されている．「Players1」の表には属性として id, position, name, height, weight があり,「Players2」の表の属性には id, team, teamID がある．図 9 のプログラムを実行した結果が図 10 である．

表 1　実装した主なオブジェクトとメソッド

メソッド名	用途
TableObject	
insert	レコードの挿入
select	射影
where	選択
from	TableGroupObject 生成
remove	レコードの削除
update	レコードの更新
next	次のレコードを取り出す
orderby	並び替え
groupby	グループ化（TabGroupObj 生成）
DBObject	
table	テーブルの検索
add	データベースに表の追加
remove	データベースから表の削除
name	データベースの名前変更
save	サーバーに保存
TableGroupObject	
where	結合 (inner join)
select	groupby の時の複数射影
having	groupby の条件指定
join	結合（共通の属性を検索して結合）

4.3　実装したオブジェクトとメソッド

本研究で実装した主なメソッドを表 1 に示す．実装した主なオブジェクトは TableObject，DBObject，TableGroupObject である．表 1 は三つのオブジェクトのいずれかにカスケーディングする形で使用することができる．

5.　評価

5.1　評価方法

本研究の評価方法として次の二つの方法で評価を行った．

(1) SQL 教材を使用した比較評価

(2) JS-RDBAPI の性能評価

一つ目は，著者の研究室学生を被験者として，SQL の教材にある練習問題を JS-RDB API を使って解いてもらう．JS-RDB API を簡単に説明したサンプルファイルをテスト用に作成した BitArrow のアカウントに配布する．BitArrow のアカウントを使って SQL の問題を JS-RDB API を使って解けたかどうか，そして，JS-RDB API を使った感想をもとに評価を行う．もう一方は JS-RDB API の処理速度の評価を行う．実装したメソッドがどのくらいの処理性能なのかを評価する．

5.2　SQL との比較評価

SQL の教材を使った比較評価を行う．使用した SQL 教材は「スッキリわかる SQL 入門」[9] である．以下に「スッキリわかる SQL 入門」にある練習問題を抜粋したものを示す．なお，RDB をわかりやすく説明するために多少表現を変えている部分や省略している部分もある．

```
// 5-1 1 の例
var value = (80*4)- (86+68+91);
exam1.update({"class1":value}function(e){
  return e.id == 1046;});

// 5-1 2 の例
var result1 = exam1.where(function(e){
  return e.class1 >= 60 && e.class2+e.class3 >= 120&&
 (e.class1 + e.class2 + e.class3 + e.class4)*0.3 >= e.class4;
}).select("id");
```

図 11　ソースコード:SQL 問題 解答例

表 2　問題の例「試験結果テーブル」(exam1)

id	class1	class2	class3	class4	average
1046	86	(A)	68	91	80
1350	65	53	70	(B)	68
1877	(C)	59	56	36	856

問題

1. 現在登録されているデータ (表 2) を基に，0 の部分を update 文を使ってそれぞれ作成する

2. この試験に合格するには次の条件をすべて満たす必要がある．

（1）class1 の点数は 60 以上であること

（2）class2 class3 を合計した点数が 120 以上であること

（3）class4 の点数が全クラスの合計点の三割以上であること

これらの条件をもとに，合格者の受験 ID を抽出せよ．

この問題に対する SQL での解答を一部紹介する．

SQL の解答 (一部)

```
UPDATE exam1
SET class1 = (80*4) -(86+68+91)
WHERE id = '1046'
```

```
SELECT id
FROM exam1
WHERE class1 >= 60
AND class2 + class3 >= 120
AND 0.3 * (class1 + class2 + class3) <= class4
```

そして，JS-RDB API での解答例を図 11 のソースコードに示す．図 11 のソースコードではあらかじめ，exam1 の表のデータを入れておき，それを DBObject で読み込み，DBOBJ.table() でテーブルオブジェクトを読み込んだコンストラクタが exam1 であるものとする．

本研究室の被験者（6 人）に先ほどの問題の 2 を解いてもらった．簡単な RDP API の説明のためのサンプルファ

イルを BitArrow の教員用のアカウントで作成し，被験者にテスト用の BitArrow の生徒アカウントに配布することで，その生徒アカウントで問題を解いてもらった．つまり，BitArrow でのデータベースの配布の容易に行うことができた．その結果として，全員問題を解くことができた．そして，RDB API を使用した感想を以下にまとめる．

- RDB の各機能をこのように JS により扱えるのは，教育用に限らず様々な場面で活用できるように感じた．メソッド名や使い方も直感に沿っているため分かりやすかったが，RDB.m1. の部分が何を指しているのかわかりにくかった
- where の条件を無名関数で渡す部分が好きで，条件が多い時などは見やすくて良いと思った．
- JavaScript の特徴をうまく活かしている．Table Object に対して where のチェーンを繋げていくことができる点はとても良い．独自データのため ORM として用いることは難しいと思うが，教育用途として RDB の基本的な概念を習得することはできる． where メソッドの conditional function 内で undefined な変数を参照したときのエラー文が「Error: where: not found」なのはわかりにくいと感じた．またチェーンを繋げている場合のエラー発生箇所が必ずチェーンの先頭になってしまう点もデバッグがしづらいのではないかと感じた．
- 普段から JavaScript をよく利用しており、一方でデータベースはそこまで利用頻度が高くないですが、JavaScript で書けるので使いやすかった． ただ、SQL を使ったことがある人が使おうとすると、select from where の順番が異なっていることに多少違和感を覚えるかもしれない．JavaScript 経験者がこの API を使って RDB について学習してから実際の RDB を使う、というステップで学習するのに向いていると思った．
- 表示される表が見やすかった．実行したらすぐ確認できるのでストレスなく取り組むことができたし，1 ステップずつ確認しながら進められた．エラーが出たときにどの行でエラーが発生したのかを知らせてくれるので修正しやすかった．SQL との対応で見たときに，FROM がどうなっているのかがよくわからなかった．function() がそのような役割なのかよくわからなかった．

5.3 性能評価

性能評価に関しては，結合や BitArrow でのファイル処理の他にも計測したメソッドはあるが，本稿では代表的なメソッドの一つとして結合とファイル処理の性能評価について述べる．なお，評価では誤差を考慮して基本的に 5 回繰り返した平均を取得している．本研究で評価に用いた開発環境を次の表 3 に示す．

表 3 評価環境

ブラウザ名	バージョン
Chrome	61.0.3163.100 （64 ビット）
Firefox	57.0
Internet Explorer 11	11.674.15063.0

5.3.1 結合 評価

結合の評価を行うにあたって以下の条件でテーブルオブジェクトを作成した．ただし，N は任意のレコード数とする

- テーブル名：user
- 属性：id, name, gender
- id は 0〜N までを順番に格納し，name はランダムで生成した文字列を格納している．また，gender はランダムで F か M が格納されている．

この表と同じような表を三つ作成し，それを元に結合の処理時間を計測する．なお，JavaScript はブラウザによって処理時間が違うため，多く使われている Chrome, Firfox, Internet Explorer（IE）の三つのブラウザで計測を行った．まず，where を使った結合（inner join）をレコード数を変えて時間計測を行った．その結果を表 4 に示す．なお，表にある（x）は（*）に対して何倍かを表したものである．

表 4 ブラウザの違いによる性能比較（inner join）

各表のレコード数	Chrome[ms]	Firefox[ms]	IE[ms]
100 × 100 × 100	204(*)	50(*)	601(*)
1000 × 1000 × 100	18104(x89)	3674(x73)	35542(x59)
500 × 500 × 500	21699(x106)	4727(x95)	66622(x110)

inner join では現時点で最適化を施してない．そこで，ループを途中で抜ける仕様となっている join() の評価を行う．join では where を使った結合（inner join）と比べて，条件の指定は必要としないため，再帰関数を途中で抜ける処理方法とした．途中で抜ける方法とそうでない方法とでどのくらいの差がでるかを検証する．

計測した結果を次の表 5 に示す．A は natural join の結果で，B は inner join の結果である．表の (-) は時間がかかりすぎて計測ができなかったことを示す．

表 5 (A)natural join と (B)inner join の性能比較

	Chrome[ms]		Firefox[ms]		IE[ms]	
各表のレコード数	(A)	(B)	(A)	(B)	(A)	(B)
100 × 100 × 100	21	204	36	50	54	601
500 × 500 × 500	231	21699	573	4727	1133	66622
1000 × 1000 × 1000	552	-	2285	-	4397	-

ⓒ 2018 Information Processing Society of Japan

5.3.2 DB の読書きの性能

BitArrow 上での性能評価を行う．前述したように，本 API は，学習用として大規模なデータは想定せずに，データをオンメモリで扱うように実装した．また，5.3.1 ではブラウザごとの処理時間の違いが見て取れたため，本項では実行するブラウザを Chrome に絞って評価を行う．

データベースにおいてデータの読み込みに時間がかかっていては，教育現場で使われにくいことが推測できる．そのため，データを保存または読み込む時間計測を行う．次の条件のもと，時間計測を Bit Arrow 上で行う．ただし，n,N には個数が入り， 格納するデータはランダムに決められた長さの文字列を格納する．ただし，その文字列の長さは一定とする．

- 条件 1：テーブル名：Emp n ，属性数 3 つ，格納するデータの文字列数 3，レコード数 N
- 条件 2：テーブル名：Emp n, 属性数 5，格納するデータの文字列数 5，レコード数 N

DB1 では条件 1 のテーブルオブジェクトを二つ生成し，DBObject としたものである．DB2 では条件 2 のテーブルオブジェクトを 5 つ生成し，格納したものである．
まず，サーバーに保存する処理時間の結果を次の表 6 に示す．レコード数について計測したところ，10000 レコードまでは正常実行できたが，このレコード数を越えると、実行は不可能であった．

表 6 サーバーに保存する時間

DB の種類	表のレコード数	データサイズ (KB)	時間 [ms]
DB1	100	2.008	35
DB1	1000	21.808	44
DB1	10000	237.808	110
DB2	10000	1444.575	162

次に，データの読み込みも保存する時にデータ指定がある場合とない場合でどれほど時間に差がでるのかを検証するために，データの型指定ありとなしで時間計測を行った．その結果を次の表 7 に示す．

表 7 サーバーから読み込む時間

DB の種類	表のレコード数	データサイズ (KB)	時間 [ms]	
			指定なし	指定あり
DB1	100	2.008	64	66
DB1	1000	21.808	68	72
DB1	10000	237.808	169	175
DB2	10000	1444.575	164	223

6. 考察

6.1 記述性についての考察

JS-RDB API の使用感の調査を行った結果，SQL の問題を調査に協力してもらった全員が JS-RDB API で解くことができた．JS-RDB API を使って SQL の問題をプログラムとして記述することができたということである．また，時間の関係上，調査人数は少なかったものの，JS-RDB API を使用した感想を見る限り，JavaScript の特性を活かした仕様や無名関数を使った条件の渡し方は高評価となった．本研究で実装した JS-RDB API は，データベースを JavaScript で書けることによって，表の表示のさせ方も自由に設定でき，かつプログラミングも自由に行えるものとなった．また，SQL の問題の調査を行う過程で BitArrow で容易にデータベースやファイルを複数の人に配布できることも確認できた．しかし，エラーの表示，エラー処理の仕方や一部のメソッドがわかりにくいなどの指摘もあった．確かに，時間の関係上，エラー処理の設計や SQL でのトランザクション処理の設計と実装は行えなかったため，今後の課題である．

6.2 実行時性能についての考察

join の性能評価を行った結果，教育現場でも扱うことのできる性能速度であると判断できる．なぜなら，表 5 をみてわかるように，1000 × 1000 × 1000 の結合でも Chrome や Firefox は一秒とかからずに結合の処理が行われている．全体的に処理速度が遅い InternetExplorer でも約 4 秒ほどで処理が完了している．教育現場での基準があるわけではないが，授業で扱うデータサイズはレコード数 100 でも十分であると考えられる．そのため，教育現場で扱う速度の面では十分に速いといえるだろう．

次に，DB 読書きの時間時間計測を行った結果，DB 読書きの処理（データの保存，変換，読み込み）に時間がかからないことが分かった．1MB ほどのデータ量であっても，時間は 1 秒とかからないことが結果からみてわかる．また，データの型指定の有無に関しては，ない方が時間がかかっていないことが分かった．しかし，微々たる差であり，5 回繰り返した平均であっても誤差の振れ幅は数十ミリ秒となっているため，データ指定の有無はファイル処理に依存しないといえる．

7. おわりに

プログラミング学習環境 BitArrow においての JavaScript によるデータベース API の設計と実装，そしてその評価を行った．JS-RDB API を触ってもらう人数は少なかったものの，プログラミング学習環境で RDB が扱え，かつプログラミングも行える環境を構築することができた．RDBAPI の性能評価は処理速度の面に関して，実習に利用可能な結果となった．

本報告では RDB 操作やプログラムが書けるような教育支援機能の構築に主眼を置いたが，今後の課題として，RDBAPI のエラー処理やトランザクション処理などの設

計と実装などが挙げられる．また，大規模共有データの扱いなどを考慮した場合，通常の RDBMS を用いる方法が考えられる．この場合は，本 API の内容を SQL に変換することで大規模な共有データも例題として扱えるようにすることが課題である．

参考文献

[1] Kanemune Susumu, Nagataki Hiroyuki, Nobe Midori, Nakano Yoshiaki:“Design and Implementation of an Online Tool for Database Education”, IPSJ SIG Technical Report, pp.1-8, 2013.
[2] PDO, http://php.net/manual/ja/book.pdo.php
[3] Sequelize, 入手先 ⟨http://docs.sequelizejs.com/⟩
[4] 間辺 広樹, 長島 和平, 並木 美太郎, 長 慎也, 兼宗 進: 高校教育における複数言語によるプログラミング教育の提案，情報処理学会研究報告 コンピュータと教育（CE），Vol.3 No.3,pp.29-41(2017).
[5] オンラインプログラミング環境 Bit Arrow 入手先 ⟨http://bitarrow.eplang.jp/⟩
[6] Pietro Colombo, Elena Ferrari; “Efficient Enforcement of Action-Aware Purpose-Based Access Control within Relational Database Management Systems”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, Vol.27, 8, pp.2134-2147, 2015
[7] Inoue, Takeru; Asakura, Hiroshi; Uematsu, Yukio; et al. “Web API Database Systems for Rapid Web Application Development”, IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS, Vol.E93D, pp.3181-3193, 2010
[8] “A Visual Learning Tool for Database Operation” Proceedings of 8th Workshop in Primary and Secondary Computing Education (WiPSCE 2013), pp.40–41, 2013-11-11.
[9] スッキリわかる SQL 入門，中山清喬・飯田理恵子 2013，株式会社インプレスジャパン
[10] Saez-Lopez, Jose-Manuel; Roman-Gonzalez, Marcos; Vazquez-Cano, Esteban; “Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools”, Computers & Education Vol.97, pp.129-141 , 2016

ⓒ 2018 Information Processing Society of Japan