Project 3. No SQL

- Group 7 -
岡 亮 (81918015), 田部　悠介 (81919076),
永山　悠人 (81919273), 和久井　拓 (81920343)

テーマ：**ポケモン**

ポケモンとは, ゲームボーイ用ソフト「ポケットモンスター赤・緑」から始まるポケットモンスターシリーズ内に登場する架空の生き物である.各々のポケモンには重さ, 身長といった生物的特徴の他に, ほのお, みず, くさ, といったタイプ (type), 覚えられる技 (move) などの要素が結び付けられている. ポケモンのメインとも言えるバトルシステムでは, ポケモンのタイプや技の威力などを考慮したポケモンの選択と技の選択が戦略の要である.

このレポートでは, バトルに影響しない重さや身長などのデータを除き, ポケモンとそのポケモンの持つタイプや技などの情報の管理をテーマとする.

データベース：**MySQL** vs **MongoDB**

表 1. 使用したデータベースとバージョン

RDB	MySQL v14.14
NoSQL	mongoDB v3.6.0

RDB は MySQL v14.14 を使用した. 一方 NoSQL は mongoDB を使用した. mongoDB では外部の collection と join する場合に lookup operator を使用するが, mongoDB 3.2 以降でしか利用できないため，比較的新しい mongoDB 3.6 を使用した.

データ詳細

ポケモン一覧(pokemon)，技一覧(move)，ポケモンが覚えられる技(pokemon_move)の一覧に関して ゲーム情報サイト [1] から Web スクレイピングした.
pythonのBeautiful Soupというライブラリを使用.

・MySQL

MySQL の構造を以下に示す. また各テーブルのレコード数を表 2 に示す. 下線が引かれた属性は主キーであり，”[table_name(column_name)]”が後ろ書いてある属性は外部キーである.

- pokemons (no, name, type1[types(id)], type2[types(id)])
  ポケモンの基本情報. no はポケモンの図鑑番号であり pokemons テーブルの主キー. ポケモンは最低 1 種類, 最大で 2 種類のタイプを持つ. タイプが 1 種類のポケモンは type2 は null.

- moves (id, name, type[types(id)], tech, power, accuracy, pp)
  ポケモンの覚える技の情報. 図1. の "覚える技" に対応するテーブル.

- pokemon_move (pokemon_no[pokemons(no)], move_id[moves(id)])
  ポケモンと, そのポケモンが覚える技の対応表. ポケモンは複数の技を覚え, 技は様々なポケモンが覚えるため多対多の関係.

- types (id, name)
  ポケモンやポケモンの技のタイプの一覧.

- comps(m_type[types(id)], p_type[types(id)], rate)
  攻撃技のタイプ(m_type)と対象のポケモン(p_type)の相性によるダメージ倍率(rate)テーブル.

表 2. 各テーブルのレコード数

pokemons	moves	pokemon_move	types	comps
809 件	668 件	45673 件	18 件	324件

図 1. ポケモンの情報

- MongoDB

MongoDB のデータ構造は以下のように定義した.

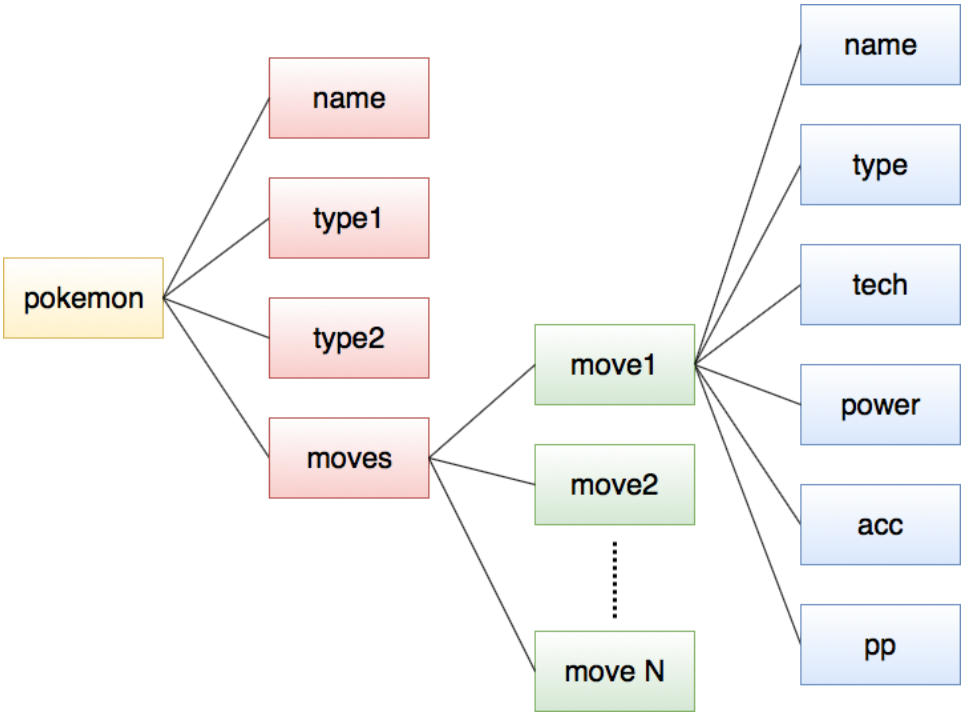

図2. MongoDB のスキーマ

図3. MongoDB のドキュメントの例

MongoDBに挿入したcollectionとその形式について以下に示す．MongoDBでは collection に挿入したデータそれぞれに対して自動的に識別子である "_id" 属性が付与される．以降では "_id" の説明は省略する．""内が属性の名前であり，後ろに: [{}] が続く属性は中身に配列を持つことを意味する．

- pokemon ("_id", "no", "name", "type1", "type2",
  "move": [{"m_name", "m_type", "m_tech", "m_power",
  "m_acc", "m_pp",},...
  ]
)

  ポケモンの基本情報を管理
  - **no**: ポケモン図鑑の番号
  - **name**: ポケモンの名前
  - **type1**, **type2**: ポケモンのタイプID（独自に作成）．**type** collection の id と対応している．ポケモンによってはタイプは1つの場合と2つの場合があるため，type2がない場合もある．
  - **move**: そのポケモンが覚える技一覧，階層構造を取り技は配列になる
    - **m_name**: 技の名前
    - **m_type**: 技のタイプID
    - **m_tech**: 技の種類を示し，「物理」「特殊」のどちらか

    - **m_power**: 技の威力（攻撃技でない場合，また自身や相手のポケモンの重さ・なつき度などに応じて威力が変わる場合は "null" ）
    - **m_acc**: 技の命中率
    - **m_pp**: 技を使用できる上限回数

実験環境

Mongo shell に直接クエリを打ち込むのは面倒であるため， 別途プログラミング言語を使用してクエリを実行することを考えた．以下に実験環境を示す
.

表3. 使用マシンスペック

OS	Ubuntu 16.04.5 LTS
CPU	Intel(R) Xeon(R) CPU E5-2643 v2 @3.50GHz
RAM	32GB

以下に使用した言語のバージョンとモジュールのバージョンを示す.

- 使用言語: **Node.js v11.10.1**
    - **mongodb@3.2.7**
      javascriptファイルからクエリを実行可能にするために使用.

    - **csvtojson@2.0.10**
      mongoDB の検索クエリが json 形式であるため，挿入するデータも json形式であることが望ましいと考えた．Webスクレイピングによって取得した csv ファイルを json ファイルに変換するために使用.

    - **assert**
      エラーチェックに使用.

Sample Query

同じ条件を表示するクエリを MySQL と Mongo DB にそれぞれ適用し比較した.
検索する条件は段階的に制約されるように以下の 3 種類を試行した.

1) ほのおタイプのポケモンもに表示
2) ほのおタイプのポケモンで, かくとうタイプの技を覚えるポケモンの表示
3) ほのおタイプのポケモンで, 威力 100 以上のかくとうタイプの技を覚えるポケモンの表示

本章ではそれぞれの条件について MySQL と MongoDB で検索するクエリとその結果を示す.

1) ほのおタイプのポケモンのみ表示

- MySQL

・クエリ

```
select p.no,p.name from pokemons p, types t
where (p.type1 = t.id or p.type2 = t.id) and t.name = 'ほのお';
```

・結果

```
| 004 | ヒトカゲ          |
| 005 | リザード          |
| 006 | リザードン        |
| 037 | ロコン            |
| 038 | キュウコン        |
| 058 | ガーディ          |
～～～～～～～～～～～～
| 758 | エンニュート      |
| 776 | バクガメス        |
| 806 | ズガドーン        |
64 rows in set (0.00 sec)
```

- MongoDB

・クエリ

```
db.pokemon.find({{$or:[{type1:“ほのお”}, {type2:“ほのお”}]},
{_id:0, type1:0, type2:0, move:0});
```

・結果

```
{“no”:”004”,”name”:“ヒトカゲ”}
{“no”:”005”,”name”:“リザード”}
{“no”:”006”,”name”:”リザードン”}
{“no”:”037”,”name”:“ロコン”}
{“no”:”038”,”name”:”キュウコン”}
{“no”:”059”,”name”:”ウインディ”}
～～～～～～～～～～～～～～～～～～～～～
{“no”:”758”,”name”:”エンニュート”}
{“no”:”776”,”name”:”バクガメス”}
{“no”:”806”,”name”:”ズガドーン”}
```

2) ほのおタイプのポケモンで, かくとうタイプの技を覚えるポケモンの表示

- MySQL

・クエリ

```
select p.no,p.name from pokemons p, pokemon_move pm,
types pt, types mt, moves m
where (p.type1 = pt.id or p.type2 = pt.id)
and pt.name = ‘ほのお’ and p.no = pm.pokemon_no
and pm.move_id = m.id and m.type = mt.id
and mt.name = ‘かくとう’;
```

・結果

```
| 004 | ヒトカゲ          |
| 005 | リザード          |
| 006 | リザードン        |
| 058 | ガーディ          |
| 059 | ウインディ        |
| 077 | ポニータ          |
～～～～～～～～～～～～
| 726 | ニャヒート        |
| 727 | ガオガエン        |
| 776 | バクガメス        |
48 rows in set (0.02 sec)
```

- MongoDB

・クエリ

```
db.pokemon.find({$and:[{$or:[{"type1":"ほのお"}, {"type2":"ほのお"}]}
,{"move.m_type":"かくとう"}]} ,{_id:0, type1:0 ,type2:0,move:0});
```

・結果

```
{“no”:”004”,”name”:“ヒトカゲ”}
{“no”:”005”,”name”:“リザード”}
{“no”:”006”,”name”:”リザードン”}
{“no”:”058”,”name”:“ガーディ”}
{“no”:”059”,”name”:”ウインディ”}
{“no”:”077”,”name”:”ポニータ”}
～～～～～～～～～～～～～～～～～～
{“no”:”726”,”name”:”ニャヒート”}
{“no”:”727”,”name”:”ガオガエン”}
{“no”:”776”,”name”:”バクガメス”}
```

3) ほのおタイプのポケモンで, 威力 100 以上の
かくとうタイプの技を覚えるポケモンの表示

- MySQL

・クエリ

```
select p.no,p.name from pokemons p, pokemon_move pm,
types pt, types mt, moves m
where (p.type1 = pt.id or p.type2 = pt.id)
and pt.name = 'ほのお' and p.no = pm.pokemon_no
and pm.move_id = m.id and m.type = mt.id
and mt.name = 'かくとう' and m.power >= 100;
```

・結果

```
| 004 | ヒトカゲ          |
| 005 | リザード          |
| 006 | リザードン        |
| 058 | ガーディ          |
| 059 | ウインディ        |
| 126 | ブーバー          |
～～～～～～～～～～～～
| 721 | ボルケニオン      |
| 727 | ガオガエン        |
| 776 | バクガメス        |
30 rows in set (0.01 sec)
```

- MongoDB
  - クエリ

```
db.pokemon.find({$and:[{move:
{$elemMatch:{m_type:"かくとう",m_power:{$gte: 100}}}}
,{$or:[{type1:"ほのお"},{type2:"ほのお"}]}]},{_id:0, type1:0,type2:0,move:0});
```

  - 結果

{“no”:”004”,”name”:“ヒトカゲ”}
{“no”:”005”,”name”:“リザード”}
{“no”:”006”,”name”:”リザードン”}
{“no”:”058”,”name”:“ガーディ”}
{“no”:”059”,”name”:”ウインディ”}
{“no”:”126”,”name”:”ブーバー”}
～～～～～～～～～～～～～～～～
{“no”:”721”,”name”:”ボルケニオン”}
{“no”:”727”,”name”:”ガオガエン”}
{“no”:”776”,”name”:”バクガメス”}

また, 後述する, "MongoDB で複数 collection を管理した場合" のクエリと結果を以下に示す.

2') ほのおタイプのポケモンで, かくとうタイプの技を覚えるポケモンとその技の名前を表示

- MongoDB
  - クエリ（一部）

```
collection
   .aggregate([
     {$lookup: {
         from: 'type',
         localField: 'type1',
         foreignField: 'id',
         as: 'type1'
       }},
     {$lookup: {
         from: 'type',
         localField: 'type2',
         foreignField: 'id',
```

```
      as: 'type2'
    }},
  {$lookup: {
    from: 'move',
      localField: 'moves.move_id',
      foreignField: 'id',
      as: 'moves'
    }},
  {$match:
        {$or: [
            {"type1.name": 'ほのお'},
            {"type2.name": 'ほのお'}
          ]}},
  {$unwind: "$moves"},
  {$lookup: {
      from: 'type',
      localField: 'moves.type',
      foreignField: 'id',
      as: 'moves.type'
    }},
  {$match: {
      "moves.type.name": "かくとう",
    }},
  {$project: {
    "_id": 1,
    "no" : 1,
      "name": 1,
      "type1.name": 1,
      "type2.name": 1,
      "moves.name": 1,
      "moves.type.name": 1,
      "moves.power": 1
    }},
  {
    $group: {
      "_id": "$_id",
      "no": {
        $first: "$no"
      },
      "name": {
        $first: "$name"
      },
      "type1": {
        $first: "$type1.name"
      },
      "type2": {
        $first: "$type2.name"
      },
      "moves": {
        $push: "$moves"
      }
    }
  }
```

```
  ])
  .toArray((err, docs) => {

    /** アサーションテスト */
    assert.equal(err, null);

    /** 検索結果を整形してコンソールに出力 */
    console.log(JSON.stringify(docs, null, 2));
    ...
```

・結果


```
[
  {
    "_id": "5d2ac4f275821727d8d8c032",
    "no": "776",
    "name": "バクガメス",
    "type1": [
      "ほのお"
    ],
    "type2": [
      "ドラゴン"
    ],
    "moves": [
      {
        "name": "きあいだま",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 120
      },
      {
        "name": "リベンジ",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 60
      },
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
  {
    "_id": "5d2ac4f275821727d8d8bfc3",
    "no": "667",
    "name": "シシコ",
    "type1": [
      "ほのお"
    ],
    "type2": [
      "ノーマル"
    ],
    "moves": [
      {
        "name": "いわくだき",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 40
      }
    ]
  }
]
```


3') ほのおタイプのポケモンで, 威力 100 以上の
かくとうタイプの技を覚えるポケモンとその技の名前を
表示

・クエリ（一部）

```
collection
   .aggregate([
     {$lookup: {
         from: 'type',
         localField: 'type1',
         foreignField: 'id',
         as: 'type1'
       }},
     {$lookup: {
         from: 'type',
```

```
      localField: 'type2',
      foreignField: 'id',
      as: 'type2'
    }},
  {$lookup: {
    from: 'move',
      localField: 'moves.move_id',
      foreignField: 'id',
      as: 'moves'
    }},
  {$match:
        {$or: [
            {"type1.name": 'ほのお'},
            {"type2.name": 'ほのお'}
          ]}},
  {$unwind: "$moves"},
  {$lookup: {
      from: 'type',
      localField: 'moves.type',
      foreignField: 'id',
      as: 'moves.type'
    }},
  {$match: {
      "moves.type.name": "かくとう",
      "moves.power": {$gte : 100}
    }},
  {$project: {
    "_id": 1,
    "no" : 1,
      "name": 1,
      "type1.name": 1,
      "type2.name": 1,
      "moves.name": 1,
      "moves.type.name": 1,
      "moves.power": 1
    }},
  {
    $group: {
      "_id": "$_id",
      "no": {
        $first: "$no"
      },
      "name": {
        $first: "$name"
      },
      "type1": {
        $first: "$type1.name"
      },
      "type2": {
        $first: "$type2.name"
      },
      "moves": {
        $push: "$moves"
```

```
        }
      }
    }
  ])
  .toArray((err, docs) => {

    /** アサーションテスト */
    assert.equal(err, null);

    /** 検索結果を整形してコンソールに出力 */
    console.log(JSON.stringify(docs, null, 2));
    ...
```

・結果


```
[
  {
    "_id": "5d2ac4f275821727d8d8c000",
    "no": "727",
    "name": "ガオガエン",
    "type1": [
      "ほのお"
    ],
    "type2": [
      "あく"
    ],
    "moves": [
      {
        "name": "きあいだま",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 120
      },
      {
        "name": "きあいパンチ",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 150
```

```
      },
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    "_id": "5d2ac4f275821727d8d8bf18",
    "no": "499",
    "name": "チャオブー",
    "type1": [
      "ほのお"
    ],
    "type2": [
      "かくとう"
    ],
    "moves": [
      {
        "name": "きあいだま",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 120
      },
      {
        "name": "きあいパンチ",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 150
      },
      {
        "name": "ばかぢから",
        "type": [
          {
            "name": "かくとう"
          }
        ],
        "power": 120
      }
    ]
  }
```

```
]
```

考察

1) Sample Query に関する考察

まず Sample Query で試行した検索条件を導くクエリについて考察する. 1 つ目の条件 「ほのおタイプのポケモンのみ表示」では MySQL, MongoDB 共にクエリは短く簡潔である. しかし条件をさらに絞ったそれ以降のクエリでは MySQL の where 節が長くなっていることが分かる. MySQL は複雑な検索条件のとき複数のテーブルに跨った検索が必要になることがある. 1 つ目の条件では 2 つのテーブル (pokemons, types) のみ用いているが, 2 つ目の条件では 4 つのテーブル (pokemons, types, pokemon_move, move) のテーブルを用いた上にポケモン自体のタイプと技のタイプをそれぞれ検索するのに from 文で types テーブルを 2 回指定する必要がある. これらのテーブルを where 節で検索するため 1 つ目の条件と比べクエリが長く, また可読性も低くなっている. 3 つ目の条件では使用するテーブル数は 2 つ目の条件と同じであるため, 2 つ目と 3 つ目のクエリの差は先ほどよりも大きくない. 一方の Mongo DB は MySQL よりもクエリは短く, また検索文も可読性に優れている. 木構造を root から辿るようにクエリを書くことができる. これは 2 つのデータベースの構造によるものである.

MySQL はデータを複数のテーブルで管理し, 各テーブルは作成時にデータの型を定義する. 一方のドキュメント指向の Mongo DB は図2. 図3. で示したように 1 つの要素の中に入れ子構造のように要素を格納している. そのため Mongo DB は MySQL よりもデータ構造は柔軟であると言える. また Mongo DB は JSON 形式のデータを格納して扱うことができるため可読性が高い一方, MySQL と比べデータ自体が冗長である.

2) データ管理に関する考察

続いてデータを実用的に管理していく際の有用性について比較し考察する. 実際にデータベースを元にゲームやゲームの攻略情報サイトを管理すると仮定した時, 既に存在するデータベースに対する操作はデータの追加と変更の 2 つが考えられる. ポケモンは新作ゲームが発表されるたびに新たなポケモンや技が追加される. 一方で既に存在していたポケモンが消えていなくなることはないのでデータの削除は考慮する必要はない. また技の威力などのデータが変

更された例がある. ポケモンGO のようなスマートフォン向けゲームでは技の威力やポケモン自体の個体値 (攻撃力や防御力など) の変更は特に多い. そのため実用性を評価するにはこの 2 つの操作について考察する必要がある.

まず, あるポケモンが追加 1 体追加された場合の操作を考える. この場合 MySQL では pokemons, pokemon_move に対しレコードを 1 件追加する必要がある. types, moves などには追加や変更の必要はない. 一方 Mongo DB で同様の追加操作を行う場合, 図3. に示した様な記述を追加する必要がある. 既に存在する技に関する情報も, その威力やタイプなどを書く必要がある. この操作は MySQL での追加に比べ, 非常に手間がかかると同時にタイプミスなどを誘発しやすいと考えられる.

次に, ある技について, その威力が変更になる場合を考える. この場合 MySQLでは moves テーブルにおいて当該の技の威力を書き換えるのみである. 一方で Mongo DB で同様の変更操作を行う場合, その技を覚える全てのポケモンについて, その中の当該する技の情報を書き換える必要がある. この操作は MySQL での操作に比べ圧倒的に手間がかかる.

以上より, データ管理していく中で必要となると考えられるデータの追加, 変更の様な操作は MySQL の方が簡潔で優れていることが示された.

3) タイプ相性関連計算を実現する方法に関する考察

ポケモンは捕まえたポケモンを育成し, 他のプレイヤーとお互いのポケモン同士を戦わせるバトルシステムがある. バトルにおいてポケモン同士のタイプ相性とそれに伴うポケモンや技の選択は戦略の要と言える. この節ではポケモンのタイプ相性を考慮した検索を実現するための方法を考察する.

まず MySQL では comps テーブルを用いる. comps テーブルは 2 タイプ間の相性をダメージ倍率で管理している. 18 種類のタイプの相関表であるため 324 (18 × 18) レコードとなる. rate の値が 2.0 のとき, m_type (攻撃する側のタイプ) は p_type (攻撃される側のタイプ) に対し "こうかはばつぐん" 即ち有利な相性であると言える. 相性はポケモンのタイプにも技のタイプにも共通するため, comps テーブルは, pokemons テーブルの type にも, moves テーブルの type にも検索をすることができる. そのため相手のポケモンに対し有利なポケモンの選択も有利な技の選択も可能となる.

一方で今回実装した Mongo DB の構成では複数テーブルを跨いでの検索が困難なため, 図4. に示すスキーマの中にタイプ相性に関するノードを追加する必要がある. 有利なポケモンの選択をするためには, ポケモン (pokemon) の type1, type2 の下にタイプ相性に関する配列を, 有利な技の選択をするためには, 技 (move) の type の下にタイプ相性に関する配列をそれぞれ格納する必要がある.

以上の様に, MySQL はデータ構造が厳密である一方で, 複雑なクエリの発行も可能であり, MongoDB はデータ構造が柔軟であるが検索については柔軟

性に欠ける. データ構造の柔軟性を活かし, よりデータ構造を冗長にすることで実現する.

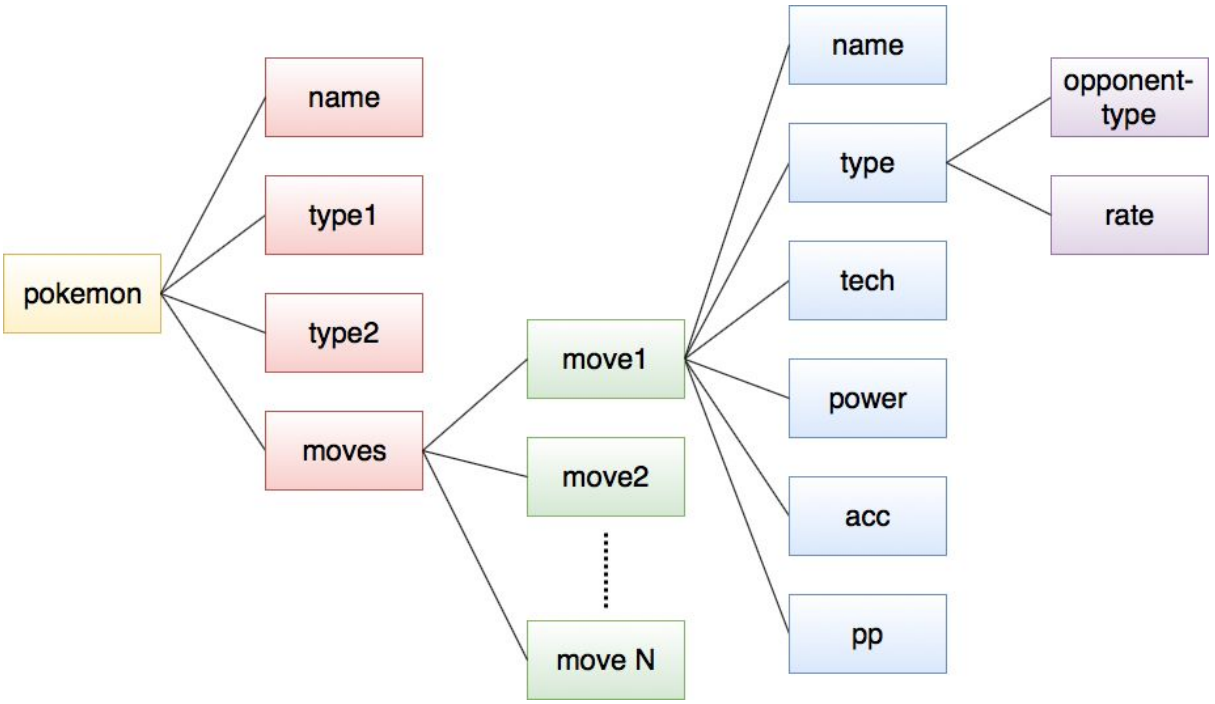

図4. type 以下にタイプ相性を追加したスキーマ

ここで MongoDB でタイプ相性の計算をするために複数の collection でデータを管理し join する方法を試みた.

MongoDB に挿入した collection とその形式について以下に示す.

- pokemon ("_id", "no", "name", "type1", "type2", "moves": [{"move_id": }, ... {"move_id": }])
  ポケモンの基本情報を管理
  - **no**: ポケモン図鑑の番号
  - **name**: ポケモンの名前
  - **type1**, **type2**: ポケモンのタイプID（独自に作成）. **type** collectionの id と対応している. ポケモンによってはタイプは1つの場合と2つの場合があるため, type2がない場合もある.
  - **moves**: そのポケモンが覚える技のID（独自に作成）を"**move_id**"の配列として保持.

- type ("_id", "id", "name")
  - **id**: ポケモンのタイプに付与した独自のID
  - **name**: タイプの名前

- move ("_id", "name", "type", "tech", "power", "acc", "pp", "pokemons": [{"pokemon_no": }, ... {"pokemon_no": }])

    - **name**: 技の名前
    - **type**: 技のタイプID
    - **tech**: 技の種類を示し，「物理」「特殊」のどちらか
    - **power**: 技の威力（攻撃技でない場合，また自身や相手のポケモンの重さ・なつき度などに応じて威力が変わる場合は "null"）
    - **acc**: 技の命中率
    - **pp**: 技を使用できる上限回数
    - **pokemons**: その技を習得可能なポケモンの図鑑Noを "**pokemon_no**" の配列として保持

- comp ("_id", "m_type", "p_type", "rate")
    - **m_type**: 攻撃する技のタイプID
    - **p_type**: 攻撃を受ける側のタイプID

複数の collection で管理することで構成を複雑化することなくタイプ相性が関係する検索が可能になる. またこの方法を用いれば, 前節で述べたデータの追加や変更に関しても, 該当する collection のみを MySQL と同様に追加, 変更するのみで解決することができる.

4)計算時間に関する考察

Sample Query で取り上げた 3 つのクエリについて, それぞれ実行時間を計測し比較した. なお, MongoDB はもとの単一 collection による構成に加え, 複数 collection によるものでも計測した. それぞれ 10 回ずつクエリを実行し, 実行時間の平均を計算した. 結果を表4. に示す.

表4. Sample Query のアプローチ別実行時間

	MySQL	MongoDB (単一 collection)	MongoDB (複数 collection)
query 1	0.009 s	0.114 s	0.474 s
query 2	0.018 s	0.111 s	1.005 s
query 3	0.01 s	0.11 s	0.584 s

3 つ全てのクエリで MySQL が最も短い計算時間を記録した. MongoDB はデータの構造上, 45 万行余りのデータを全探索の様にしてクエリを実行するため, 自ずとそこで時間がかかってしまうと考えられる.

MongoDB の 2 つのアプローチを比較すると, 全てのクエリで複数 collection を join するアプローチの方が計算時間がかかった. 特に, join する collection が 2 つの query 1 より, 3 つの collection を join する query2, query3

の方が計算時間が必要であった. 一方のアプローチでは複数 collection を用いたアプローチよりも query 同士の差が小さい. このことから MongoDB では collection の join が計算時間に大きな影響を与えていることがわかる. 単一 collection の方は join は無く, 木構造を探索する様に検索するためいずれのクエリでも同様に短い計算時間を記録していると考えれる.

また, MySQL でも 複数 collection による Mongo でも, 必要な table 数, collection 数は等しいが, 全てのアプローチで query 2 よりも query 3 の方が計算時間が短い. これは条件を満たすデータの数が query 2 は 48 件, query 3 は 30 件と query 2 の方が多く, その出力に時間を要していると考えられる. この特徴が影響して, 単一 collection の MongoDB でも, 件数が最も多い query 1 が最も計算時間がかかっている.

作業分担

- 岡　　: MongoDB データベースの作成, 考察
- 田部　　: MySQL データベースの作成, 考察
- 永山　　: スライド作成, 発表, 考察
- 和久井　　: スライド作成, 発表, 考察

参考文献

[1] Gamewith (https://gamewith.jp)