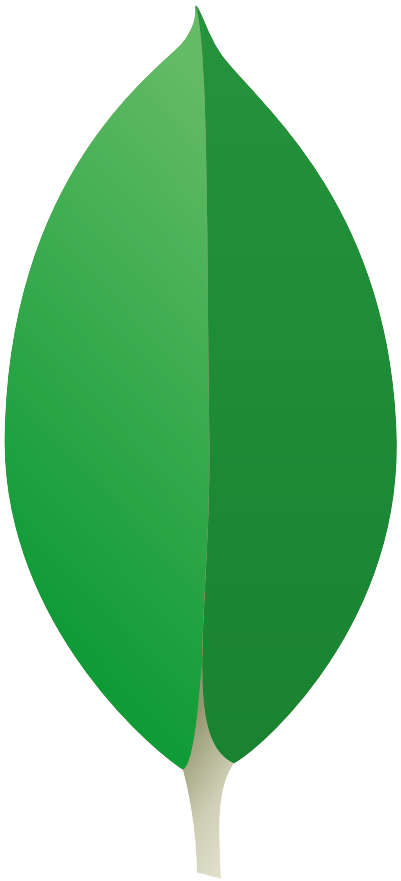
第 2.0.5 版

MongoDBの薄い本

The Little MongoDB Book

Karl Seguin 著 / 濱野　司 訳



目次

この本について

ライセンス

MongoDB の薄い本は Attribution-NonCommercial 3.0 Unported に基づいてライセンスされています。**あなたはこの本を読む為にお金を支払う必要はありません。**

この本を複製、改変、展示することは基本的に自由です。しかし、この本は常に私 (カール・セガン) に帰属するように求めます。そして私はこれを商用目的で使用する事はありません。以下にライセンスの全文があります:

http://creativecommons.org/licenses/by-nc/3.0/legalcode

著者について

カール・セガンは幅広い経験と技術を持った開発者です。彼は.Net のエキスパートであると同時に Ruby の開発者です。彼は OSS プロジェクトのセミアクティブな貢献者であり、テクニカルライターや時々講演を行っています。MongoDB に関して、彼は C#の MongoDB ライブラリ NoRM の主要な貢献者であり、インタラクティブ・チュートリアル mongly や Mongo Web Admin を書きました。彼のカジュアルなゲーム開発者の為のサービス、mogade.com は MongoDB で稼動しています。

カールは過去に Redis の薄い本も書いています。

彼のブログは http://openmymind.net 、つぶやきは@karlseguin で見つかります。

謝辞

Perry Neal が私に彼の目と意見と情熱を貸してくれたことに感謝します。ありがとう。

訳者より

内容の誤りや誤訳などありましたら@hamano まで連絡下さい。翻訳を手伝ってくれた@tamura__246 さんと誤字、誤訳を指摘下さった matsubo さん、honda0510 さん、@ponta_さん、ttaka さんに感謝します。

翻訳版のソース: http://github.com/hamano/the-little-mongodb-book

原書のソース: http://github.com/karlseguin/the-little-mongodb-book

序章

> この章が短いことは私の誤りではありません、MongoDB を学ぶ事はとても簡単です。

技術は激しい速度で進歩しているとよく言われます。それは新しい技術と技術手法が公開され続けているという点で真実ですが、私の見解ではプログラマによって利用される基礎的な技術の進歩は非常に遅いと考えています。何年もかけて習得した技術には少なからず基礎技術と関連があります。注目すべき点は確立した技術が置きかえられる速度です。気がつくと、長い歴史を持つ技術がまるで一晩で開発者の関心の移り変わりに脅かされているようです。

既に確立されているリレーショナルデータベースに対して発展してきた NoSQL はこの様な急転換の典型的な事例です。いつの日か、RDBMS で運用される Web は少なくなり、NoSQL が実用に足るソリューションとして確立されるかもしれません。

たとえこれらの技術が一晩で移り変わったとしても、実際的な実務でこれらが受け入れられるには何年もかかります。初期の段階では比較的小規模な会社や開発者の情熱によって突き動かされます。新しい技術は彼らのような人々の挑戦によってゆっくりと普及しソリューションや教育環境が洗練されていきます。念の為言っておくと、NoSQL は昔ながらのストレージソリューションを置き換える手段ではないという事について大部分は事実です。しかしある特定の分野では従来のものに優る価値を提供します。

何よりもまず初めに、NoSQL が何を意味しているのかを説明すべきでしょう。それは人によって異なる意味を持つ広義の用語です。私は個人的にそれをデータストレージの役割を果たすシステムという意味で使用しています。言い換えれば、(私にとっての)NoSQL はあなたが執着する単一のシステムに必ずしも責任を持つようなものではありません。歴史的に見て、リレーショナルデータベースベンダーはどんな規模にも適応する万能なソリューションとしてソ

フトウェアを位置づけてきました。NoSQL スタックを MySQL といったリレーショナルデータベースの様に利用する事も出来るでしょうが、NoSQL は与えられた仕事を達成する為の最適なツールとしてより小さい単位の役割に向かう傾向があります。そして、Redis もまた参照性能というシステムの特定部位に執着し、同じように Hadoop もデータ処理に集中的です。簡単に言うと、NoSQL はオープンであり、既存のものの代替や付加的なパターンを意識し、データを管理する為のツールです。

あなたは MongoDB が多くの状況に適応する事に驚くかもしれません。Mongo はドキュメント指向データベースとしてより NoSQL ソリューションに汎用化されています。それはリレーショナルデータベースの代替の様に見られるでしょうが、リレーショナルデータベースと比較すると、もっと専門化した NoSQL ソリューションにおいて恩恵を得ることが出来ます。MongoDB には利点と欠点がありますのでそれには後ほどこの本の中で触れます。

もう気がついているかもしれませんが、私たちは MongoDB という用語と同じ意味で Mongo と呼ぶことがあります。

はじめよう

この本の大部分は MongoDB の機能面に注目します。その為に私たちは MongoDB シェルを利用します。MongoDB ドライバを利用するようになるまで、MongoDB シェルは学習に役立つだけでなく、便利な管理ツールとなるでしょう。

MongoDB についてまず最初に知るべきことを取り上げます: それはドライバです。MongoDB は各種プログラミング言語向けに数多くの公式ドライバが用意されています。これらのドライバは恐らくあなたがすでに慣れ親しんでいる各種データベースのドライバと似たようなものだと考えて良いでしょう。これらのドライバに加えて、開発コミュニティでは更にプログラミング言語/フレームワーク用のライブラリが開発されています。例えば、NoRM は LINQ を実装した C#のライブラリで、MongoMapper は ActiveRecord と親和性の高い Ruby ライブラリです。プログラムから直接 MongoDB のコアドライバを利用するか、他の高級なライブラリを選択するかはあなた次第です。何故、公式ドライバとコミュニティライブラリの両方が存在するのかについて MongoDB に不慣れな多くの人に混乱があるようなので説明しておきます。前者は MongoDB の中核的な通信と接続性に、後者はプログラミング言語や特定のフレームワークの実装により特化しています。

これを読みながらあなたは私の実習を実際に反復し、自分自身で疑問を探っていく事を推奨します。MongoDB の準備と実行は簡単です、今から数分の時間をかけてセットアップしてみましょう。

1. 公式ダウンロードページへ進み、一番上の行から OS を選択してバイナリを手に入れましょう (安定バージョンを推奨)。開発目的であれば 32 ビット 64 ビットのどちらを選んでも構いません。
2. アーカイブを適当な場所に展開し、`bin`サブフォルダへ移動します。まだ何も実行しないこと、`mongod`がサーバープロセスであり、`mongo`がクライアントシェルであること

は知っておいて下さい。その 2 つはこれから私たちが最も時間を費やす実行ファイルです。

3. binサブフォルダの中に mongodb.configという名前で新しいテキストファイルを作成します
4. mongodb.configに以下の 1 行を追記します:

```
dbpath=データーベースファイルを格納する場所
```

例えば、Windows では bpath=c:\mongodb\dataを指定し、Linux では dbpath=/var/lib/mongodb/dataと指定します。

5. 指定した dbpathを作成する
6. mongod を--config /path/to/your/mongodb.configパラメーターを付けて起動します。

Windows ユーザーの為の例を示すと、もしダウンロードファイルを c:\mongodb\に展開したのなら c:\mongodb\bin\mongodb.configに dbpath=c:\mongodb\data\を指定すると、c:\mongodb\data\が作成されます。次に、コマンドプロンプトから c:\mongodb\bin\mongod --config c:\mongodb\bin\mongodb.configを実行して mongodを起動します。

無駄を少なくする為に、ご自由に binフォルダをパスに追加して下さい。MacOSX と Linux ユーザーはほとんど同じやり方に従うことが出来ます。あなたがすべきことはパスを変更することくらいです。

うまくいけば今あなたは MongoDB を実行しているでしょう。もしエラーが起こったのならメッセージを注意深く読んで下さい。サーバーは何がおかしいのかを丁寧に説明してくれます。

あなたは mongo(**d** は付かない) を起動し、実行中のサーバーのシェルに接続する事が出来ます。全てが動作しているか確認するために db.version()と入力してみて下さい、うまくいっていればインストールされているバージョンを確認することが出来るでしょう。

第 1 章

基礎

MongoDB の動作の基本的な仕組みを知ることからはじめましょう。当然これは MongoDB の核心を理解することです。しかしこれは MongoDB についての高レベルな質問の答えを見つけ出す為にも役立ちます。

最初に、私たちは 6 つの概念を理解する必要があります。

1. MongoDB はあなたが既に慣れ親しんでいる**データベース**と同じ概念を持っています (あるいは Oracle でいうところのスキーマ)。MongoDB インスタンスの中には 0 個以上のデータベースを持つことが出来、それぞれは高レベルコンテナの様に作用します。
2. データベースは 0 個以上の**コレクション**を持つことが出来ます。コレクションは従来の**テーブル**とほぼ共通しているので、この 2 つが同じものだと思っても支障は無いでしょう。
3. コレクションは 0 個以上の**ドキュメント**を作成できます。先ほどと同様にドキュメントを**行**と思って構いません。
4. ドキュメントは 1 つ以上の**フィールド**を作成できます。あなたは恐らくこれを**列**に似ていると推測できるでしょう。
5. MongoDB での**インデックス**機能は RDBMS のものとよく似ています。
6. **カーソル**はこれまでの 5 つの概念とは異なりますが、とても重要で見落とされがちですので詳しく説明する必要があると思います。カーソルについて理解すべき重要な事があります。カウントやスキップの様な操作は実際にデータを引き出すことはありません、MongoDB にデータを問い合わせた際に返却されるカーソルに対して行われます。

要点をまとめると、MongoDB は**データベース**を作成し、その中には**コレクション**を含みます。**コレクション**は**ドキュメント**を作成します。それぞれの**ドキュメント**は**フィールド**を作成

します。**コレクション**は**インデックス化**可能であり、これは参照やソートの性能を改善します。最後に、MongoDB からデータを取得する際、**カーソル**を経由して操作を行います。これは実際に実行する際に必要不可欠なものです。

なぜ新しい専門用語を利用するのでしょうか (コレクションとテーブル、ドキュメントと行、フィールドと列)。物事を複雑にするだけでしょうか?これらの概念がリレーショナルデータベースの機能と良く似ているという点ではその通りですが、これらは全く同じではありません。主要な違いは、リレーショナルデータベースが**テーブル**のレベルに**列**を定義しているのに対し、ドキュメント指向データベースは**ドキュメント**のレベルに**フィールド**を定義している事です。**コレクション**の中の**ドキュメント**はそれ自身の独自の**フィールド**を持つことが出来ると言えます。という訳で、**コレクション**は**テーブル**に比べ、より使い易いコンテナとなり、さらに**ドキュメント**は**行**に比べてより多くの情報を持つようになります。

これを理解することは重要ですが、もしこれをまだ完全に理解出来ていなくても心配することはありません。この本当の意味を確かめる為にはまだもう少し説明が必要でしょう。突き詰めていくとコレクションの中に何が入るかが厳密ではない事が要点です (スキーマレスの事)。フィールドは特定のドキュメントに対して追従します。あとの章で利点と欠点に気がつくでしょう。

さあハンズオンをはじめましょう。まだ動かしていないのなら、どうぞ mongodサーバーと mongo シェルを起動して下さい。シェルは JavaScript を実行できます。helpや exitの様な、幾つかのグローバルコマンドを実行することが出来ます。例えば、db.help()や db.stats()といったコマンドは、現在のデータベース dbオブジェクトに対して実行します。db.unicorns.help()や db.unicorns.count()といったコマンドは、db.COLLECTION_NAMEオブジェクトの様に、指定したそれぞれのコレクションに対して実行します。

どうぞ、db.help()と入力してみて下さい。dbオブジェクトに対して実行可能なコマンド一覧を得ることが出来るでしょう。

余談ですが、あなたが丸カッコ ()を含めずにメソッドを実行した場合、メソッドの実行ではなくメソッドの本体が表示されます。なぜならこれは JavaScript シェルだからです。あなたが最初にこれをやった時、function (...){という応答が返ってきても驚かないように、この事に触れておきました。たとえば、丸カッコ無しで db.helpと入力すると helpメソッドの内部実装を見ることが出来ます。

私たちは最初にグローバルな useメソッドを利用してデータベースを切り替えます。どうぞ use learnと入力してみて下さい。そのデータベースが実際に存在していなくても構いません。最初のコレクションを learnに作成しましょう。今あなたはデータベースの中にいて、db.getCollectionNames()という様なデータベースコマンドを発行できます。これを実行すると、恐らく空の配列 ([ ]) が返ってくるでしょう。コレクションはスキーマレスですので、それらを明確に作成する必要はありません。私たちは、単純にドキュメントをコレクションに作

成する事が出来ます。それでは、insertコマンドを使ってコレクションにドキュメント挿入してみましょう。

```
db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})
```

上記のコマンドは一つの引数を受け取り unicornsコレクションに対して insertを行います。MongoDB の内部ではシリアライズされた JSON フォーマットを利用します。今、db.getCollectionNames()を実行すると、実際には 2 つのコレクション unicornsと system.indexesを確認できます。system.indexesコレクションはデータベースのインデックス情報が格納され、データベース毎にひとつ作成されます。

これで、unicornsに対し findコマンドを使用してドキュメントのリストを取得出来るようになりました。

```
db.unicorns.find()
```

あなたが指定したデータには_idフィールドが追加されていることに注目して下さい。全てのドキュメントはユニークな_idフィールドを持たなければなりません。あなたは、MongoDB に生成させるかこの ObjectIDを自分自身で生成する事になります。先程 system.indexesコレクションが作成された理由は、デフォルトで_idフィールドはインデックス化されているからであると説明できます。あなたは以下のようにして system.indexesを参照できます:

```
db.system.indexes.find()
```

あなたはインデックスを含むフィールドに対して作成されたデータベースやコレクションのインデックス名を確認することが出来ます。

スキーマレスコレクションの話に戻りましょう。unicornsコレクションに以下の様な完全に異なるドキュメントを入れてみます:

```
db.unicorns.insert({name: 'Leto', gender: 'm', home: 'Arrakeen', worm: false})
```

再度 findを利用してドキュメントを表示してみて下さい。MongoDB の興味深い振る舞いについて前に少しだけ話しました、何故従来の技術がうまく適応しなかったのかが解り始めて来たのではないでしょうか。

1.1 セレクターの習得

次の話題に進む前に、先程説明した 6 つの概念に加え、MongoDB の実用面でしっかりと理解すべきことがあります。それはクエリーセレクターです。MongoDB のクエリーセレクターは SQL 構文の where節によく似ています。そういうわけで、これはドキュメントを見つけ出

したり、数えたり、更新したり、削除したりする際に使用します。セレクターは JSON オブジェクトです。最も単純な{}は全てのドキュメントにマッチします (nullも同じです)。もし女性ドキュメントを見つけたい場合、{gender:'f'}と指定します。

セレクターについて掘り下げていく前に、演習の為の幾つかのデータをセットアップしましょう。まず最初に、これまでに unicornsコレクションに入れたドキュメントを db.unicorns.remove()を実行して削除します (セレクターを指定していないので、全てのドキュメントが削除されます)。さて、以下を実行して演習に必要なデータを挿入しましょう (コピペ推奨):

```
db.unicorns.insert({name: 'Horny', dob: new Date(1992,2,13,7,47),
                    loves: ['carrot','papaya'], weight: 600,
                    gender: 'm', vampires: 63});
db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0),
                    loves: ['carrot', 'grape'], weight: 450,
                    gender: 'f', vampires: 43});
db.unicorns.insert({name: 'Unicrom', dob: new Date(1973, 1, 9, 22, 10),
                    loves: ['energon', 'redbull'], weight: 984,
                    gender: 'm', vampires: 182});
db.unicorns.insert({name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44),
                    loves: ['apple'], weight: 575,
                    gender: 'm', vampires: 99});
db.unicorns.insert({name: 'Solnara', dob: new Date(1985, 6, 4, 2, 1),
                    loves:['apple', 'carrot', 'chocolate'], weight:550,
                    gender:'f', vampires:80});
db.unicorns.insert({name:'Ayna', dob: new Date(1998, 2, 7, 8, 30),
                    loves: ['strawberry', 'lemon'], weight: 733,
                    gender: 'f', vampires: 40});
db.unicorns.insert({name:'Kenny', dob: new Date(1997, 6, 1, 10, 42),
                    loves: ['grape', 'lemon'], weight: 690,
                    gender: 'm', vampires: 39});
db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57),
                    loves: ['apple', 'sugar'], weight: 421,
                    gender: 'm', vampires: 2});
db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53),
                    loves: ['apple', 'watermelon'], weight: 601,
                    gender: 'f', vampires: 33});
db.unicorns.insert({name: 'Pilot', dob: new Date(1997, 2, 1, 5, 3),
                    loves: ['apple', 'watermelon'], weight: 650,
                    gender: 'm', vampires: 54});
db.unicorns.insert({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15),
                    loves: ['grape', 'carrot'], weight: 540,
                    gender: 'f'});
db.unicorns.insert({name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18),
                    loves: ['grape', 'watermelon'], weight: 704,
                    gender: 'm', vampires: 165});
```

さて、データが入りましたのでセレクターを習得しましょう。{field: value}は fieldというフィールドが valueと等しいドキュメントを検索します。{field1: value1, field2: value2 }は and式で検索します。$lt、$lte、 $gt、 $gte、$neはそれぞれ、未満、以下、より大きい、以上、非等価、を意味する特別な演算子です。例えば、性別が男で体重が 700 ポンドより大きいユニコーンを探すにはこのようにします:

```
db.unicorns.find({gender: 'm', weight: {$gt: 700}})
// このセレクターは以下と同等です。
db.unicorns.find({gender: {$ne: 'f'}, weight: {$gte: 701}})
```

$exists演算子はフィールドの存在や欠如のマッチに利用します。例えば:

```
db.unicorns.find({vampires: {$exists: false}})
```

ひとつのドキュメントが返ってくるはずです。AND ではなく OR を利用したい場合、$or演算子を利用して、OR をとりたい式を配列で指定します。

```
db.unicorns.find({gender: 'f', $or: [{loves: 'apple'},
                                     {loves: 'orange'},
                                     {weight: {$lt: 500}}]})
```

上記は全ての女性のユニコーンの中から、りんごかオレンジが好き、もしくは体重が 500 ポンド未満の条件で検索します。

最後に示した例にはとても素敵なものがあります。すでに知っていると思いますが lovesフィールドは配列です。MongoDB はファーストクラスオブジェクトとしての配列をサポートしています。これはとんでもなく便利な機能です。一度これを使ってしまうと、これ無しでは生活できなくなる恐れがあります。何よりも興味深いのは配列の値に基づいて簡単に選択できることです。{loves: 'watermelon'} は lovesの値に watermelonを持つドキュメントを返します。

これまでに見てきた他に、演算子はまだまだあります。最も柔軟な$whereは指定した JavaScript をサーバー上で実行します。MongoDB の Web サイトの Advanced Queries の項に全てが記載されています。これまでに紹介してきたものはあなたが使い始めるのに必要な基本です。もっと使いこなせるようになるには多くの時間がかかるでしょう。

これまでセレクターは findコマンドで利用できることを見てきました。これらは以前ちょっとだけ利用した removeコマンドやまだ使っていない countコマンド、後で出てくる updateコマンドでも利用できます。

ObjectIdは MongoDB が生成した_idフィールドを選択するために利用します:

```
db.unicorns.find({_id: ObjectId("TheObjectId")})
```

1.2 章のまとめ

私たちはまだ updateコマンドや、findで出来る幾つかの手の込んだ操作については学んでいませんが、insertコマンドと removeコマンドについて簡単に学びました (これまで見てきた以上のものはそれほど多くありません)。私たちは findの紹介と MongoDB の selectorsというものも見てきました。私たちは順調なスタートと、来たるべき時の為の盤石な基礎を築く事が出来ました。信じようと信じまいと、実際にあなたは MongoDB についての殆どの事を知っています (素早く学習し、簡単に使えるようになるという意味でです)。次に移る前に、演習のデータをローカルコピーすることを強く推奨します。恐らく、新しいコレクションで違うドキュメントを挿入したり、セレクターと似たようなものを使います。findや countや removeも使います。いろいろ試しているうちに、なにかマズイ事が起こった場合に最初の状態に戻れるようにしておいた方が良いでしょう。

第 2 章

更新

1 章では CRUD(作成、読み込み、更新、削除) の 4 つのうちの 3 つの操作を紹介しました。この章では、省略していた updateに専念します。その理由は、updateには幾つかの意外な振る舞いがあるからです。

2.1 置換 と $ set

最も単純な形式では、updateは 2 つの引数をとります: セレクター (where 条件) とアップデートするフィールドです。もし Roooooodles の体重を少し増やしたい場合、これを実行します:

```
db.unicorns.update({name: 'Roooooodles'}, {weight: 590})
```

(もし前回の演習で作成した unicornsコレクションを残していない場合、1 章に戻って、全てのドキュメントを removeし、挿入し直して下さい。)

実際のコードでは_idを指定してレコードを更新するでしょうが、MongoDB が生成する_idはまだ知らないので、ここでは nameを指定します。アップデートされたレコードを確認する場合、以下のようにします:

```
db.unicorns.find({name: 'Roooooodles'})
```

あなたは最初の updateに驚いてしまうでしょう。2 番目に指定したパラメータは元のデータを**置き換える**為に使われてしまい、元のドキュメントは見つかりません。言い換えると、updateはドキュメントを nameで検索し、ドキュメント全体を 2 番目のパラメータで置き換えます。これは SQL の update文と異なる動作です。忠実に動的な更新を行う目的の幾つかの状

況では、これは理想的な動作です。しかし、ひとつか複数のフィールドの値を変更したい場合は、MongoDB の$set修飾子を利用するのが最適でしょう:

```
db.unicorns.update({weight: 590}, {$set: {name: 'Roooooodles',
                                    dob: new Date(1979, 7, 18, 18, 44),
                                    loves: ['apple'],
                                    gender: 'm',
                                    vampires: 99}})
```

これで、失われたフィールドをリセットします。weightを指定しなければ上書きできません。以下を実行します:

```
db.unicorns.find({name: 'Roooooodles'})
```

期待する結果が得られました。従って、最初に行いたかった体重を変更する正しい方法は以下の通りです:

```
db.unicorns.update({name: 'Roooooodles'}, {$set: {weight: 590}})
```

2.2 更新修飾子

$setに加えて、その他の修飾子を利用するともっと粋なことが出来ます。これらの更新修飾子は、フィールドに対して作用します。なのでドキュメント全体が消えてしまうことはありません。例えば、$inc修飾子はフィールドの値を増やしたり、負の値で減らす事が出来ます。もし Pilot が vampireを倒した数が間違っていて 2 つ多かった場合、以下のようにして間違いを修正します:

```
db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})
```

もし Aurora が突然甘党になったら、$push修飾子を使って、lovesフィールドに値を追加することが出来ます:

```
db.unicorns.update({name: 'Aurora'}, {$push: {loves: 'sugar'}})
```

その他の有効な更新修飾子は MongoDB Web サイトの Updating に情報があります。

2.3 Upsert

updateの使い方にはもっと驚く愉快なものがあります。その一つは upsertを完全にサポートしている事です。upsertはドキュメントが見つかった場合に更新を行い、無ければ挿入を行います。upsertは見ればすぐ解るし、よくあるシチュエーションで重宝します。updateの 3 番

目の引数を'true' に設定する事で upsertを利用することが出来ます。

一般的な例は Web サイトのカウンターです。複数のページのカウンターをリアルタイムに動作させたい場合、ページのレコードが既に存在しているか確認し、更新を行うか挿入を行うか決めなければなりません。3 番目の引数を省略 (もしくは false に設定) して実行すると、以下のようにうまくいきません:

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}});
db.hits.find();
```

しかし、upsertを有効にすると違った結果になります。

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);
db.hits.find();
```

pageというフィールドの値が unicornsのドキュメントが存在していなければ、新しいドキュメントが挿入されます。2 回目を実行すると既存のドキュメントが更新され、hitsは 2 に増えます。

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);
db.hits.find();
```

2.4 複数同時更新

最後の驚きは、updateはデフォルトで一つのドキュメントに対してのみ更新を行う事です。これまでの様に、まず例を見ていきましょう。以下のように実行します:

```
db.unicorns.update({}, {$set: {vaccinated: true }});
db.unicorns.find({vaccinated: true});
```

あなたは全てのかわいいユニコーン達が予防接種を受けた (vaccinated) と期待するでしょう。あなたが望む様な振る舞いを行うには、4 番目のパラメータを true に設定する必要があります:

```
db.unicorns.update({}, {$set: {vaccinated: true }}, false, true);
db.unicorns.find({vaccinated: true});
```

2.5 章のまとめ

この章でコレクションに対して行う基本的な CRUD 操作を紹介し終えました。私たちはupdateの詳細を確認し、3 つの興味深い振る舞いを観察しました。まず、MongoDB の updateは SQL の updateと異なり、ドキュメント全体を置き換えます。そんな訳で$set修飾子はとても

便利です。次に、updateは直感的な upsertをサポートしています。これは$inc修飾子と一緒に使うと非常に便利です。最後に、デフォルトで updateは最初にみつけたドキュメントしか更新しません。

私たちはシェルの視点から MongoDB を見てきた事を思い出して下さい。ドライバやライブラリの場合でも、デフォルトの振る舞いを切り替えて使用したり、異なる API に触れる事になります。

例えば、Ruby のドライバでは最後の 2 つのパラメータを一つのハッシュにまとめています:

```
{:upsert => false, :multi => false}
```

同様に、PHP のドライバも最後の 2 つのパラメータを配列にまとめています。

```
array('upsert' => false, 'multiple' => false)
```

第 3 章

検索の習得

1 章では、findコマンドについて簡単に説明しました。ここでは findやセレクターについての理解を深めていきます。findが**カーソル**を返却することについては既に述べましたので、もっと正確な意味を見ていきましょう。

3.1 フィールド選択

カーソルついて学ぶ前に、findに任意で設定出来る 2 番目のパラメータについて知る必要があります。このパラメーターは取得したいフィールドのリストです。例えば、以下の様に実行して、全てのユニコーンの名前を取得出来ます。

```
db.unicorns.find(null, {name: 1});
```

デフォルトで、_idフィールドは常に返却されます。明示的に{name:1, _id: 0}を指定する事でそれを除外する事が出来ます。

_idに関する余談ですが、包含条件と排他条件を混ぜることが出来るのかどうか、気になるかもしれません。あなたはフィールドを含めるか除くかのどちらかを選択することが出来ます。

3.2 順序

これまでに何度か、findが必要な時に遅延して実行されるカーソルを返却する事に言及しました。しかし、まだあなたはこれをシェルから直接'find' を実行して自分の目で観測していません。この振る舞いはシェルのみとなります。カーソルの本当の振る舞いは findに一つのメソッドを連結することで観測することが出来ます。昇順でソートを行いたい場合はフィールド

と 1 を指定し、降順で行いたい場合は-1 を指定します。例えば:

```
//重いユニコーンの順
db.unicorns.find().sort({weight: -1})

//ユニコーンの名前と vampires の多い順:
db.unicorns.find().sort({name: 1, vampires: -1})
```

リレーショナルデータベースと同様に、MongoDB もソートの為にインデックスを利用出来ます。インデックスの詳細は後で詳しく見ていきますが、MongoDB にはインデックスを使用しない場合にソートのサイズ制限があることを知っておく必要があります。すなわち、もし巨大な結果に対してソートを行おうとするとエラーが返ってきます。実際の話、その他のデータベースにも、最適化されていないクエリーを拒否する機能があった方が良いと考えています。(私はこの動作を MongoDB の欠点とは考えていませんし、データベースの最適化が下手な人達にこの機能を使って欲しいと強く願っています。)

3.3 ページング

ページングの結果は cursor の limitメソッドや skipメソッドを利用して遂行できます。2 番目と 3 番目に重いユニコーンを得るにはこうやります:

```
db.unicorns.find().sort({weight: -1}).limit(2).skip(1)
```

limitと sortを組み合わせると、インデックス化されていないフィールドでソートする場合の問題を避けるのに役立ちます。

3.4 カウント

シェルでは collection に対して直接 countを呼び出す事が出来ます。例えば:

```
db.unicorns.count({vampires: {$gt: 50}})
```

実際には countは cursorのメソッドであり、シェルは単純なショートカットを提供しているだけです。この様なショートカットを提供しないドライバでは以下の様に実行する必要があります (これはシェルでも動きます):

```
db.unicorns.find({vampires: {$gt: 50}}).count()
```

3.5 章のまとめ

findや cursorの率直な使われ方を見てきました。これらの他に、あとの章で触れるコマンドや特別な状況で使われるコマンドが存在しますが、あなたは既に MongoDB の基礎を理解し、mongo シェルを安心して触れるようになったでしょう。

第 4 章

データモデリング

さて、MongoDB のもっと抽象的な話題に移っていきましょう。幾つかの新しい用語や、些細な機能の新しい文法について説明していきます。新しいパラダイムであるモデリングについての話題は簡単ではありません。モデリングに関する新しい技術について、大抵の人々はまだ何が役に立ち、役に立たないのかをよく知りません。まずは講話から始めますが、最終的には実際のコードで学び、実践を行っていきます。

モデリングに関して言えば、ドキュメント指向データベースである多くの NoSQL ソリューションとリレーショナルデータベースを比較して大した違いは在りません。しかし違いが少ないからといってそれらが重要で無い訳ではありません。

4.1 Join はありません

まず最初に、最も根本的な違いである MongoDB に Join が存在しない事に対して安心する必要があるでしょう。MongoDB が何故 join の文法をサポートしていないのか、特別な理由を私は知りませんが、Join がスケーラブルでない事は一般的に知られています。すなわち、一度データの水平分割を行うと、最終的にクライアント (アプリケーションサーバー) 側で Join を行う事になります。理由はどうあれ、データはリレーショナルである事に変り在りませんが、MongoDB は Join をサポートしていません。

とにかく、Join 無しの世界で生活するためにはアプリケーションのコード内で Join を行わなくてはなりません。それには基本的に 2 度目の `find`クエリーを発行してデータを取得する必要があります。これから準備するデータはリレーショナルデータベースの外部キーと違いはありません。しばらく素敵な `unicorns`コレクションから視点を外して、`employees`コレクションに注目してみましょう。まず最初に、社員を作成します。(分かり易く説明する為に、

_idフィールドを明示的に指定しています)

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d730"), name: 'Leto'})
```

さて、Letoがマネージャーとなる様に設定した社員を何人か追加してみましょう:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d731"), name: 'Duncan',
                     manager: ObjectId("4d85c7039ab0fd70a117d730")});
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d732"), name: 'Moneo',
                     manager: ObjectId("4d85c7039ab0fd70a117d730")});
```

(上記に倣って、_idはユニークになる必要があります。ここで実際に指定した ObjectIdを、以降も同じ様に使用する事になります。)

言うまでもなく、Leto の社員を検索するには単純に以下を実行します:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

これは何の変哲もありません。最悪の場合、join の欠如はただ単に余分なクエリーが多くの時間を占めるかもしれません (恐らくインデックス化されているでしょうが)。

4.1.1 配列と埋め込みドキュメント

MongoDB が join を持たないからといって、切り札が無いという意味ではありません。MongoDB のドキュメントがファーストクラスオブジェクトとしての配列をサポートしている事を簡単に確認したのを思い出してください。これは、多対一、多対多の関係を表現する際にとても器用に役立つ事が分かります。簡単な例として、社員が複数のマネージャーを持つ場合、単純にこれらを配列で格納する事が出来ます:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d733"), name: 'Siona',
                     manager: [ObjectId("4d85c7039ab0fd70a117d730"),
                               ObjectId("4d85c7039ab0fd70a117d732")]})
```

特に興味深い事は、ドキュメントはスカラ値であっても構わないし、配列であっても構わないという点です。最初の findクエリーはどちらであっても動作します:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

これによって、多対多の join テーブルよりもっと便利に素早く配列の値を見つけることが出来ます。

配列に加えて、MongoDB は埋め込みドキュメントをサポートしています。次に進んで入れ子になったドキュメントを挿入してみてください:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d734"), name: 'Ghanima',
                     family: {mother: 'Chani',
                              father: 'Paul',
                              brother: ObjectId("4d85c7039ab0fd70a117d730")}})
```

驚くでしょうが、埋め込みドキュメントはクエリーにドット表記を使用できます:

```
db.employees.find({'family.mother': 'Chani'})
```

後ほど、埋め込みドキュメントがどの様な場所に適合し、どの様に使用するかを簡単に説明します。

4.1.2 DBRef

MongoDB は DBRefと言われる習慣を多くのドライバでサポートしています。ドライバがDBRefに遭遇すると、自動的に参照先のドキュメントを取得します。DBRefはコレクションやドキュメントの参照 ID を含みます。これは一般的に特定の目的に対して提供される機能です。例えばドキュメントが同じコレクションのドキュメントから参照され、異なるコレクションからも同じドキュメントを参照するような場合です。つまり、ドキュメント 1 が managersのドキュメントを指し示す DBRefである一方、ドキュメント 2 が employeesのドキュメントを指し示す事が出来ます。

4.1.3 非正規化

join を使う事の代わりのもうひとつの代替は、データを非正規化する事です。従来、非正規化はパフォーマンス特化の為やデータの速記 (ログの様な) の為に利用されてきました。ところが、NoSQL の人気が高まるにつれて、join を行わないことや非正規化は次第に一般的なモデリング手法として認められるようになってきました。これは全てのドキュメントであらゆる情報が重複するという意味ではありません。けれども、その方がかえって重複を恐れずに、データがどの情報に基づいていて、どのドキュメントに属しているかをよく考えてモデリングし、設計を行う傾向があります。

例えば、掲示板の WEB アプリケーションを作っているとします。伝統的な方法では、postsテーブルにある**ユーザー ID** によって**ユーザー**と**投稿**を紐付けます。この様なモデルでは usersテーブルを検索 (もしくは join) しなければ**投稿**を表示することが出来ません。埋め込みドキュメントがあればこういう事も出来るでしょう:

```
`user: {id: ObjectId('Something'), name: 'Leto'}`
```

もちろんこうした場合、ユーザーが名前を変更すると全てのドキュメントを更新しなければ

なりません (一度のクエリーで済みます)。

この種の取り組みを調整する事はそれほど簡単ではありません。多くの場合、この様な調整は非常識で効果が無いかもしれません。しかし、この取り組みへの実験を恐れないでください。状況によっては適切ではありませんが、その取り組みが効果的で正しい対処になることもあるでしょう。

4.1.4 どちらを選ぶ?

1 対多や多対多の関係のシナリオで ID を配列にする事は有用な戦略です。DBRefは実験的に利用することは出来ますがそれほど利用頻度は多くないと言っても間違いではないと思います。一般的な新しい開発者にとって、埋め込みドキュメントを利用するか、手動で参照を行うか悩んでしまう事がよくあります。

まず、個々のドキュメントのサイズは 16MByte までに制限されていることを知らなければなりません。ドキュメントのサイズに制限があるとわかった所で、気前よく替りにどの様にすれば良いかのアイディアを提供しましょう。現在の所、開発者が巨大なリレーションを行いたい場合、大抵は手動で参照しなければならない様に思われます。埋め込みドキュメントは頻繁に利用されますが、データの小片は親ドキュメントと同時に取得したい場合が殆どです。実例として、ユーザーのアカウント情報を格納する例を利用します。

```
db.users.insert({name: 'leto', email: 'leto@dune.gov',
                 account: {allowed_gholas: 5, spice_ration: 10}})
```

これを単に手っ取り早く書き込むための只のユーティリティだと過小評価してはいけません。直接ドキュメントを持つ事は、データモデルをより単純にし、多くの場合 Join の必要性を無くします。これは、埋め込みドキュメントのインデックスフィールドや、クエリーを考慮すると、特にあてはまります。

4.2 少ないコレクションと多いコレクション

コレクションがスキーマを強要しないのであれば、単一のコレクションに様々なドキュメントをごちゃ混ぜにしたシステムを構築することも可能です。私の見た所、多くの MongoDB のシステムはよくあるリレーショナルデータベースと同じように設計されて使われているようです。言い換えると、リレーショナルデータベースのテーブルは多くは MongoDB のコレクションで置き換えることが可能です。(多対多の Join は重要な例外です)。

埋め込みドキュメントについて考えてみると、もっと面白い話題があります。よくある例はブログシステムです。postsコレクションと commentsコレクションを持っているとすると、

postドキュメントにコメントの配列を埋め込むことも出来るはずです。多くの開発者はまだ、コレクションを分割することを好むようですが、16MByte の制限はひとまず考えないようにしてみてはどうでしょうか (**ハムレット**の全文は 200KByte 以下です、あなたのブログはこれより有名なのですか?)。それは単純明快でしょう。

16MByte の制限はそれほど難しいルールではありません。ぜひ今までと違った手法を試してみて、何が上手く行って何がうまく行かないのかを自分で確かめてみて下さい。

4.3 章のまとめ

この章の目標は MongoDB でデータをモデリングする為のガイドラインを示すことでした。ドキュメント指向システムのモデリングはリレーショナルデータベースの世界と異なりますが、それほど多くの違いはありません。あなたは少しの柔軟性と一つの制約を知りましたが、新しいシステムであれば上手く適合させることが出来るでしょう。誤った方向に進む唯一の方法は挑戦を行わない事です。

第 5 章

どんな時 MongoDB を利用するか

そろそろ、MongoDB を何処にどの様にして既存のシステムに適合させる為の感覚をつかむ必要があるでしょう。MongoDB には、その他多くの選択肢を簡単に凌駕する十分新しい競合ストレージ技術があります。

私にとって最も重要な教訓は MongoDB とは関係がありません。それはあなたがデータを扱う際に単一の解決手段に頼らなくてもよい様にすることです。たしかに、単一の解決手段を利用することは利点があります。多くのプロジェクトで単一の解決手段に限定することは場合によっては賢明な選択でしょう。異なるテクノロジを使用しなければならないと言うことではなく、異なるテクノロジを使用できるという発想です。あなただけが、新しいソリューションを導入することの利益がコストを上回るかどうかを知っています。

そんな訳で、これまで見てきた MongoDB の機能は一般的な解決手段として見なすことを期待しています。ドキュメント指向データベースがリレーショナルデータベースと共通する所が多い点については既に何度か言及してきました。そのために、MongoDB が単純にリレーショナルデータベースの代替になると言い切る事を慎重に扱って来ました。Lucene が全文検索インデックスによってリレーショナルデータベースを強化し、Redis が永続的な Key-Value ストアと見なすことが出来るように、MongoDB はデータの中央レポジトリとして見なすことが出来ます。

MongoDB はリレーショナルデータベースをそのまま置き換える様なものではなく、どちらかというと別の代替手段であると言っていることに注意して下さい。それはその他のツールと同様にツールなのです。MongoDB に向いている事もあれば、向いていない事もあります。それではもう少し詳しく分析してみましょう。

5.1 スキーマレス

ドキュメント指向データベースの利点としてよくもてはやされるのはスキーマレスである事です。これは従来のデータベーステーブルに比べてはるかに柔軟性をもたらします。私はスキーマレスは素晴らしい機能だと認めますが、それが主な理由でない事に多くの人は言及しません。

人々はスキーマレスについて、突然狂った様にごちゃ混ぜなデータを格納し始めるのではないか、という様な事を話します。たしかにリレーショナルデータベースと同様のモデルで実際に痛みを伴うデータセットと領域が存在しますが、特殊なケースでしょう。スキーマレスは凄いのですが、殆んどのデータは高度に構造化されてしまいます。特に新しい機能を導入する場合、時々不整合を起こしやすいのは確かです。しかし NULL カラムが実際に上手く解決出来ない様な問題となる事は無いでしょう。

私にとって、スキーマレスの本当の利点はセットアップの省略とオブジェクト指向プログラミングとの摩擦の低減です。これは、静的型付け言語を利用している場合に特に当てはまります。私はこれまで C#と Ruby で MongoDB を利用してきましたが、違いは顕著です。Ruby のダイナミズムと有名な ActiveRecord 実装は既にオブジェクトとリレーショナル DB の摩擦を十分低減しています。それは実際に MongoDB が Ruby と上手く適合していないと言っているわけではありません。どちらかというと多くの Ruby 開発者は MongoDB を追加の改善として見ると思います。一方、C#や Java 開発者はこれらのデータ相互作用を根本的な変化として見るでしょう。

ドライバ開発者の視点で考えてみて下さい。オブジェクトを保存する際に JSON(正確には BSON だけど大体同じ) にシリアライズして MongoDB に送信します。プロパティマッピングや、型マッピングもありません。アプリケーション開発者が単純明快に実装できます。

5.2 書き込み

MongoDB が適合する専門的な役割のひとつはロギングです。MongoDB には書き込みを速くするための 2 つの特徴があります。ひとつ目は、送信した書き込み命令は実際の書き込みを待たず即座に復帰する事です。ふたつ目は、バージョン 1.8 で導入され、2.0 で強化されたジャーナリング機能です。これはデータの耐久性に関する振る舞いを制御できる機能です。書き込み毎にこれらの設定を指定することで、サーバーは書き込みを完了する前にデータを取得します。これによって素晴らしい書き込み性能と耐久性が得られます。

パフォーマンスに加え、ログデータはスキーマレスの利点を活かす事が出来るデータセットの一つです。最後に、MongoDB の Capped コレクションと呼ばれる機能を紹介します。これ

まで作成してきたコレクションは暗黙的に普通のコレクションを作成してきました。capped コレクションは db.createCollectionコマンドにフラグを指定して作成します:

```
// このCapped コレクションを 1Mbyte で制限します
db.createCollection('logs', {capped: true, size: 1048576})
```

この場合、Capped コレクションが 1MByte の上限に達した時、古いドキュメントは自動的に削除されます。ドキュメントのサイズではなく数で制限を行いたい場合、maxを設定出来ます。Capped コレクションは幾つかの興味深い性質を持っています。例えば、ドキュメントの更新を行なってもサイズが増え続ける事はありません。また、挿入した順序を維持しているため、時間でソートする為のインデックスを貼る必要はありません。

発生した書き込みエラーの内容を知りたい場合、ここが説明する丁度良い機会でしょう (一方向通信書き込みとは対照的)。それは簡単なコマンドで取得できます: db.getLastError ()多くのドライバは insertの 2 番目のパラメータに{:safe => true}を指定するなどして、安全な書き込みをカプセル化しています。

5.3 耐久性

MongoDB バージョン 1.8 までは単一サーバーの耐久性はありませんでした。サーバーがクラッシュした場合データを失う可能性が高かったのです。ジャーナリングという重要な機能が 1.8 に追加されるまで、解決策は MongoDB を複数台で構成するしかありませんでした (MongoDB はレプリケーションをサポートしています)。この機能を有効にするには、最初に MongoDB をセットアップした時に生成した mongodb.configに journal=trueという行を追記します (反映させるには再起動が必要です)。これで恐らくジャーナリング機能が有効になったと思います (将来これはデフォルトになるでしょう)。とはいっても、ジャーナリングを無効にした状況と比べて余計な処理が発生するリスクがあります。(それはアプリケーションがデータを見失う可能性があることを指摘しておきます)

ここで耐久性に関して触れた理由は、MongoDB に関する多くの情報の中で単一サーバーの耐久性に関する情報が不足していたからです。Google で検索してみれば解りますが、見つかる情報は古くて既に存在しない機能です。

5.4 全文検索

全文検索は将来の MongoDB の将来のリリースに期待されている機能です。それは配列もサポートしているし、基本的な全文検索をとても簡単に実装できます。もっと強力な検索機能が必要ならば、Lucene や Solr などと連携させる必要があるでしょう。もちろん、これは他の

リレーショナルデータベースでも同様です。

5.5 トランザクション

MongoDB はトランザクションを持っていません。２つの代替手段を持っていて、１つめは素晴らしいのですが利用に制限があります、もうひとつの方法は柔軟性は高いのですが面倒です。

１つめの方法はアトミック操作です。それは素晴らしく実際の問題に適合します。既に`$inc`や`$set`などの単純な例を見てきました。`findAndModify`というドキュメントの更新と削除をアトミックに行うコマンドもあります。

２つめは、アトミック操作が不十分で二層コミットをフォールバックする際に利用します。二層コミットは Join に手動参照するトランザクションです。それはコードの中で行うストレージから独立した解決手段です。二層コミットは複数のデータベースにまたがってトランザクションを行う為の方法としてリレーショナルデータベースの世界ではとても有名です。MongoDB の Web サイトに一般的なシナリオを解説した例があります (銀行口座)。基本的な考え方は、実際のドキュメントの中にトランザクションの状態を格納し、init-pending-commit/rollback の段階を手動で行います。

MongoDB がサポートしている入れ子のドキュメントとスキーマレスな設計は二層コミットの痛みを少し和らげます。しかし初心者にとってはまだあまり良い方法では無いでしょう。

5.6 データ処理

MongoDB はデータ処理の仕事の殆んどを MapReduce に頼っています。幾つかの基本集約機能がありますが、あまり良いものではないのであなたはきっと MapReduce を利用したいと思うでしょう。次の章で、MapReduce の詳細を見ていきます。今の所、MapReduce はとても強力で、`group by`とは別のものだと考えてもかまいません (控えめに言っても)。MapReduce の魅力のひとつは巨大なデータを並列処理出来ることです。しかし、MongoDB の実装はシングルスレッドの JavaScript に依存しています。なにが問題でしょうか?巨大なデータを処理するためには Hadoop の様なものを必要とするでしょう。幸いにも、２つのシステムはお互いに補完する関係にあります。ここに MongoDB adapter for Hadoop があります。

もちろん並列データ処理はリレーショナルデータベースの得意とするものでもありません。MongoDB の将来のバージョンで巨大なデータセットをもっと上手く扱えるようにする計画があります。

5.7 位置情報

非常に強力な機能として MongoDB は位置情報インデックスをサポートしています。x と y の座標をドキュメントに格納し、`$near`で指定した座標で検索したり、`$within`で指定した四角や円で検索を行えます。この機能は図で説明したほうが分かりやすいので 5 分間位置情報チュートリアルを試すことをお勧めします。

5.8 ツールと成熟度

既に知っていると思いますが、MongoDB はリレーショナルデータベースより新しいシステムです。何をどの様に行いたいかに依りますが、この事はよく理解しておくべきでしょう。それにしても率直に評価すると MongoDB は新しく、あまり良いツールが在るとは言えません。(とはいえ、成熟したリレーショナルデータベースのツールにも怖ろしく酷いものはあります!) 例を挙げると、10 進数での浮動小数点の欠如はお金を扱うシステムでは明らかな懸念点です。(それほど致命的ではありませんが)

良い面を挙げると、多くの有名な言語のドライバが存在します。プロトコルは単純で現代的で目まぐるしい速度で開発されています。MongoDB は多くの企業の製品に利用され、成熟度に関する懸念は急速に過去のものになりつつあります。

5.9 章のまとめ

この章で伝えたかったことは、MongoDB は大抵の場合リレーショナルデータベースを置き換えられるということです。もっと率直に言えば、それは速さの代わりに幾つかの制約をアプリケーション開発者に課します。トランザクションの欠如は正当で重要な懸念です。また、人々は尋ねます「**MongoDB は新しいデータストレージ分野の何処に位置するのでしょうか?**」答えは単純です: 「**ちょうど真ん中あたりだよ**」

第 6 章

MapReduce

MapReduce は、従来のソリューションを上回る 2 つの有用な利点を持ったデータ処理手法です。最初であり、かつ主要な目的はパフォーマンスを発達させる事です。理論上では、MapReduce は並行化によって、巨大なデータセットを多くの CPU やマシンで処理する事が出来ます。最初に述べておくと、MongoDB は現在の所この利点はありません。MapReduce の 2 番目の利点は実際に書いたコードで処理することが出来るという事です。SQL と比べて何を行えるのか比較すると、MapReduce のコードは特別なソリューションを利用すること無く機能を拡張し、飛躍的に豊かな柔軟性を提供します。

MapReduce は注目を集めているパターンです、あなたは、C#, Ruby, Java, Python など、ほとんど全ての実装でこれを利用することが出来ます。それらはまったく異なっていて複雑に見える事を警告しておきます。挫折せず時間をかけて学んでみてください。これは MongoDB の利用に関わらず理解しておく価値があります。

6.1 理論と実践

MapReduce は 2 段階の処理に分かれています。最初に map を行い、次に reduce を行います。mapping の段階で入力されたドキュメントを変換し、key=>value のペアを emit します (キーと値は複合化可能です)。次に key/value ペアを key 毎にグループ化します。reduce の段階で emit されたキーと値の配列から処理を行った最終的な結果を集約します。

それでは各段階での出力を見ていきましょう。

ここでは、(Web ページの) リソースに対して日別のアクセス数のレポートを生成する例を使用します。これは MapReduce の **hello world** です。目的は、`hits`コレクションに 2 つのフィールド: `resource`と `date`という入力を利用して、`resource`、`year`、`month`、`day`、`count`とい

う出力を行います。

hitsに以下のデータがあります:

```
resource date
index Jan 20 2010 4:30
index Jan 20 2010 5:30
about Jan 20 2010 6:00
index Jan 20 2010 7:00
about Jan 21 2010 8:00
about Jan 21 2010 8:30
index Jan 21 2010 8:30
about Jan 21 2010 9:00
index Jan 21 2010 9:30
index Jan 22 2010 5:00
```

以下の出力を期待しているとします:

```
resource year month day count
index 2010 1 20 3
about 2010 1 20 1
about 2010 1 21 3
index 2010 1 21 2
index 2010 1 22 1
```

この種のアクセス解析の良い所は、レポートを素早く生成して出力を保存し、増え続けるデータを抑制出来る事です。(1 日毎に解析するページの数だけのドキュメントが追加されます。)

当面は、概念の理解に集中します。章の最後の方でサンプルデータとコードを利用して自分自身で試してみましょう。

まず初めに、以下の map 関数を見てください。map の目的は reduce 出来るような値を生成し、emit する事です。map は 0 回以上 emit する事が可能です。今回の場合、全て共通に一度だけ emit を行います。この map 関数は hits コレクションのドキュメント毎にループしていると想像して下さい。ドキュメント毎に、キーを resource, year, month, day 指定し、値には単純に 1 を指定して emit します:

```
function() {
    var key = {
        resource: this.resource,
        year: this.date.getFullYear(),
        month: this.date.getMonth(),
        day: this.date.getDate()
    };
```

```
    emit(key, {count: 1});
}
```

thisはループ中のドキュメントを参照します。恐らくは、map 段階の後にどの様なデータが出力されるかを確認する事が、理解の助けになるでしょう。前記した入力データを利用すると、map 完了後のデータは以下の様になり、emitされた値はキー毎に配列としてグループ化されます:

```
{resource: 'index', year: 2010, month: 0, day: 20}
=> [{count: 1}, {count: 1}, {count:1}]

{resource: 'about', year: 2010, month: 0, day: 20}
=> [{count: 1}]

{resource: 'about', year: 2010, month: 0, day: 21}
=> [{count: 1}, {count: 1}, {count:1}]

{resource: 'index', year: 2010, month: 0, day: 21}
=> [{count: 1}, {count: 1}]

{resource: 'index', year: 2010, month: 0, day: 22}
=> [{count: 1}]
```

この中間段階を理解することが、MapReduce を理解する事の鍵です。.NET や Java 開発者は IDictionary<object, IList<object>>(.Net) や HashMap<Object, ArrayList>(Java) の様な物だと思って構いません。

それでは、map 関数を不自然に変更してみましょう:

```
function() {
    var key = {
        resource: this.resource,
        year: this.date.getFullYear(),
        month: this.date.getMonth(),
        day: this.date.getDate()
    };
    if (this.resource == 'index' && this.date.getHours() == 4) {
        emit(key, {count: 5});
    } else {
        emit(key, {count: 1});
    }
}
```

中間段階の出力は以下の様に変わります:

```
{resource: 'index', year: 2010, month: 0, day: 20}
=> [{count: 5}, {count: 1}, {count:1}]
```

キーに対応して emit で生成される値が、どの様にまとめられているかに注目して下さい。

reduce 関数はこれらの中間結果を受け取り、最終的な結果として出力します。以下を見て下さい:

```
function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count'];
    });
    return {count: sum};
};
```

以下の出力を得られます:

```
{resource: 'index', year: 2010, month: 0, day: 20} => {count: 3}
{resource: 'about', year: 2010, month: 0, day: 20} => {count: 1}
{resource: 'about', year: 2010, month: 0, day: 21} => {count: 3}
{resource: 'index', year: 2010, month: 0, day: 21} => {count: 2}
{resource: 'index', year: 2010, month: 0, day: 22} => {count: 1}
```

正確には、MongoDB はこの様に出力します:

```
_id: {resource: 'index', year: 2010, month: 0, day: 20}, value: {count: 3}
```

これが目的の結果である事に気が付きましたでしょうか。

注意深く見て来たのであれば、あなたはこんな疑問を持つかもしれません、なぜ `sum = values.length`を利用しないのですか? 原則として`{count: 1}`しか合計しない場合、この方法は効果的の様に見えます。しかしこの動作は保証されません。Reduce は何度か部分的に Reduce された値で呼び出される可能性があります。これの目的は、reduce 関数を複数のスレッドやプロセス、あるいはコンピュータに分散する事を可能にする為です。reduce 関数の結果は同じ reduce 関数にフィードバックされます。(データセットが大きくなると、何度も行われるでしょう)

例に戻ると、reduce は以下のような入力で呼び出されるかもしれません:

```
{resource: 'index', year: 2010, month: 0, day: 20}
=> [{count: 1}, {count: 1}, {count:1}]
```

もしくは以下のように 2 段階に分かれて呼ばれるかもしれません:

```
// ステップ 1
{resource: 'index', year: 2010, month: 0, day: 20}
 => [{count: 1}, {count: 1}]

// ステップ 2
{resource: 'index', year: 2010, month: 0, day: 20}
=> [{count: 2}, {count: 1}]
```

‘sum = values.length’ を利用した場合、ステップ 2 の例では誤った答えが返るでしょう。

一般的に、reduce の出力は同様に入力になり得る構造になるべきです。そして複数回呼び出された reduce の結果は同じ結果になる必要があります (これは冪等性として知られています)。

最後に、ここではあまり触れませんでしたがより複雑な解析を行う場合に reduce メソッドを連鎖する事は一般的です。

6.2 ひたすら練習

MongoDB では、コレクションに対して mapReduceコマンドを呼び出して MapReduce を実行します。mapReduceには引数に map 関数と reduce 関数、そして出力ディレクティブを引き渡します。mongodb のシェルでは JavaScript の関数を定義して解釈します。多くのライブラリでは関数を文字列で引き渡します (ちょっとカッコ悪いけど)。まずはこれらのデータを入力してみましょう:

```
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 4, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 5, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 20, 6, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 7, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 9, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 9, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 22, 5, 0)});
```

続いて map と reduce 関数を定義します (MongoDB のシェルは複数行の命令文を解釈します。...は引き続きテキストが入力される事を期待しています):

```
var map = function() {
    var key = {resource: this.resource,
               year: this.date.getFullYear(),
               month: this.date.getMonth(),
               day: this.date.getDate()
```

```
            };
    emit(key, {count: 1});
};

var reduce = function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count'];
    });
    return {count: sum};
};
```

この map、reduce関数を mapReduceコマンドに渡して実行します:

```
db.hits.mapReduce(map, reduce, {out: {inline:1}})
```

上記を実行すると、期待した出力が表示されます。out: {inline:1}を設定すると、mapReduceの処理結果が順次表示されます。この機能は現在の所結果のサイズが 4MByte 以下に制限されています。代わりに、{out: 'hit_stats'}と指定することで結果を hit_statsコレクションに格納することが出来ます:

```
db.hits.mapReduce(map, reduce, {out: 'hit_stats'});
db.hit_stats.find();
```

これを行うと、既存の hit_statsにある既存のデータは消えてしまいます。もし{out: {merge: 'hit_stats'}}を指定したのであれば、既存のキーは上書きされ、新しいキーは新しいドキュメントとして挿入されます。最後に、高度な使い方として、reduce関数で outを操作することが可能です (upsert の様な使い方が出来ます)

3 番目の引数は追加のオプションを渡します。例えば、解析したいドキュメントを制限したり、フィルタやソートを行うことが出来ます。finalizeメソッドを渡すと、reduce後の段階結果にこの関数を適用することもできます。

6.3 章のまとめ

これは、これまでに触れてきた内容とはまったく異なる最初の章でした。もし不安が残るようであれば、MongoDB のその他の aggregation capabilities を参照することが出来ます。最後になりますが、MapReduce は MongoDB の最も強力な機能のひとつです。正しく理解するための鍵は、あなたの書いた map と reduce 関数を思い浮かべ、mapを出てから reduceに入る前の中間データを理解することです。

第 7 章

パフォーマンスとツール

最後の章では、パフォーマンスに関する話題と MongoDB 開発者に有効な幾つかのツールを見ていきます。どちらの話題にも深くは追求しませんがそれぞれの最も重要な側面を分析します。

7.1 インデックス

まず最初に、特別な system.indexesコレクションの中に含まれるデータベースのインデックス情報を見ていきましょう。MongoDB のインデックスはリレーショナルデータベースのインデックスと同じように動作します。すなわち、これらはクエリーやソートのパフォーマンスを改善するのに役立ちます。インデックスは ensureIndexを呼んで作成されます:

```
// この "name" はフィールド名です
db.unicorns.ensureIndex({name: 1});
```

そして、dropIndexを呼んで削除します:

```
db.unicorns.dropIndex({name: 1});
```

2 番目のパラメーターに{unique: true}に設定することでユニークインデックスを作成できます。

```
db.unicorns.ensureIndex({name: 1}, {unique: true});
```

インデックスは埋めこまれたフィールドと配列フィールドに対して作成できます。複合インデックスも作成できます:

```
db.unicorns.ensureIndex({name: 1, vampires: -1});
```

インデックスの順序 (1 は昇順、-1 は降順) は単一のキーインデックスでは問題となりませんが、複合キーでソートやレンジ条件を利用した際に影響があります。

インデックスに関する詳しい情報は indexes page にあります。

7.2 Explain

インデックスを使用しているかに関わらず、カーソルに対し explainメソッドを使うことが出来ます:

```
db.unicorns.find().explain()
```

出力は BasicCursorが利用され (インデックスを使用していない事を意味します)、あの 12 個のオブジェクトをスキャンしてどれくらいの時間がかかったのかなど、その他の便利な情報も教えてくれます。

もしインデックスを利用するように変更した場合 BtreeCursorが利用されていることを確認できます。この場合、インデックスはうまく利用できているでしょう:

```
db.unicorns.find({name: 'Pilot'}).explain()
```

7.3 一方向通信書き込み

前述したように、MongoDB の書き込みはデフォルトでは一方向通信 (fire-and-forget) です。これによってクラッシュするとデータを失うリスクと引き換えに素晴らしいパフォーマンスを得ることが出来ます。興味深い副作用として、挿入や更新などの書き込みでユニーク制約の違反が発生した場合でもエラーを返却しません。書きこみ失敗について知る為には挿入を行った後に db.getLastError()を呼ぶ必要があります。多くのドライバは安全な書き込みを行う方法を追加のパラメーターによって抽象化して提供しています。

あいにく、シェルは自動的に安全な挿入を行います。従って私たちはこの動作の振る舞いを簡単に確認することが出来ません。

7.4 シャーディング

MongoDB は自動シャーディングをサポートしています。シャーディングはデータを複数のサーバーに分割してスケーラビリティを高める手法です。単純な実装ではデータの名前が A

〜M で始まるものをサーバー 1 に、残りをサーバー 2 に格納するでしょう。有り難いことに、MongoDB のシャーディング能力はその単純なアルゴリズムを上回ります。シャーディングの話題はこの本では取り上げませんが、単一サーバーのデータが限界まで増えた際に、あなたはシャーディングの存在と、それついてよく知っている必要があるでしょう。

7.5 レプリケーション

MongoDB のレプリケーションの動きはリレーショナルデータベースのそれとよく似ています。単一のマスターサーバーに対し書き込みが行われると、他のスレーブサーバに同期します。あなたはスレーブに対して読み込みリクエストを発生させるかどうかを制御できます。これは古いデータを読み込むリスクを低減させるのに役立ちます。マスターが落ちた場合、スレーブが新しいマスターの役割に昇格することが出来ます。MongoDB のレプリケーションもまた、この本の主題から外れます。

レプリケーションの主要な目的は信頼性の向上ですが、読み込みリクエストを分散することでパフォーマンスを改善することも出来ます。レプリケーションとシャーディングを組み合わせることは一般的な方法です。例えば、それぞれのマスターとスレーブシャードを共有することが出来ます。(厳密には、調停者が 2 つのスレーブの均衡を破って、マスターになれる様に助ける必要があります。その為に調停者は若干のリソースと、複数のシャードを利用できる事を要求します。)

7.6 統計

あなたは `db.stats()` とタイプすることでデータベースの統計を取得できます。データベースのサイズは最もよく扱う情報です。`db.unicorns.stats()` とタイプすることで `unicorns` というコレクションの統計を取得することも出来ます。同様にこのコレクションのサイズに関する情報も有用です。

7.7 Web インターフェース

MongoDB を起動すると、Web ベースの管理ツールに関する情報が含まれています (`mongod` を起動した時点までターミナルウィンドウをスクロールすればその様子を確認できるでしょう)。あなたはブラウザで http://localhost:28017/ を開いてアクセス出来ます。設定ファイルに `rest=true` を追加して `mongod` プロセスを再起動すると、さらにこれを有効に活用出来るでしょう。この Web インターフェースはサーバの現在の状態についての洞察を与えてくれます。

7.8 プロファイラ

以下を実行をすることで MongoDB プロファイラを有効にできます:

```
db.setProfilingLevel(2);
```

有効にした後に、以下のコマンドを実行します:

```
db.unicorns.find({weight: {$gt: 600}});
```

そして、プロファイラを観察して下さい:

```
db.system.profile.find()
```

この出力は何がいつどれ程のドキュメントを走査し、どれ程のデータが返却されたかを教えてくれます。

再度、setProfileLevelの引数を 0に変えて呼び出すことでプロファイラを無効に出来ます。他のオプションは 1を指定することで 100 ミリ秒以上のクエリーのみをプロファイリングします。さらに、2 番目のパラメータに最小時間をミリ秒で指定することが出来ます。

```
// 1秒以上のクエリーをプロファイルする
db.setProfilingLevel(1, 1000);
```

7.9 バックアップとリストア

MongoDB には binの中に mongodumpという実行ファイルが付属しています。単純に mongodumpを実行すると、ローカルホストに接続して全てのデータベースを dumpサブフォルダ以下にバックアップします。mongodump --helpとタイプすると追加のオプションを確認できます。指定したデータベースをバックアップする--db DBNAMEと、指定したコレクションをバックアップする--collection COLLECTIONNAMEは共通です。次に、同じ binフォルダにある mongorestore実行ファイルを使うことで、以前のバックアップをリストアできます。同じように、--dbと--collectionはオプションはリストアするデータベースとコレクションを指定します

例えば、learnデータベースを backupフォルダにバックアップするには以下を実行します (これは実行ファイルですので mongo シェルではなく、ターミナルウィンドウでコマンドを実行します):

```
mongodump --db learn --out backup
```

unicornsコレクションのみをリストアするにはこの様に実行します:

```
mongorestore --collection unicorns backup/learn/unicorns.bson
```

mongoexportと mongoimportという２つの実行ファイルは JSON または CSV 形式でエクスポートとインポートできることを指摘しておきます。例えば JSON 形式で出力するには以下のようにします:

```
mongoexport --db learn -collection unicorns
```

そして、CSV 形式での出力はこうします:

```
mongoexport --db learn -collection unicorns --csv -fields name,weight,vampires
```

mongoexportと mongoimportは完全にあなたのデータを表現できないことに注意して下さい。mongodumpと mongorestoreのみを実際のバックアップでは利用すべきです。

7.10 章のまとめ

この章では、MongoDB で利用する様々なコマンドやツールやパフォーマンスの詳細を見てきました。全てに触れることは出来ませんでしたが最も一般的なものを紹介しました。MongoDB のインデックス化がリレーショナルデータベースのインデックス化とよく似ているのと同様に、ツールの多くも同じです。しかし MongoDB の方がわかり易くて単純に利用できるものが多いでしょう。

まとめ

実際のプロジェクトで MongoDB を使い始める為には十分な情報を持つべきです。ここで紹介してきた事の他にも、MongoDB の情報はまだまだあります。しかし、あなたが次に優先すべき事は、これまでに学んできた事を組み合わせて、これから利用するドライバに慣れる事です。MongoDB のウェブサイトには数多くの役立つ情報があります。公式な MongoDB ユーザーグループは質問を尋ねるには最適な場所です。

NoSQL は必要性によってのみ生み出されただけではなく、新しいアプローチへの興味深い試みでもあります。有難いことにこれらの分野は常に発展しており、私たちが時々失敗し、挑戦し続けなければ成功はありえません。思うにこれは、私たちがプロとして活躍するための賢明な方法となるでしょう。

MongoDB の薄い本

2016 年 4 月 24 日 2.0.5 版発行

著者　Karl Seguin
翻訳　HAMANO Tsukasa