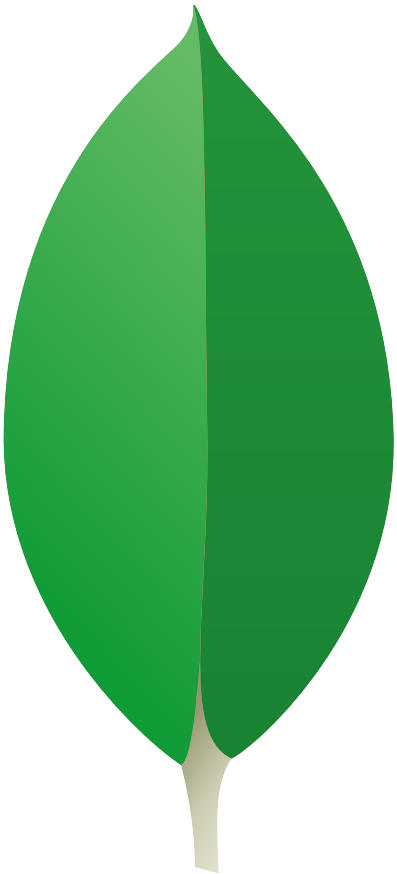
2.6対応版

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

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。

最新バージョン

この本は MongoDB 2.6 に対応するよう Asya Kamsky によって更新されました。

原書の最新のソースはこちら:

http://github.com/karlseguin/the-little-mongodb-book

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

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。長い歴史を持つ技術に対する開発者の関心はあっという間に移り変わる恐れがあります。

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。MongoDB 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。

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点は、MongoDB にデータを問い合わせるとカーソルと呼ばれる結果データへのポインタを返します。カーソルは実際のデータを取り出す前に、カウントやスキップの様な操作はを行うことが出来ます。

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。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。

まず私たちはグローバルな use ヘルパー使っ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 BSON というバイナリでシリアライズされた JSON フォーマットを利用します。外側から見ると JSON をいろいろ使っているように見えます。ここで db.getCollectionNames() を実行すると unicorns コレクションが表示されます。

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()
```

あなたが挿入したデータに_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くてはいけません。これは ObjectID 型の値を自分で生成するか MongoDB に生成させる事になります。ほとんどの場合 MongoDB に生成させたいと思うでしょう。_id フィールドには既定でインデックスが貼られます。これは getIndexes コマンドで確認できます。

```
db.unicorns.getIndexes()
```

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)。もしメスのユニコーンを見つけたい場合、{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、演習の為のデータを幾つかセットアップしましょう。まず、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。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、性別がオスで体重が 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。$in 演算子は配列で渡した値のいずれかにマッチします。例えば:

```
db.unicorns.find({loves: {$in:['apple','orange']}})
```

これで apple または orange が好きなユニコーンが返ります。

異なるフィールドに対して AND などではなく OR を利用したい場合、$or 演算子を利用して、OR をとりたいセレクターの配列を指定します。

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

上記は全てのメスのユニコーンの中からりんごが好き、もしくは体重が 500 ポンド未満の条件で検索します。

最後 2 つはとてもイカした用例です。お気づきと思いますが 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 のドキュメントの Query Selectors 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 を使い始めるために知るべき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 更新: 置換 と$set

最も単純な形式では update に 2 つの引数を渡します。セレクター (where 条件) と更新を適用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は以下を実行します:

```
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。)

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 update の動作に驚くはずです。2 番目のパラメータに更新演算子を指定しなかったので元のデータを**置き換える**動作となり、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。SQL の update 文にこれと同等の機能はありません。本当に動的な更新を行いたい場合などの特定の状況で、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 演算子を利用すべきです。

以下のように実行して消えたフィールドをもとに戻します:

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

このように 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 更新演算子

$set に加え、その他の演算子を利用するともっと粋なことが出来ます。これらの更新演算子は、フィールドに対して作用します。なのでドキュメント全体が消えてしまうことはありません。例えば、$inc 演算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 演算子を使って、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演算子は MongoDB マニュアルの更新演算子 ((http://docs.mongodb.org/manual/referenc operators) 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 は可読性が良く、よくあるシチュエーションで重宝します。update の 3 番目の引数に{upsert:true}を渡す事で upsert を利用できます。

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。upsert パラメータを省略 (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 オプション加えると違った結果になります。

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, {upsert: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}}, {upsert: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に予防接種を受けさせた (vaccinated) としましょう。これを行うには、multi オプションを true に設定します:

```
db.unicorns.update({}, {$set: {vaccinated: true }}, {multi: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 は更新演算子を使わないと既存のドキュメントを置き換えます。通常は$set などの様々な演算子を利用してドキュメントを変更します。次に、update は直感的な upsert オプションをサポートしています。これはドキュメントが存在するかどうか分からない時に便利です。最後に、デフォルトで update は最初にマッチしたドキュメントしか更新しません。マッチしたすべてのドキュメントを更新したい場合は multi オプションを指定してください。

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

カーソルについて学ぶ前に、find に任意で設定出来る “projection” という 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、もしくは除外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その他のフィールドを除外して取得します。

```
db.unicorns.find({}, {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すと説明しました。しかしシェルは find を即座に処理を実行しているように見えるます。これはシェルだけの動作です。カーソルの本当の動作は find にメソッドをひとつ連結することで観測出来

ます。まずはソートを見てみましょう。JSON ドキュメントのフィールドを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

ページングは cursor の `limit` メソッドや `skip` メソッドを利用して実現できます。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ことができます。

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いうと、NoSQL データベースの中でドキュメント指向データベースはリレーショナルデータベースと多くの部分で共通しています。しかし重要な違いがあります。

4.1 Join がありません

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 度の `find` クエリーを発行して 2 つ目のコレクション内の関連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'})
```

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。

2 つのコンセプトを組み合わせて埋め込みドキュメントの配列を埋め込むこともできます。

```
db.employees.insert({_id: ObjectId(
                     "4d85c7039ab0fd70a117d735"),
                     name: 'Chani',
                     family: [ {relation:'mother',name: 'Chani'},
                               {relation:'father',name: 'Paul'},
                               {relation:'brother', name: 'Duncan'}]})
```

4.1.2 非正規化

join を代替するもうひとつの方法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スナップショットデータ (監査ログの様な) 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。これは各ドキュメントで各情報の断片を重複させろという意味ではありません。データの重複を恐れながら設計するのではなく、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ましょう。

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 テーブルにユーザー ID を持たせて**ユーザー**と**投稿**を関連付けます。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。有効な代替案は各投稿エントリに単純にユーザー ID に対応するユーザー名を保持することです。埋め込みドキュメントでは以下のようにしします。

```
`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 (1 回のマルチアップデートで済みます)。

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最良の対処になることもあるでしょう。

4.1.3 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。しかし一般的に新しい開発者は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でしょう。

まず、各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、ほとんどの場合親ドキュメントと同時に取得したい小さなデータです。以下は各ユーザーのアカウントに住所を持たせる実例です:

```
db.users.insert({name: 'leto',
                 email: 'leto@dune.gov',
                 addresses: [{street: "229 W. 43rd St",
                              city: "New York", state:"NY",zip:"10036"},
                             {street: "555 University",
                              city: "Palo Alto", state:"CA",zip:"94107"}]})
```

これを単に手っ取り早く書き込むための短縮記法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。これは特に埋め込みドキュメントや配列のインデックスフィールドのクエリーに適応します。

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制しないのであれば、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作ることも出来ますが、こんなことをしてはいけませ

ん。多くの MongoDB システムはよく見るリレーショナルデータベースと同じ様に設計されますが、それよりもコレクションは少なくなります。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の懸念について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` コレクションを別々に持つべきか、`post` ドキュメントにコメントの配列を埋め込むべきでしょうか?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?)。それはより単純明快で、より良いパフォーマンスが得られます。MongoDB の柔軟なスキーマはこの 2 つのアプローチを組み合わせて、別々のコレクションに分けたまま、少ない数のコメントを投稿に埋め込む事が出来ます。これは 1 回のクエリーで欲しいデータをまとめて取得するという原則に従っています。

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。あなたは多く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 動的スキーマ

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動的スキーマです。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。私は動的スキーマを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。

私にとって動的スキーマ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 2 つの性質があります。1 つ目は、送信した書き込み命令は実際の書き込み完了を待たず即座に戻ってくるオプションがあります。2 つ目は、データの耐久性に関する動作を制御できる事です。これらの設定は何台のサーバーに書き込んだら成功とするかを設定することで書き込み性能と耐久性を絶妙にコントロール出来ます。

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})
```

Capped コレクションが 1MByte の上限に達すると、古いドキュメントは自動的に削除されます。max オプションを指定することでドキュメントのサイズではなく数で制限できます。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は増えません。挿入した順序を維持するので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。Unix の tail -f コマンドのように、1 回のクエリーで Capped コレクションを tail できます。

データを時間で有効期限切れにしたい場合、TTL(有効期間) インデックスを利用できます。

5.3 耐久性

MongoDB バージョン 1.8 までは 1 台構成のサーバーに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。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)。ジャーナリングは 1.8 で追加された重要な機能のひとつです。MongoDB バージョン 2.0 以降ジャーナリングは既定で有効になっているのでクラッシュや電源喪失からの迅速な復旧が可能です。

ここで耐久性に関して触れたのは、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。ググってみると MongoDB にはジャーナリングが無いという古い情報が見つかります。

5.4 全文検索

正真正銘の全文検索機能は最近の MongoDB に追加されました。15 の言語でステミング (語幹抽出) とストップワードに対応しています。MongoDB は配列と全文検索に対応していますので、もっと強力な検索機能が必要な場合のみ、その他の全文検索エンジンと組み合わせる必要があるでしょう。

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

バージョン 2.2 より前の MongoDB はデータ処理の殆どの仕事を MapReduce に頼っていました。バージョン 2.2 以降、アグリゲーション・フレームワークやパイプラインと呼ばれる強力な機能が追加されましたので、MapReduce を使う必要があるのはパイプラインでサポートしていない複雑な関数での集約が必要な稀なケースに限られます。アグリゲーション・パイプラインと MapReduce の詳細については次の章で説明します。いまの所これらはグループ化する為の機能豊富な方法がいくつかある、と思っていてください。(控えめな言い方ですが) 巨大なデータを並列処理するには Hadoop の様なものと連携する必要があるでしょう。ありがたい事に、お互いに補完し合う 2 つのシステムをつなげるための MongoDB コネクタが Hadoop にはあります。

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もそれほど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は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。geoJSON または 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
データの集計

6.1 アグリゲーション・パイプライン

アグリゲーション・パイプラインはコレクション内のドキュメントを変換・結合する方法です。ドキュメントをパイプラインに渡すという方法は出力データを別のコマンドに渡す Unix の「パイプ」とちょっと似ています。

最も単純なアグリゲーション（集約）はたぶん知っていると思いますが SQL の GROUP BY です。既に count() 関数を説明しましたが、ユニコーンをオスとメスで別々に集約したい場合はどうやるのでしょうか?

```
db.unicorns.aggregate([{$group:{_id:'$gender',
                                total: {$sum:1}}}])
```

シェル内ではパイプライン演算子を配列で渡す aggregate ヘルパー関数があります。何かを単純にグループ化して集約するには$group というオペレーターを利用します。これは SQL の GROUP BY と似たようなもので、_id で指定したフィールドでグループ化 (ここでは gender) とその他フィールドの集約結果 (ここでは特定の性別にマッチするドキュメントの合計数) で新しいドキュメントを生成します。多分気がついていると思いますが_id フィールドには gender でなく$gender を渡しています。フィールド名の前の$はドキュメントのフィールド値に置き換えられます。

他にはどんなパイプライン演算子があるのでしょうか。$match 演算子は$group 演算子の前後でよく使われます。これは find メソッドと同じ様に条件にマッチする、あるいはマッチしないドキュメントの部分集合を集約します。

```
db.unicorns.aggregate([{$match: {weight:{$lt:600}}},
                       {$group: {_id:'$gender',  total:{$sum:1},
                                 avgVamp:{$avg:'$vampires'}}},
                       {$sort:{avgVamp:-1}} ])
```

ここで紹介したパイプライン演算子$sort はたぶんあなたの期待通りに動作します。もちろん$skip や$limit もあります。また$group 演算子で$avg を使用しています。

MongoDB の配列は強力で DB に格納されているデータはなんでも集約できます。これらを全て正しく数え上げるにはデータを「平ら」にしてやる必要があります。

```
db.unicorns.aggregate([{$unwind:'$loves'},
                       {$group: {_id:'$loves',  total:{$sum:1},
                                 unicorns:{$addToSet:'$name'}}},
                       {$sort:{total:-1}},
                       {$limit:1} ])
```

これは最も多くのユニコーン達に好まれている食べ物と各好物を好きなユニコーンの名前のリストを表示します。$sort と$limit の組み合わせはいわゆる「トップ N」のランキングを集計します。

find で指定する “projection” とよく似た$project という強力なパイプライン演算子があります。これは特定のフィールドを含めるだけでなく、既存のフィールドの値に基づいて新しいフィールドを作成・計算することができます。たとえば、算術演算子を利用して複数のフィールドを足し合わせて平均値を求めたり、文字列演算子を利用して既存のフィールドの文字列を結合した新しいフィールドを生成できます。

ここで説明したのはアグリゲーションで出来ることのほんの一部です。2.6 でアグリゲーションは更に強力になりました。集約命令の結果をカーソルに戻したり$out パイプライン演算子を利用して結果を新しいコレクションに書き込むことが出来ます。サポートされている更に多くのパイプラインと式演算子の利用例は MongoDB マニュアルを参照してください。

6.2 MapReduce

MapReduce は 2 段階のステップに分かれています。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 のペアを出力します (キーと値は複合できます)。次に key/value ペアを key 毎にグループ化します。reduce の段階で出力されたキーと値の配列から最終的な結果を生成します。map と reduce 関数は JavaScript で記述します。

MongoDB では、コレクションに対して mapReduce 命令を呼び出して MapReduce を行います。mapReduce の引数には 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 の関数を定義して引数に指定します。多くのライブラリでは関数を文字列で渡してやります (ちょっとカッコ悪いけど)。3 番目の引数にはドキュメント解析するためのフィルターやソート、リミットなどの追加のオプションを指定します。さらに、reduce 段階の後に評価される finalize メソッドを指定することも出来ます。

恐らくあなたは殆どのケースで MapReduce を使う必要はないでしょう。もしその必要があった場合は私のブログや MongoDB マニュアルを参照にしてください。

6.3 章のまとめ

この章では MongoDB のアグリゲーション機能を紹介しました。アグリゲーションパイプラインはデータをグループ化する為の強力な手段であり、データ構造を把握していれば比較的単純に記述できます。MapReduce はより理解するのが難しいですが JavaScript で記述できるのでその能力は際限がありません。

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紹介します。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部分を分析します。

7.1 インデックス

まず最初に getIndexes コマンドで表示されるコレクション内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) は単一キーのインデックスでは影響ありませんが、複合キーで複数のインデックスフィールドをソートする際に違いがあります。

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 レプリケーション

MongoDB のレプリケーションはリレーショナルデータベースとよく似た仕組みで動作します。理想的に、本番環境には同じデータを持つ 3 台以上のレプリカセットを配備してください。単一のプライマリサーバーに対し書き込みが行われると、すべてのセカンダリサーバへ非同期に複製されます。あなたはセカンダリサーバに対して読み込みリクエストを許可するかを制御できます。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、セカンダリサーバの 1 つが新しいプライマリサーバに昇格します。MongoDB のレプリケーションもこの本の主題の範囲外です。

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やクラスタ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、単一レプリカセットのデータが限界まで増えた時、あなたはシャーディングの存在を思い出して利用することを検討してください。

レプリケーションは時間の掛かる参照リクエストをセカンダリサーバに分散することでパフォーマンスの向上にも役立ちますが、主要な目的は信頼性の向上です。シャーディングは MongoDB クラスタのスケーラビリティを向上させる事が主要な目的です。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スケーラビリティと高可用性を向上させるための常套手段です。

7.5 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。

7.6 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。

7.7 プロファイラ

以下を実行して MongoDB プロファイラを有効にします:

```
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、setProfilingLevel の引数を 0 に変えて呼び出すとプロファイラが無効化されます。最初の引数に 1 を指定すると 100 ミリ秒以上掛かるクエリーをプロファイリングします。100 ミリ秒は既定のしきい値です。2 番目の引数に最小のしきい値を指定できます。

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

7.8 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。mongodump と mongorestore は BSON という MongoDB の独自フォーマットを操作します。

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 --db learn --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 はデータを完全に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。詳細は MongoDB マニュアルのバックアップの選択肢を読んでください。

7.9 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一般的なものをいくつか紹介しました。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進展しており、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

2018 年 10 月 1 日 2.6.0 版発行

著者　Karl Seguin
翻訳　HAMANO Tsukasa