# Groonga ではじめる全文検索

版　　発行

# MySQL で作る全文検索システム

Groonga は様々なソフトウェアに組み込み、全文検索機能を提供することができます。その一つ、MySQL に組み込むための Mroonga を使って、全文検索システムを作ってみましょう。

# 第 1 章

# Mroonga の概要

Mroonga は、MySQL のストレージエンジンの一つです[*1]。MySQL のストレージエンジンには MyISAM や InnoDB などがあり、聞いたことがある人も多いでしょう。それぞれにパフォーマンスや対障害性などに特徴があり、テーブルごとに使い分けることができます。

その一つとして Mroonga は、Groonga を組み込んで、高速かつ多機能な全文検索を可能にしたストレージエンジンです。これを MySQL に組み込んで使うことで、SQL を使って全文検索を行えるようになります。

Mroonga を全文検索システムの本番環境で使うには、公式マニュアルのインストールのページを参照してください。ここでは、本書で試す目的で、簡易に環境を準備する方法を紹介します。

[*1] MySQL から派生した MariaDB というデータベースシステムでも使用できます。最新の MariaDB では始めから Mroonga が組み込まれています。

# 第 2 章

# Mroonga の環境の準備

本書では、MySQL 及びその他の環境の準備に Docker を使います。

以後、PHP と MySQL、Mroonga を使って PDF 文書の全文検索システムを作成していきますが、普段使っているシステムにインストール済みの物を使うと、バージョンが異なって本書の内容が当てはまらなかったり、また、将来別件で異なるバージョンをインストールしたい場合などに手間が増えてしまいます。設定ファイルの競合も問題になるかも知れません。Docker を使うと、システムの方に影響を与えずに、本書で必要なバージョンのソフトウェアを揃えることができます。不要になった場合には削除することも簡単で、その場合もシステムの方には影響を及ぼしません。

## 2.1 Docker のインストール

（後で書く）

## 2.2 Docker コンテナの起動

本書用に必要なソフトウェアをインストールした Docker イメージを作成してあります。ターミナルで上の `docker run` を実行したディレクトリーに移動し、以下のコマンドを実行してください。

リスト 2.1　Docker イメージの取得とコンテナの起動

```
% cd path/to/project
% docker run --detach --name=pdfsearch --publish=8080:80 \
    --volume=$PWD:/var/lib/pdfsearch kitaitimakoto/grnbook-mroonga
```

`docker run` コマンドにより、以下の三つのことが行われます。

1. インターネット経由の Docker イメージの取得（もしシステム上に存在しない場合）
2. 取得した Docker イメージを元にした Docker コンテナの作成
3. 作成した Docker コンテナの起動

Docker イメージの取得とコンテナの作成は一度行えば充分なので、コンテナの二度目以降の起動には別のコマンドを使用します。

1zwDocker コンテナの起動

```
% docker start pdfsearch
```

1zwDocker コンテナの停止

```
% docker stop pdfsearch
```

コンテナの操作に使用している pdfsearch という名前は、リスト 2.1 における name オプションで指定した物を使用しています。docker run 実行時に異なる名前を指定した場合は、そちらを使いましょう。(name をつけ忘れた時は docker ps で探して消すが、説明いる？)

もし、二度目も docker run コマンドを使用してしまうと、新たに**別の**コンテナを作成してしまいます。MySQL のデータ等は引き継がれませんし、同じ目的のコンテナが二つ以上あると混乱の元ですので、通常は一つになるようにしておきましょう。

## 2.3 動作確認

Docker コンテナが実際に動作していることを確認しましょう。このイメージには PHP が含まれているので、docker run を実行したディレクトリーで以下のようなファイルを作成することで動作確認ができます。

リスト 2.2 info.php

```
<?php
phpinfo();
```

ブラウザーで http://localhost:8080/info.php にアクセスしてください。PHP の情報が表示されれば、環境の準備は成功しています。(Docker Toolbox 使っている場合は localhost じゃなさそう)

「Not Found」の画面が表示されてしまう場合は、どこかで間違えてしまったようです。これまでの手順を見ながらやり直してみてください。

docker run をやり直すには、以下のコマンドを実行して一旦 Docker コンテナを停止し、削除する必要があります。

1zwDocker コンテナの停止と削除

```
% docker stop pdfsearch
% docker rm pdfsearch
```

ありがちな間違いとして、Docker コンテナとの共有ディレクトリーの指定ミスがあります。Docker コンテナとホストマシンは、docker run コマンドの volume (v) オプションで指定したディレクトリーを共有します。上の例では$PWD:/var/lib/pdfsearch を指定しています。コロンの左側がホストマシンのディレクトリー、右側がコンテナ内のディレクトリーです。$PWD は (docker run を実行した) 現在のディレクトリーを意味しますので、そこと別のディレクトリーにファイルを置いた場合は、コンテナ内の PHP が認識できません。事前にプロジェクトのディレクトリーに移動してから実行するようにしましょう (コンテナ内は/var/lib/pdfsearch がドキュメントルートとなるよう設定されており、こちらは常にこの値で大丈夫です)。

# 第 3 章

# 作成する全文検索システムの概要

それでは、PHP と Mroonga を使って、全文検索システムを作っていきましょう。以下のようなシステムを作ることにします。

**概要**

登録済みの PDF ファイルを全文検索するシステムである

**できること**

検索対象となる文書をウェブ UI で登録することができる

ウェブ UI 上のテキストフィールドを使って検索することができる

**実装方針**

ウェブ UI は PHP を使用して作成する

全文検索用のデータは MySQL に格納する

PDF からテキストを抜き出す処理は本書では扱わない（Docker イメージ付属のツールを使う）

# 第 4 章

# データベースの作成

まず、PHP を使って、MySQL にデータベースを作成します。作成の手順は通常の MySQL でのデータベース作成と同様です。Docker コンテナ内では既に MySQL が動作しており、root ユーザーでパスワード無しでログインできます。

ソースコードは以下のようになります。[*1]

リスト 4.1　create-database.php

```
<?php
define('DBNAME', 'pdfsearch');

$dbh = new PDO('mysql:host=localhost;charset=utf8', 'root', '');

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $result = $dbh->exec('CREATE␣DATABASE␣' . DBNAME);

    header("Location:␣{$_SERVER['REQUEST_URI']}");
}

$result = $dbh->query('SHOW␣DATABASES')->fetchAll();
$databases = array_map(function($row) {
    return $row[0];
}, $result);
?>
<!doctype html>
<html>
  <head>
    <title>データベースの作成</title>
  </head>
  <body>
    <h1>データベースの作成</h1>
    <?php if (! (in_array(DBNAME, $databases))): ?>
    <form method="post">
      <input type="submit" value="pdfsearchデータベースの作成">
    </form>
    <?php endif; ?>
    <h2>現在のデータベース一覧</h2>
    <ul>
    <?php foreach ($databases as $database): ?>
      <li><?php echo htmlspecialchars($database, ENT_QUOTES, 'UTF-8'); ?></li>
    <?php endforeach; ?>
    </ul>
  </body>
</html>
```

通常の MySQL に於けるデータベースと全く同じ手順でデータベースを作成していることが分かると思います。Mroonga を使う場合にも、データベースの作成には特別なことはありません。

うまくできたら、http://localhost:8080/create-database.php を表示して、実際に画面上のボタ

[*1] 限定された状況での一度きりの処理のため、褒められない書き方をしている箇所もあります。実際のアプリケーションで使用する場合には、フレームワークなどのやり方に則った適切な方法でデータベースを操作してください。

ンを押してデータベースを作成してみましょう。「現在のデータベース一覧」に「pdfsearch」というデータベースが加わるはずです。

■コラム: PHP のデバッグ

PHP が期待通りに動作しない場合や画面が真っ白になって何が悪いか分からない場合は、`docker logs` コマンドでログを確認することができます。

リスト 4.2　docker logs で PHP のログを確認

```
% docker log pdfsearch
13.03.2016 09:41:25 INFO  -- [web:php] switch :starting [:unmonitored => :starting] (
reason: monitor by user)
160313 09:41:26 mysqld_safe Can't log to error log and syslog at the same time.  Remove
all --log-error configuration options for --syslog to take effect.
160313 09:41:26 mysqld_safe Logging to '/var/log/mysql/error.log'.
160313 09:41:27 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql
[Sun Mar 13 09:41:54 2016] 172.17.0.1:58804 [200]: /create-database.php
[Sun Mar 13 09:41:55 2016] 172.17.0.1:58808 [404]: /favicon.ico - No such file or
directory
[Sun Mar 13 09:42:01 2016] PHP Parse error:  syntax error, unexpected '$databases' (
T_VARIABLE) in /var/lib/pdfsearch/create-database.php on line 14
[Sun Mar 13 09:42:01 2016] 172.17.0.1:58812 [500]: /create-database.php - syntax error,
unexpected '$databases' (T_VARIABLE) in /var/lib/pdfsearch/create-database.php on line
14
```

`-f` オプションを付け

1zw

```
% docker logs -f pdfsearch
```

と実行すると、ログが出力する度にそのログが表示されます。Docker コンテナの中では PHP のビルトインサーバーが動作しているため、ログの内容もそれに準じます。

また、ブラウザーでエラー内容を確認したい場合には、PHP ファイルの冒頭で

1zw

```
<?php
ini_set('dispaly_errors', 'stdout');
```

と設定するとよいでしょう。

# 第 5 章

# テーブルの作成

データベースが出来たので、以下のカラムを持つ pdfs テーブル作りましょう。

表 5.1　作成する pdfs テーブル

| カラム | 内容 | データ型 | 備考 |
|---|---|---|---|
| path | システム内のパス（場所） | VARCHAR(255) | 主キー。検索対象ではない |
| title | PDF 文書のタイトル | VARCHAR(255) | 検索対象 |
| content | PDF 内のテキスト | LONGTEXT | 検索対象 |

やり方はデータベースの作成と同様です。

リスト 5.1　create-table.php

```
<?php
define('DBNAME', 'pdfsearch');
define('TABLE_NAME', 'pdfs');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$dbh = new PDO($dsn, 'root', '');
$table = TABLE_NAME;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $statement = <<<STATEMENT
CREATE TABLE `{$table}` (
    id       INT PRIMARY KEY AUTO_INCREMENT,
    file     VARCHAR(255),
    title    VARCHAR(255),
    content LONGTEXT,
    FULLTEXT INDEX (title,content)
) ENGINE = Mroonga DEFAULT CHARSET utf8;
STATEMENT;
    $dbh->exec($statement);

    header("Location: {$_SERVER['REQUEST_URI']}");
}

$result = $dbh->query('SHOW TABLES')->fetchAll();
$tables = array_map(function($row) {
    return $row[0];
}, $result);

?>
<!doctype html>
<html>
  <head>
    <title>テーブルの作成</title>
  </head>
  <body>
    <h1>テーブルの作成</h1>
    <?php if (! (in_array(TABLE_NAME, $tables))): ?>
    <form method="post">
```

```
      <input type="submit" value="pdfsテーブルの作成">
    </form>
    <?php endif; ?>
    <h2>現在のテーブル一覧</h2>
    <ul>
    <?php foreach ($tables as $table): ?>
      <li><?php echo htmlspecialchars($table, ENT_QUOTES, 'UTF-8') ?></li>
    <?php endforeach; ?>
    </ul>
  </body>
</html>
```

http://localhost:8080/create-table.php にアクセスして pdfs テーブルを作成してみましょう。事前に pdfsearch テーブルを作っておかないとエラーになるので注意してください。

データベース作成時と違って、Mroonga を使ったテーブルを作る時にはいくつか注意点があります。

一番大事なのは、CREATE TABLE 文の最後に、ストレージエンジンとして Mroonga を指定することです。

1zw

```
CREATE TABLE (
-- ...
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

普段は InnoDB や MyISAM を指定しているところを切り替えるだけなので、難しいことはないでしょう。

次に、インデックスの作成方法が InnoDB などと異なります。PDF のタイトルや内容を完全一致で検索することはまずないでしょうから、通常のインデックスは必要ありません。代わりに、全文検索のための特別なインデックスを作る、FULLTEXT INDEX 構文を使用します。これは、MySQL に元々備わっている、全文検索機能のための構文です。Mroonga でも用途は同じなので、全く同じ構文で使用できます。

それ以外は、通常の MySQL 使い方と同じです。主キーにはパスといった意味のある物ではなく、ただレコードを識別するためだけのサロゲートキーを使う人もいることでしょう。それでも構いません。

# 第 6 章

# データ入力機能の作成

いよいよアプリケーションらしい所を作っていきます。始めから検索機能に着手したいところですが、データがないことには何もできないので、入力機能から作成します。

PDF ファイルの内容を MySQL に登録するために、以下のようなステップを踏むことになります。

1. PHP によるウェブ UI から一時ディレクトリーにアップロードする
2. PDF ファイルからタイトルと本文テキストを抽出する
3. タイトルと本文を MySQL の適切なカラムに挿入する

## 6.1 PDF のアップロード機能を作る

まずは PDF のアップロード機能を作りましょう。PDF ファイルその物は後で使うことはないので、一時ディレクトリーに置いたままで構いません。

ユーザーが表示させる起点の画面を index.php、そのファイルから呼び出すアップロード機能の部分を upload.php として実装することにします。

リスト 6.1 index.php

```
<?php
require_once __DIR__ . '/upload.php';

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
}

?><!doctype html>
<title>PDF Search</title>
<h1>PDF Search</h1>

<h2>アップロードされたファイル</h2>
<?php if (! empty($uploads)): ?>
<?php foreach ($uploads as $upload): ?>
<p><?php echo h($upload->getName()); ?></p>
<?php endforeach; ?>
<?php endif; ?>

<form enctype="multipart/form-data" method="post">
  <input name="pdf[]" type="file" multiple>
  <input type="submit">
```

```
</form>
```

リスト 6.2　upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    protected $name;
    protected $tmpName;

    public static function fromFilesInfo(array $info)
    {
        $uploads = array();
        $fileCount = count($info['name']);
        for ($i = 0; $i < $fileCount; $i++) {
            $name = $info['name'][$i];
            $tmpName = $info['tmp_name'][$i];
            $uploads[] = new Upload($name, $tmpName);
        }
        return $uploads;
    }

    public function __construct($name, $tmpName)
    {
        $this->name = $name;
        $this->tmpName = $tmpName;
    }

    public function getName()
    {
        return $this->name;
    }
}
```

今の所、単にアップロードされたファイルの名前を表示しているだけです。後でその他の機能を実装していきます。複数のファイルをアップロードすると[*1]、全てが表示されることも確認してみてください。

[*1] ファイル選択ダイアログで Ctrl キーを押しながらファイルをクリックしていくことで、選んだファイルを同時にアップロードできます。また、Shift を押しながら最初のファイルと最後のファイルをクリックすることで、その間にあるファイルをすべて選択状態にできます。


図 6.1　ファイルアップロード画面

紙幅を減らすため、細かな検証は省略しています。実際に運用するアプリケーションでは安全性やエラーのチェックなどをしっかりと行ってください。

## 6.2　PDF からの情報を抽出する

次に、PDF ファイルからタイトルと本文を抜き出す処理を実装します。これを行うにはいくつか方法がありますが、ここでは php-poppler というライブラリーを使うことにします。php-poppler を使うと PDF を解析してタイトルや本文を抜き出すことができます[*2]。

php-poppler は Poppler 名前空間にいくつかのクラスを定義しており、以下のようにして使います。

1zwphp-poppler の基本的な使い方

```
<?php

// pdfinfoコマンドのラッパー
$pdfinfo = \Poppler\Driver\Pdfinfo::create();
// pdftotextコマンドのラッパー
$pdftotext = \Poppler\Driver\Pdftotext::create();
// pdftohtmlコマンドのラッパー
$pdftohtml = \Poppler\Driver\Pdftohtml::create();

$pdf = new \Poppler\Processer\Pdffile($pdfinfo, $pdftotet, $peftohtml);

// PDFからタイトルや著者、作成日時などの情報を連想配列として抜き出す
echo $pdf->getInfo('path/to/document.pdf');
// PDFから本文テキストを抜き出す
```

[*2] C 製の PDF ライブラリーで Poppler という物があり、PDF を扱う様々なコマンドラインツールの提供もしています。php-poppler はこれらのコマンド呼び出しをラップして PHP から扱いやすくした物です。

```
echo $pdf->toText('path/to/document.pdf');
```

HTML 関連の機能を使わないとしても、\Poppler\Processer\Pdffile のコンストラクターには\Poppler\Driver\Pdftohtml のインスタンスを渡す必要があるので注意してください。

Docker イメージの中には、既に Poppler と php-poppler がインストール済みです。php-poppler は Composer でインストールしているので、vendor/autoload.php を読み込むことで、オートロードが有効になります。

これを使って、先ほど作った\PDFSearch\Upload クラスに PDF 関連の機能を追加します。

リスト 6.3　upload.php

```
<?php
namespace PDFSearch;

class Upload
{
    protected $name;
    protected $tmpName;

    public static function fromFilesInfo(array $info)
    {
        $uploads = array();
        $fileCount = count($info['name']);
        for ($i = 0; $i < $fileCount; $i++) {
            $name = $info['name'][$i];
            $tmpName = $info['tmp_name'][$i];
            $uploads[] = new Upload($name, $tmpName);
        }
        return $uploads;
    }

    public function __construct($name, $tmpName)
    {
        $this->name = $name;
        $this->tmpName = $tmpName;
    }

    public function getName()
    {
        return $this->name;
    }
}
```

実際に PDF ファイルをアップロードして、タイトルなどの情報が取得できているか確認しましょう。


図 6.2　PDF 内の情報を取得

## 6.3　データベースにデータを挿入する

それでは、いよいよ Mroonga にデータを入力しましょう。と言っても、いつも通り PDO クラスで INSERT 文を実行するだけです。Mroonga の方で自動的に検索用のインデックスを作成してくれます。Docker イメージ内に既に MySQL 用のアダプターはインストールされていますので、単に PHP から呼び出すだけで使用できます。

データベースを扱う\PDFSearch\Table クラスを実装し、index.php でそのクラスを使うようにしたのが以下の内容です。\PDFSearch\Upload には変更がないため省略しています。

リスト 6.4　index.php

```
<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
require_once __DIR__ . '/table.php';

define('DBNAME', 'pdfsearch');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$table = new \PDFSearch\Table($dsn, 'root', '');

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
    try {
        foreach ($uploads as $upload) {
            $tableb->insert($upload);
        }
        header('Location: .');
    } catch (\RuntimeException $e) {
        header('HTTP', true, 500);
        header('content-type: text/plain');
        echo $e;
    }
```

```
    exit;
}

$records = $table->records();

?><!doctype html>
<title>PDF Search</title>
<h1>PDF Search</h1>

<h2>登録済みPDF一覧</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($records as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
    </tr>
  <?php endforeach; ?>
</table>

<form enctype="multipart/form-data" method="post">
  <input name="pdf[]" type="file" multiple>
  <input type="submit">
</form>
```

リスト 6.5　table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT␣INTO␣`pdfs`␣(file,␣title,␣content)
␣␣␣␣␣␣␣␣VALUES(:file,␣:title,␣:content);";
    const SELECT = "SELECT␣*␣FROM␣`pdfs`␣ORDER␣BY␣`id`;";

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file'    => $upload->getName(),
            ':title'   => $upload->getTitle(),
            ':content' => $upload->getContent()
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
    }

    public function records()
    {
        $sth = $this->pdo->prepare(self::SELECT);
        $isSucceeded = $sth->execute();
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }
}
```

ウェブブラウザーでアクセスして、データベースの中身を表示してみましょう。


図 6.3　アップロードした PDF の情報を表示

# 第 7 章

# 全文検索機能の作成

ようやく、全文検索機能の実装です。実は、これも Mroonga ならではの特別なことは必要なく、通常の MySQL の全文検索機能、即ち `MATCH() ... AGAINST` 構文を使用するだけです。テーブル作成時に、ストレージエンジンに Mroonga を選択しておけば、他の操作は通常の MySQL と同じように実行できるようにデザインされているのです。

`MATCH() ... AGAINST` 構文は元々 MySQL に備わっている全文検索用の構文です。`MATCH` には検索対象にするカラム名を渡します。ここで指定できるのは、テーブル作成時に `FULLTEXT` インデックスを指定したカラムのみです。Docker 内のテーブルでは、予め `title` カラムと `content` カラムに `FULLTEXT` インデックスを張っています。`AGAINST` には検索クエリーを渡します。オプションを渡すこともできます。詳細は MySQL のドキュメントを参照してください。

Mroonga ではこの構文を流用しており、元の挙動を知っている人にはおおよそ期待通りに動作するようになっていますが、厳密に同じではありません。Mroonga でのみ使える様々な機能が使えるようになっています。これについては、一旦基本の検索機能を作ったあとで触れたいと思います。

今の「登録済み PDF 一覧」の上に検索フォームを作成し、検索を実行した場合にはフォームの下に検索結果を表示するようにしてみましょう。そのために、`\PDFSearch\Table` クラスに検索用のメンバー関数を追加します。

リスト 7.1　search.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT␣INTO␣‘pdfs‘␣(file,␣title,␣content)
␣␣␣␣␣␣␣␣VALUES(:file,␣:title,␣:content);";
    const SELECT = "SELECT␣*␣FROM␣‘pdfs‘␣ORDER␣BY␣‘id‘;";
    const SEARCH = <<<EOS
SELECT * FROM ‘pdfs‘ WHERE MATCH(content) AGAINST(:query IN BOOLEAN MODE);
EOS;

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file'    => $upload->getName(),
            ':title'   => $upload->getTitle(),
            ':content' => $upload->getContent()
```

```
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
    }

    public function records()
    {
        $sth = $this->pdo->prepare(self::SELECT);
        $isSucceeded = $sth->execute();
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }

    public function search($query)
    {
        $sth = $this->pdo->prepare(self::SEARCH);
        $params = [':query' => $query];
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }
}
```

SEARCH 定数と search メンバー関数を追加しています。

先ほど言ったような検索フォームと結果のエリアを追加すると、index.php はこうなります。

リスト 7.2　index.php

```
<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
require_once __DIR__ . '/table.php';

define('DBNAME', 'pdfsearch');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$table = new \PDFSearch\Table($dsn, 'root', '');

function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

// 検索処理
$searchQuery = null;
if ($_SERVER['REQUEST_METHOD'] === 'GET' &&
    array_key_exists('q', $_GET) && $_GET['q']) {
    $searchQuery = $_GET['q'];
    $searchResult = $table->search($searchQuery);
}

// ファイルアップロード処理
if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
    try {
        foreach ($uploads as $upload) {
            $tableb->insert($upload);
        }
        header('Location: .');
    } catch (\RuntimeException $e) {
        header('HTTP', true, 500);
        header('content-type: text/plain');
        echo $e;
    }
    exit;
}

$records = $table->records();
```

```
?><!doctype html>
<title>PDF Search</title>
<h1>PDF Search</h1>

<h2>検索</h2>
<form method="get">
  <input name="q" type="search" value="<?php␣echo␣h($searchQuery);␣?>">
  <input type="submit" value="検索">
</form>

<?php if ($searchResult && count($searchResult) > 0): ?>
<h2>「<?php echo h($searchQuery); ?>」を含むPDF</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($searchResult as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
    </tr>
  <?php endforeach; ?>
</table>
<?php endif; ?>

<h2>登録済みPDF一覧</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($records as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
    </tr>
  <?php endforeach; ?>
</table>

<form enctype="multipart/form-data" method="post">
  <input name="pdf[]" type="file" multiple>
  <input type="submit">
</form>
```

SQL 一文で済むので簡単ですね。では、検索してみましょう。

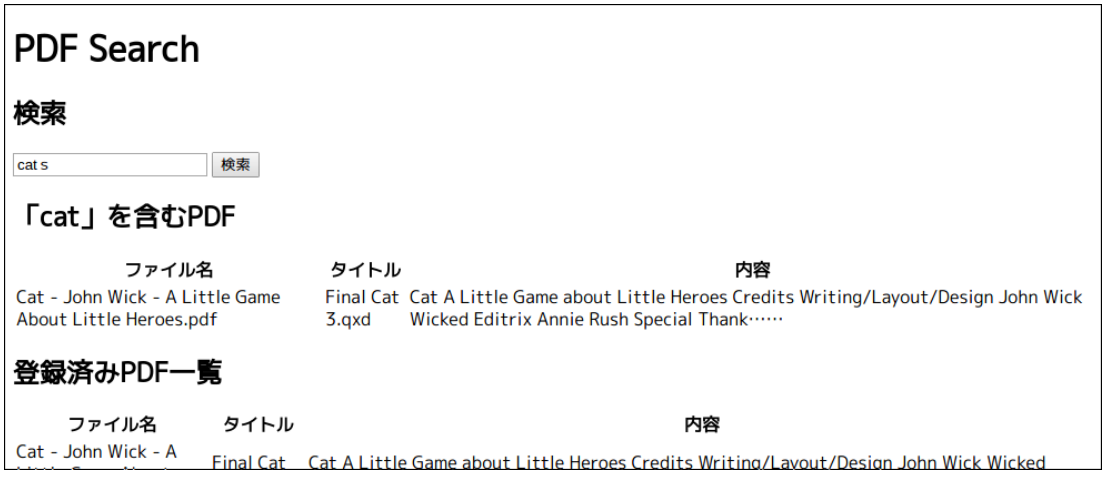


図 7.1　全文検索結果の表示

以上で、基本的な機能は完成です。以降では、Mroonga に特徴的な幾つかの機能を紹介し、実装していきます。

# 第 8 章

# Mroonga ならではの機能の作成

ここからは、MySQL の全文検索機能にはない、Mroonga ならではの特徴を活かした機能を実装していきます。

## 8.1 検索語のハイライト

現在の検索機能では、検索結果中、どこに検索語があるのか、自分で探さなくてはなりません。次の画像のように、検索語だけがハイライトされていると、とても見やすくなります。また、猫についての PDF を探すつもりで「cat」で検索し「application」といった関係ない語がヒットした場合に、すぐにそのことに気付けるというメリットもあります。


図 8.1　検索語をハイライトして表示

このように検索語をハイライトする機能を実装してみましょう。

すぐに思い付く実装方法は、取得した検索結果の content カラムを、再度 PHP で解析してタグを挿入することですが、実は Mroonga には、このための機能が備わっています。SQL 中で利用可能な mroonga_snippet_html() 関数です。これは特定のカラム（ここでは content）のうち、指定した語（「cat」）の前後数十バイトを抜き出し、文字列として返す関数です。

mroonga_snippet_html() 関数が返すスニペットは ... というタグで囲まれ、更に検索語は...で囲まれます[*1]。一レコード中に検索語が複数回現れる場合は、その数に応じて複数のスニペットを結合した一つの文字列を返します。余談ですが「スニペット」は「断片」という意味で、この場合は content カラムのうちの一部を返すことを意味しています。

mroonga_snippet_html() は以下のようにして使用します。

1zwmroonga_snippet_html の簡単な使い方

```
SELECT mroonga_snippet_html(content, 'cat') FROM `pdfs`;
```

結果は、例えば次のようになります[*2]。

1zwmroonga_snippet_html() の結果例

```
<div class="snippet"><span class="keyword">Cat</span>

A Little Game about Little Heroes

Credits

Writing/Layout/Design
John Wick
Wicked Editrix
Annie Rush
Special Thanks
to Jared Sorensen
For helping me throw out the (litter) box.
Extra Special Tha</div>
<div class="snippet">Still
Who protects little heroes.

http://www.wicked-dead.com/<span class="keyword">cat</span>

<span class="keyword">Cat</span> uses the Advantage System.
http://www.wicked-dead.com/advantage

<span class="keyword">Cat</span>: A Little Game about Little Heroes is ©
 and ™ 2004 by Joh</div>
<div class="snippet"> Except boggins. Those are real.
Go give your <span class="keyword">cat</span> a hug.

T a b l e o f C o n t e n t s
“I Know I’m Dreaming”
Introduction
What You Need
Making a <span class="keyword">Cat</span> Character
Step 1: The Traits
Step 2: Names
St</div>
```

ここでは、本書のアプリケーションに必要なことのみ説明しているので、詳細については公式ドキュメント（5.5.5. mroonga_snippet_html()）を参照してください。

現在の実装では検索でヒットした時に content カラムの抜粋を表示していますが、mroonga_snippet_html() の戻り値で置き換えてみましょう。PDFSearch\Table::search() で

[*1] タグや class 属性を変更したい場合は、柔軟性の高い mroonga_snippet() 関数を使用してください。

[*2] 読みやすさのため、改行を調整しています。

使用している SQL（PDFSearch\Table::SEARCH 定数）を変更します。mroonga_snippet_html() も COUNT() などと同様の関数なので、AS 句を使って別名を付けることができます。

リスト 8.1　table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT␣INTO␣‘pdfs‘␣(file,␣title,␣content)
␣␣␣␣␣␣␣␣VALUES(:file,␣:title,␣:content);";
    const SELECT = "SELECT␣*␣FROM␣‘pdfs‘␣ORDER␣BY␣‘id‘;";
    const SEARCH = <<<EOS
SELECT file, title, mroonga_snippet_html(content, :query) AS snippets
FROM ‘pdfs‘ WHERE MATCH(content) AGAINST(:query IN BOOLEAN MODE);
EOS;

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file'    => $upload->getName(),
            ':title'   => $upload->getTitle(),
            ':content' => $upload->getContent()
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
    }

    public function records()
    {
        $sth = $this->pdo->prepare(self::SELECT);
        $isSucceeded = $sth->execute();
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }

    public function search($query)
    {
        $sth = $this->pdo->prepare(self::SEARCH);
        $params = [':query' => $query];
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }
}
```

HTML 中で content カラムを表示していた所を、snippets に置き換えます。

リスト 8.2　index.php

```
<?php
// Dockerイメージ内にComposerでインストール済みのライブラリーを読み込む
require_once 'vendor/autoload.php';
require_once __DIR__ . '/upload.php';
require_once __DIR__ . '/table.php';

define('DBNAME', 'pdfsearch');

$dsn = 'mysql:host=localhost;dbname=' . DBNAME . ';charset=utf8';
$table = new \PDFSearch\Table($dsn, 'root', '');
```

```
function h($string, $flags = ENT_QUOTES, $encoding = 'UTF-8')
{
    return htmlspecialchars($string, $flags, $encoding);
}

// 検索処理
$searchQuery = null;
if ($_SERVER['REQUEST_METHOD'] === 'GET' &&
    array_key_exists('q', $_GET) && $_GET['q']) {
    $searchQuery = $_GET['q'];
    $searchResult = $table->search($searchQuery);
}

// ファイルアップロード処理
if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    array_key_exists('pdf', $_FILES)) {
    $uploads = \PDFSearch\Upload::fromFilesInfo($_FILES['pdf']);
    try {
        foreach ($uploads as $upload) {
            $table->insert($upload);
        }
        header('Location: .');
    } catch (\RuntimeException $e) {
        header('HTTP', true, 500);
        header('content-type: text/plain');
        echo $e;
    }
    exit;
}

$records = $table->records();

?><!doctype html>
<title>PDF Search</title>
<style>
  .keyword {
    font-weight: bold;
    color: red;
  }
</style>
<h1>PDF Search</h1>

<h2>検索</h2>
<form method="get">
  <input name="q" type="search" value="<?php echo h($searchQuery); ?>">
  <input type="submit" value="検索">
</form>

<?php if ($searchResult && count($searchResult) > 0): ?>
<h2>「<?php echo h($searchQuery); ?>」を含むPDF</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($searchResult as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo $record['snippets']; ?></td>
    </tr>
  <?php endforeach; ?>
</table>
<?php endif; ?>

<h2>登録済みPDF一覧</h2>
<table>
  <tr>
    <th>ファイル名</th>
    <th>タイトル</th>
    <th>内容</th>
  </tr>
  <?php foreach ($records as $record): ?>
    <tr>
      <td><?php echo h($record['file']); ?></td>
      <td><?php echo h($record['title']); ?></td>
      <td><?php echo h(mb_substr($record['content'], 0, 120)); ?>&hellip;&hellip;</td>
    </tr>
```

```
  <?php endforeach; ?>
</table>

<form enctype="multipart/form-data" method="post">
  <input name="pdf[]" type="file" multiple>
  <input type="submit">
</form>
```

snippets の表示では HTML エスケープをしていないことに気が付いたでしょうか。mroonga_snippet_html() に HTML タグが含まれているため、ここで HTML エスケープしてしまうと、タグとしての役を果たさなくなってしまいます。そのため、直接出力しています。

ここにはセキュリティ上の懸念があります[*3]。mroonga_snippet_html() の戻り値では、カラム中の HTML タグなどは全てエスケープされますが、取得したスニペットと他の文字列を PHP で結合すると、セキュリティホールになり得ます。加工せず、可能であれば単独で出力するようにしてください。加工や他の文字列との結合が必要な場合は、慎重に行ってください。

これで、検索結果のハイライトも出来ました。実際に検索して結果を楽しみましょう。

## スコアによる並び替え

これまでは、検索結果の並び順については考慮していませんでした。やはり、検索語に近い PDF が先頭に並んでほしいものです。

この「検索語に近い」または「より望ましい」という状態を、Mroonga ではスコアという形で、数値として計算することができます。「検索語がより多く含まれている PDF の方が望ましい」「検索語が本文に含まれているよりもタイトルに含まれている方がより望ましい」といった判断が可能になるのです。

Mroonga が使っているスコアの値は、WHERE 句に指定している MATCH()...AGAINST() の戻り値として取得できます。AS で別名を付けると使いやすいでしょう。

1zw スコアの取得

```
SELECT MATCH(content) AGAINST('cat' IN BOOLEAN MODE) as score FROM `pdfs`;
```

1zw

```
+-------+
| score |
+-------+
|   315 |
|    10 |
|     0 |
|     0 |
|     4 |
+-------+
5 rows in set (0.47 sec)
```

このスコアでソートすることで、より望ましい結果を上の方に表示することができます。

では、このスコアというのは何によって決められているのでしょうか。一つは、結果文書の中に踏まれている検索語の数です。検索語が多く含まれている文書ほど検索する人の要求に合っていると見做して、検索上位に表示させられるわけです。

以下の用に PDFSearch\Table::SEARCH の SQL にちょっとした変更を加えるだけで、多くの場合、期待される検索結果になるでしょう。

[*3] 実際には HTML を正しく出力するために必要な注意点です。仮にセキュリティ上問題とならなくても、HTML が正しく出力されず、アプリケーションが壊れてしまう可能性があります。

1zw 変更前の\PDFSearch\Table::SEARCH 定数

```
SELECT file, title, mroonga_snippet_html(content, :query) AS snippets
FROM `pdfs` WHERE MATCH(content) AGAINST(:query IN BOOLEAN MODE);
```

1zw 変更前の\PDFSearch\Table::SEARCH 定数

```
SELECT file, title, mroonga_snippet_html(content, :query) AS snippets,
MATCH(content) AGAINST(:query IN BOOLEAN MODE) AS score
FROM `pdfs` WHERE MATCH(content) AGAINST(:query IN BOOLEAN MODE)
ORDER BY score DESC;
```

もう一つの基準は重みと呼ばれます。重みは、どのカラムをより重視するかという指標です。例えば、次のような二つの PDF が登録されているとします。

表 8.1　Groonga という語を含む二つの PDF

| タイトル | 内容 |
|---|---|
| **Groonga** で作る全文検索システム | **Groonga** を使って全文検索システムを作ります。 |
| わたしのエッセイ集 | 今日、**Groonga** という言葉を耳にしました。不思議な言葉ですね、**Groonga**。検索してみる |

Groonga のことが知りたくて検索する時に、どちらがヒットしてほしいでしょうか。始めの「Groonga で作る全文検索システム」の方ではないでしょうか。でも、「Groonga」という言葉が多く含まれているのは「わたしのエッセイ集」の方なので、今までの実装だと、こちらが上位に表示されることになってしまいます。前者を上位に持って来るために、「検索語がタイトルに含まれている方が、本文に含まれているよりも 100 倍くらい重要だ」という指標を導入することにしましょう。Mroonga または全文検索の言葉で「`title` カラムの重みを `content` カラムの 100 倍にする」ということになります。


図 8.2　検索語がタイトル含まれている場合スコアが大きくなる

上の画像にある三つの PDF ではどれも本文中に「すべて」という語が含まれますが、タイトルにも含まれる「mruby のすべて」のスコアが、含まれない「API デザインケーススタディ」「Docker エキスパート養成読本」に比べ、非常に大きくなっていることが分かります。実際、タイトルの重視をやめると、「mruby のすべて」と「API デザインケーススタディ」の順位は逆転します。

このように重み付けするには、`W` プラグマを使用します。

1zw

```
MATCH(title,content) AGAINST('*W1:100,2:1␣API')
```

W が重み付けのためのプラグマ使用を指示し、その後はカラムごとの重み付けをカンマ区切りで指定します。「1:100」は「(MATCH で指定した) 1 カラム目の重みを 100 に」、「2:1」は「2 カラム目の重みを 1 に」という命令になります。これにより、「検索語が 1 つタイトルに含まれていると、本文に 100 個含まれているのと同等のスコアになる」ということを実現しています (実際には 100 倍の差が付きさえすればいいので、2 と 200 でも、10 と 1000 でも構いません)。

まとめると、実装は次のようになります。

リスト 8.3　table.php

```
<?php
namespace PDFSearch;

class Table
{
    const INSERT =
        "INSERT␣INTO␣`pdfs`␣(file,␣title,␣content)
␣␣␣␣␣␣␣␣VALUES(:file,␣:title,␣:content);";
    const SELECT = "SELECT␣*␣FROM␣`pdfs`␣ORDER␣BY␣`id`;";
    const SEARCH = <<<EOS
SELECT file, title, mroonga_snippet_html(content, :query AS query) AS snippets,
MATCH(title,content) AGAINST(:query_with_pragma IN BOOLEAN MODE) AS score
FROM `pdfs` WHERE MATCH(title,content) AGAINST(:query_with_pragma IN BOOLEAN MODE)
ORDER BY score DESC;
EOS;

    protected $pdo;

    public function __construct($dsn, $username, $password)
    {
        $this->pdo = new \PDO($dsn, $username, $password);
    }

    public function insert(Upload $upload)
    {
        $params = [
            ':file'    => $upload->getName(),
            ':title'   => $upload->getTitle(),
            ':content' => $upload->getContent()
        ];
        $sth = $this->pdo->prepare(self::INSERT);
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
    }

    public function records()
    {
        $sth = $this->pdo->prepare(self::SELECT);
        $isSucceeded = $sth->execute();
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }

    public function search($query)
    {
        $sth = $this->pdo->prepare(self::SEARCH);
        $params = [
          ':query' => $query,
          ':query_with_pragma' => '*D+W1:100,2:1␣' . $query
        ];
        $isSucceeded = $sth->execute($params);
        if (! $isSucceeded) {
            throw new \RuntimeException(implode(PHP_EOL, $sth->errorInfo()));
        }
        return $sth->fetchAll();
    }
}
```

W プラグマを色々な値に変えて試してみてください。本書のようなチュートリアルではなく、実際の全文検索システムに置いてもスコアリングは非常に大切なポイントです。重み付けを使いこなしたスコアリングができると、いかにも全文検索システムを作っているという実感が出て、楽しいと思います。

- （D+ プラグマはもっと前でやる）

# 付録 A

# 付録

## A.1　Mroonga のインストール

Mroonga を使った全文検索システムに興味が出て来たら、ぜひ自分でインストールしてみてください。以下に、典型的なインストール方法をご案内します。

### Debian/GNU Linux

Groonga プロジェクト（？）は、本書執筆時点で Debian/GNU Linux jessie の APT 用のリポジトリーを持っており、Mroonga もそこからインストールできます。サンプルで用いた Docker イメージでもこのパッケージを使用しています。

まず、以下の内容で/etc/apt/sources.list.d/groonga.list というを作成し、ソースを登録します。

1zw/etc/apt/sources.list.d/groonga.list

```
deb http://packages.groonga.org/debian/ jessie main
deb-src http://packages.groonga.org/debian/ jessie main
```

次にインストールします。

1zw

```
% sudo apt-get update
% sudo apt-get install -y --allow-unauthenticated groonga-keyring
% sudo apt-get update
% sudo apt-get install -y -V mysql-server-mroonga
```

以上でインストールは終了です。

実際にインストールする際には細部が変わっている可能性もあるため、一度以下の公式ドキュメントを確認しましょう。

http://mroonga.org/ja/docs/install/debian.html

### Windows、OS X、Ubuntu、CentOS

Windows 用にも公式に Mroonga のインストーラー及びコンパイル済みパッケージの ZIP アーカイブが提供されています。OS X の Homebrew、Ubuntu の APT、CentOS の Yum 用にはリポジトリーが提供されています。

いずれも公式サイトにインストール方法がありますので、参照してください。

http://mroonga.org/ja/docs/install.html

### Docker

Mroonga の Docker イメージも Docker Hub から配布されています。以下のように `docker run` コマンドを実行することで、自動的にイメージの取得からコンテナの起動までが行われます。

1zwMroonga の Docker イメージの取得

```
% docker run --detach --publish=3306:3306 groonga/mroonta
```

これでポート番号 3306 をリスンしながら Mroonga インストール済みの MySQL サーバーが起動します。通常の MySQL クライアントを使って接続が可能です。

1zwMroonga への接続

```
% mysql --host=127.0.0.1 --port=3306 --user=root
```

（Linux 以外は--host が違いそう）

但し、Docker コンテナは、削除と同時にデータが失われてしまいます。これは多くの場合望ましい挙動ではないでしょう。次のように `volume` オプションを追加することで、ホスト上のディレクトリーをコンテナと共有できます。

1zwvolume オプションの指定

```
% docker run --detach --publish=3306:3306 --volume=/tmp/mroonga:/var/lib/mysql groonga/
mroonga
```

これで、 ホスト側の/tmp/mroonga 以下に MySQL のデータファイルが作成されるようになります。

Mroonga を使ったテーブルの作成については、「A.2 データベースとテーブルの作成」を参照してください。

また、Mroonga の Docker イメージは、MySQL と Mroonga それぞれのバージョンの組み合わせごとに多数のイメージが作られています。入手可能な組み合わせは Docker Hub の Mroonga のページで確認できます。

https://hub.docker.com/r/groonga/mroonga/

## A.2 データベースとテーブルの作成

データベースの作成には、通常の MySQL と変わることは何もありません。MySQL サーバーに接続し、`CREATE DATABASE` 文を実行することでデータベースを作成します。

1zw データベースの作成

```
CREATE DATABASE pdfsearch;
```

必要に応じて権限設定などを行ってください。

テーブルの作成には気を付けることがあります。まず、ストレージエンジンを Mroonga にする必要があります。MySQL で何も指定せずに `CREATE TABLE` を実行した場合、ストレージエンジンはデフォルトの InnoDB になります。これを他のストレージエンジンに変更するには、`ENGINE` オプションを使用します。

1zw テーブルのストレージエンジンに Mroonga を指定

```
CREATE TABLE `pdfs` (
-- カラム定義等
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

また、検索対象にしたいカラムに、（通常のインデックスとは異なる）全文検索用んインデックスを作成する必要があります。

1zw 全文検索インデックスの追加

```
ALTER TABLE `pdfs` ADD FULLTEXT INDEX (titke, content);
```

これ自体は通常の MySQL での操作なので、ALTER TABLE のほか、CREATE TABLE でインデックスを作成することもできます。

1zw テーブル作成時の全文検索インデックスの作成

```
CREATE TABLE `pdfs` (
    id      INT PRIMARY KEY AUTO_INCREMENT,
    file    VARCHAR(255),
    title   VARCHAR(255),
    content LONGTEXT,
    FULLTEXT INDEX (title, content)
) ENGINE = Mroonga DEFAULT CHARSET utf8;
```

これは、本書で使用した Docker イメージを作る際に、実際に使用した SQL 文です。

また、SQL を使わず phpMyAdmin の UI でテーブルを作成することもあるかも知れません。Mroonga がインストール済みであれば、phpMyAdmin でもストレージエンジンの選択肢に Mroonga が追加されていることを確認しているので、同じように Mroonga を使い始めることができます。

## A.3 Mroonga のコミュニティ、サポート

Mroonga を含む Groonga については、開発者が日本人であることもあり、日本語で気軽に問い合わせられます。公式サイトにはメーリングリスト、チャット、Twitter、Facebook ページについて紹介されているので、質問や要望がある際にはぜひ活用してください。

http://mroonga.org/ja/docs/community.html

また、開発や運用などの有償のサポートサービスもあります。業務で使用する場合には検討するといいでしょう。

http://groonga.org/ja/support/

## A.4 Docker コンテナの作り直し

本書のサンプルでは Docker コンテナを使って Apache、PHP、MySQL を動かしています。時には始めからやり直したくなることもあるかも知れません。そうした場合は、以下のように一度コンテナを削除して作り直すことができます。

1zw コンテナの削除

```
% docker rm pdfsearch
```

1zw コンテナの作成

```
% docker run --detach --name=pdfsearch --publish=8080:80 \
    --volume=$PWD:/var/lib/pdfsearch kitaitimakoto/grnbook-mroonga
```

コンテナを削除すると、データベース内のデータは全て失われます。PDF の登録からやり直しとなるので注意しましょう。ホストと共有している PHP ファイルは消えることはありません。

## A.5　phpMyAdmin によるデータの確認・操作

本書の Docker イメージには phpMyAdmin もインストールされており、`http://localhost:8080/phpmyadmin` でアクセスできます。名前が `root`、パスワードはなし、というユーザーが登録されており、全権限を所有しています（PHP で使用しているユーザーと同じです）。

これを使って実際に入っているデータを確認・削除したり、コンテナはそのままでデータベースやテーブルの再作成を行ったりできます。テーブル作成時はストレージエンジンに Mroonga を指定することを忘れないようにしてください。詳細は「A.2 データベースとテーブルの作成」で解説しています。

## A.6　PHP のデバッグ

本書サンプルで作成する Docker コンテナの中では Apache の mod_php モジュールを使って PHP 5.6 が動作しています。もし、PHP スクリプトがうまく動作しない場合は、Apache のエラーログを確認するといいでしょう。

まず、`docker exec` コマンドを使ってコンテナの中に入ります（正確にはコンテナの中で bash を実行します）。

1zwdocker exec でコンテナの中に入る

```
% docker exec --interactive --tty pdfsearch /bin/bash
```

Apache のエラーログは/var/log/apache2/error_log なので、これを `less` や `tail` などのコマンドで見ながら PHP を実行することで、どんなエラーが起きているのかを知ることができます。

1zw ログファイルを監視する

```
# less /var/log/apache2/error_log
# tail -f /var/log/apache2/error_log
```