MySQL/PostgreSQL
サーバー運用者向け
基礎まとめ

2018-04-08 : @lf_ur_ : uraku

対象者

- 主にサーバー（OS・各種ミドルウェア）の構築・運用をする人。MySQLのインストールや既存dumpデータの流し込み、mysqldump等はするが、SQLはあまり叩いたことがない方。

- スペックアップ以外の方法でMySQL（やPostgreSQL）の高速化を頼まれ困っている人。

話す事

- データベースでどういうことができるのか。
- どういう処理が重くなるのか。
- それに対してどういう対処法があるか。
- MySQL 5.x / PostgreSQL 9.xを扱う上で知っておいて欲しいこと。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

1. RDBMSとは

- MySQLだったりPostgreSQLだったりOracleだったりSQL ServerはRDBMSと呼ばれる。

- リレーショナルデータベースのマネジメントシステム。

- リレーション: 関係、繋がり。

- 複数の表（テーブル）として保存したデータをリレーションで繋でデータを管理・処理するデータベース。

- 実際はもっと細かい話もあるけど気にしない方向で……

1. RDBMSとは

staffsテーブル

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

locationsテーブル

location_id	location_name	address
1	名古屋事務所	愛知県xxx..
2	神戸事務所	兵庫県xxx..

データの分け方：正規化

1. RDBMSとは

staffsテーブル

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

カラム（列）のデータ型
数値 int etc…
文字列 varchar, text, etc…
日付 date, datetime, etc…

locationsテーブル

location_id	location_name	address
1	名古屋事務所	愛知県xxx..
2	神戸事務所	兵庫県xxx..

RDBではないDB

- RDBMSは**SQL**というクエリ言語で操作したりデータを取り出したりする。

- RDBではないデータベースはSQLを使わないこと(が多い事)から**NoSQL**とまとめて呼ばれたり呼ばれなかったりする。

  - KVS（Key-Value Store）： Redis, Cassandra など

  - Document DB : MongoDB など

  - 時系列DB, etc…

DBの使われ方

- Webアプリケーションでは、各種データの保存場所に使われる（永続化）。
- 例えばブログなら
  - ログインアカウント情報のテーブル
  - カテゴリ情報のテーブル
  - 投稿した記事内容のテーブル

話す事

1. RDBMSとは
2. **mysqlコマンドとpsqlコマンドの使い方**
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

2. mysqlコマンドと psqlコマンドの使い方

- 実際にOSSのRDBMSではよく使われるMySQLとPostgreSQLの使い方を軽くおさらいします。

mysqlコマンドの使い方

mysqlコマンドでMySQLに接続

```
$ mysql -u <USER> -p -h <HOST> <DB-NAME>
```

-u ユーザー名指定

-p ユーザーが要パスワードの場合指定

-h ホスト名指定（自ホストの場合は不要）

-P ポート番号指定が必要な場合は使う。大文字であることに注意。

<DBNAME> 利用するDB名。ログイン後に後から指定可

mysqlコマンドの使い方

データベースの一覧を表示

```
mysql> show database;
```

操作するデータベースを指定（切り替え）

```
mysql> use <DB-NAME>;
```

mysqlコマンドの使い方

データベース内テーブル一覧を表示

```
mysql> show tables;
```

テーブルの詳細を表示

```
mysql> desc <TABLE-NAME>;
```

テーブル作成時の定義を表示

```
mysql> show create tables <TABLE-NAME>;
```

MySQLとの接続を終了する

```
mysql> quit;
```

psqlコマンドの使い方

psqlコマンドでPostgreSQLに接続

```
$ psql -U <USER> -h <HOST> <SCHEMA-NAME>
```

-U ユーザー名指定。大文字であることに注意。

※ パスワードの指定が必要な場合も自動的に聞かれる

-h ホスト名指定（自ホストの場合は不要）

-p ポート番号指定が必要な場合は使う。小文字であることに注意。

<SHEMA-NAME> 利用するスキーマ名。MySQLでいうDB名にあたる。

psqlコマンドの使い方

データベースの一覧を表示

```
psql> \l
```

操作するデータベースを指定（切り替え）

```
psql> \connect <SCHEMA-NAME>
```

psqlコマンドの使い方

データベース内テーブル一覧を表示

```
psql> \dt
```

テーブルの詳細を表示

```
psql> \d <TABLE-NAME>
psql> \d+ <TABLE-NAME> --より詳しい表示になる
```

PostgreSQLとの接続を終了する

```
psql> \q
```

2. mysqlコマンドと psqlコマンドの使い方

コマンド内ののヘルプを表示

```
mysql> \?
psql> \?
```

どちらも同じだが表示される内容は若干若干毛色が違う。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. **CRUD操作の基礎**
4. MySQLのDBエンジン
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

3. CRUD操作の基礎

- Create - データの作成(追加)
- Read - データの読み込み
- Update - データの更新
- Delete - データの削除

3. CRUD操作の基礎

SQLを使ったテーブル行の操作

- Create → INSERT
- Read → SELECT
- Update → UPDATE
- Delete → DELETE

SELECT文

テーブルから全てのデータを読み込む。

```
SELECT * FROM <TABLE-NAME>; -- 大きなテーブルだと凄く重いので注意
```

テーブルからカラム指定でn行を読み込む。

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
LIMIT <n-LINE>;

-- どういうデータが入っているか軽く確認したい時に
-- LIMIT句をつけて数行だけ取得したりする。
```

SELECT文 WHERE句

条件を絞り込んでデータを取得する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
WHERE <COL-NAME> = xxx;
```

```
-- 例
SELECT id, s_name
FROM staffs
WHERE id = 2;
```

```
-- 例
SELECT id, s_name
FROM staffs
WHERE s_name = '花子';
```

staffsテーブル

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

SELECT文 WHERE句

条件を絞り込んでデータを取得する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
WHERE <COL-NAME> LIKE 'xxx';
```

```
-- 例
SELECT id, s_name
FROM staffs
WHERE s_name LIKE '%郎';

-- LIKE内での % は何らかの文字列
-- LIKE内での _ は何らかの文字（1文字）
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

SELECT文 WHERE句

条件を絞り込んでデータを取得する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
WHERE <COL-NAME> IN(xxx, xxx);
```

```
-- 例
SELECT id, s_name
FROM staffs
WHERE id IN(1, 3);
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

SELECT文 WHERE句

条件を絞り込んでデータを取得する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
WHERE <COL-NAME> BETWEEN xxx AND xxx;
```

```
-- 例
SELECT id, s_name
FROM staffs
WHERE id BETWEEN 2 AND 4;

-- 上記の結果にはid 2 と 4 も含む
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

SELECT文 WHERE句

条件を絞り込んでデータを取得する

```
WHERE id > 2    -- 2より大きい
WHERE id >= 2   -- 2以上
WHERE id != 2   -- 2ではない
WHERE name IS NULL -- NULL(空データ)である
WHERE name IS NOT NULL -- NULL(空データ)ではない
```

複数の条件で絞り込む

```
WHERE id > 2 AND id < 100 -- idが2より大きく100より小さい
WHERE AAA = true AND (week == ‘sunday’ OR week == ‘saturday’)
```

SELECT文 ORDER BY句

並び順を制御する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME>
ORDER BY <COL-NAME> <ASC/DESC>;
```

```
-- 例
SELECT id, s_name, section_id
FROM staffs
ORDER BY id DESC;

-- ASC 昇順
-- DESC 降順
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

SELECT文 JOIN句

テーブルを結合する

```
SELECT <COL-NAME>, <COL-NAME>, <COL-NAME>
FROM <TABLE-NAME-1>
LEFT JOIN <TABLE-NAME-2>
          ON <TABLE-NAME-1>.<COL-NAME> = <TABLE-NAME-2>.<COL-NAME>;
```

```
-- 例
SELECT s_name, location_name
FROM staffs
LEFT JOIN locations
  ON staffs.location_id =
     locations.location_id;
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

location_id	location_name	address
1	名古屋事務所	愛知県xxx..
2	神戸事務所	兵庫県xxx..

SELECT文 GROUP BY句

データをグルーピングして集計する

```
-- 例
SELECT location_id, location_name, COUNT(id) AS 'staff_count'
FROM staffs
LEFT JOIN locations ON staffs.location_id = locations.location_id
GROUP BY location_id
ORDER BY staff_count desc;

-- スタッフをlocation_idごとにグループ化。
-- 件数をカウントして所属人数を求める。
-- 所属人数順に降順ソートする。

-- AS 列に別名をつける
-- COUNT 件数を数える
-- 他にSUM, AVG, MAX, MIN等
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

location_id	location_name	address
1	名古屋事務所	愛知県xxx..
2	神戸事務所	兵庫県xxx..

INSERT文

テーブルにデータ（行）を追加する

```
-- 例
INSERT INTO staffs (s_name, section_id, location_id)
    VALUES ('五郎', 1, 2);
```

```
-- idカラムはAUTO INCREMENTになっている想定。
-- AUTO INCREMENTのカラムは自動で採番される。
-- この場合idは自動で5が設定される。
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

UPDATE文

テーブルのデータ（行）を更新する

```
-- 例
UPDATE staffs SET s_name = '六郎', section_id = 1
WHERE id = 4;
```

-- WHERE句で対象を絞り込まないとテーブル内の全ての行が書き換わる。

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

DELETE文

テーブルからデータ（行）を削除する

```
-- 例
DELETE FROM staffs WHERE id = 3;
-- WHERE句で対象を絞り込まないとテーブル内の全ての行が削除される。
```

id	s_name	section_id	location_id
1	太郎	1	1
2	花子	2	2
3	三郎	2	1
4	四郎	3	1

DDL (実例省略)

- ここまでのSELECT, INSERT, UPDATE, DELETEは「DML」と呼ばれます。
  以下はDBやテーブル構造自体などに手を入れる「DDL」について。

- CREATE DATABASE : DBを作成

- CREATE TABLE : テーブルを作成

- DROP TABLE : テーブルを削除

- DROP DATABASE : DBを削除

- ALTER TABLE : テーブルの構造を変更

- そのほかにもユーザー作成など色々……

重たいクエリとは

- WHEREの条件、ソート、テーブルの結合（JOIN）、サブクエリなど、抽出条件やソートが増えるほど重くなりやすい。この際、**抽出条件や結合条件に関わるカラムに適切な「インデックス」が設定されていない**場合、設定されているが上手く利用できないクエリが実行された場合はさらに重くなる（→フルテーブルスキャンなどが起きる）。

- 特にテーブルの結合とサブクエリは処理中のデータを「**一時テーブル**」として保持する。**一時テーブルがメモリに乗りきらない**サイズになるとディスク上に一時テーブルを書き込み更に重くなる。

- 大量のデータを取得する場合ももちろん重くなる。

- **テーブルのデータがメモリに乗りきらなくなるとディスクアクセスで重くなる**（MySQLであれば例えばinnodb_buffer_pool_size不足など）。

重たいクエリとは

- WHEREの条件、ソート、テーブルの結合（JOIN）、サブクエリなど、抽出条件やソートが増えるほど重くなりやすい。この際、**抽出条件や結合条件に関わるカラムに適切な「インデックス」が設定されていない**場合に重くなる（→フルテーブルスキャンなどが起きる）。

クエリの実行計画を確認する
クエリの見直し
DB設計（スキーマ）の見直し　等

- 特にテーブルの結合とサブクエリは処理中のデータを「**一時テーブル**」として保持する。**一時テーブルがメモリに乗りきらない**サイズになるとディスク上に一時テーブルを書き込み更に重くなる。

クエリの見直し
メモリを増やす+それに併せてチューニング　等

- 大量のデータを取得する場合ももちろん重くなる。

クエリの見直し　等

- **テーブルのデータがメモリに乗りきらなくなるとディスクアクセスで重くなる**（MySQLであれば例えばinnodb_buffer_pool_size不足など）

メモリを増やす+それに併せてチューニング
DB設計（スキーマ）の見直し　等

ロックされるクエリとは

- MySQLの場合、InnoDBとMyISAMではロックされるタイミングが異なる。

- ストレージエンジンにMyISAMを使っている場合はINSERTやUPDATE中にテーブル全体がロックされ、他の接続から書き込みも読み込みもできなくなる。

- InnoDBの場合は行ごとのロックが行われるので、書き込みを行なっていても他の接続が実行するクエリは読み込みや別の行の書き換えが行える。

- テーブルの構造を変更する（ALTER TABLE文）ような場合にもロックされる。カラムを追加したりなど。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. **MySQLのDBエンジン**
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

4. MySQLのDBエンジン

- 最低限知っておいた方がいいのは**InnoDB**と**MyISAM**

	InnoDB	MyISAM
トランザクション	対応	非対応
ロック	行ロック ※	テーブルロック
全文検索	5.6以降対応	対応
地理データ	5.7以降対応	対応
オンラインALTER TABLE	5.6以降対応	非対応
速度	新しいバージョンほど速い※	速いけどロックが……

※ロックされる行数は場合によって異なり、大量の行がロックされ実質的にテーブルロックに近い状態になる場合もある。
※ベンチマーク内容にもよって変わるが基本的には新しい方が速い。

トランザクション

- トランザクション： DBにおいては複数のクエリをまとめて扱う仕組み。

- Aさんの口座からBさんの口座に100円を移動する場合
  →トランザクション開始。
  →Aさんの口座残高から100円減らすクエリを実行。
  →Bさんの口座残高を100円増やすクエリを実行。
  →トランザクション完了。

- トランザクションの途中で何らかのエラーが起きた場合にまとめて処理を失敗させる。これにより中途半端でおかしなデータになってしまう事を防ぐ。

- 安全なアプリケーションを作るために大切な仕組み。DBとプログラム側の両方でトランザクションに対応する必要がある。

4. MySQLのDBエンジン

- 必ずしもトランザクションが必要でないアプリもあるが、
  安全なアプリを作るなら基本的には有った方がいい。
  →InnoDBの方が良い。

- Webアクセスの度にDB上にアクセス数をカウントするなど、
  書き込みが多いDBもInnoDBが有利（ロックの問題）。

- データベースの定期バックアップを取得したい。
  →テーブル全体をロックしないInnoDBの方が楽。

  ※MySQL自体の管理系のテーブルはMyISAMなので、その部分について編集がかからないように注意。全ての管理系テーブルがInnoDBになるのはMySQL 8.0から。

InnoDBのファイルフォーマット

- InnoDBにはInnoDB自体の「ファイルフォーマット」とその中で行のデータをどういう形式で保存するかという「行フォーマット」というものがある。

- MySQLのバージョンによってデフォルトは異なる。
  MySQL 5.6までのデフォルトはAntelope。

- 古いMySQL DBでよく使われているファイルフォーマット Antelope では1行あたり8KBまでしか保存できない。
  長文や大きなバイナリデータを保存したい場合は要注意。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. **文字コードと照合順**
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

文字コード

- UTF-8、Shift-JIS、Latin-1など

- MySQLのデフォルトの文字コード指定はバージョンによって異なるので注意。最近のバージョンはutf8

- データベース、テーブル、（文字列型の）カラムごとにも文字コードを持っている。

- MySQLのutf8は若干問題のあるutf8。

utf8とutf8mb4

- MySQLのutf8は「サロゲートペア」と呼ばれる文字に対応していない。

- 具体的には各種絵文字 😄 であったり

- 吉野家の 吉（下の棒が長いつちよし）のような文字など

- utf8mb4でないと「絵文字が保存できない！」などが起きるのでユーザーが自由に内容を投稿するようなWebアプリの場合は要注意。

- PostgreSQLのutf8はサロゲートペアに対応します。

文字照合順

- 文字コードと合わせて「照合順(collation)」というものがある。文字を比較するルール。

- 照合順によって **パ** と **ハ** が同じ文字とみなされたり、🍣 と 🍺 が同じ文字とみなされたり……

- 寿司ビール問題、パパハハ問題なのでググるとより詳しい情報が出てきます。これによって様々な問題が出てくるのですが、ここではまず「そういうものがある」とだけ知っておいてください。

5. 文字コードと照合順

- 開発者的には困ったら文字コード utf8mb4 の
  照合順 utf8mb4_bin でいいのではないかと思っている。
  ※MySQL 5.5以降の場合。utf8mb4は5.5以降でしか使えません。
- あくまで私個人の見解です。
- DBサーバー移行の場合は元の文字コードに合わせるのが原則。
- Webアプリ上の文字コード、MySQL接続時の文字コード、DB上の文字コードで差異があると文字化けになる場合があります。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. 文字コードと照合順
6. **クエリのパフォーマンス**
7. インデックス
8. レプリケーション
9. パーティショニング
10. 設定と設定値の確認

基本的にはこれ

- WHEREの条件、ソート、テーブルの結合（JOIN）、サブクエリなど、抽出条件やソートが増えるほど重くなりやすい。この際、**抽出条件や結合条件に関わるカラムに適切な「インデックス」が設定されていない**場合に重くなる（→フルテーブルスキャンなどが起きる）。

  クエリの実行計画を確認する
  クエリの見直し
  DB設計（スキーマ）の見直し　等

- 特にテーブルの結合とサブクエリは処理中のデータを「**一時テーブル**」として保持する。**一時テーブルがメモリに乗りきらない**サイズになるとディスク上に一時テーブルを書き込み更に重くなる。

  クエリの見直し
  メモリを増やす+それに併せてチューニング　等

- 大量のデータを取得する場合ももちろん重くなる。

  クエリの見直し　等

- **テーブルのデータがメモリに乗りきらなくなるとディスクアクセスで重くなる**（MySQLであれば例えばinnodb_buffer_pool_size不足など）

  メモリを増やす+それに併せてチューニング
  DB設計（スキーマ）の見直し　等

6. クエリのパフォーマンス

- クエリがどのように実行されるか = 実行計画
  を確認することでクエリの改善点の材料の一つになる。
  インデックスが使われていない！など。

- EXPLAIN <任意のSQL>;
  で調べる。

- ほかにもスロークエリログを有効化する事で、処理の遅いクエリをログに残すことができる。極端に遅いクエリをまとめる事でプログラム（内のクエリ）の改善の材料になる。

7. インデックス

- インデックスはテーブルに対する目次のようなもの。抽出条件に使われるようなカラムにはインデックスを作成しておくべき。

- テーブルのPRIMARY KEY（idカラムなど）は、MySQL、PostgreSQLの場合は自動的にインデックスが作成されている。それ以外のカラムで必要なものがあれば作成。

- 無駄なインデックスを増やすとかえってテーブルサイズが大きくなり重くなる。むやみにインデックスを増やしていいわけではない。

- インデックスの設定はALTER TABLEを使うので、インデックス追加の際はロックの発生に注意。

8. レプリケーション

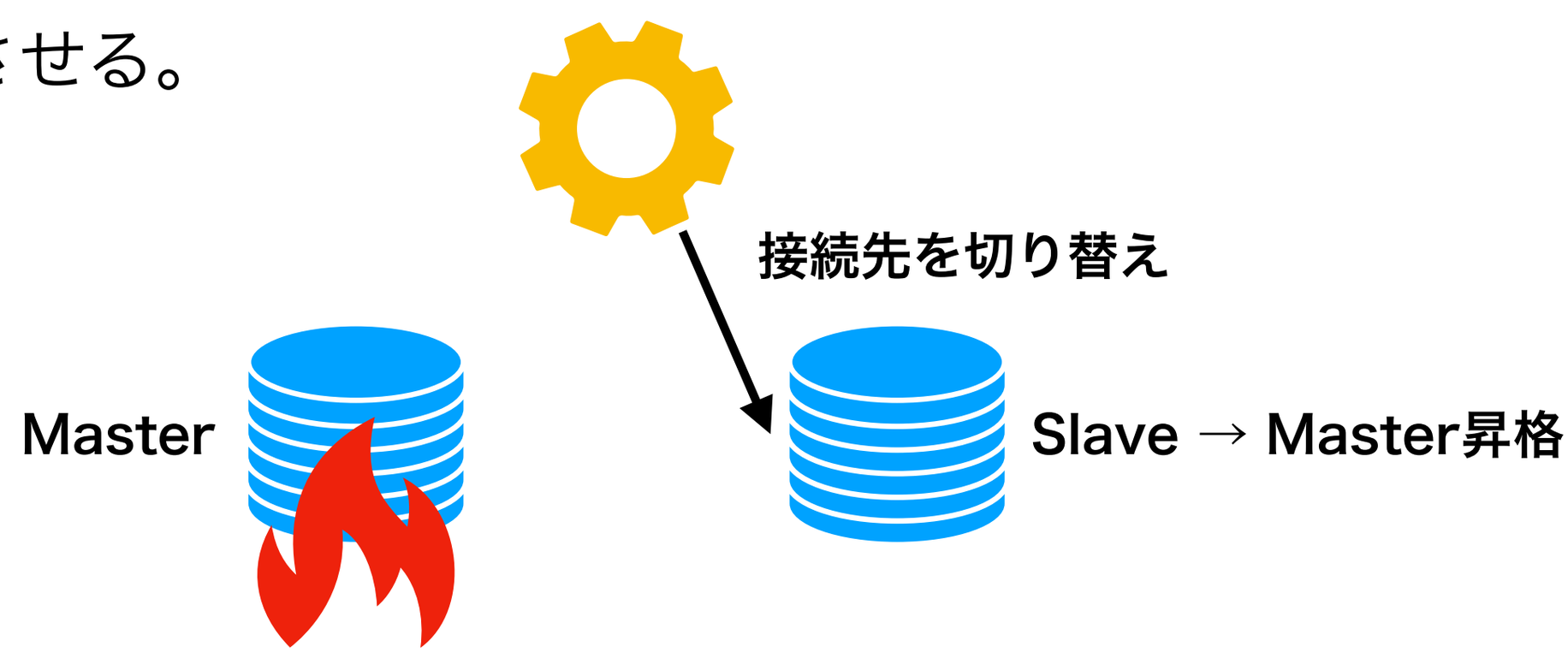
- レプリケーション：
  複数のRDBMSインスタンス間でデータを同期する仕組み

- マスタ側のMySQLでデータを更新するとスレーブ側の
  MySQLでも同様にデータが更新される。

- 可用性だけでなくパフォーマンスの観点からもレプリケーションが使われる。

レプリケーションの目的

- 可用性： Master側のサーバーが何らかの理由で利用できなくなった場合、Slave側を新しいMasterとして利用する。アプリ側から参照するDBを切り替えてシステムを継続稼働させる。



レプリケーションの目的

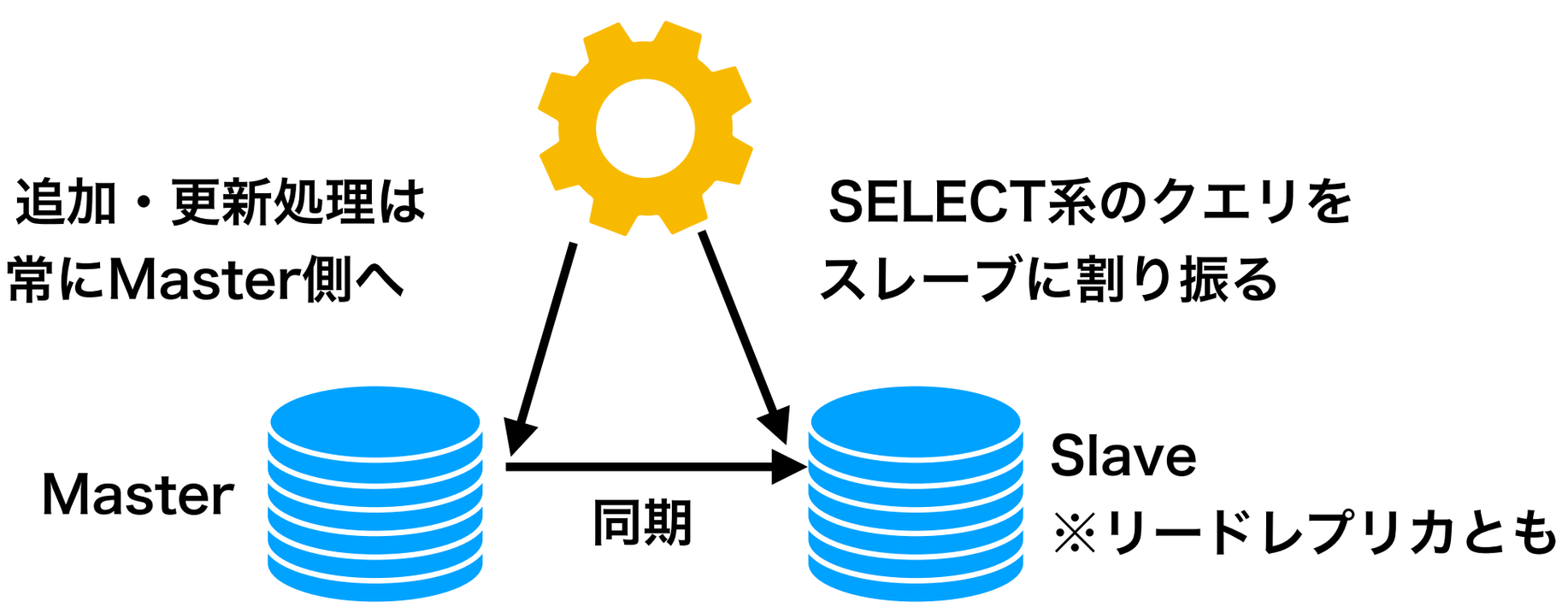
- あくまで同じデータが同期されているので、Masterで消されたデータはSlaveでも消される。Master側で誤った改変が行われたデータもSlave側に反映される。

- そういう意味でレプリケーションはデータのバックアップにはならない。この例ではあくまでシステムとしての可用性を高めるために使われている。

- 切り替えはMySQLでもPostgreSQLでも基本的に手動（スイッチオーバー）。切り替えを自動化（フェイルオーバー）する方法はいくつか有るがある程度自分で構築する必要あり。
  →ほぼ全自動のAmazon RDS便利！

レプリケーションの目的

- 分散： 参照系のクエリを読み込み専用のリードレプリカに向けてDB1台あたりの負荷を下げる。



レプリケーションの目的

- 大きなシステムでよく用いられる手法。リードレプリカが増えるほどシステムのSELECT処理能力が上がる。

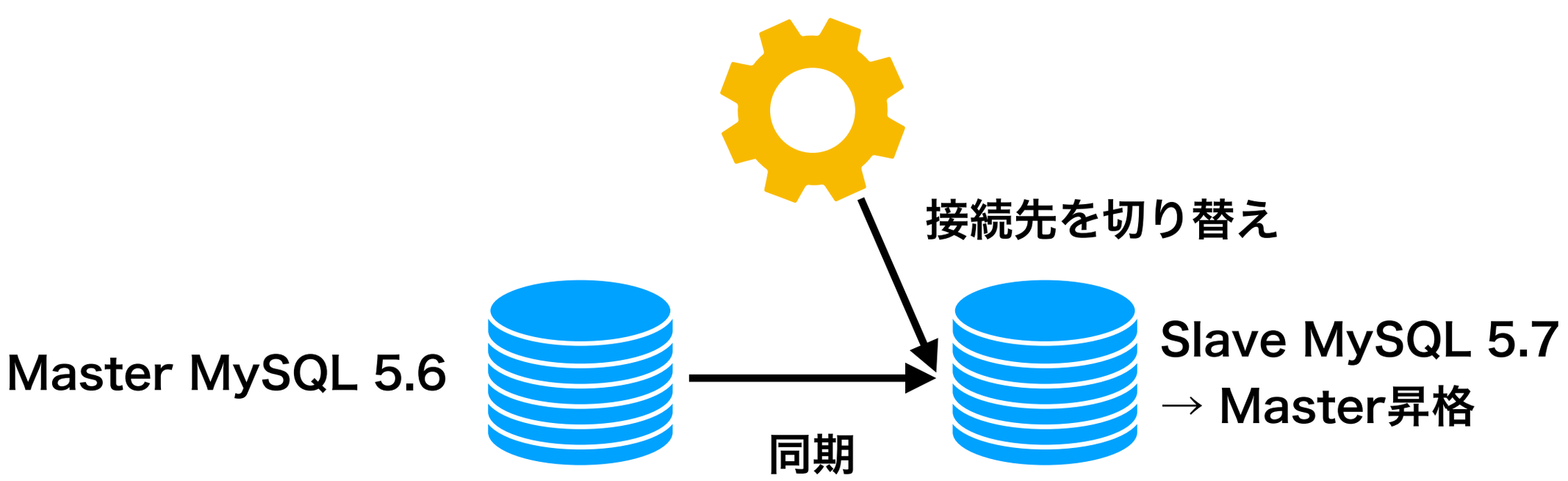
- クエリの割り振りはプログラム側、もしくはプログラムとの間に挟む別のミドルウェアによって制御する必要がある。

- 非同期レプリケーションの場合、スレーブ側へ全ての同期が完了してない場合もある。そういう前提の設計をしておく必要がある。

- バックアップ処理など重たい読み込み処理をスレーブ（リードレプリカ）側で実行すると効果的。定期バックアップがメインDB（マスター側）に負荷を与えなくなる。

レプリケーションの目的

- 移行： レプリケーションでデータを同期しておき、接続先を切り替えることでダウンタイムの短いDB移行が行える。



※ダウンタイムは短くて済むが、切り替える瞬間に関してはMasterへの書き込みをストップさせておく事。

MySQLのレプリケーション

- バイナリログと呼ばれるものを転送してレプリケーションを実現する。

- 実行したSQLの内容を転送する「ステートメントのレプリケーション」と行の内容を転送する「行のレプリケーション」、その混合など転送方法に種類があり、バイナリログ形式の設定によって変わってくる。ステートメントのレプリケーションは転送量が軽くなるが、実行するたびに結果がわかるようなタイプのクエリの実行は危険なものになる。
  →Amazon RDSのMulti-AZではMIXEDの設定で固定されている。

PostgreSQLの
ストリーミングレプリケーション

- WALと呼ばれる先行書き込みログを転送してレプリケーションを実現する。PostgreSQL9.1以降からの機能。

- WALに書き込まれた物理データを転送し、それを元にスタンバイ側がデータをリカバリし続けるような処理になっている。実データを転送する仕組み上、異なるPostgreSQLバージョン間でのレプリケーションは構築できない。

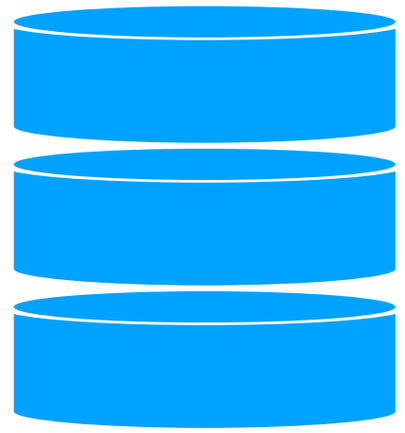
- PostgreSQL内蔵機能ではなくpg_poolと呼ばれるミドルウェアを利用した冗長化方法もある。pg_poolは受け取ったクエリを複数のPostgreSQLサーバに送信することでDBを冗長化する。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. **パーティショニング**
10. 設定と設定値の確認

9. パーティショニング

- 1つのテーブルを複数の領域に分割する仕組み。
- 巨大なテーブルを扱う際の処理が軽くなる。



- テーブルの全ての領域にアクセスしない場合は消費メモリも抑えられる。
- パーティションを使うことによる制限事項も有ることには有るが、リードレプリカにクエリを振り分けるようなプログラム改修よりは敷居が低め。

9. パーティショニング

- 2018年1月分のデータが入るパーティション、2018年2月分の、3月分の……とあらかじめパーティションを作成しておく必要がある。先に10年分ぐらい作っておくなどしておく。

- 例えばログを保存したテーブルから古いデータを削除する場合、DELETEを実行するのではなく古いデータの入ったパーティションを削除することで削除処理が軽くなる。ものすごい勢いでログが増えるようなシステムで便利。
  →オンラインゲームなど。

- RDBMSの種類、バージョンによってテーブルあたりの最大パーティション数は異なるので数年分作成しておく際は要確認。

話す事

1. RDBMSとは
2. mysqlコマンドとpsqlコマンドの使い方
3. CRUD操作の基礎
4. MySQLのDBエンジン
5. 文字コードと照合順
6. クエリのパフォーマンス
7. インデックス
8. レプリケーション
9. パーティショニング
10. **設定と設定値の確認**

MySQLの設定

- MySQL自体の設定：/etc/my.cnf
- 既存データベースやテーブルへの設定：ALTER TABLE実行

MySQLのチューニング

- これまで見てきたように、高速化のために設定ファイルの修正だけで出来る事は少ない。

- ただしマシンスペックに合わせた設定は必要であり、例えばスペックアップしてメモリ容量が増えた場合などにはかならず合わせて設定の調整が必要。

- 設定値はMySQLの再起動が必要なものとそうでないものがある。MySQLのバージョンによっても異なるので注意。設定項目もバージョンごとに増えたり、選択肢が変わっていたり、デフォルト値が変更されていたり。
  →**必ず対応するバージョンの公式マニュアルを読む。**

MySQLのチューニング

- 現在の設定値をクエリで確認
  SHOW GLOBAL VARIABLES LIKE ‘<VARIABLE_NAME>’;

- 現在のステータスをクエリで確認
  SHOW GLOBAL STATUS LIKE ‘<STATUS_NAME>’;

- GLOBALを付けない場合は現在の接続に対しての状態を確認する。
  基本的にはGLOBAL付けっぱなしでいい。

MySQLのチューニング

- `innodb_buffer_pool_size`
  ディスクから読み込まれた(InnoDBの)テーブルとインデックスが載る共有バッファ。データがバッファメモリに乗り切らない場合、ディスクI/Oが増える。MySQLでは**一番大事なパラメータ**。DB専用サーバの場合物理メモリサイズの5〜7割ぐらいを割り当てるが、同時アクセス数にもよる。

- 共有バッファとは別でMySQL接続ごとにもメモリが消費される(スレッドバッファ)。

- 現在の設定を確認
  ```
  SHOW GLOBAL VARIABLES LIKE ‘innodb_buffer_pool_size’;
  ```

MySQLのチューニング

- key_buffer_size
  MyISAM DBのインデックス情報が載る共有バッファ。
  MyISAMのデータベースがメインで動いている、かつ適切にインデックスが設定されているシステムなら増やしておくと速くなるかもしれない。

MySQLのチューニング

- max_connections

  MySQLに接続できる最大の接続数。接続ごとにスレッドバッファを消費するので無尽蔵に増やしていいわけではない。

- 実際どのくらい接続できるのか、は単純には求められず、計算方法には諸説ある……。

- MySQLが起動してからこれまでの最大接続数を見る

  SHOW GLOBAL STATUS ‘Max_used_connections’;

  これがmax_connectionsに近いようなら要調整。

MySQLのチューニング

- thread_cache_size
  接続を待ち受けるスレッドの数を指定する。同時アクセス数が多い場合はあらかじめスレッドの数を増やしておく。

MySQLのチューニング

- `slow_query_log`
  1を指定してスロークエリログの出力を有効。
  long_query_timeでスロークエリとする秒数の指定、
  slow_query_log_fileでスロークエリログの保存場所。

- 重いクエリが原因でシステムが遅くなっている気配がしたら確認する。クエリを特定したらEXPLAINで実行計画を見る。

MySQLのチューニング

- innodb_file_per_table
  テーブルごとにファイルを分ける設定。MySQL 5.6からはデフォルトで有効。テーブルを削除したりする場合、これが有効になっている方が速く動きかつディスク容量の消費も抑えられる。

- テーブル数がすごい(数千〜数万)場合にファイルディスクリプタの消費が増えるなどの問題もあるが、基本的には有効でいい。

- 後述のinnodb_file_formatをBarracudaに指定するにはinnodb_file_per_tableの有効かは必須。

MySQLのチューニング

- `innodb_file_format`
  InnoDBのファイルフォーマット。前述の1行8KBの壁対策のためにもデフォルトのAntelopeではなくBarracudaを指定しておく。

- MySQL 5.7以降からはBarracudaがデフォルト。

- Barracudaを指定するときはかならず innodb_fire_per_tableを有効にしておくこと。

MySQLのチューニング

- `innodb_log_file_size`
  InnoDBへの更新ログファイルのサイズ指定。これがりていないとDB更新が多い場合にディスクI/Oが増えます。
  innodb_buffer_pool_sizeの4分の1程度が適切と言われる。ただし大きくしすぎるとInnoDBがクラッシュした際に(特にMySQL 5.5以前では)リカバリ時間が伸びます。

- MySQL 5.5ではデフォルト5MB、5.6からは50MB。

- 個人的な感覚としては最大でも256MBまでで。

MySQLのチューニング

- `max_heap_table_size`
  `tmp_table_size`

  上記の2項目はセットで同じ値を設定しておく。テーブルの結合時にメモリに乗せておく上限。ここで指定したサイズを超えるとディスク上に一時テーブルが書き込まれディスクI/Oで遅くなる。巨大な一時テーブルが作られないようなクエリにしておくべきだが、どうしてもサイズが大きい場合は調整しておく。

- MySQLが起動してからこれまでにディスク上に一時テーブルを作成した数

  `SHOW GROBAL STATUS LIKE ‘Created_tmp_disk_tables’;`

MySQLのチューニング

- `sql_mode`
  SQLモードの設定。様々な設定値が追加できる。MySQLはデフォルトでは怪しいSQLでも受け入れてしまう傾向が強いが、SQLモードを設定しておく事でそれらをエラーにすることができる。

  - 怪しい・問題のあるSQL： 例えば10文字しか入らない設定のカラムに15文字入れようとしたりなど。SQLモードで禁止されていない場合は勝手に5文字削除されて10文字目までが保存される。

  - MySQLのバージョンによってデフォルト値が異なり、新しいほど厳しい。古いMySQLを対象に書かれたアプリだと、アプリ側に怪しいSQLが書かれているせいでエラーになる場合があるので（本来プログラム側を修正するべきだが、それができない場合は）移行元とSQLモードを合わせておく。

MariaDBのチューニング

- MySQLとMariaDBは**SQL互換**であり、設定ファイルに互換性はありません。特にMySQL 5.6/MariaDB 10.0世代以降は注意。

- 概ね同じ設定値が使えますが、上記バージョン以降は異なる設定項目が増えているのでMySQLと同じ感覚で設定するのは要注意。

- SQL互換についてもMySQL、MariaDBそれぞれでどんどん機能が増え続けているので、**MySQL 5.5〜5.6の頃のSQLならどちらでも動く**ぐらいの認識で。

- 実際のところ違いで大きく困る事はあまりない。数年先の、より差異が大きくなってきた頃には困ることが増えるかも？

PostgreSQLの設定

- 必ず触る設定ファイルは
  postgresql.conf と pg_hba.conf

- pg_hba.conf はロール(ユーザー)ごとの接続許可設定。
  接続元IP指定や認証方法などを設定する。

- PostgreSQLは日本語のマニュアルが充実しているので
  →**必ず対応するバージョンの公式マニュアルを読む。**

PostgreSQLのチューニング

- shared_buffers
  共有バッファ。デフォルトでは128MBしかない（古いバージョンでは32MBしかない場合も）、メインメモリの25%程度を割り当てる。

PostgreSQLのチューニング

- checkpoint_segments
  共有バッファ上の変更内容ログ（WAL）をDBに書き込む頻度。更新頻度が高いとディスクに書き込む頻度が増えI/Oで遅くなるので設定値を増やしておく。デフォルトの3から16〜32あたりに増やす。

- WALファイルのサイズ(16MB) * checkpoint_segments 更新サイズごとにディスクに書き込まれる。デフォルトの3では48MB相当。

- 上記のサイズだけでなくcheckpoint_timeoutで指定された時間ごとにもディスクに書き込み。

- PostgreSQL 9.5からはこの設定項目は無くなり、max_wal_sizeとmin_wal_sizeを使って設定します。1GB相当。

PostgreSQLのチューニング

- `max_connections;`
  最大接続数の設定です。接続数 * work_mem のメモリを消費していくのでメモリ消費の様子を見ながら設定。

PostgreSQLのチューニング

- `log_min_duration_statement;`
  PostgreSQLで言うところのスロークエリログ設定。時間を指定するとその長さを超えたクエリがログに出力される。単位はミリ秒。デフォルト値は-1(無効)。

- PostgreSQLのログファイルのフォーマットは log_line_prefix で調整する。設定によってはログに日時が出力されないので注意。

DBの為にサーバー運用・構築者ができること

- 最初に設定しておくべき項目は設定しておく。
- スペックアップやスペックダウンを行う際は、サイズ指定関連の設定も併せて調整する。ただしそれ以上の高速化を求めるなら設定値変更だけでできることは少ない。
- 項目内容や設定値、デフォルト値はバージョンによって異なるので公式のドキュメントを確認する。
- クエリが重ければソフトウェア開発側に協力を仰ぐ。