# Groonga族 2016

須藤功平　株式会社クリアコード

*Groonga Meatup 2017*
*2017-02-09*

# 最近のニュース（1）

Groonga 7.0.0
リリース！

# 最近のニュース (2)

Mroonga 7.00
リリース！

# 最近のニュース（3）

## MySQL Workbench

（Oracle製のGUIなMySQL管理ツール）

## Mroongaをサポート！

# 最近のニュース（4）

## Zulip

（Dropbox製のチャットツール）

## PGroongaをサポート！

# 2016年の概要

- Groongaが凄くよくなった
  - 速くもなったし便利にもなった
- Mroongaはそこそこ
  - でもGroongaがよくなったので Mroongaもよくなった
- PGroongaも凄くよくなった

# 最新情報：Groonga


ぐるんが

# Groonga: 1

索引指定の
等価検索を
サポート

# 索引指定

```
column == 29
# ↑Groongaが適切な索引を選んで使用
Numbers.index == 29
# ↑指定した索引で検索
```

# Groonga: 2

# シーケンシャルサーチのパフォーマンス向上

（索引なしでも速くなる）

# 向上パターン

```
true
#  ↑定数
x == 29
#  ↑定数値との二項演算
```

# Groonga: 3

config_*
コマンドを追加

# config_*コマンド

DB全体で使える
キー・バリュー・ストア

- config_set：設定
- config_get：取得
- config_delete：削除

http://groonga.org/ja/docs/reference/configuration.html

# 用途例

- エイリアス情報を格納するテーブル名の指定
- プラグインの設定
  - 例：クエリー展開辞書のパス指定
  - 例：ステミングの言語指定

# Groonga: 4

# エイリアス
# サポート

# エイリアス

- 同一テーブル・カラムに複数の名前でアクセス
  - 例：tags/Tagsどちらでも**OK**

http://groonga.org/ja/docs/reference/alias.html

# 用途例

- 既存クライアントを変更せずに
  テーブル・カラムの名前を変更
  - 古いテーブルを削除→
    古い名前で新しいテーブルを参照
- ダウンタイムなしで
  カラムの型を変更
  - 変更後にエイリアスを切り替え

# 使い方：設定

```
# Aliasesテーブルのキーを別名、
# real_nameカラムを実名とする
config_set \
  alias.column \
  Aliases.real_name
```

# 使い方：登録

```
table_create Aliases \
  TABLE_HASH_KEY ShortText
column_create Aliases real_name \
  COLUMN_SCALAR ShortText
# Users.ageをUsers.yearsに展開
load --table Aliases
[
{"_key": "Users.age",
 "real_name": "Users.years"}
]
```

# エイリアスの解決

- 再帰的
  - Users.age→
    Users.years→
    People.years
    もアリ

# Groonga: 5

## lock_*
## コマンドを追加

# lock_*コマンド

- lock_acquire：取得
- lock_release：解放

# 用途

テスト

# Groonga: 6

# あいまい検索サポート

# あいまい検索

- 多少表記が違ってもマッチ
- 索引を使うので高速
- @naoa_y作

http://blog.createfield.com/entry/2016/02/28/014432

# 用途例

- 名前・住所の検索
  - 表記が揺れやすい
- オートコンプリート
  - ユーザーの入力はミスが多め

# 使い方：準備

```
# パトリシアトライなら
# 索引なしでキーをあいまい検索できる
table_create Products TABLE_PAT_KEY ShortText
load --table Products
[
{"_key": "Groonga"},
{"_key": "Grooonga"},
{"_key": "Mroonga"}
]
```

# 使い方：検索

```
select Products \
  --filter 'fuzzy_search(_key, "Groonga")' \
  --output_columns '_key, _score'
# ... 一致しているほど_scoreが高い ...
# ["Groonga", 2],
# ["Grooonga", 1], 多少違ってもヒット
# ["Mroonga", 1]   多少違ってもヒット
# ...
```

# Groonga: 7

object_inspect
コマンドを追加

# object_inspect

- テーブル・カラムの情報を返す
  - レコード数とか総キーサイズとか
- 用途：調査

http://groonga.org/ja/docs/reference/commands/object_inspect.html

# Groonga: 8

# ベクターの指定位置要素の索引検索をサポート

# 例

```
# agesの1番目の要素が29未満ならヒット
# agesの0番目の要素が29未満でも関係ない
ages[1] < 29
```

# 使い方：索引

```
table_create Teams TABLE_NO_KEY
column_create Teams ages \
  COLUMN_VECTOR UInt8

table_create Ages TABLE_PAT_KEY
# 索引にはWITH_POSITIONを指定すること！
column_create Ages teams_age \
  COLUMN_INDEX|WITH_POSITION Teams age
```

# 使い方：検索

```
load --table Teams
[
{"ages": [ 1, 30,  2]},
{"ages": [30, 28, 29]}
]
select Teams --filter 'ages[1] < 29'
# "ages": [ 1, 30,  2]はヒットしない
# "ages": [30, 28, 29]だけヒット
```

# Groonga: 9

object_remove
コマンドを追加

# object_remove

- テーブル・カラム…を削除
  - 壊れていても強制削除可！
  - 要注意コマンド
- 用途：障害対応

http://groonga.org/ja/docs/reference/commands/object_remove.html

# Groonga: 10

# クエリー最適化

# AND+非等価→AND NOT+等価

```
# 最適化前（索引を使えない）
XXX && column != xxx
# 最適化後（索引を使える）
XXX &! column == xxx
```

# Groonga: 11

# Memcachedプロトコルでのカラム値の取得・更新をサポート

# 使い方：スキーマ

```
# テーブルのキーはShortText
table_create Memos \
  TABLE_HASH_KEY ShortText
# カラムはスカラーでText
#  (ShortText/LongTextも可)
column_create Memos content \
  COLUMN_SCALAR Text
```

# 使い方：サーバー

```
# Memcachedプロトコルで
# Memos.contentにアクセス
% groonga \
    --protocol memcached \
    --port 11211 \
    --memcached-column Memos.content \
    -s db
```

# 使い方：クライアント

```
import bmemcached
servers = ("127.0.0.1:11211",)
client = bmemcached.Client(servers)
print(client.get("abc")) # None
client.set("abc", "Hello")
print(client.get("abc")) # "Hello"
```

# 使い方：確認

```
% groonga db select Memos
# {"_key":    "abc",   # キー
#  "content": "Hello", # バリュー
#  ...他にいくつかメタデータ...}
```

# 用途例

- 永続化対応キーバリューストア
- 頻繁な参照・更新処理の高速化
  - HTTPより速い
  - 索引は更新される
    →更新したデータはHTTPで全文検索

# 参考：hog

- GroongaベースのKVS
  by @takiuchi
- 既存DBのカラムを読み書き
  - --protocol memcachedと同様
  - ↑より軽そうで機能が多い
- 独自プロトコル

https://github.com/genki/hog

# Groonga: 12

# ハッシュテーブルの最大総キーサイズUP
# 4GiB→1TiB

# 使い方

```
# デフォルトは4GiBのまま
# KEY_LARGE指定で1TiBに
table_create \
  --name Large \
  --flags TABLE_HASH_KEY|KEY_LARGE \
  --key_type ShortText
```

# Groonga: 13

即
シャットダウン
機能

# 使い方

```
# デフォルトは処理中の接続が
# 終わってからシャットダウン
# immediate：処理中の接続が
# あってもすぐにシャットダウン
shutdown --mode immediate
```

# 用途例

- Windowsシャットダウン時
  - 指定時間内に終了しないと強制終了
  - 強制終了するとDB破損の可能性アリ
  - 強制終了よりimmediateの方がマシ
  - （Windowsには指定時間を伸ばすAPIアリ）

# Groonga: 14

依存テーブル
依存カラムも
一緒に
テーブル削除

# 対象例

```
table_create Numbers \
  TABLE_PAT_KEY Int64
table_create IDs \
  TABLE_HASH_KEY Numbers
# ↑IDsはNumbersを参照
table_remove Numbers
# ↑Numbersは参照されている→エラー
```

# 使い方

```
# --dependent yesを指定
# デフォルトはno
table_remove Numbers \
  --dependent yes
# IDsも一緒に削除
```

http://groonga.org/ja/docs/reference/commands/table_remove.html

# Groonga: 15

# HTTPでの タイムアウト サポート

# タイムアウト

- タイムアウトなし
  →クライアント側で切断
  - 切断後もサーバーのCPUは空かない
- タイムアウトあり
  →サーバー側で中断
  - サーバーのCPUが空く

# 使い方

```
# 2.9秒で完了しなかったら
# 408 Request Timeoutを返す
/d/select.json?...&
  request_timeout=2.9
```

http://groonga.org/ja/docs/reference/command/request_timeout.html

# Groonga: 16

# 多段
# ドリルダウン
# サポート

# 多段ドリルダウン

- ドリルダウン結果を
  ドリルダウン
  - 利用例：大中小分類それぞれのヒット件数を1リクエストで集計
- データを正規化できる
  - データの更新に強い
- @naoa_y作

# イメージ

| 商品テーブル | 小分類テーブル | 中分類テーブル | 大分類テーブル |
|---|---|---|---|
| ThinkPad E470 | 2ドア冷蔵庫 | 冷蔵庫 | 家電 |
| ThinkPad E570 | 3ドア冷蔵庫 | 洗濯機 | コンピューター |
| VAIO C15 | ThinkPad | ノートPC | |
| Nexus 7 | Let's NOTE | タブレット | |
| | VAIO | | |
| | Nexus | | |

小分類でドリルダウン

ThinkPad（2件）
VAIO（1件）
Nexus（1件）

中分類でドリルダウン

ノートPC（3件）
タブレット（1件）

大分類でドリルダウン

コンピューター（4件）

# 使い方

```
select Items \
  --drilldowns[s].keys category \  # 小分類でドリルダウン
  --drilldowns[s].output_columns _key,medium,_nsubrecs \
  --drilldowns[m].table s \        # 小分類でのドリルダウン結果を
  --drilldowns[m].keys medium \    # 中分類でドリルダウン
  --drilldowns[m].calc_target _nsubrecs \
  --drilldowns[m].calc_types SUM \ # 件数（_nsubrecs）を合計
  --drilldowns[m].output_columns _key,large,_sum \
  --drilldowns[l].table m \        # 中分類でのドリルダウン結果を
  --drilldowns[l].keys large \     # 大分類でドリルダウン
  --drilldowns[l].calc_target _sum \
  --drilldowns[l].calc_types SUM \ # 合計（_sum）をさらに合計
  --drilldowns[l].output_columns _key,large,_sum
```

# Groonga: 17

# ウィンドウ関数サポート

# ウィンドウ関数

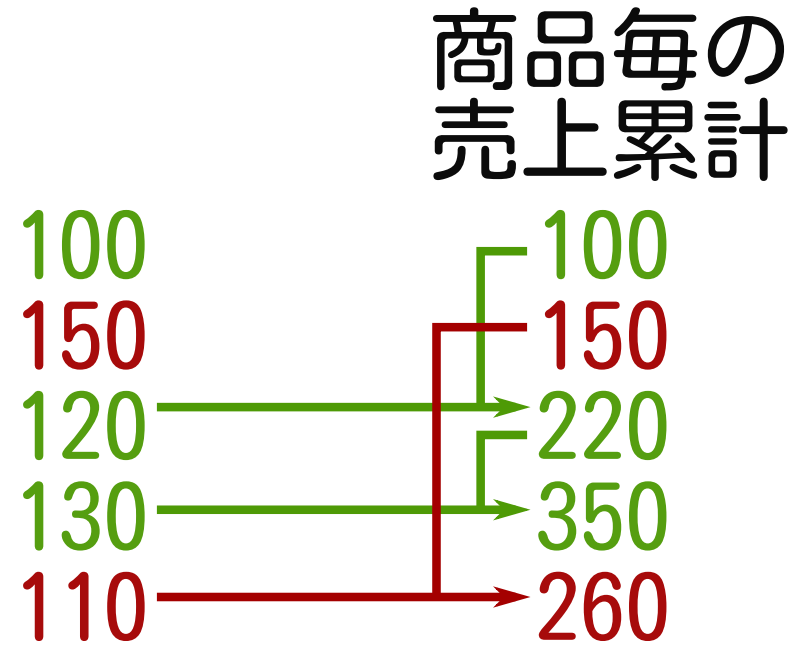
1. グループ毎に処理
   - 普通の関数は1レコード毎に処理
   - ドリルダウンは全レコードを処理
2. 指定したソート順で処理

# 用途例

- データ分析
- 例：
  - 商品毎（商品でグループ化）の売上累計（日付でソート）

# 商品毎の売上累計

| 商品 | 金額 | 日 |
|---|---|---|
| A | 100 | 2/1 |
| B | 150 | 2/1 |
| A | 120 | 2/2 |
| A | 130 | 2/3 |
| B | 110 | 2/3 |




商品で
グループ化

日付で
ソートして
累積

# 使い方

```
select Logs \
  # 商品毎の累計売上はamount一時カラムに入れる
  --columns[amount].stage initial \
  --columns[amount].type UInt32 \
  # window_sum()は累計するウィンドウ関数
  --columns[amount].value 'window_sum(sales)' \
  # 製品でグループ化
  --columns[amount].window.group_keys product \
  # 日付でソートした順に処理
  --columns[amount].window.sort_keys day \
  --output_columns *,amount # 出力
```

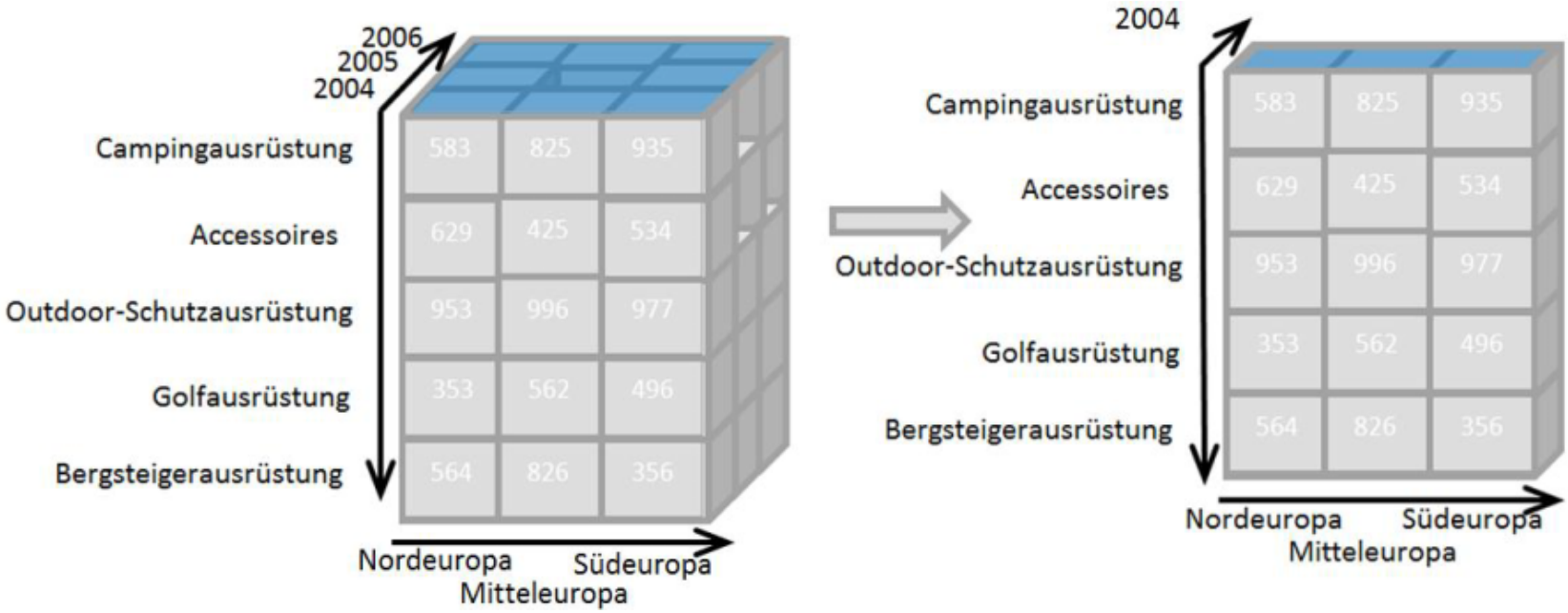
# Groonga: 18

# スライス
# サポート

# スライス

- OLAPの用語
  豆知識：ドリルダウンもOLAPの用語
  - 軸を1つ決めて絞り込む
- 例：
  全期間中今年のログだけに着目
  - 「今年」という軸で絞り込んでいる

https://en.wikipedia.org/wiki/OLAP_cube



# 用途例

## 1リクエストで<br>オススメ絞り込み結果も表示

「Webアプリ」の仕事の検索結果（29件）

1. ECサイト開発（PHP）
2. SNS開発（Ruby）
3. 社内ポータル開発（Java）
4. 検索サイト開発（Ruby）
…

オススメはRuby！(5件)

1. SNS開発
2. 検索サイト開発
…

「Webアプリ」での検索結果

さらに「Ruby」で絞り込み

スライス

# 使い方

```
select Works \
  --match_columns description \
  --query Webアプリ \
  # ↑までが通常の検索
  # ↓で追加の絞り込み
  --slices[ruby].filter language:Ruby
```

# Groonga: 20

## 数値・時間の分類関数を追加

（分類関数という言い回しはよくない気がする）

# 分類関数

- 値をグループ化
  - 例：100-199は100、200-299は200
  - 例：2017年1月も2017年2月も2017年

# 用途例

- 価格帯でのドリルダウン
- 年や月でのドリルダウン

# 使い方

```
select Logs \
  --columns[year].stage filtered \
  --columns[year].type UInt16 \
  --columns[year].flags COLUMN_SCALAR \
  --columns[year].value \
    'time_classify_year(timestamp)' \
  --output_columns timestamp,year
  # yearカラムに年だけが入っている
```

# Groonga: 21

## コマンド バージョン3 追加

# コマンドバージョン3

- レスポンスのフォーマット変更
  - 配列→オブジェクト
- デフォルトは1のまま

# 出力例

```
status --command_version 3
# {
#   "header": {
#     "return_code": 0,
#     "start_time": 1486275098.294436,
#     "elapsed_time": 0.00002,
#   },
#   "body": {
#     "alloc_count": 275,
#     ...
#   }
# }
```

# Groonga: 22

query_expand
コマンドを追加

# query_expandコマンド

- クエリー展開結果を返す
- 同義語の確認に便利

# 使用例：準備

```
table_create Words TABLE_HASH_KEY ShortText
# 類義語のリスト
column_create Words synonyms \
  COLUMN_VECTOR ShortText
# 「焼肉」は「焼き肉」とも書かれる
load --table Words
[
{"_key": "焼肉",
 "synonyms": ["焼肉", "焼き肉"]}
]
```

# 使用例：展開

```
query_expand Words.synonyms 焼肉
# "((焼肉) OR (焼き肉))"
# ↑「焼肉」で検索すると
# 「焼肉」と「焼き肉」のORで検索
```

# Groonga: 23

# デフォルトのロックタイムアウト短縮

# ロックタイムアウト

- ロック獲得を諦めるまでの時間
- 2.7時間→15分
  - クラッシュによるロック残留に気づくまでの時間が短くなった

# Groonga: 24

# テーブルの最大レコード数更新

# 最大レコード数

- 従来：$2^{28}-1$（約2億7千万件）
- 新：
  - NO_KEY, PAT_KEY: 約10億件
  - HASH_KEY: 約5億件
  - DAT_KEY: 約2億7千万件

# Groonga: 25

# Zstandard サポート

# Zstandard

- 新しい圧縮ライブラリー
  - zlib並の圧縮率
  - zlibの数倍の速度
- カラム値の圧縮に利用可能

# 使い方

```
table_create Memos \
  TABLE_HASH_KEY ShortText
# COMPRESS_ZSTDを指定するだけ
column_create Memos content \
  COLUMN_SCALAR|COMPRESS_ZSTD \
  Text
```

# Groonga: 26

# 選択的カラム圧縮サポート

# 選択的カラム圧縮

- 小さなデータの圧縮は無意味
  - ヘッダーの追加でむしろ大きくなる
  - ムダにリソースを使う
- ある程度大きなデータだけ圧縮
  - 257バイト以上のデータのみ圧縮
  - →カジュアルに圧縮指定できる！

# Groonga: 27

# loadコマンドがレコードIDの返却をサポート

# loadコマンド

- バルクロード用コマンド
  - 何件ロードしたかだけ返す
  - ロードしたレコードのIDは返さない
- 1件のみロードしたい時に不便
  - 追加したらIDを知りたい！

# 使い方

```
# _keyは数値
table_create Numbers TABLE_HASH_KEY Int32
# --output_ids yesと--command_version 3の指定が必要！
load --table Numbers \
     --output_ids yes --command_version 3
[
{"_key": 29},      # OK
{"_key": "Hello"} # NG
]
# "body": {
#   "loaded_ids": [1, 0]
# }
```

# Groonga: 28

# loadコマンドがエラー情報の返却をサポート

# loadコマンド

- バルクロード用コマンド
  - エラーはログに記録のみ
  - バッチのときは妥当な挙動
- インタラクティブなとき
  - ログ確認は不便
  - エラーをすぐに確認したい！

# 使い方

```
# --output_errors yesと--command_version 3の指定が必要！
load --table Numbers \
     --output_errors yes --command_version 3
[
{"_key": 29},     # OK
{"_key": "Hello"} # NG
]
# "body": {
#   "errors": [
#     {"return_code": 0, "message": null},
#     {"return_code": -22, "message": "failed to..."}
#   ]
# }
```

# Groonga: 29

速くなった！
検索も！
更新も！

# 高速化

- 検索
  - AND検索の高速化 by @naoa_y
    （10%-40%高速化）
  - フレーズ検索の高速化 by @naoa_y
    （0%-100%高速化）
- 更新
  - 非自然言語の索引更新性能劣化解消
    （分単位→ミリ秒単位）

# Groonga: まとめ

便利になって
速くもなった

# 最新情報：Mroonga


むるんが

# Mroonga: おしらせ

初心者向けの電子書籍ができた！

https://grnbook-ja.tumblr.com/
by @KitaitiMakoto

# Mroonga: 1

## mroonga_normalize UDFを追加

UDF: User Defined Function

# mroonga_normalize

- 正規化した文字列を返す
- 全文検索索引以外でも使える

# 使い方

```
SELECT mroonga_normalize('ｱﾊﾞ㊗ｻﾝﾁｰﾑ');
-- +-----------------------------------+
-- | mroonga_normalize('ｱﾊﾞ?ｻﾝﾁｰﾑ')       |
-- +-----------------------------------+
-- | アバ㊗サンチーム                    |
-- +-----------------------------------+
-- 半角カタカナ→全角カタカナ（濁点含む）
-- 絵文字→そのまま
-- 合成済み文字（？）→展開
```

# Mroonga: 2

## *SSプラグマ追加

# *SSプラグマ

```
SELECT ...
  WHERE
    MATCH (title)
-- Groongaの検索条件を↓で使える
    AGAINST ('*SS ...'
             IN BOOLEAN MODE);
```

# Groongaの検索条件

- 豊富な検索処理
  - 例：あいまい検索
- 複数の索引で検索可→速い！
  - MySQLは1つしか使えない

書式：http://groonga.org/ja/docs/reference/grn_expr/script_syntax.html

# Mroonga: 3

mroonga_
snippet_html
UDFが便利に

# mroonga_snippet_html

```
SELECT
  mroonga_snippet_html(content,
                       '+肉 -魚 +野菜' AS query)
  -- クエリーをそのまま指定できる↑
  -- （肉、野菜は対象。魚は非対象。）
  -- 「AS query」がポイント
  ...
  WHERE
    MATCH (content)
    AGAINST ('+肉 -魚 +野菜' IN BOOLEAN MODE);
```

# Mroonga: 4

マルチバイトな
カラム名を
サポート

# 使い方：定義

```
CREATE TABLE メモ (
  内容 text,
  FULLTEXT INDEX (内容)
) ENGINE=Mroonga
  DEFAULT CHARSET=utf8mb4;
```

# 使い方：確認

```
INSERT INTO メモ
  VALUES ('むるんがは速い！');
SELECT * FROM メモ WHERE
  MATCH (内容)
  AGAINST ('+速い' IN BOOLEAN MODE);
-- +--------------------------+
-- | 内容                     |
-- +--------------------------+
-- | むるんがは速い！         |
-- +--------------------------+
```

# Mroonga: 5

# FOREIGN KEY制約サポート

# FOREIGN KEY制約

- INSERT
  - 存在しない参照の挿入はエラー
- UPDATE
  - 被参照レコードの更新はエラー
- DELETE
  - 被参照レコードの削除はエラー

# 使い方：定義

```
CREATE TABLE tags (
  name VARCHAR(128) PRIMARY KEY
) ENGINE=Mroonga
  DEFAULT CHARSET=utf8mb4;
CREATE TABLE memos (
  tag VARCHAR(128),
  FOREIGN KEY (tag) REFERENCES tags (name)
) ENGINE=Mroonga
  DEFAULT CHARSET=utf8mb4;
```

# 使い方：INSERT

```
INSERT INTO memos VALUES ('Mroonga');
-- ↑はエラー：ERROR 1452 (23000):
-- Cannot add or update a child row:
-- a foreign key constraint fails
-- (foreign record doesn't exist:
--  <tag>:<"Mroonga">)
INSERT INTO tags VALUES ('Mroonga');
INSERT INTO memos VALUES ('Mroonga');
-- ↑タグ追加後は成功
-- Query OK, 1 row affected (0.00 sec)
```

# 使い方：UPDATE

```
UPDATE memos SET tag = 'MySQL';
-- ↑はエラー：ERROR 1452 (23000):
-- Cannot add or update a child row:
-- a foreign key constraint fails
-- (foreign record doesn't exist:
--  <tag>:<"MySQL">)
INSERT INTO tags VALUES ('MySQL');
UPDATE memos SET tag = 'MySQL';
-- ↑タグ追加後は成功
-- Query OK, 1 row affected (0.00 sec)
```

# 使い方：DELETE

```
DELETE FROM tags WHERE name = 'MySQL';
-- ↑参照されているタグは消せない
-- ERROR 1451 (23000):
-- Cannot delete or update a parent row:
-- a foreign key constraint fails
-- (one or more child rows exist in <memos>)
DELETE FROM tags WHERE name = 'Mroonga';
-- ↑参照されていないタグは消せる
-- Query OK, 1 row affected (0.00 sec)
```

# Mroonga: 6

# 最新 MySQL・MariaDB サポート

# サポートバージョン

- MySQL: 5.7.17
  - 8.0.0は未確認
- MariaDB: 10.2.3

# Mroonga: 7

# マルチカラムインデックスの性能劣化解消

# 性能劣化

- マルチカラムインデックス
  - キー：非自然言語
    →Groongaが苦手なパターンだった
  - Groonga側の改良で解消

# Mroonga: まとめ

- 便利になった！
- 速くもなった！
  - Groongaが速くなったから

# 最新情報：PGroonga

pgroonga

ぴーじーるんが

# PGroonga: 1

# TABLESPACE サポート

# TABLESPACE

- 別のパスにファイルを置く機能
  - 使用例：索引はSSDに置く

https://www.postgresql.org/docs/current/static/manage-ag-tablespaces.html

# 使い方

```
-- SSDを使うテーブルスペースを作成
CREATE TABLESPACE ssd
  LOCATION '/ssd/data';
CREATE TABLE memos (
  content text
); -- テーブルはHDDに置く
CREATE INDEX content_search
  ON memos
  USING pgroonga (content)
  TABLESPACE ssd; -- 索引はSSDに置く
```

# PGroonga: 2

## 複合主キーサポート

# 複合主キー

- 複数のカラムで主キーを構成
- pgroonga.scoreを使うには主キーが必要
  - 複合主キーでもpgroonga.scoreを使えるようになった

# 使い方：複合主キー定義

```
CREATE TABLE items (
  company_id int,
  product_id int,
  description text,
  -- 会社IDと製品IDを合わせて主キー
  PRIMARY KEY (company_id, product_id)
);
```

# 使い方：索引定義

```
CREATE INDEX description_search
  ON items
  -- 複合主キーの全カラムを含める
  USING pgroonga (company_id,
                  product_id,
                  description);
```

# 使い方：検索

```
-- 索引を使わないとスコアは常に0になる
SET enable_seqscan = 'no';
SELECT description, pgroonga.score(items)
  FROM items
  WHERE description @@ 'すごい';
--  description | score
-- -------------+-------
--  すごい！    |     1   ← 0じゃない！
-- (1 row)
```

# PGroonga: 3

マルチバイトな
カラム名を
サポート

# マルチバイトなカラム名

- 例：日本語のカラム名
- 制限：UTF-8のみサポート

# 使い方

```
-- 日本語テーブル名
CREATE TABLE メモ (
  内容 text -- 日本語カラム名
);
CREATE INDEX 内容検索
  ON メモ
  USING pgroonga (内容);
```

# PGroonga: 4

# 類似文書検索サポート

# 使い方：定義

```
CREATE TABLE memos (content text);
CREATE INDEX content_search
  ON memos
  USING pgroonga (
    content
    -- v2オペレータークラスを指定
    pgroonga.text_full_text_search_ops_v2
  ) -- 形態素解析ベースのトークナイザーを指定
  WITH (tokenizer='TokenMecab');
```

# 使い方：検索

```
INSERT INTO memos VALUES
  ('全文検索が速い'),
  ('更新が速い'),
  ('集計が速い');
SELECT * FROM memos
  -- 類似文書検索用演算子は「&~?」
  WHERE content &~? '全文検索どう？';
--      content
-- ---------------
--  全文検索が速い
-- (1 row)
-- ↑同じ話題。「どう？」は含んでいない。
```

# PGroonga: 5

## 前方一致検索サポート

# 使い方：定義

```
-- タグをヨミガナで検索
CREATE TABLE tags (reading text);
CREATE INDEX reading_search
  ON tags
  USING pgroonga (
    reading
    -- ↓のオペレータークラスを指定
    pgroonga.text_term_search_ops_v2
  );
```

# 使い方：検索

```
INSERT INTO tags VALUES
  ('ゼンブンケンサク'),
  ('ゼンポウイッチ'),
  ('シュウケイ');
SELECT * FROM tags
  -- 前方一致用演算子は「&^」
  WHERE reading &^ 'ゼン';
--       reading
-- --------------------
--  ゼンブンケンサク
--  ゼンポウイッチ
```

# PGroonga: 6

# 前方一致RK検索サポート

# RK：ローマ字・カナ

- ローマ字（zen）で
  カナ（ゼンブンケンサク）を
  前方一致検索
  - = 前方一致RK検索
- 用途：入力補完

# 使い方：検索

```
-- データは前方一致検索と同じ
SELECT * FROM tags
  -- 前方一致RK用演算子は「&^~」
  -- 「ゼン」でも「ぜん」でも同じ結果
  WHERE reading &^~ 'zen';
--       reading
-- --------------------
--  ゼンブンケンサク
--  ゼンポウイッチ
```

# PGroonga: 7

pgroonga.highlight_html追加

# pgroonga.highlight_html

- キーワードをハイライト
- 用途：検索結果の表示

# 使い方：基本

```
SELECT pgroonga.highlight_html(
  'Groongaは速い',    -- ハイライト対象
  ARRAY['Groonga']); -- キーワードは複数指定可
--                   highlight_html
-- ---------------------------------------
--  <span class="keyword">Groonga</span>は速い
-- (1 row)
-- ↑キーワードは<span>で囲まれる。
```

# 使い方：応用

```
SELECT pgroonga.highlight_html(
  'Groongaは速い',   -- ハイライト対象
  -- クエリーからキーワードを抽出
  pgroonga.query_extract_keywords(
    'Groonga OR PostgreSQL')
);
--                  highlight_html
-- ------------------------------------------
--  <span class="keyword">Groonga</span>は速い
-- (1 row)
-- ↑キーワードは<span>で囲まれる。
```

# PGroonga: 8

# ヒット件数の見積サポート

# ヒット件数の見積

- 実行計画決定に利用
  - 見積精度が上がると
    適切な計画を選びやすくなる
  - →速くなる！

# PGroonga: 9

# ストリーミング
# レプリ
# ケーション
# サポート

# レプリケーション

- リアルタイムでデータコピー
- 用途例：検索性能向上
  - 検索性能が足りない！
  - →検索専用ノード追加で対応

https://pgroonga.github.io/ja/reference/replication.html

# PGroonga: 10

# Zstandard サポート

# 使い方

- ユーザーはなにもしなくてよい
- PGroongaが自動で使う
  - テキストカラムで利用

# PGroonga: 11

# 生Groonga検索サポート強化

# 生Groonga検索

- PostgreSQLを介さずにGroongaで検索
  - pgroonga.commandを使う
- 課題：セキュリティー
  - インジェクション怖い…

# セキュリティー対策

## 自動エスケープ

# 使い方

```
SELECT pgroonga.command(
  'select',
  ARRAY[ -- 配列で渡すと自動エスケープ
    'table', pgroonga.table_name('memos'),
    -- ...,  ↓自動エスケープ
    'query', 'ユーザーからの入力'
  ]);
```

# PGroonga: まとめ

- 便利になった！
- 速くもなった！
  - Groongaが速くなったから

# 2016年のまとめ

- Groongaが凄くよくなった
  - 速くもなったし便利にもなった
- Mroongaの本ができた
  - https://grnbook-ja.tumblr.com/
    by @KitaitiMakoto
- PGroongaも凄くよくなった

# 2017年の抱負

データ分析に
活用したい！

# データ分析に活用？

- 分析システムの一部でGroonga
  - 特徴を活かせるのでは！？
- Groongaの特徴
  - リアルタイムな自然言語処理
  - 高速なフィルター・集計処理

データの前処理・選別にどう？

（一緒に活用方法から考えていきたい！）

# データ分析への最初の一歩

- Cythonでバインディングを開発
  - Pythonで使えると分析システムで使いやすいかも？
- 興味がある人たちで集まる
  - 手を動かす人たちがよさそう？