Geographical Searching

-全文検索エンジンgroonga を囲むタベ #2-

Gurunavi, Inc.
塩畑公一
2011/11/29

groonga との歩み その壱

2008年06月 ～

新規検索システム構築プロジェクト開始

商用パッケージからオープンソース化

a. ファセット分類機能
b. HTTP によるQuery 操作機能
c. 緯度経度範囲検索機能

2010年01月 ～

senna 後継検索エンジン、groonga が誕生

有限会社未来検索ブラジル様協力のもと、各種機能を開発
パフォーマンス向上を目指す

groonga との歩み その弐

2010年04 ～ 現在

1. groonga を利用したサービス開始

   リリース後も協力関係を継続し、新規機能開発やパフォーマンス向上に従事

2. 弊社内での主な利用コンテンツ

   a. レストラン検索
   b. 地図検索
   c. 駅検索
   d. GPS 検索 (モバイル)
   etc...

緯度経度検索機能の実現 その壱

□ 緯度経度検索とは

二点の座標から形成される範囲以内を対象としたレコードを検索

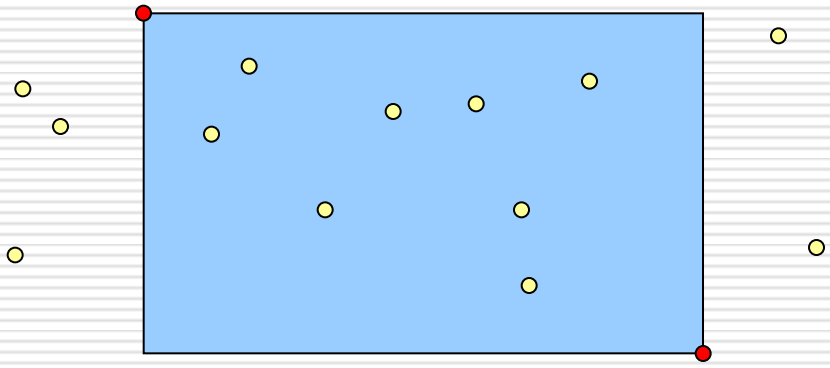
※ groonga の機能として、**矩形**と**円形**にて対応

緯度経度検索機能の実現 その弐

□ 矩形による範囲検索

左上と右下の座標から形成される矩形以内に存在するデータを検索

図例 – 1.



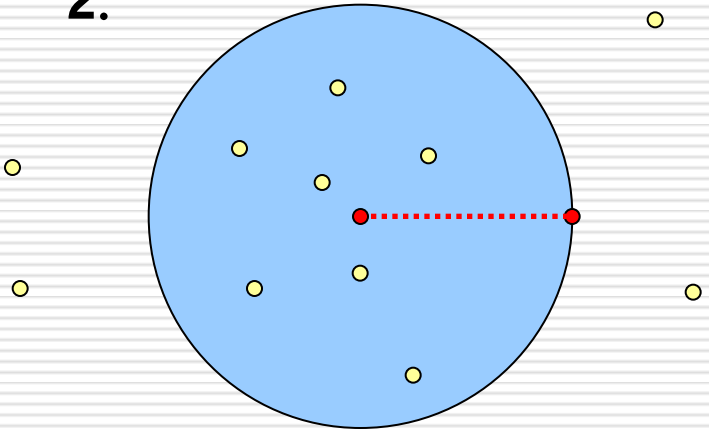
geo_in_rectangle () 関数にて実現

緯度経度検索機能の実現 その参

□ 円形による範囲検索

中心と半径から形成される円形以内に存在するデータを検索

図例 – 2.



geo_in_circle () 関数にて実現

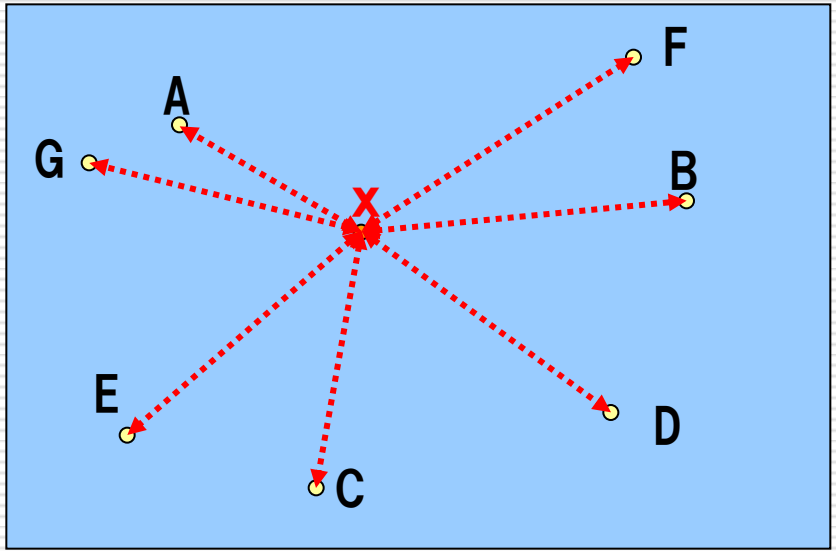
緯度経度検索機能の実現 その四

□ 距離について

指定された一点の座標から検索結果対象が保持する座標までの距離

※ groonga での取り扱い単位はm（メートル）

図例 － 3.



座標点 x から各検索データが保持している座標までの距離を算出できる

※ 図例 － 3. は矩形だが、
円形でも可能

緯度経度検索機能の実現 その伍

□ 距離の計算手法について　（壱）

三つの手法にて距離を算出

a. 方形近似

平面地図上にて距離を算出する手法

メリット）

アルゴリズムがシンプルで計算速度が速い

→ 三平方の定理

デメリット）

精度の高い距離算出ができない

geo_distance () 関数にて実現

緯度経度検索機能の実現 その伍

□ 距離の計算手法について　（弐）

b. 球面近似

球形地図（e.g. 地球儀）上にて距離を算出する手法

↓

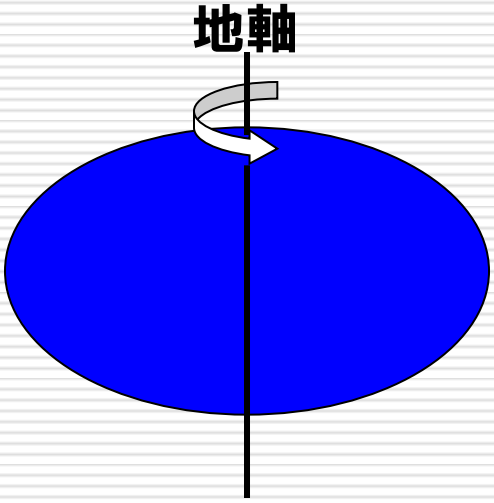
geo_distance2 () 関数にて実現

緯度経度検索機能の実現 その伍

□ 距離の計算手法について　（参）

c. ヒュベニ

楕円体上にて距離を算出する手法
※ 地球は自転の遠心力により、
　楕円体となっている為



メリット）
　精度の高い距離計算が可能

デメリット）
　複雑な計算式を用いる為、計算速度が遅い

geo_distance3 () 関数にて実現

緯度経度検索機能の実現 その陸

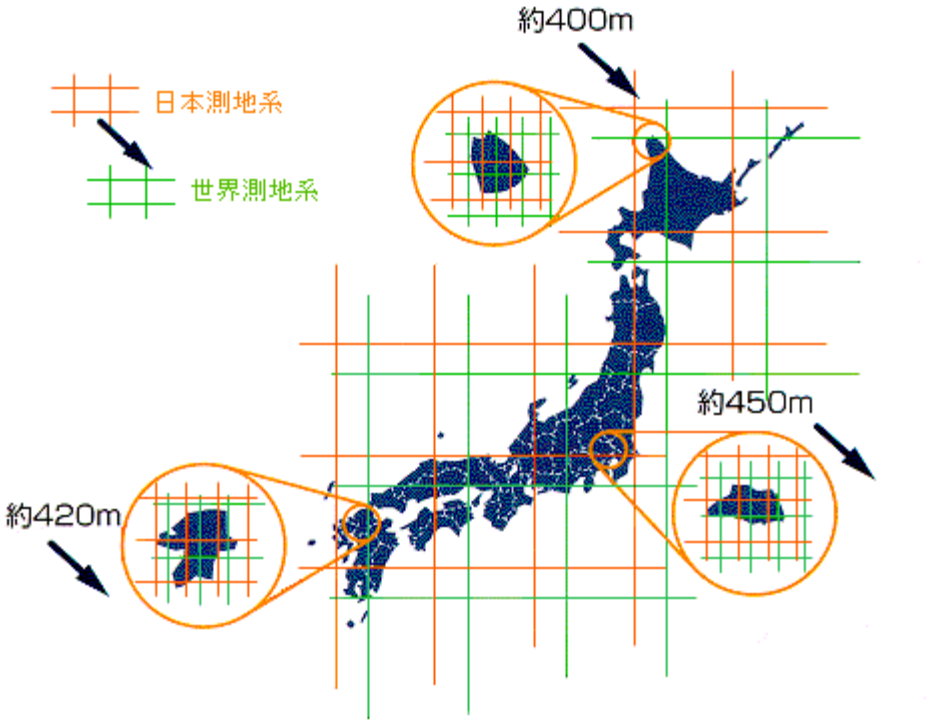
□ 測地系について　（壱）

日本測地系と世界測地系といわれる二系統の座標の存在

それぞれの座標では、地域によって差異が生じる

e.g.）

北海道稚内市
東京都近辺
福岡県近辺



緯度経度検索機能の実現 その陸

□ 測地系について　(弐)

二つの測地系座標を利用した検索

a. 日本測地系座標

1. 日本各地に設置された基準点から日本天文台が作り上げた測地基準系座標
   - ベッセル楕円体を準拠楕円体として扱う
   - 日本周辺でのみ利用可能
   - 2002年04月以前まで使用されていた

   e.g.)

   Yahoo! Japan 地図情報、Mapion、Mapfan etc...

2. TokyoGeoPoint 型にて対応

緯度経度検索機能の実現 その陸

□ 測地系について　(参)

b. 世界測地系座標

1. WGS84 (World Geodetic System 1984 の略) を採用
   - GPS 等にて使用されている測地基準系座標
     → GPS からのフィードバックにて精度を向上
   - GRS 楕円体を準拠楕円体として扱う
   - 世界標準として扱える

     e.g.)

     Google Maps etc...

2. WGS84GeoPoint 型にて対応

設定・使用方法について その壱

□ DDL の構成について（壱）

groonga にて緯度経度検索を行う為には、下記の様な
DDL の設定となる

- 2010年04月時点

```
create_table     gnavi          TABLE_HASH_KEY    ShortText
column_create    gnavi name     COLUMN_SCALAR     ShortText
column_create    gnavi lct_wgs  COLUMN_SCALAR     WGS84GeoPoint
```

HASH 型のテーブルにより検索速度を向上させ、
WGS84GeoPoint 型のカラムに対して、検索を実施

期待していた検索速度が出なかった

設定・使用方法について その壱

□ DDL の構成について（弐）

転置インデックス用テーブルを追加

- 2010年08月以降

```
create_table     gnavi          TABLE_HASH_KEY      ShortText
column_create    gnavi  name    COLUMN_SCALAR       ShortText
column_create    gnavi  lct_wgs COLUMN_SCALAR       WGS84GeoPoint

create_table             wgs    TABLE_PAT_KEY WGS84GeoPoint
column_create    wgs     index  COLUMN_INDEX gnavi  lct_wgs
```

緯度経度用のカラムに転置インデックスを施し、
検索速度の向上を図る

パフォーマンスについて その壱

テスト環境について

パフォーマンステストに利用したサーバスペックやデータ数

CPU　　：Intel(R) Xeon(R) 2.00GHz x4
メモリ　　：8GB
総データ数　：約54万件

パフォーマンスについて その弐

Query パラメータについて

指定座標から半径1km（1000m）以内のレコードを検索した場合

```
$ /usr/local/bin/groonga
> --log-path /var/log/groonga.log
> /db/gnavi.db
> select --table gnavi --offest 0 --limit 15
> --filter geo_in_circle(lct_wgs, "128418599x503159518", 1000)
> --scorer _score=geo_distance(lct_wgs, "128418599x503159518")
> --output_columns _key, name, _score
> --sortby _score
```

擬似カラム”_score”へ算出した距離を代入する事で、絞込み条件（範囲検索）とは独立した形で、距離の表示やソートが可能な仕様となっている

Cf.）--sortby geo_distance() とする事も可

パフォーマンスについて その参

□ パフォーマンス比較結果

	2010年04月Ver.	2010年08月Ver.
総データ数	：約54万件	
ヒット件数	：3,734件	
レスポンス時間	：約0.34秒	：約0.03秒

約90%のパフォーマンスアップを実現!!