SQL入門

データベースと情報検索　第3回

横浜市立大学　坂巻顕太郎
kentaro.sakamaki@gmail.com

SQL

83の言語版

ページ　ノート　　閲覧　編集　履歴表示

出典: フリー百科事典『ウィキペディア（Wikipedia）』

目次 [非表示]

ページ先頭

この項目では、データベース言語について説明しています。マーケティング活動の文脈での利用については「潜在顧客#リードの品質認定」をご覧ください。

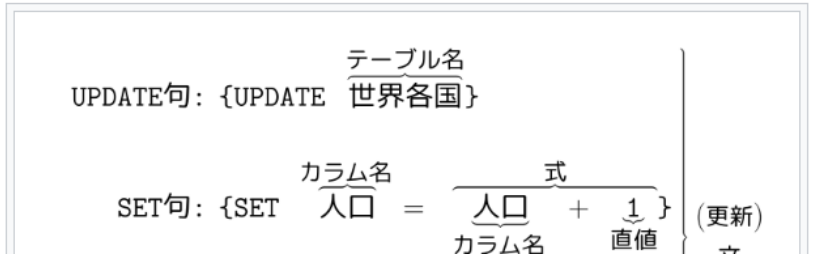
SQL（エスキューエル[1][ɛ́s kjuːɛ́l] (音声ファイル)、シークェル[1][ś ikwəl] (音声ファイル)、シーケル[2]）は、関係データベース管理システム (RDBMS) において、データの操作や定義を行うためのデータベース言語（問い合わせ言語）、ドメイン固有言語である。プログラミングにおいてデータベースへのアクセスのために、他のプログラミング言語と併用される。

SQLが使われるRDBは「エドガー・F・コッドによって考案された関係データベースの関係モデルにおける演算体系である、関係代数と関係論理（関係計算）に基づいている」と宣伝されていることが多い。しかし、SQLについては、そのコッド自身をはじめ他からも、関係代数と関係論理にきちんと準拠していないとして批判されてはいる（The Third Manifesto - クリス・デイト、ヒュー・ダーウェン）。

SQL

パラダイム	宣言型
登場時期	1974年
設計者	レイモンド・F・ボイス (英語版) ドナルド・D・チェンバリン
最新リリース	SQL:2016 (2016) /

テンプレートを表示



標準SQL規格 [編集]

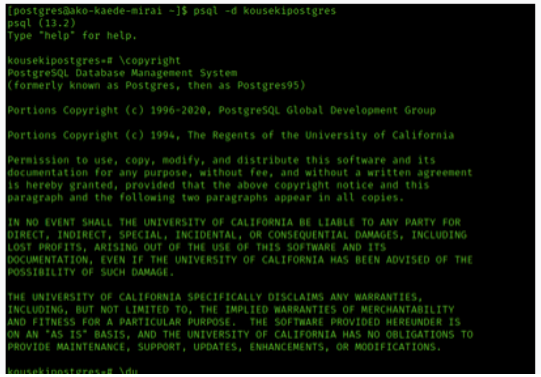
PostgreSQL

出典: フリー百科事典『ウィキペディア（Wikipedia）』

PostgreSQL（ポストグレス キューエル[※ 1]）は、拡張性とSQL準拠を強調するフリーでオープンソースの関係データベース管理システム（RDBMS）である。Postgresとしても知られている。もともとは、カリフォルニア大学バークレー校で開発されたIngresデータベースの後継としてその起源を根拠としたPOSTGRESという名前であった。1996年に、プロジェクトはSQLのサポートを反映してPostgreSQLに改名された。2007年の検討の結果、開発チームはPostgreSQLという名前とPostgresという別名を維持することを決定した。

PostgreSQLは、原子性、整合性、独立性、耐久性 (ACID)プロパティを持つトランザクション、自動更新可能なビュー、マテリアライズドビュー、トリガ、外部キー、ストアドプロシージャを特徴としている。単一マシンからデータウェアハウスや多数の同時使用ユーザを持つWebサービスまで、さまざまなワークロードを扱えるように設計されている。macOS Serverのデフォルトデータベースであり、Linux、FreeBSD、OpenBSD、Windowsでも利用可能である。

PostgreSQL



概要 [編集]

PostgreSQL 14.5文書

PostgreSQLグローバル開発グループ

目次

テーブルの作成

- CREATE TABLE ...

```
-- 天候に関するテーブル
CREATE TABLE weather (
    city  varchar(80),
    temp_lo       int,          -- 最低気温
    temp_hi       int,          -- 最高気温
    prcp          real,         -- 降水量
    date          date
);

-- 都市に関するテーブル
CREATE TABLE cities (
    name          varchar(80),
    location      point
);
```

Tips

- 複数の行に分けて入力できる
- セミコロンで終わるまでそのコマンドは継続していると認識する
- 空白文字（つまり空白、タブ、改行）を自由に使用できる
- キーワードと識別子に対して大文字小文字を区別しなし
  - ただし、識別子が二重引用符で括られていた場合は大文字小文字を区別する

型

- varchar(80)
  - 80文字までの任意の文字列を格納できるデータ型
- int
  - 一般的な整数用の型です
- Real
  - 単精度浮動小数点数値を格納する型
- date
  - 日付
  - （date型の列の名前をdateとするのは、わかりやすいかもしれないし、逆に混乱を招くかもしれず、名付け方は好みによる）

第8章 データ型

目次

8.8. 幾何データ型

幾何データ型は2次元空間オブジェクトを表現します。 表 8.20は、PostgreSQLで使用可能な幾何データ型を列挙したものです。

表8.20 幾何データ型

型名	格納サイズ	説明	表現
point	16バイト	平面における座標点	(x,y)
line	32バイト	無限の直線	{A,B,C}
lseg	32バイト	有限の線分	((x1,y1),(x2,y2))
box	32バイト	矩形	((x1,y1),(x2,y2))
path	16+16nバイト	閉経路（多角形に類似）	((x1,y1),...)
path	16+16nバイト	開経路	[(x1,y1),...]
polygon	40+16nバイト	多角形（閉経路に類似）	((x1,y1),...)
circle	24バイト	円	<(x,y),r>（中心と半径）

データの挿入1

- INSERT INTO …

```
INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11-27');
```

  - 単純な数値以外の定数は、単一引用符（‘）で括る
  - date型が受け付ける入力はかなり柔軟だが、曖昧さがない書式で書く

```
INSERT INTO cities VALUES ('San Francisco', '(-194.0, 53.0)');
```

  - point型は入力として座標の組み合わせが必要

データの挿入2

- 列を指定すれば、順番はスキーマと違ってもよい

```
-- すべての列に挿入
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
    VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');

-- prcpがわからない
INSERT INTO weather (date, city, temp_hi, temp_lo)
    VALUES ('1994-11-29', 'Hayward', 54, 37);
```

テーブルへの問い合わせ

- SELECT …

```
-- *は「全ての列」の省略形
SELECT * FROM weather;

-- 上と同じ
SELECT city, temp_lo, temp_hi, prcp, date FROM weather;
```

- 式の指定

```
SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date FROM weather;
```

  - AS句：使用した出力列（テーブル）の再ラベリング（AS句は省略可）

「条件付け」問い合わせ

- WHERE句

```
SELECT * FROM weather WHERE city = 'San Francisco' AND prcp > 0.0;
```

  - AND（OR、NOT）: 論理演算子

9.1. 論理演算子

通常の論理演算子が使用できます。

```
boolean AND boolean → boolean
boolean OR boolean → boolean
NOT boolean → boolean
```

SQLはtrue、false、そして「不明」を意味するnullの3値の論理システムを使用します。 以下の真理値表を参照してください。

a	*b*	*a* **AND** *b*	*a* **OR** *b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL
NULL	NULL	NULL	NULL

ソート

- 都市だけでソート

```
SELECT * FROM weather ORDER BY city;
```

- 都市の次に最低気温でソート

```
SELECT * FROM weather ORDER BY city, temp_lo;
```

重複の無いデータの抽出

- DISTINCT句

```
SELECT DISTINCT city FROM weather;
```

- DISTINCTとORDER BYを一緒に使用して一貫した結果を得る

```
SELECT DISTINCT city
    FROM weather
    ORDER BY city;
```

テーブル間を結合１

- JOIN句

```
SELECT * FROM weather JOIN cities ON city = name;
```

  - weatherテーブルの各行のcity列をcitiesテーブルの全ての行のname列と比較

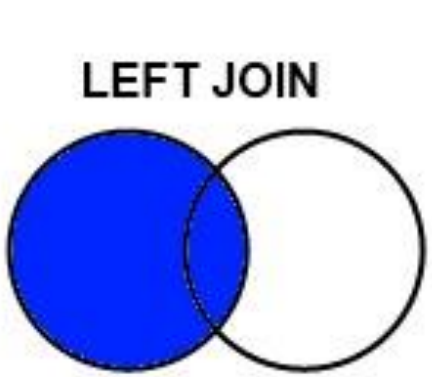
- 結果の解釈
  - Haywardの結果がないのはcitiesテーブルにHaywardに一致する項目がないから
    - 結合の際にweatherテーブル内の一致されなかった行は無視される
  - weatherテーブルとcitiesテーブルからの列のリストが連結されるため、都市名を持つ列が2つになる

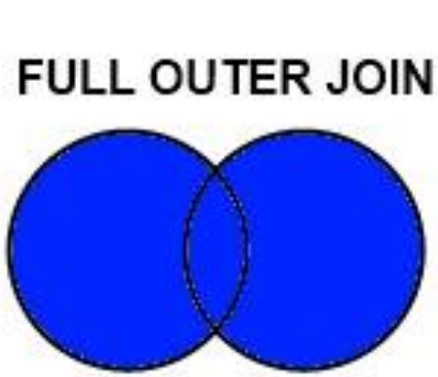
テーブル間を結合２

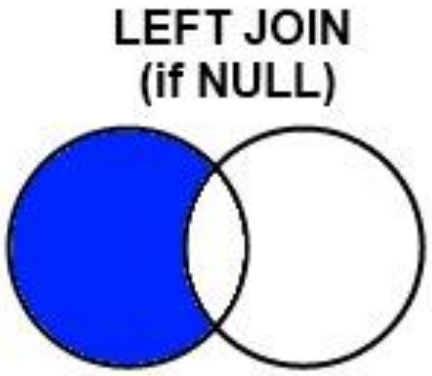
- 列名とテーブル名を明示

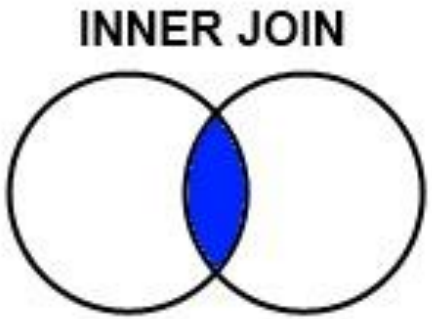
```
-- 必要な列名を列挙
SELECT city, temp_lo, temp_hi, prcp, date, location
    FROM weather JOIN cities ON city = name;

-- どのテーブルに入っているかを
SELECT weather.city, weather.temp_lo, weather.temp_hi, weather.prcp, weather.date,
    cities.location FROM weather JOIN cities ON weather.city = cities.name;
```

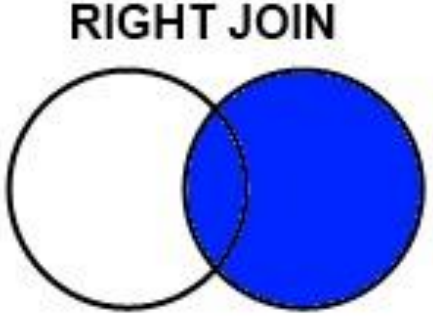
結合の種類

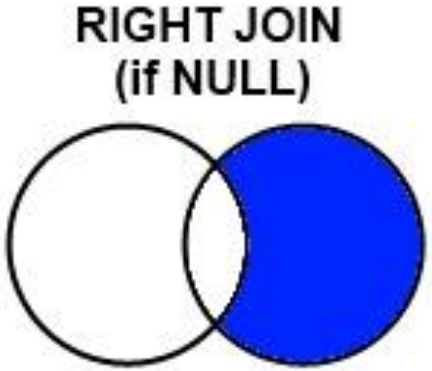

LEFT JOIN


FULL OUTER JOIN


LEFT JOIN
(if NULL)


INNER JOIN


RIGHT JOIN


RIGHT JOIN
(if NULL)

左外部結合

- LEFT OUTER JOIN句

```
SELECT * FROM weather LEFT OUTER JOIN cities ON weather.city = cities.name;
```

  - 結合演算子の左側に指定したテーブルの各行が最低でも一度出力される一方で、右側のテーブルでは左側のテーブルの行に一致するもののみが出力される
  - 右側のテーブルに一致するものがない左側のテーブルの行を出力する時、右側のテーブルの列は空の値（NULL）で置換される

集約関数の例

- max集約

```
SELECT max(temp_lo) FROM weather;
```

- 集約関数はWHERE句で使用することができない

```
SELECT city FROM weather WHERE temp_lo = max(temp_lo);    --間違い
```

  - WHERE句はどの行を集約処理に渡すのかを決定するもので、集約関数の演算を行う前に評価されなければならない

集約関数

関数	
SUM	引数の総和を求める（NULLの場合は集計対象外）
MAX	引数の最大値を求める
MIN	引数の最小値を求める
AVG	引数の平均値を求める（NULLの場合は集計対象外）
COUNT	引数の値の総数を求める（NULLの場合は集計対象外）COUNT(*)とすべての列を扱うことも可能

副問い合わせ

- クエリ内にクエリを書く

```
SELECT city FROM weather WHERE temp_lo = (SELECT max(temp_lo) FROM weather);
```

グループごとに集約

- GROUP BY句
  - 指定した列名（または列名を含んだ式）によるグループごとに集約

```
SELECT city, max(temp_lo) FROM weather GROUP BY city;
```

  - 列別名は使用できない
- HAVING句
  - GROUP BY句で集計した結果に対して抽出条件を指定する際に使用する

```
SELECT city, max(temp_lo) FROM weather GROUP BY city HAVING max(temp_lo) < 40;
```

WHEREとHAVINGの基本的な違い

- WHERE
  - グループや集約を演算する前に入力行を選択する（どの行を使用して集約演算を行うかを制御する）
  - 集約関数を持つことはできない

- HAVING
  - グループと集約を演算した後に、グループ化された行を選択する
  - 常に集約関数を持つ

更新

- UPDATEコマンド

```
UPDATE weather
    SET temp_hi = temp_hi - 2,  temp_lo = temp_lo - 2
    WHERE date > '1994-11-28';
```

削除

- DELETEコマンド

```
-- Haywardに関するデータの削除
DELETE FROM weather WHERE city = 'Hayward';

-- すべてのデータの行が削除される
-- DELETE * FROM weather;
```

Tableに保存

- CREATE文に副問い合わせ

```
CREATE TABLE whether2 as SELECT city, max(temp_lo) FROM weather GROUP BY city;
```

課題

- iris.csvを読み込み、種類ごとに平均などを計算せよ
  - プログラムをmanabaから提出