

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

目次 [非表示]

ページ先頭

標準SQL規格

SQLとオンライン処理

SQLとバッチ処理

SQLの対話的実行

SQL文法

主な SQL

注釈

出典

参考文献

関連項目

SQL

文A 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）』

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][*ɛs kjuː.əl*]^[1] 音声ファイル）、シーケル^[1][*sɪkwəl*]^[2] 音声ファイル）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**）。

標準SQL規格 [編集]

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

UPDATE句: {UPDATE

テーブル名
世界各国

}

SET句: {SET

カラム名
人口

 =

式

人口

 +

直値
1

 } (更新)

PostgreSQL

📄 46の言語版 ▼

目次 [非表示]

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）』

ページ先頭

- 概要
- 特徴
- 基本的機能
- 性能
- 周辺ツール
- 歴史

注目すべきユーザー

受賞

注釈

出典

参考書籍

外部リンク

PostgreSQL（ポストGRES 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う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。

概要 [編集]

PostgreSQL



```
postgres@eko-kaede-miral:~$ psql -d housekipostgres
psql (13.2)
Type "help" for help.

housekipostgres=# \copyright
PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright (c) 1996-2020, PostgreSQL Global Development Group
Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose, without fee, and without a written agreement
is hereby granted, provided that the above copyright notice and this
paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING
LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS
DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO
PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

housekipostgres=# \q
```

[他のバージョンの文書](#)

[PostgreSQL 14.5文書](#)

[次へ](#)

PostgreSQL 14.5文書

PostgreSQLグローバル開発グループ

製作著作 © 1996–2022 PostgreSQL グローバル開発グループ

[法的告知](#)

目次

[はじめに](#)

- [1. PostgreSQLとは?](#)
- [2. PostgreSQL小史](#)
- [3. 規約](#)
- [4. より進んだ情報](#)
- [5. バグレポートガイドライン](#)

[I. チュートリアル](#)

- [1. さあ始めましょう](#)

2. SQL 言語

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)

[他のバージョンの文書](#)

[PostgreSQL 14.5文書](#)

[パート II. SQL言語](#)

第8章 データ型

[前へ](#)[上へ](#)[次へ](#)

第8章 データ型

目次

[8.1. 数値データ型](#)

[8.1.1. 整数データ型](#)

[8.1.2. 任意の精度を持つ数](#)

[8.1.3. 浮動小数点データ型](#)

[8.1.4. 連番型](#)

[8.2. 通貨型](#)

[8.3. 文字型](#)

[8.4. バイナリ列データ型](#)

[8.4.1. byteaのhex書式](#)

[8.4.2. byteaのエスケープ書式](#)

[8.5. 日付/時刻データ型](#)

PostgreSQL 14.5文書

第8章 データ型

8.8. 幾何データ型

[前へ](#)[上へ](#)[次へ](#)

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

[他のバージョンの文書](#)

PostgreSQL 14.5文書

[第9章 関数と演算子](#)

9.1. 論理演算子

[前へ](#)[上へ](#)[次へ](#)

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。

<i>a</i>	<i>b</i>	<i>a AND b</i>	<i>a OR b</i>
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

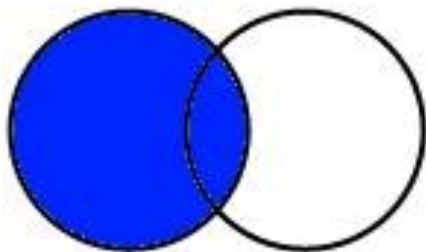
- SELECT city, temp_lo, temp_hi, prcp, date, location
FROM weather JOIN cities ON city = name;

- どのテーブルに入っているかを

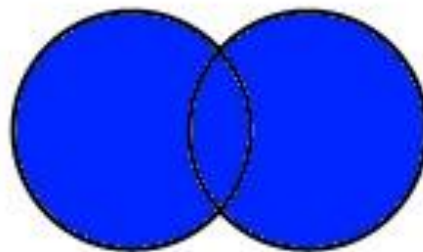
- SELECT weather.city, weather.temp_lo, weather.temp_hi, weather.prcp, weather.date,
cities.location FROM weather JOIN cities ON weather.city = cities.name;

結合の種類

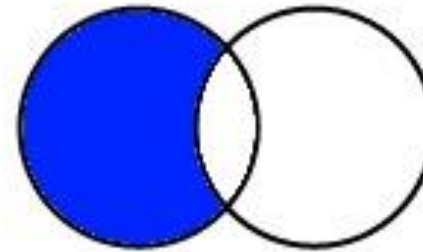
LEFT JOIN



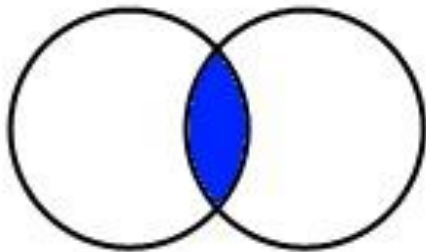
FULL OUTER JOIN



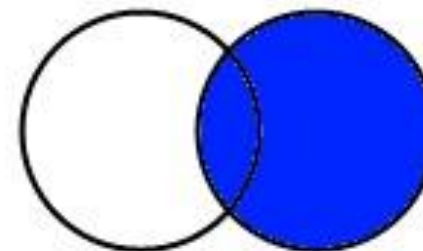
LEFT JOIN
(if NULL)



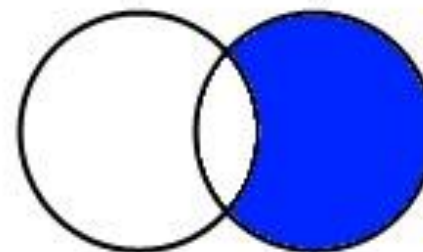
INNER JOIN



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句で使うことができない

```
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