

SQL勉強会

第1回 基礎編

本日のお題目

- 対象者、勉強会のゴール
- なぜSQLを学ぶのか
- SQLとは？
- SQLハンズオン

対象者、ゴール

対象者

- SQLの勉強かーと思っている...が、環境も無いし...つってやっていない人
- SQL、、あ———SQLね。はいはい。。。SQLねう——ん。っていう人
- データ集計はエクセルを使ってやっているエクセラー

勉強会のゴール

- SQLを実務で使ってみることができるようになる。
- SQLで欲しいデータが取れるようになる。
- 他の人の書いたSQLがなんとなく読めるようになる。
- こういうのあったな一位の事を脳にとどめておいて調べることでSQLがかけられるようになる。

勉強会の進め方

- ハンズオン形式。
- 話を聞くだけではなく必ず自分でSQLを書いて結果を見る。
- 解答ものせますが、まず自分で考えて書いてみる事

1回目は基礎編です。

JOINやサブクエリー、UNIONやWindow関数

みたいな概念を一切扱わず内容をだいぶ絞っています。

(つまりRDBだけどリレーションという概念を置いてけぼりにしている)

複数回に分けて開催します。

次回開催は2ヶ月以内を目指しています。

あなたは誰？

齋藤 晃 / Saito Akira

- 新潟生まれ新潟育ち。エンジニア？歴15年？
- プログラムは面倒だからなるべく書きたくないタイプの人です。
- 唯一の自慢は30歳から3年で174kg → 94kgに減量したことです。(現在120kgにリバウンド)
- 趣味はバイクとキャンプでしたが、もう5年近くほとんど乗ってないし行ってないのでそう言ってはいけない気がしている。

一応やったことがある業務

(システム開発系は基本的に全部ネットショップ向け) Windows業務アプリ開発 / システム間データ連携 / Web業務アプリ開発 / Windows&Macのクロスプラットフォームアプリ開発 (Electron) / サーバ構築運用 (メール/ファイル/Web/VPN...等) / ヘルプデスク業務 / ハンズオンセミナー講師 / ネットショップ構築運営代行 / 印刷物作成 (チラシ・パンフレット・ポスター) / 社長の、お別荘のお清掃・お別荘のビザ窯設計・お別荘の監視カメラ設置運用 / 社長が開催する人生が成功できるセミナーへのサクラ参加 (自腹) / 工場の製造ラインへひたすら部品を供給するお仕事 / ドラム缶をひたすら洗うお仕事 / プレハブ小屋のパーツにひたすら断熱材を注入するお仕事 / 足場をひたすら解体するお仕事 / ハンターランク上げ

得意なこと

ない...

多分人並みにできる

SQLならなんとか ...

なんか仕事で触ったことがある程度のキーワードたち (色々つまんで触っているがどれも浅い理解 ...)

Linux(Redhat系) / Apache / shell / Swagger / TypeScript / C++/CLI / Java / GO / PHP / Redis / AWS(S3, EC2, SQS, SES, Lambda, RDS ポスグレ Aurora, Athena, DMS, Bedrock, ApiGateway) / React / Docker / Fluentd / GoogleCloud(GCE, GAE, CloudRun, GCS, BigQuery) / Laravel / FuelPHP / WordPress / JavaScript / Bootstrap / GitHub Actions / HTML / CSS / vim / nginx / Ansible / Mitamae / PostgreSQL / MySQL / SQLServer / FileMaker / Access / さくらのクラウド / Cloud N / GMO クラウド / Next.js / .NetFramework / Python / LlamaIndex / LangChain / C# / VB.Net / VB6 / ASP.NET / KUSANAGI / RESTAPI / SOAP-RPC / Windows/バッチ / Illustrator / Photoshop



なぜSQL

なぜSQLを学ぶのか

- データベースからデータを抽出、集計、分析することができます。
- データに基づいた意思決定により、業務効率化や生産性向上につながります。
- SQLを使ってデータ分析を行い、その結果を共有・議論できるようになります。
 - ビジネス部門とのコミュニケーションが円滑になり、要件定義などがスムーズに行えます。
- SQLスキルは多くのエンジニア職で求められており、キャリア開発に役立ちます。

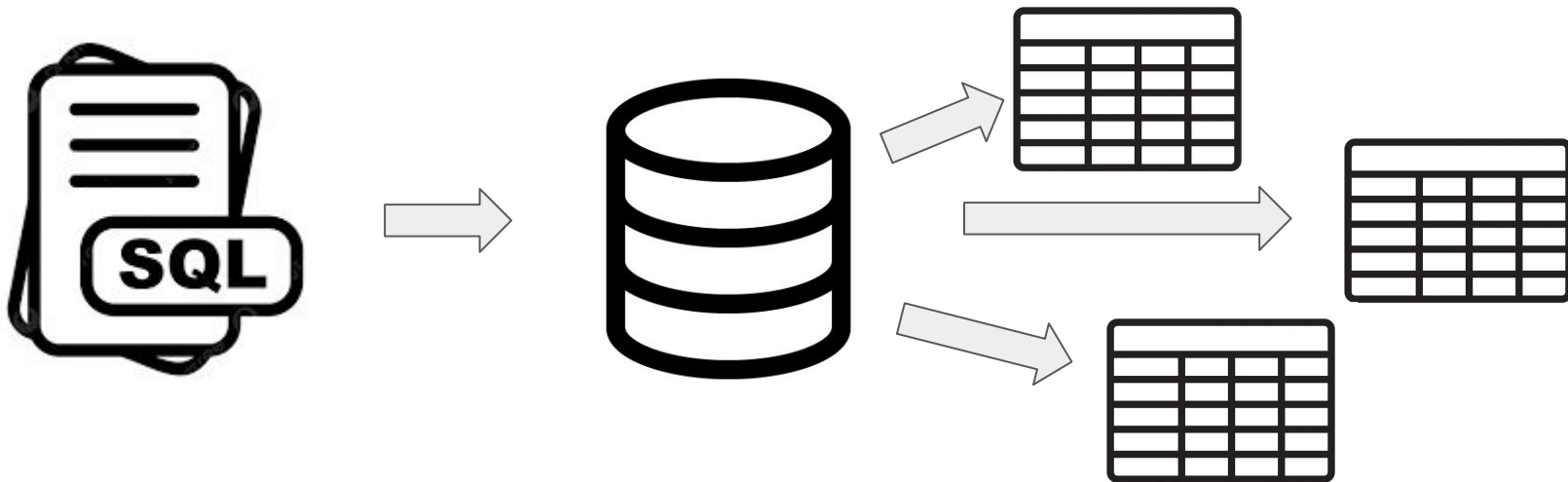
SQLとは？

SQLとは

- ストラクチャードクエリーランゲージの略(覚えなくていい)
- データベース上のデータを参照・操作するのに使用する

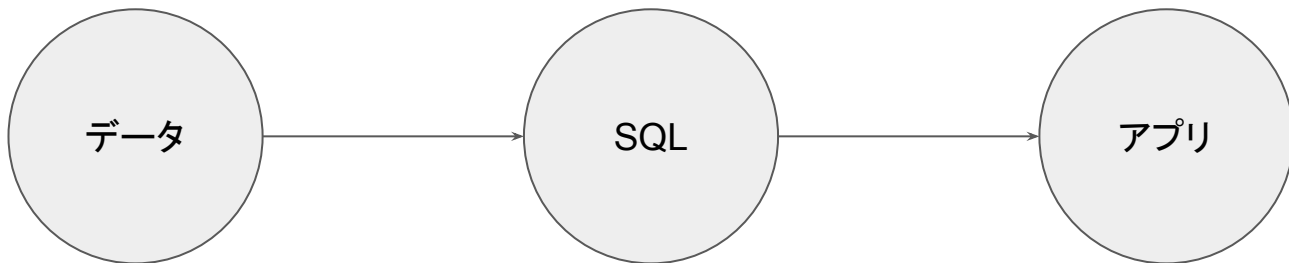
データベースとは

- ※ここではデータベース＝リレーショナルデータベースとして扱います。その他は一旦無視。
- データの集合体。
- データベースの中にはテーブルがありそこにデータが蓄積される。
- この世の中に、データを持たないシステムは存在しない。



データと密接にくっついているアプリ

- 「データ」とはある形式に揃えられている事実
- 「情報」とはデータを何らかの「目的」に沿って加工したもの。
- 「データ」から「目的」を使って「情報」を生み出すものがSQL
- SQLを使って生み出した「情報」をきれいにして見せるのがアプリ



データベースいろいろ

(多少の方言の違いはありますが)

いろいろあるデータベースに対して共通言語として使えるのがSQL



SQL初級(?)講座

まずは一つのテーブルを基本的な関数でコネコネできるようになろう。の会

その前に...

ハンズオン環境を用意していますのでサインアップしてログインして下さい。

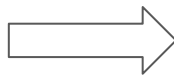
ブラウザでハンズオン環境のURLにアクセス。

「Sign up」からメールアドレスと自分の好きなパスワードを設定→Sign up

ログイン画面に戻るのでメールアドレスと設定したパスワードを入力して→Sign in

SQLPad

Sign up



SQLPad

Sign in

Sign Up

SQLの基本的な組み立て方

1. **取得したい情報の明確化**
 - 何を知りたいのか、どのような情報が必要かを明確にする
2. **対象となるテーブルの特定**
 - 必要な情報が格納されているテーブルを特定する
3. **列の選択**
 - 取得したい情報に対応した列を選択する
 - 必要な列だけを選択し、不要な列は除外する
4. **条件の設定**
 - 取得したい情報に合わせて、WHERE句などで条件を設定する
 - 条件を適切に組み合わせ、目的の情報を絞り込む
5. **集計・グループ化**
 - 集計関数を使って合計、平均などの集計を行う
 - GROUP BY句でグループ化し、集計結果を表示する
6. **並び替えの指定**
 - 取得結果を特定の順序で表示したい場合、ORDER BY句を使う

これから、SQLの書き方を説明するんですが...

①が最も重要、そして②がすぐわかる事が次に重要。

SQL基本構文

記述例

```
SELECT
  item_code,
  COUNT(*) AS '件数'
FROM orders
WHERE category = '食品'
GROUP BY item_code
ORDER BY '件数' DESC
```

SELECT	列の選択
FROM	テーブルの選択
WHERE	条件の設定
GROUP BY	集計のグループ化
ORDER BY	並び順

①SELECT 列の選択

とLIMIT

①-①まずは実行してみようSQL

```
SELECT  
    product_name,  
    unit_price  
FROM  
    products
```

product_name	unit_price
Product A	100.00
Product B	150.00
Product C	120.00
Product D	150.00
Product E	80.00
Product F	90.00
Product G	100.00
Product H	130.00
Product I	110.00

まずは実行してみようSQL:解説

product_name	unit_price
Product A	100.00
Product B	150.00
Product C	120.00
Product D	150.00
Product E	80.00
Product F	90.00
Product G	100.00
Product H	130.00
Product I	110.00

SELECTで指定した列の情報が全部返ってくる。

以上。

①-②列指定は省略できる。が...

```
SELECT
  *
FROM
  products
```

0.01 seconds	30 rows	↓ .csv	↓ .xlsx	↓ .json	table 🔗
product_id	product_name	category	unit_price		
1	Product A	Electronics	100.00		
2	Product B	Office Supplies	150.00		
3	Product C	Electronics	120.00		
4	Product D	Home Goods	150.00		
5	Product E	Office Supplies	80.00		
6	Product F	Electronics	90.00		
7	Product G	Home Goods	100.00		
8	Product H	Office Supplies	130.00		
9	Product I	Electronics	110.00		
10	Product J	Home Goods	160.00		
11	Product K	Office Supplies	95.00		
12	Product L	Electronics	105.00		
13	Product M	Home Goods	140.00		
14	Product N	Office Supplies	125.00		
15	Product O	Electronics	115.00		
16	Product P	Home Goods	155.00		
17	Product Q	Office Supplies	85.00		
18	Product R	Electronics	95.00		

列指定は省略できる。が...: 解説

0.01 seconds 30 rows [.csv](#) [.xlsx](#) [.json](#) [table](#)

product_id	product_name	category	unit_price
1	Product A	Electronics	100.00
2	Product B	Office Supplies	150.00
3	Product C	Electronics	120.00
4	Product D	Home Goods	150.00
5	Product E	Office Supplies	80.00
6	Product F	Electronics	90.00
7	Product G	Home Goods	100.00
8	Product H	Office Supplies	130.00
9	Product I	Electronics	110.00
10	Product J	Home Goods	160.00
11	Product K	Office Supplies	95.00
12	Product L	Electronics	105.00
13	Product M	Home Goods	140.00
14	Product N	Office Supplies	125.00
15	Product O	Electronics	115.00
16	Product P	Home Goods	155.00
17	Product Q	Office Supplies	85.00
18	Product R	Electronics	95.00

すべての列データを取得したい場合は「*」アスタリスクを使う。

出力する項目が多くなると処理が重くなる。

SQLが複雑化したときに、、、あ`ああ`あ`——一つ
てなる。といった副作用があるので用法用量を守って使いましょう。

とりあえずテーブルの中身を眺めたいときはコレを使う。

* 注意点:なんかよくわからないけど見てみる場合、最後にLIMITを使うことで負荷を抑える事。

取得件数を抑えるLIMIT: 解説

0.019 seconds5 rows

[.csv](#)[.xlsx](#)[.json](#)

product_id	product_name	category	unit_price
1	Product A	Electronics	100.00
2	Product B	Office Supplies	150.00
3	Product C	Electronics	120.00
4	Product D	Home Goods	150.00
5	Product E	Office Supplies	80.00

LIMIT [数字]

で、本来取得される表の先頭何件までを表示するかを制限することができる。

負荷を抑えた状態でクエリを試したいときによく使う。
諸事情により必ず低負荷になるわけではない。
ただし、何も条件を指定しない(列指定は可)ただのSELECT文であれば必ず低負荷になる。

①-④ヘッダが分かりづらいな...ってとき

```
SELECT
    product_name AS 製品名,
    unit_price AS 単価
FROM
    products
;
```

0.012 seconds		30 rows	↓ .csv
製品名		:	単価
Product A			100.00
Product B			150.00
Product C			120.00
Product D			150.00
Product E			80.00
Product F			90.00
Product G			100.00
Product H			130.00
Product I			110.00
Product J			160.00
Product K			95.00

ヘッダが分かりづらいな...ってとき: 解説

0.012 seconds 30 rows [↓ .csv](#)

製品名	単価
Product A	100.00
Product B	150.00
Product C	120.00
Product D	150.00
Product E	80.00
Product F	90.00
Product G	100.00
Product H	130.00
Product I	110.00
Product J	160.00
Product K	95.00

AS を使うことで列に別名をつけることができる。
名前はわかりやすくつけましょう。

①-⑤定数値を表示することもできる

```
SELECT
  'サンプル' AS タイプ,
  product_name AS 製品名,
  unit_price AS 単価
from products;
```

0.012 seconds		30 rows	.csv	.xlsx	↓
タイプ	製品名	単価			
サンプル	Product A	100.00			
サンプル	Product B	150.00			
サンプル	Product C	120.00			
サンプル	Product D	150.00			
サンプル	Product E	80.00			
サンプル	Product F	90.00			
サンプル	Product G	100.00			
サンプル	Product H	130.00			
サンプル	Product I	110.00			
サンプル	Product J	160.00			
サンプル	Product K	95.00			
サンプル	Product L	105.00			
サンプル	Product M	140.00			

②集合関数

②-①細かいことは良いので使ってみよう集合関数

```
SELECT
  count(*),
  sum(salary),
  max(salary),
  min(salary),
  avg(salary)
from employees;
```

0.01 seconds	1 rows	↓ .csv	↓ .xlsx	↓ .json	table 🔗
count	⋮ sum	⋮ max	⋮ min	⋮ avg	⋮
50	2472500.00	55000.00	45000.00	49450.0000000000...	

使ってみよう集合関数: 解説

0.01 seconds	1 rows	↓ .csv	↓ .xlsx	↓ .json	table 🔗
count	sum	max	min	avg	
50	2472500.00	55000.00	45000.00	49450.000000000...	

集合関数とは、
複数のレコードを集計して結果を返す奴。

SUM	合計
COUNT	件数
AVG	平均
MAX	最大
MIN	最小

③GROUP BY グループ化

③-①やってみようGROUP BY

GROUP BYは、データを特定の列で「グループ化」するための機能です。

例えば、従業員テーブルがあり、部門ごとの平均給与を知りたい場合、GROUP BYを使うと便利。

通常、従業員テーブルから給与の平均を求めると、全従業員の平均給与が表示されます。

しかし、GROUP BYを使うと、部門ごとに平均給与を表示することができます。

```
SELECT
  department_id,
  avg(salary)
FROM employees
GROUP BY department_id;
```

0.01 seconds	4 rows	↓ .CSV
department_id	avg	
3	47125.0000000000...	
4	48458.3333333333...	
2	49038.461538461...	
1	52923.076923076...	

④DISTINCT

重複排除

試しにやってみようDISTINCT

普通にSELECT

```
SELECT  
  category  
FROM  
  products;
```

重複排除SELECT

```
SELECT  
  DISTINCT category  
FROM  
  products;
```

試しにやってみようDISTINCT

普通にSELECT

0.011 seconds	30 rows	↓ .CSV
category		
⋮		
Electronics		
Office Supplies		
Electronics		
Home Goods		
Office Supplies		
Electronics		
Home Goods		
Office Supplies		
Electronics		
Home Goods		
Office Supplies		

重複排除SELECT

0.013 seconds	3 rows	↓ .CSV
category		
⋮		
Office Supplies		
Electronics		
Home Goods		

④-① 什么时候使用...

```
SELECT
  COUNT(*),
  COUNT(category),
  COUNT(DISTINCT category)
FROM
  products;
```

categoryの種類の数を出したいな——。

と思って、COUNT(category)としてもproductsテーブルはcategoryが重複しているテーブルなので集計できません。

COUNT(*)もCOUNT(category)も同じ件数を返す。

(COUNT(category)という書き方はNULLを除外してカウントしたい場合に役立ちます)

そこでCOUNT(DISTINCT category)とするとcategoryが重複して出現しているテーブルで重複を排除した上で、その件数を集計できます。

0.009 seconds			1 rows	↓ .csv	↓ .xlsx	↓ .json
count	count_category	count_distinct_category				
30	30	3				

⑤WHERE

条件設定

WHEREとは

条件を指定して列を抽出したいときに使用する。

=: 等しい

>: 大きい

<: 小さい

>=: 以上

<=: 以下

<>: 等しくない

BETWEEN A AND B: AとBの間

IN: 一覧のいずれかに一致

LIKE: 文字列マッチ

AND: 2つ以上の条件、両方が一致

OR: 2つ以上の条件、どちらかが一致

NOT: 条件を否定

⑤-①使ってみようWHERE

全件数取得

```
SELECT
  COUNT(*)
FROM
  orders
;
```

注文日2024-04-01の件数取得

```
SELECT
  count(*)
FROM
  orders
WHERE
  order_date = '2024-04-01'
;
```

使ってみようWHERE

全件数取得

0.011 seconds	1 rows
count	:
36	

注文日2024-04-01の件数取得

0.01 seconds	1 rows
count	:
2	

⑤-②BETWEENの使い方

単価が100以上120以下の製品を抽出したい時

```
SELECT
    *
FROM
    products
WHERE
    unit_price >= 100
    AND unit_price <= 120
;
```

BETWEENを使うと。。。

```
SELECT
    *
FROM
    products
WHERE
    unit_price BETWEEN 100 AND 120
;
```

BETWEENの方がわかり易くないですか？？

⑤-③LIKEの使い方

WHEREの中であいまい検索したい ...
ってときに使います。
いわゆるワイルドカード。

%: 0文字以上の任意の文字列
_: 任意の1文字

ファーストネームがJで始まる従業員リスト。

```
SELECT
  *
FROM
  employees
WHERE
  first_name LIKE 'J%'
;
```

0.011 seconds

4 rows

[.csv](#)

[.xlsx](#)

[.json](#)

[table](#)

employee_id	first_name	last_name	email	department_id	hire_date	salary
1	John	Doe	john.doe@example.com	1	2020-01-01	50000.00
2	Jane	Smith	jane.smith@example.com	2	2021-03-15	45000.00
8	Jessica	Thompson	jessica.thompson@example.com	3	2019-11-01	49000.00
15	Jacob	Gutierrez	jacob.gutierrez@example.com	1	2018-11-01	52500.00

⑥ORDER BY 並び替え

⑥-①使ってみようORDER BY

ただのSELECT。

```
SELECT
    *
FROM
    products
ORDER BY
    unit_price
;
```

unit_priceの降順に並び替える。

```
SELECT
    *
FROM
    products
ORDER BY
    unit_price desc
;
```

⑥-②並び替える基準項目は複数指定できます。

hire_dateの昇順で並べ替えた状態で
salaryの降順で並びかえ。

```
SELECT  
    *  
FROM  
    employees  
ORDER BY  
    hire_date,  
    salary desc
```

何もつけない(またはASC)と昇順。
DESCをつけると降順。

昇順＝小さいから大きい

降順＝大きいから小さい

⑦HAVING

条件設定

条件設定=WHEREって言った

HAVINGは実行されるタイミングが違います。

実はこんなときWHEREでは条件設定できない

以下の、平均サラリーを求めるSQL

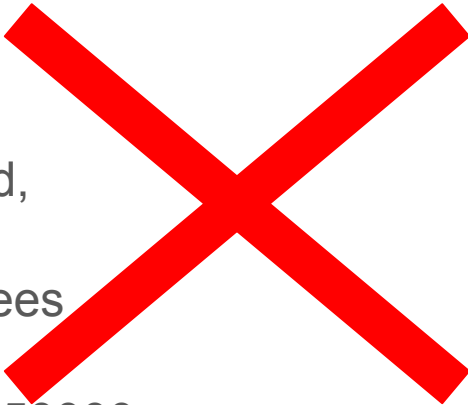
↓

```
SELECT
  department_id,
  avg(salary)
FROM employees
GROUP BY department_id;
```

サラリーが5万超の部署を取得したい

↓

```
SELECT
  department_id,
  avg(salary)
FROM employees
WHERE
  avg(salary) > 50000
GROUP BY department_id;
```



⑦-①そんな時にHAVINGの出番

以下の、平均サラリーを求めるSQL

↓

```
SELECT
    department_id,
    avg(salary)
FROM employees
GROUP BY department_id;
```

サラリーが5万超の部署を取得したい

↓

```
SELECT
    department_id,
    avg(salary)
FROM employees
GROUP BY department_id
HAVING avg(salary) > 50000;
```

なぜこうなるのか？

実はSQLには評価される順序があります。
素直に先頭から順番に評価されるわけではありません。

SQLの評価順序(左から右)



⑧CASE

条件分岐

条件分岐といえばIF

でもSQLにIFはありません。CASEを使います。

CASE式の基本構文

```
CASE
```

```
    WHEN 条件 THEN 結果
```

```
    WHEN 条件 THEN 結果
```

```
    ELSE 結果(どの条件にも一致しない時)
```

```
END
```


⑧-①使ってみようCASE

Salaryが50,000を超えている
データに「★」をつける。

```
SELECT
    case
        when salary > 50000 then '★'
        else "
    end,
    *
FROM
    employees;
```

実行結果

50,000を超えている
従業員に「★」がつけました。

0.012 seconds		50 rows		.csv	.xlsx	.json	table
case	employee_id	first_name	last_name	email	department_id	hire_date	salary
	1	John	Doe	john.doe@example.com	1	2020-01-01	50000.00
	2	Jane	Smith	jane.smith@example.com	2	2021-03-15	45000.00
★	3	Michael	Johnson	michael.johnson@example.com	1	2019-06-01	55000.00
	4	Emily	Brown	emily.brown@example.com	3	2022-02-01	48000.00
★	5	David	Lee	david.lee@example.com	2	2018-09-01	52000.00
	6	Sarah	Wilson	sarah.wilson@example.com	4	2021-04-15	47000.00
★	7	Robert	Anderson	robert.anderson@example.com	1	2020-07-01	53000.00
	8	Jessica	Thompson	jessica.thompson@example.com	3	2019-11-01	49000.00
	9	Daniel	Martinez	daniel.martinez@example.com	2	2022-01-01	46000.00
★	10	Olivia	Hernandez	olivia.hernandez@example.com	4	2018-05-01	51000.00
★	11	William	Gonzalez	william.gonzalez@example.com	1	2021-02-15	54000.00
	12	Sophia	Diaz	sophia.diaz@example.com	3	2020-09-01	47500.00
	13	Alexander	Reyes	alexander.reyes@example.com	2	2019-03-01	48500.00
	14	Isabella	Morales	isabella.morales@example.com	4	2022-06-01	50000.00
★	15	Jacob	Gutierrez	jacob.gutierrez@example.com	1	2018-11-01	52500.00
	16	Ava	Ramirez	ava.ramirez@example.com	3	2021-05-15	46000.00
	17	Benjamin	Castillo	benjamin.castillo@example.com	2	2019-08-01	49000.00
	18	Emma	Flores	emma.flores@example.com	4	2020-03-01	48000.00
★	19	Mason	Jimenez	mason.jimenez@example.com	1	2021-09-01	53000.00
	20	Abigail	Vargas	abigail.vargas@example.com	3	2018-02-01	47500.00
	21	Lucas	Rojas	lucas.rojas@example.com	2	2020-12-01	50000.00
	22	Mia	Ortiz	mia.ortiz@example.com	4	2019-06-15	48500.00
★	23	Ethan	Mendoza	ethan.mendoza@example.com	1	2021-03-01	52000.00
	24	Isabella	Ramos	isabella.ramos@example.com	3	2018-08-01	47000.00
	25	Noah	Salazar	noah.salazar@example.com	2	2020-05-01	49500.00

その他Tips

コメントが使えます

コメントは

-- (ハイフン2つ)

で表現します

紹介した集合関数以外にも使える関数は大量にあります

表 9-45. 汎用集約関数

関数	引数のデータ型	戻り値型	説明
array_agg(<i>expression</i>)	any	引数型の配列	配列に連結されたNULLを含む入力値
avg(<i>expression</i>)	smallint, int, bigint, real, double precision, numeric, or interval	整数型の引数であれば全てnumeric、浮動小数点の引数であればdouble precision、それ以外は引数のデータ型と同じ	全ての入力値の平均値（算術平均）
bit_and(<i>expression</i>)	smallint, int, bigint, or bit	引数のデータ型と同じ	全ての非NULLの入力値のビット積、非NULLの ければNULL
bit_or(<i>expression</i>)	smallint, int, bigint、または bit	引数のデータ型と同じ	全ての非NULLの入力値のビット和、非NULLの ければNULL
bool_and(<i>expression</i>)	bool	bool	全ての入力が真ならば真、そうでなければ偽
bool_or(<i>expression</i>)	bool	bool	少なくとも1つの入力値が真ならば真。そうでなければ偽
count(*)		bigint	入力行の数
count(<i>expression</i>)	全て	bigint	<i>expression</i> が非NULL値を持つ入力行の個数
every(<i>expression</i>)	bool	bool	bool_andと等価
max(<i>expression</i>)	全ての配列、数値、文字列、または日付時刻型	引数の型と同じ	全ての入力値にわたり <i>expression</i> の最大値
min(<i>expression</i>)	全ての配列、数値、文字列、または日付時刻型	引数の型と同じ	全ての入力値にわたり <i>expression</i> の最小値
string_agg(<i>expression</i> , <i>delimiter</i>)	(text, text)または(bytea, bytea)	引数と同じ型	入力された値が指定したデリミタで区切られた 列に連結されます。
sum(<i>expression</i>)	smallint, int, bigint, real, double precision, numeric、またはinterval	smallintまたはint型の引数であればbigint、bigint型の引数であればnumeric、浮動小数点の引数であればdouble precision、それ以外は引数のデータ型と同じ	全ての入力値に渡り <i>expression</i> の和
xmlagg(<i>expression</i>)	xml	xml	XML値の連結（ 項9.14.1.7 も参照）

表 9-46. 統計処理用の集約関数

関数	引数の型	戻り値の型	
corr(<i>Y</i> , <i>X</i>)	double precision	double precision	相関係数
covar_pop(<i>Y</i> , <i>X</i>)	double precision	double precision	母共分散
covar_samp(<i>Y</i> , <i>X</i>)	double precision	double precision	標本共分散
regr_avgx(<i>Y</i> , <i>X</i>)	double precision	double precision	独立変数の平均値 (sum(<i>X</i>)/ <i>N</i>)
regr_avgy(<i>Y</i> , <i>X</i>)	double precision	double precision	依存変数の平均値 (sum(<i>Y</i>)/ <i>N</i>)
regr_count(<i>Y</i> , <i>X</i>)	double precision	bigint	両式が非NULLとなる入力行
regr_intercept(<i>Y</i> , <i>X</i>)	double precision	double precision	(<i>x</i> , <i>y</i>)の組み合わせで決まる
regr_r2(<i>Y</i> , <i>X</i>)	double precision	double precision	相関係数自乗値
regr_slope(<i>Y</i> , <i>X</i>)	double precision	double precision	<i>x</i> , <i>y</i>)の組み合わせで決まる、
regr_sxx(<i>Y</i> , <i>X</i>)	double precision	double precision	sum(<i>X</i> ^2) - sum(<i>X</i>)^2/ <i>N</i> (依存
regr_sxy(<i>Y</i> , <i>X</i>)	double precision	double precision	sum(<i>X</i> * <i>Y</i>) - sum(<i>X</i>) * sum(<i>Y</i>)/ <i>N</i>
regr_syy(<i>Y</i> , <i>X</i>)	double precision	double precision	sum(<i>Y</i> ^2) - sum(<i>Y</i>)^2/ <i>N</i> (独立
stddev(<i>expression</i>)	smallint、int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	stddev_sampの歴史的な別名
stddev_pop(<i>expression</i>)	smallint、 int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	入力値に対する母標準偏差
stddev_samp(<i>expression</i>)	smallint、 int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	入力値に対する標本標準偏差
variance(<i>expression</i>)	smallint、 int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	var_sampの歴史的な別名
var_pop(<i>expression</i>)	smallint、 int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	入力値に対する母分散 (母標
var_samp(<i>expression</i>)	smallint、 int、 bigint、 real、 double precision、 またはnumeric	浮動小数点型の引数ではdouble precision。 それ以外ではnumeric	入力値に対する標本分散 (標

文字列を連結したいんです

例えば...

従業員情報にはfirst_nameとlast_nameが分かれて格納されてるんだけど...

結合された一つの情報としてほしいんだよな...

って場合。

「||」で文字列を結合することができます。

従業員情報から

結合されたnameとメールアドレスの一覧を取得

```
SELECT
```

```
  first_name || ' ' || last_name as name  
, email
```

```
FROM employees
```

このような文字列演算子、算術演算子は大量にあります

表 9-2. 算術演算子

演算子名	説明	例	結果
+	和	2 + 3	5
-	差	2 - 3	-1
*	積	2 * 3	6
/	商（整数の割り算では余りを切り捨て）	4 / 2	2
%	剰余（余り）	5 % 4	1
^	累乗	2.0 ^ 3.0	8
/	平方根	/ 25.0	5
/	立方根	/ 27.0	3
!	階乗	5 !	120
!!	階乗（前置演算子）	!! 5	120
@	絶対値	@ -5.0	5
&	バイナリのAND	91 & 15	11
	バイナリのOR	32 3	35
#	バイナリのXOR	17 # 5	20
~	バイナリのNOT	~1	-2
<<	バイナリの左シフト	1 << 4	16
>>	バイナリの右シフト	8 >> 2	2

表 9-5. SQL文字列関数と演算子

関数	戻り値型	
string string	text	文字列結合
文字列 非文字列 または、 非文字列 文字列	text	ひとつの非文字列のある入力の文字列結合
bit_length(string)	int	文字列中のビット数
char_length(string) または character_length(string)	int	文字列中の文字数
lower(string)	text	文字列を小文字に変換
octet_length(string)	int	文字列中のバイト数
overlay(string placing string from int [for int])	text	部分文字列の置換
position(substring in string)	int	指定した部分文字列の場所
substring(string [from int] [for int])	text	部分文字列の取り出し
substring(string from pattern)	text	POSIX正規表現に一致する副文字列を取り出す
substring(string from pattern for escape)	text	SQL正規表現に一致する部分文字列を取り出す
trim([leading trailing both] [characters] from string)	text	characters（デフォルトでは空白）で指定された文字列の両端を削除
upper(string)	text	文字列を大文字に変換

ランダムに行を取り出したいんだよね。。。

random()を使うと
ランダムサンプリングができます。

※ただし、postgresqlの場合。

random()は0以上1未満の値を返します。

例: 0.20442030777220532

これが1行毎に実行されて条件設定として評価されます。

この値が0.1未満となる確率は10%となるので

$\text{random()} < 0.1$

と、することで全体の10%の行を取得することができます。

注文情報全体の10%くらいをランダムに取得したい

```
SELECT
```

```
  *
```

```
FROM
```

```
  orders
```

```
WHERE
```

```
  random() < 0.1
```

この行だけは最初にして後は降順で。。。みたいな奴

従業員テーブルを給料昇順で並び替えたいんだけど、「ジョン」は必ず先頭に来て欲しい。。。

ORDER BY の条件で

first_name = 'John' DESC

と、することでジョンが先頭に來ます。

(DESCをつけないとジョンは最後に行きます)

```
SELECT
```

```
*
```

```
FROM
```

```
employees
```

```
ORDER BY
```

```
first_name = 'John' DESC
```

```
, salary
```

※ただしこれは postgresql の場合 (他の DB のことはわかりません)

GROUP BYとORDER BYの指定には数字も使えます

department_idでグループ化

```
SELECT
  department_id,
  avg(salary)
FROM employees
GROUP BY 1
```

従業員情報をsalaryの降順で並び替えて取得

```
SELECT
  email
  ,salary
FROM
  employees
ORDER BY 2 DESC
```

Q練習問題

Q-1: Employeesテーブルから全ての従業員の first_nameとlast_nameを選択します。
first_nameには「名」last_nameには「姓」と別名を付けて下さい。

0.01 seconds	50 rows	 .c
名	姓	
John	Doe	
Jane	Smith	
Michael	Johnson	
Emily	Brown	
David	Lee	
Sarah	Wilson	
Robert	Anderson	
Jessica	Thompson	
Daniel	Martinez	
Olivia	Hernandez	
William	Gonzalez	

Q-1: employeesテーブルから全ての従業員の first_nameとlast_nameを選択します。

回答例:

```
select first_name as 名, last_name as 姓 FROM employees;
```

Q-2: Productsテーブルからunit_priceの高い順で10商品表示して下さい。

0.011 seconds

10 rows

[↓ .csv](#)

[↓ .xlsx](#)

[↓ .json](#)

[table](#) [↗](#)

product_id	product_name	category	unit_price
10	Product J	Home Goods	160.00
16	Product P	Home Goods	155.00
28	Product BB	Home Goods	150.00
2	Product B	Office Supplies	150.00
4	Product D	Home Goods	150.00
22	Product V	Home Goods	145.00
13	Product M	Home Goods	140.00
20	Product T	Office Supplies	135.00
25	Product Y	Home Goods	130.00
8	Product H	Office Supplies	130.00

Q-2: Productsテーブルからunit_priceの高い順で10商品表示して下さい。

回答例:

```
SELECT * FROM products ORDER BY unit_price DESC LIMIT 10;
```


Q-3: ordersテーブルから、order_dateが2024年1月1日以降の注文を抽出してください。


0.01 seconds	17 rows	.csv	.xlsx	.json	table
order_id	customer_name	product_id	quantity	order_date	total_amount
20	XYZ Corp.	3	5	2024-01-01	600.00
21	Stark Enterprises	1	10	2024-01-15	1000.00
22	Acme LLC	5	3	2024-02-01	240.00
23	Globex Inc.	2	7	2024-02-10	1050.00
24	ABC Inc.	3	6	2024-03-01	720.00
25	XYZ Corp.	4	4	2024-03-15	600.00
26	Stark Enterprises	5	5	2024-04-01	400.00
27	Acme LLC	1	8	2024-04-10	800.00
28	Globex Inc.	2	6	2024-05-01	900.00
29	ABC Inc.	3	9	2024-05-15	1080.00
30	XYZ Corp.	4	3	2024-06-01	450.00
31	XYZ Corp.	4	4	2024-03-15	600.00

Q-3: ordersテーブルから、order_dateが2024年1月1日以降の注文を抽出してください。

回答例:

```
SELECT * FROM orders WHERE order_date >= '2024-01-01';
```

Q-4: ordersテーブルから、各 product_idごとの合計注文量を抽出して下さい。

0.01 seconds		5 rows	
product_id		sum	
3		53	
5		26	
4		36	
2		45	
1		68	

Q-4: ordersテーブルから、各product_idごとの合計注文量を抽出して下さい。

回答例:

```
SELECT product_id, SUM(quantity) FROM orders GROUP BY product_id;
```

Q-5: ordersテーブルから、合計注文量が 40以上のproduct_idを抽出して下さい。

0.015 seconds	3 rows
product_id	⋮
3	
2	
1	

Q-5: ordersテーブルから、合計注文量が 40以上のproduct_idを抽出して下さい。

回答例:

```
SELECT product_id FROM orders GROUP BY product_id HAVING SUM(quantity)
>= 40;
```

Q-6: employeesテーブルから、department_idが1の全ての従業員を選択します。

0.01 seconds	13 rows	↓ .csv	↓ .xlsx	↓ .json	table 🔗		
employee_id	first_name	last_name	email	department_id	hire_date	salary	
1	John	Doe	john.doe@example.com	1	2020-01-01	50000.00	
3	Michael	Johnson	michael.johnson@example.com	1	2019-06-01	55000.00	
7	Robert	Anderson	robert.anderson@example.com	1	2020-07-01	53000.00	
11	William	Gonzalez	william.gonzalez@example.com	1	2021-02-15	54000.00	
15	Jacob	Gutierrez	jacob.gutierrez@example.com	1	2018-11-01	52500.00	
19	Mason	Jimenez	mason.jimenez@example.com	1	2021-09-01	53000.00	
23	Ethan	Mendoza	ethan.mendoza@example.com	1	2021-03-01	52000.00	
27	Oliver	Suarez	oliver.suarez@example.com	1	2021-07-01	53500.00	
31	Alexander	Fernandez	alexander.fernandez@example.com	1	2021-12-01	54000.00	
35	Mason	Gutierrez	mason.gutierrez@example.com	1	2021-05-01	52500.00	
39	Ethan	Castillo	ethan.castillo@example.com	1	2021-10-01	53000.00	
43	Oliver	Rojas	oliver.rojas@example.com	1	2021-02-01	52000.00	
47	Alexander	Suarez	alexander.suarez@example.com	1	2021-06-15	53500.00	

Q-6: employeesテーブルから、department_idが1の全ての従業員を選択します。

回答例:

```
SELECT * FROM employees WHERE department_id = 1;
```


Q-7: ordersテーブルから、各 product_idごとの平均total_amountを選択します。

0.011 seconds	5 rows	↓ .csv	↓ .xlsx	↓ .json
product_id	avg			
3	900.0000000000000000			
5	297.1428571428571429			
4	675.0000000000000000			
2	964.2857142857142857			
1	971.4285714285714286			

Q-7: ordersテーブルから、各product_idごとの平均total_amountを選択します。

回答例:

```
SELECT product_id, AVG(total_amount) FROM orders GROUP BY product_id;
```

Q-8: employeesテーブルから、hire_dateが最も新しい従業員の first_nameとlast_nameを抽出します。

0.011 seconds		1 rows	↓ .csv
first_name	:	last_name	
Isabella		Morales	

Q-8: employeesテーブルから、hire_dateが最も新しい従業員の first_nameとlast_nameを抽出します。

回答例:

```
SELECT first_name, last_name FROM employees ORDER BY hire_date DESC  
LIMIT 1;
```

Q-9: ordersテーブルから、各 customer_nameごとの最大total_amountを抽出します。

0.012 seconds	5 rows	↓ .csv	↓ .xlsx	↓ .json
customer_name	max			
ABC Inc.	1200.00			
Stark Enterprises	1050.00			
Globex Inc.	1200.00			
Acme LLC	1200.00			
XYZ Corp.	1100.00			

Q-9: ordersテーブルから、各 customer_nameごとの最大total_amountを抽出します。

回答例:

```
SELECT customer_name, MAX(total_amount) FROM orders GROUP BY  
customer_name;
```

Q-10: ordersテーブルから、total_amountが500以上の注文を行っている customer_nameを抽出します。重複は排除して下さい。

0.009 seconds	5 rows	↓ .csv	↓ .xlsx
customer_name		⋮	
ABC Inc.			
Stark Enterprises			
XYZ Corp.			
Globex Inc.			
Acme LLC			

Q-10: ordersテーブルから、total_amountが500以上の注文を行っている customer_nameを抽出します。重複は排除して下さい。

回答例:

```
SELECT DISTINCT customer_name FROM orders WHERE total_amount >= 500;
```


Q-11 : ordersテーブルから、各 product_idごとの最小、最大、平均 total_amountを抽出します。

0.011 seconds	5 rows	↓ .csv	↓ .xlsx	↓ .json	table 🔗
product_id	⋮ min	⋮ max	⋮ avg	⋮	
3	600.00	1080.00	900.000000000000...		
5	160.00	400.00	297.14285714285...		
4	450.00	1200.00	675.000000000000...		
2	750.00	1200.00	964.28571428571...		
1	800.00	1200.00	971.42857142857...		

Q-11: ordersテーブルから、各 product_idごとの最小、最大、平均 total_amountを抽出します。

回答例:

```
SELECT product_id, MIN(total_amount), MAX(total_amount), AVG(total_amount)
FROM orders GROUP BY product_id;
```

Q-12: ordersテーブルから、各 customer_nameごとの注文数を抽出し、注文数が多い順に並べてください。ただし、結果は最初の 5行だけを表示します。

0.013 seconds	5 rows	↓ .csv	↓ .xlsx	↓
customer_name	count			
XYZ Corp.	8			
ABC Inc.	7			
Stark Enterprises	7			
Globex Inc.	7			
Acme LLC	7			

Q-12: ordersテーブルから、各 customer_nameごとの注文数を抽出し、注文数が多い順に並べてください。ただし、結果は最初の 5行だけを表示します。

回答例:

```
SELECT customer_name, COUNT(*) FROM orders GROUP BY customer_name  
ORDER BY COUNT(*) DESC LIMIT 5;
```

次回予告

SQL勉強会 第2回 （時期未定:2024-07末までに開催）

- JOIN(2つ以上のテーブルを扱う)
- サブクエリ
- UNION

SQL勉強会 第3回 (時期未定:2024-09末までに開催)

- Window関数
- その他変な関数

SQL勉強会 第4回 (時期未定:2024-11末までに開催)

- 実践的な課題を元に集計してみよう。の会。
- その他、分析における課題。