

「データベース演習」まとめ

学習内容：データベースを作成・操作する。PostgreSQLを使用する。

テキスト：SQL 第2版 ゼロからはじめるデータベース操作（翔泳社）

0. SQL の学習環境

■PostgreSQL のインストール

- <https://www.enterprisedb.com/downloads/postgres-postgresqldownloads#windows>より Windows 版の 9.5.21（テキストのバージョン）をインストールする
- Windows が 64 ビットか 32 ビットか注意する
- Windows10 では、コマンドプロンプトは Windows PowerShell という名前になっている場合もある
- 接続時に管理者パスワードの入力が求められる

1. データベースと SQL

■DBMS の種類

- 階層型データベース
- リレーショナルデータベース (RDB)
- オブジェクト指向データベース (OODB)
- XML データベース (XMLDB)
- キー・バリュー型データストア (KVS)

■リレーショナルデータベース

- 列と行からなる 2 次元表の形式でデータを管理する
- 操作する言語：SQL

Oracle Database …Oracle 社の RDBMS

SQL Server …Microsoft 社の RDBMS

DB2 …IBM 社の RDBMS

PostgreSQL …オープンソースの RDBMS

MySQL …オープンソースの RDBMS

■データベースの構成

- リレーショナルデータベースを使用する際には、クライアント/サーバ 型のシステムが一般的であり、身近なシステムとしては、Amazon や楽天などのショッピングサイトがある

■標準 SQL

- SQL (Structured Query Language) は、リレーショナルデータベースの データを効率よく操作するための言語
- リレーショナルデータベースにより違い（方言）がある
- ISO が定めた標準 SQL により、大部分データベースは同じ SQL 文で操作ができる

■SQL 文と種類

- DDL（データ定義言語）

CREATE (データベース/テーブルを作成する)

DROP (データベース/テーブルを削除する)

ALTER (データベース/テーブルを変更する)

・ DML (データ操作言語)

SELECT (テーブルから行を検索する)

INSERT (テーブルに行を登録する)

UPDATE (テーブルの行を更新する)

DELETE (テーブルの行を削除する)

・ DCL (データ制御言語)

COMMIT (データベースへの変更を確定する)

ROLLBACK (データベースへの変更を取り消す)

GRANT (ユーザに操作の権限を与える)

REVOKE (ユーザから操作の権限を奪う)

■ SQL の基本的な記述ルール

- SQL 文の最後にセミコロン (;) を付ける
- キーワードの大文字・小文字は区別されない。
ただし、テーブルに登録されているデータは 大文字・小文字が区別される
- 文字列と日付の定数はシングルクォーテーション (') で囲む
- 数字の定数は囲まない
- 単語の区切りは半角スペースか改行

■ データベースへのアクセス

コマンドプロンプト (管理者) または Windows PowerShell (管理者) を起動する

DB が作成されていない場合 c:\Program Files\PostgreSQL\9.5\bin\psql.exe -U postgres

DB が作成されている場合 c:\Program Files\PostgreSQL\9.5\bin\psql.exe -U postgres -d <データベース名>

■ データベースの作成

CREATE TABLE <テーブル名> (<列名 1> <データ型> <制約> ,

<列名 2> <データ型> <制約> , ...);

CREATE TABLE Shohin (shohin_id CHER(4) NOT NULL, ... 【途中省略】 ...);

正常終了すると、CREATE TABLE が表示

エラー終了すると、エラーの場所とエラー原因を表示

■ 命名ルール

- 使用できる文字は、A~Z、a~z、0~9、_ (半角英数字と ' _ ')
- 文字から始める (数字から始まる名前は NG)
- 一つのデータベースの中では、名前は重複しないようにする
- 長さは 1~63 バイト (PostgreSQL の場合)

以下推奨事項

- SQL のキーワード (予約語) は大文字で記述
- データベース名は小文字で記述

- ・ テーブル名は先頭だけ大文字で記述し 2 文字目以降は小文字
- ・ 列名は小文字で記述

■データ型の指定

- ・ VARCHAR(n) 長さ n 文字の可変長文字列
- ・ CHAR(n) 長さ n 文字の固定長文字データ
指定した長さより短い CHAR 値を挿入したときは、残りは空白
- ・ TEXT 長さ指定なしの可変長文字列
- ・ INTEGER 整数型(-2147483648 から+2147483647)
- ・ SMALLINT 範囲の狭い整数型(32768 から+32767)
- ・ BIGINT 範囲の広い整数型 (-9223372036854775808 から+9223372036854775807)
- ・ DECIMAL ユーザ指定精度 小数点より上は 131072 桁まで
小数点より下は 16383 桁まで
- ・ NUMERIC ユーザ指定精度 小数点より上は 131072 桁まで
小数点より下は 16383 桁まで
- ・ DATE 日付(時刻なし)
- ・ TIME 時刻(日付なし)

■制約の設定

- ・ NOT NULL 制約 (非 NULL 制約)
NULL (データが何も入っていない状態) を許可しない
- ・ 一意性制約 (UNIQUE 制約)
その列の中でデータが他と重複しない
- ・ 主キー制約 (PRIMARY KEY 制約)
データを特定するための主キーを指定する ※NOT NULL 制約 + UNIQUE 制約
- ・ 検査制約 (CHECK 制約)
列のデータを一定の範囲や種類に限定する
- ・ 外部キー制約 (FOREIGN KEY 制約)
外部の他のテーブルの特定の列の値しか指定できないようにする

■デフォルト値の設定

<列名> <データ型> DEFAULT <デフォルト値>

■テーブルの削除と変更

DROP TABLE <テーブル名>;

注意：削除したテーブルは復活できないため、誤って削除した場合は再作成するしかない
常終了時には、【 DROP TABLE 】と表示される

■テーブル定義の変更

ALTER TABLE <テーブル名> <変更内容>

ALTER TABLE 文は次のような場合に使用する

新しい列を追加する、既存の列を削除する、既存の列を変更する

新しい列にデフォルト値を定義する、テーブル名を変更する など

正常終了時には、【 ALTER TABLE 】と表示される

■テーブルへのデータ登録

BEGIN TRANSACTION;

INSERT INTO shohin VALUES ('0001', 'T シャツ', '衣服', 1000, 500, '2009-09-20');

INSERT INTO shohin VALUES ('0002', '穴あけパンチ', '事務用品', 500, 320, '2009-09-11');

：(途中省略)

INSERT INTO shohin VALUES ('0008', 'ボールペン', '事務用品', 100, NULL, '2009-11-11');

COMMIT;

※トランザクション処理でまとめてデータ登録 (INSERT 文) を行っている。

2. 検索の基本

■SELECT 文

指定した列を出力する

SELECT <列名 1>, <列名 2>, … FROM <テーブル名>;

すべての列を出力する

SELECT * FROM <テーブル名>;

■SELECT 文の処理順

- ①FROM 対象テーブルからデータを取り出す
- ②WHERE 条件に一致するレコードを絞り込み
- ③GROUP BY グループ化
- ④HAVING 集計結果から絞り込み
- ⑤SELECT 指定したカラムだけを表示

■列に別名をつける

SELECT <列名 1> AS <別名 1>, <列名 2> AS <別名 2>, … FROM <テーブル名>;

日本語で別名を付けるときは、【"】(ダブルクォーテーション)で囲む

■結果から重複行を省く

SELECT DISTINCT <列名 1>, <列名 2>, … FROM <テーブル名>;

NULL もデータのひとつとして扱われる

複数の列名を指定した場合は、すべての列のデータが同じものがまとめられ

■WHERE 句による行の選択

SELECT <列名 1>, <列名 2>, … FROM <テーブル名> WHERE <条件式>;

■算術演算子

+ (加算) - (減算) * (乗算) / (除算)

NULL との演算結果はすべて NULL になる

■比較演算子

= (等しい) <> (等しくない)

>= (以上) > (より大きい) <= (以下) < (より小さい)

NULL に比較演算子は使えず、検索結果には何も表示されない

NULL を検索するには IS NULL または IS NOT NULL を使う

■論理演算子

AND (論理積) OR (論理和) NOT (否定)

AND と OR の組み合わせでは AND が優先される

■論理演算子と真理値

SQL での真理値は真・偽・不明 (NULL) の 3 論理値で表現される

P	Q	P AND Q	P OR Q
真	真	真	真
真	偽	偽	真
真	不明	不明	真
偽	真	偽	真
偽	偽	偽	偽
偽	不明	偽	不明
不明	真	不明	真
不明	偽	偽	不明
不明	不明	不明	不明

■コメントの書き方

コメントには、1 行コメント 【 -- 】と複数行コメント 【 /* */ 】がある

3. 集約と並び替え

■集約関数

複数の検索結果を処理して、ひとつの結果を計算 (集約) するもの

件数: COUNT(列名または*) *はすべての項目の意味で、NULL があっても数える

合計: SUM(列名)

平均: AVG(列名)

最大: MAX(列名)

最小: MIN(列名)

指定した列名の値が NULL の場合は、NULL を除いて計算する

■重複値を除外する

DISTINCT キーワードを使用する

DISTINCT キーワードを使う場所が違くと、結果も違ってくる

○ SELECT COUNT(DISTINCT shohin_bunrui) FROM Shohin ;

× SELECT DISTINCT COUNT(shohin_bunrui) FROM Shohin ;

■GROUP BY 句

テーブル内のデータをグループ分けして集約する

SELECT <列名 1>, <列名 2>, <列名 3>, ... FROM テーブル名

GROUP BY <列名 1>, <列名 2>, <列名 3>, ... ;

GROUP BY 句に指定した列に NULL データが含まれる場合は NULL のグループで分類される

順番は、まず WHERE 句の条件で絞り込み、GROUP BY 句の指定によりグループ分けされる

■集約関数と GROUP BY 句での注意点

- ① SELECT 句には以下 の 3 種類しか記述できない
 - ・ 定数 (123 のような数値、'衣服' のような文字列)
 - ・ 集約関数
 - ・ 集約キー (GROUP BY 句で指定した列名)
- ② GROUP BY 句に指定する列名は、AS を使用してつけた別名を指定 できない
- ③ GROUP BY 句を使用して出力した結果は、ソートされない
- ④ WHERE 句に集約関数を記述できない

■HAVING 句

GROUP BY 句でグループ分けしたデータに対して条件を設定する

(※WHERE 句はデータ全体に対して条件を設定する)

SELECT <列名 1>, <列名 2>, … FROM <テーブル名>

GROUP BY <列名 1>, <列名 2>, …

HAVING <グループの値に対する条件>

■ORDER BY 句

検索結果を並べ替える

SELECT <列名 1>, <列名 2>, … FROM <テーブル名>

WHERE 句

GROUP BY 句

HAVING 句

ORDER BY <並べ替え基準の列 1>, <並べ替え基準の列 2>, … ;

ORDER BY 句は一番最後に記述する

降順に並べるには、DESC (デスク) キーワード

昇順に並べるには、何も指定しないか、ASC (アスク) キーワード

基準はテーブルに存在する列であれば指定できる (SELECT 句に含まれていない列も可)

NULL は先頭か末尾にまとめられる

4. データの更新

■INSERT 文 (テーブルへレコードを追加する)

INSERT INTO <テーブル名> (列 1, 列 2, 列 3, …) VALUES (値 1, 値 2, 値 3, …)

VALUE 句のリストをカンマで区切って複数のデータを一度に登録できる

テーブルの全ての列にデータを登録する場合、列リストを省略できる

NOT NULL 制約が設定されていない列には、NULL データを挿入できる

テーブルにデフォルト値を挿入するには、DEFAULT キーワードで明示的に挿入する

■DELETE 文 (テーブルからレコードを削除する)

DELETE FROM <テーブル名> WHERE <条件式> ;

削除したレコードは元に戻せないので注意する

■UPDATE 文（レコードの値を変更する）

UPDATE <テーブル名> SET <列名> = <式> WHERE <条件式>;

複数列を更新したい場合は列をカンマ区切りで並べる

UPDATE <テーブル名> SET <列名 1> = <式 1>, <列名 2> = <式 2> WHERE <条件式>;

■トランザクション

データベースでは、複数の更新処理をひとまとめにしたもの

複数の処理を COMMIT で確定させるか、ROLLBACK で元に戻すかのどちらかを実行する

トランザクション開始文; (BEGIN TRANSACTION または 開始文が無い DBMS もある)

DML 文①;

DML 文②;

DML 文③;

:

トランザクション終了文 (COMMIT または ROLLBACK);

■ACID 特性

DBMS のトランザクションで守るべき 4 つの性質

- ・原子性(Atomicity) 更新処理はすべて実行されるか、すべて実行されないかのどちらか
- ・一貫性(Consistency) 各種の規則や制約によって統一された処理を行う
- ・独立性(Isolation) トランザクションが完了するまで、他のトランザクションに影響しない
- ・永続性(Durability) トランザクションが終了したら、その時点でデータは保存される

5. 複雑な問い合わせ

■ビュー（仮想表）

テーブルは、実際のデータが入っている場所で、ディスク等の記憶装置に保存される

ビューは、内部的に SELECT 文を実行して仮想的に作成されたテーブル

ビューのデータそのものは元のテーブルにあり、記憶装置を消費しない

複雑な SELECT 文もビューを作ることで、簡単な SELECT 文で可能となる

■ビューの作成

CREATE VIEW ビュー名 (<ビューの列名 1>, <ビューの列名 2>, ...)

AS <SELECT 文>

多段ビュー（ビューの上にビュー）の階層が増えると処理が遅くなるので、なるべく避ける

■ビューの制限事項

①ビュー定義で ORDER BY 句は使えない

②ビューは仮想的なテーブルのため、更新(INSERT,DELETE,UPDATE)するには制限がある

以下の条件を満たしている場合、ビューに対する更新が可能である

- ・ SELECT 句に DISTINCT が含まれていない
- ・ FROM 句に含まれるテーブルが 1 つだけである
- ・ GROUP BY 句を使用していない
- ・ HAVING 句を使用していない

- ・集約関数が含まれていない

■ビューの削除

DROP VIEW <ビュー名>

多段ビューの参照元になっているビューの削除には <ビュー名>の後に CASCADE を使う

■サブクエリ

SELECT ... FROM (SELECT) AS サブクエリ名

FROM 句に直接ビュー定義の SELECT 文を書く入れ子構造で、内側から実行される
サブクエリは階層を増やすこともできるが、処理が遅くなり、読みにくいので避ける

■スカラ・サブクエリ

戻り値が必ず単一の値になるサブクエリのこと

スカラ・サブクエリは戻り値を比較演算子の入力（右辺）に使える

使用例：販売単価が、全体の平均の販売単価より高い商品だけを検索する

```
SELECT shohin_id, shohin_mei, hanbai_tanka FROM Shohin
WHERE hanbai_tanka > (SELECT AVG(hanbai_tanka) FROM Shohin);
```

スカラ・サブクエリは先に実行されるため、販売単価の平均が先に求められる

■相関サブクエリ

相関サブクエリはテーブルをグループごとに比較する場合に便利である

使用例：商品分類ごとに平均販売単価よりも高い商品を探る場合

```
SELECT shohin_bunrui, shohin_mei, hanbai_tanka FROM Shohin AS S1
WHERE hanbai_tanka > (SELECT AVG(hanbai_tanka) FROM Shohin AS S2
WHERE S1.shohin_bunrui = S2.shohin_bunrui
GROUP BY shohin_bunrui);
```

S1 と S2 の別名をつけた Shohin テーブルを同じ商品分類で結合して検索している
結合条件は必ずサブクエリの中に書く（スコープ内で有効にするため）

6. 関数、述語、CASE 式

■関数

関数とはある値（引数）を入力すると、それに対応した値（戻り値）を出力する機能
算術関数、文字列関数、日付関数、変換関数、集約関数がある

■算術関数

データの型 NUMERIC(全体の桁数, 小数の桁数)

加減乗除 +, -, *, /

絶対値 ABS(数値)

剰余 MOD(被除数, 除数)

四捨五入 ROUND(数値, 丸めの桁数) ※桁数はマイナスも可

■文字列関数

連結 文字列 1 || 文字列 2

文字列長 LENGTH(文字列)

小文字化	LOWER(文字列)
大文字化	UPPER (文字列)
置き換え	REPLACE(対象文字列, 置換前の文字列, 置換後の文字列)
切り出し	SUBSTRING(対象文字列 FROM 切り出し開始位置 FOR 切り出す文字数)

■日付関数

現在の日付	CURRENT_DATE	→ 2020-03-31 など
現在の時間	CURRENT_TIME	→ 18:35:50.995+09 など
現在の日時	CURRENT_TIMESTAMP	→ 2020-03-31 18:35:50.995+09 など
切り出し	EXTRACT(日付要素 FROM 日付)	※日付要素は year,month,day,hour など

■変換関数

型変換	CAST(変換前の値 AS 変換するデータ型)
NULL を値変換	COALESCE(データ 1, データ 2, ...)
※引数を左から順に見て、最初に NULL でない値を返す	
NULL を何か別の値に変えて扱いたいときに便利	

■集約関数

SUM,AVG,COUNT,MAX,MIN

■述語

関数の一種で、戻り値が真理値(TRUE/FALSE/UNKNOWN)になるもの
LIKE 述語,BETWEEN 述語,IS NULL,IN 述語,EXISTS 述語などがある

■LIKE 述語

LIKE 述語は文字列の部分一致検索をするときに使う

前方一致、中間一致、後方一致の 3 種類がある

「 % 」は 0 文字以上の任意の文字列「 _ 」(アンダーバー)は任意の 1 文字の意味

使用例: SELECT * FROM SampleLike WHERE strcol LIKE '%ddd%';

■BETWEEN 述語

WHERE 句で範囲を指定した条件を記述したいときに使う

使用例: SELECT shohin_mei, hanbai_tanka FROM Shohin

WHERE hanbai_tanka BETWEEN 100 AND 1000 ;

※WHERE hanbai_tanka >= 100 AND hanbai_tanka <= 1000 と同じ

■IS NULL

列のデータが NULL のものを検索するためには「 = 」は使えない

使用例: SELECT shohin_mei, shiire_tanka FROM Shohin WHERE shiire_tanka IS NULL ;

非 NULL の判定では IS NOT NULL を使う

■IN 述語

OR の便利な省略形

ある集合の値かどうか検索する場合に使用する

使用例: SELECT shohin_mei, shiire_tanka FROM Shohin

WHERE shiire_tanka IN (320, 500, 5000) ;

※WHERE Shiire_tanka = 320 OR Shiire_tanka = 500 OR Shiire_tanka = 5000 ; と同じ
 IN 述語の引数にサブクエリを指定することもできる (IN(SELECT ……) など)
 ある集合の値以外かどうか検索する場合は、NOT IN を使う

■EXISTS 述語

一部を除いて、ほぼ IN 述語 (NOT IN 述語) で代用できる

使用例：大阪店 (000C) に置いてある商品の販売単価を求める

```
SELECT shohin_mei, hanbai_tanka FROM Shohin AS S
      WHERE EXISTS (SELECT * FROM TenpoShohin AS TS
                    WHERE TS.tenpo_id = '000C'
                    AND TS.shohin_id = S.shohin_id) ;
```

EXISTS 述語の後には、相関サブクエリしか記述できない

サブクエリの SELECT 文で 指定する列名は何でも良いが、習慣として「 * 」を指定する

■CASE 式

条件分岐を行うための記述で、単純 CASE 式と検索 CASE 式の 2 種類がある

検索 CASE 式は単純 CASE 式の機能をすべて含む

■検索 CASE 式

CASE

WHEN <評価式 1> THEN <式 1> ※評価式 1 が TRUE の時、式 1 を戻す

WHEN <評価式 2> THEN <式 2>

:

ELSE <式 3> ※どの評価式も TRUE でない時、式 3 を戻す

END ※ELSE を省略すると ELSE NULL とみなす

使用例：CASE 式で商品分類に A～C の文字列を割り当てる

```
SELECT shohin_mei,
      CASE WHEN shohin_bunrui = '衣服' THEN 'A : ' || shohin_bunrui
            WHEN shohin_bunrui = '事務用品' THEN 'B : ' || shohin_bunrui
            WHEN shohin_bunrui = 'キッチン用品' THEN 'C : ' || shohin_bunrui
            ELSE NULL
      END AS abc_shohin_bunrui
FROM Shohin ;
```

■単純 CASE 式

CASE <式>

WHEN <値 1> THEN <式 1> ※式が値 1 の時、式 1 を戻す

WHEN <値 2> THEN <式 2>

:

ELSE <式 3> ※式がどの値でもない時、式 3 を戻す

END

7. 集合演算

■ テーブルの足し算 (UNION)

同じ構造の 2 つのテーブルに含まれていたレコードをすべて表示する (和)

```
使用例:  SELECT shohin_id, shohin_mei FROM Shohin
          UNION
          SELECT shohin_id, shohin_mei FROM Shohin2 ;
```

重複行は排除されるので、重複行を残すには UNION ALL とする

■ テーブルの共通部分の選択 (INTERSECT)

同じ構造の 2 つのテーブルに共通に含まれるレコードを表示する (交差)

```
使用例:  SELECT shohin_id, shohin_mei FROM Shohin
          INTERSECT
          SELECT shohin_id, shohin_mei FROM Shohin2 ;
```

■ テーブルの引き算 (EXCEPT)

あるテーブルからあるテーブルに含まれるレコードを除く (差)

```
使用例:  SELECT shohin_id, shohin_mei FROM Shohin
          EXCEPT
          SELECT shohin_id, shohin_mei FROM Shohin2 ;
```

どちらからどちらを引くかによって結果は異なる

■ 結合

あるテーブルに別のテーブルから列を持ってきて列を増やす

UNION や INTERSECT などは行が増えたり減ったりするが、結合は列が増える

結合するために、複数のテーブルに存在する共通の列が必要である

内部結合と外部結合がある

■ 内部結合 (INNER JOIN)

```
使用例:  SELECT TS.tenpo_id, TS.tenpo_mei, TS.shohin_id, S.shohin_mei, S.hanbai_tanka
          FROM TenpoShohin AS TS INNER JOIN Shohin AS S
          ON TS.shohin_id = S.shohin_id ;
```

内部結合では、どちらのテーブルにもあるレコードのみ出力される

FROM 句に複数のテーブルを記述し、ON 句は FROM と WHERE の間に記述する

別名は書いた方が見やすい

WHERE 句は結合した後に実行される

結合されたテーブルは SELECT 文の実行中しか存在しないので、必要な場合はビューを作る

■ 外部結合 (OUTER JOIN)

```
使用例:  SELECT TS.tenpo_id, TS.tenpo_mei, S.shohin_id, S.shohin_mei, S.hanbai_tanka
          FROM TenpoShohin AS TS RIGHT OUTER JOIN Shohin AS S
          ON TS.shohin_id = S.shohin_id ;
```

外部結合では、どちらか一方のテーブルに存在しているレコードはすべて出力される

橋渡しする列に値がない場合は、NULL が返される

どちらをマスタにするかは、LEFT または RIGHT で指定する

共通する列はマスタ側を参照する

行数固定の定型帳票を作りたい場合などに利用される

■クロス結合（クロス結合）

2つのテーブルのレコードについて、すべての組み合わせを作る結合方法（直積）

実務で使われることはないが、内部結合は必ずクロス結合の一部分になっている

8. SQL で高度な処理を行う

■ウィンドウ関数（OLAP 関数, OnLine Analytical Processing）

データベースを使ってリアルタイムにデータ分析を行う処理のこと

<ウィンドウ関数> OVER ([PARTITION BY <列リスト>] ORDER BY <ソート用リスト>)

<ウィンドウ関数>に使用できるのは、以下の関数

- ・集約関数（SUM、AVG、COUNT、MAX、MIN）
- ・ウィンドウ専用関数（RANK、DENSE_RANK、ROW_NUMBER など）

ウィンドウ関数は SELECT 句でのみ使用できる

使用例： SELECT shohin_mei, shohin_bunrui, hanbai_tanka,
RANK () OVER (PARTITION BY shohin_bunrui
ORDER BY hanbai_tanka) AS ranking
FROM Shohin ;

実行結果例：

shohin_mei	shohin_bunrui	hanbai_tanka	ranking
フォーク	キッチン用品	500	1
おろしがね	キッチン用品	880	2
包丁	キッチン用品	3000	3
圧力鍋	キッチン用品	6800	4
Y シャツ	衣服	1000	1
カッターシャツ	衣服	4000	2
ボールペン	事務用品	100	1
穴あけパンチ	事務用品	500	2

①商品分類で仕分け → ②販売単価で並び替え → ③その結果に順位付けしている
累計や移動平均を算出する場合などにも便利である

■累計

SUM 関数をウィンドウ関数として使用する

使用例： SELECT shohin_id, shohin_mei, hanbai_tanka,
SUM(hanbai_tanka) OVER (ORDER BY shohin_id) AS current_sum
FROM Shohin ;

実行結果例：

shohin_id	shohin_mei	hanbai_tanka	current_sum
0001	Yシャツ	1000	1000
0002	穴あけパンチ	500	1500
0003	カッターシャツ	4000	5500
0004	包丁	3000	8500
:	:	:	:

■移動平均

集計範囲をウィンドウ全体ではなく、さらに小さなフレームで指定 することができる

使用例： `SELECT shohin_id, shohin_mei, hanbai_tanka,`
`AVG (hanbai_tanka) OVER (ORDER BY shohin_id ROWS 2 PRECEDING)`
`AS moving_avg`
`FROM Shohin ;`

実行結果例：

shohin_id	shohin_mei	hanbai_tanka	moving_avg
0001	Yシャツ	1000	1000.00...
0002	穴あけパンチ	500	750.00...
0003	カッターシャツ	4000	1833.33...
0004	包丁	3000	2500.00...
:	:	:	:

フレームは前後の行を指定することもできる

`SELECT shohin_id, shohin_mei, hanbai_tanka,`
`AVG (hanbai_tanka) OVER (ORDER BY shohin_id`
`ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS moving_avg`
`FROM Shohin ;`

■GROUPING 演算子

合計と小計を一度に求める (ROLLUP)

使用例 1： `SELECT shohin_bunrui, SUM(hanbai_tanka) AS sum_tanka FROM Shohin`
`GROUP BY ROLLUP(shohin_bunrui) ;`

実行結果例 1：

shohin_bunrui	sum_tanka
キッチン用品	11180
衣服	5000
事務用品	600
	16780

ROLLUP(<列 1 >, <列 2 >, ...)と指定すると、集約キーのレベルで小計を求める

使用例 2： `SELECT shohin_bunrui, SUM(hanbai_tanka) AS sum_tanka FROM Shohin`
`GROUP BY ROLLUP(shohin_bunrui ,torokubi) ;`

実行結果例 2：

shohin_bunrui	torokubi	sum_tanka
キッチン用品	2009-04-28	880
キッチン用品	2009-01-15	6800
キッチン用品	2009-09-20	3500
キッチン用品		11180
衣服	2009-09-20	1000
衣服		4000
衣服		5000
事務用品	2009-09-11	500
事務用品	2009-11-11	100
事務用品		600
		16780

9. アプリケーションからデータベースへ接続する

■システム＝アプリケーション＋データベース

システム構築では、何らかのプログラムがデータベースとセットで使われている
アプリケーションには Java, Python, PHP, C# 言語などがある

■ドライバ

アプリケーションとデータベースの接続に特化した機能を持つ小さなプログラム

ODBC (Open DataBase Connectivity)

1992年にMicrosoft社が開発し、業界標準となった

JDBC (Java DataBase Connectivity)

ODBCを参考にJavaからの接続方法をまとめた規格

<https://jdbc.postgresql.org/download.html> より PostgreSQL 用の JDBC ドライバをダウンロード
フォルダ名にはすべて半角英数字を使用する

■Java から PostgreSQL へ接続

import java.sql.*; // Java からデータベースに接続するために必要な機能を取り込む

```
public class DBConnect3{
```

```
    public static void main(String[] args) throws Exception {
```

```
        /* 1) PostgreSQL への接続情報 */
```

```
        Connection con; // データベースの接続部
```

```
        Statement st; // SQL 文を格納し実行する
```

```
        ResultSet rs; // SQL 文の実行結果を格納する
```

```
        String url = "jdbc:postgresql://localhost:5432/shop";
```

```
        /* プロトコル :// ホスト名:ポート/接続するデータベース */
```

```
        String user = "postgres"; // 接続するデータベースユーザ
```

```
        String password = "*****"; // 接続するデータベースユーザのパスワード
```

```
        /* 2) JDBC ドライバの定義 */
```

```

        Class.forName("org.postgresql.Driver");
/* 3) PostgreSQL への接続 */
        con = DriverManager.getConnection(url, user, password);    //DB へ接続
        st = con.createStatement();    //SQL 文の実行準備
/* 4) SELECT 文の実行 */
        rs = st.executeQuery("SELECT shohin_id, shohin_mei FROM Shohin");
                                //SELECT 文末の ; なし
                                //INSERT,DELETE,UPDATE 文には executeUpdate を使用する
/* 5) 結果の画面表示 */
        while(rs.next()){    //レコードがなくなるまでループする
            System.out.print(rs.getString("shohin_id") + ", ");    //取得した文字列を表示
            System.out.println(rs.getString("shohin_mei"));    //数値の場合は getInt()
        }
/* 6) PostgreSQL との接続を切断 */
        rs.close();    // SQL 文の実行結果を開放して終了
        st.close();    // SQL 文の実行部を開放して終了
        con.close();    // データベースへの接続を終了
    }
}

```

■どこからでも Java を起動する

コマンドプロンプトから Java を起動する際、コマンドをフルパスで指定するのは面倒
Windows の環境設定でパスを通すことで解決する

環境変数【Path】に java/javac コマンドが格納されているフォルダを追加する

- ①ファイルエクスプローラの【PC】を右クリックして、【プロパティ】を選択
- ②【システムの環境設定】を選択する
- ③システムのプロパティから【環境変数】を選択する
- ④環境変数から【Path】を選択し、【編集】を選択する
- ⑤環境変数名の設定から【新規】を選択し、Java コマンドが格納されているパスを追加する
パスは環境により異なるため、各自で確認して記入すること
- ⑥コマンドプロンプトを再起動し、環境変数【Path】を確認