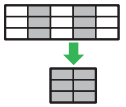
SELECTの基本　SELECT, JOIN, WHERE

SELECT文の基本的な書き方　➡2.1節

```
SELECT 列名 FROM テーブル WHERE 条件;
```

- 各単語は半角スペースまたは改行で区切り、文末には「;」を付ける
  （ⓈSQL Serverの場合**go**のみの行を入力すると実行）
- キーワード（SQL文を構成するための単語）は大文字でも小文字でも良い
- テーブル名や列名には通常の文字と「_」を使用する（半角英数字が無難だが漢字も使用できる）。キーワードを使いたい・大文字小文字の区別をしたい場合はダブルクォーテーションで囲む
- 文字列や日付は半角のシングルクォーテーションで囲み、キーワードや数字は囲まない
- コメントは「--」（半角ハイフン2つ）の後ろか「/*」と「*/」の間に書く
  （ⓂMySQLとⓂMariaDBⓈSQL Serverは「#」も使用可能。半角#以降がコメント扱いになる）
  （ⓂMySQLは半角ハイフン2つの後に半角スペースが必要）

列名の指定　➡2.7節, 6.1節

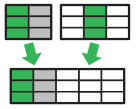


- 使いたい列を使いたい順番に「,」で区切って指定する
- 同じ列を何度指定しても良い
- 複数のテーブルに同じ名前の列がある場合は テーブル名.列名 で指定
- テーブル名.*で該当テーブルのすべての列、「*」ですべてのテーブルのすべての列
- 列名 **AS** 別名 で表示用の列名をつけることが可能

テーブルの結合　JOIN　➡2.7節, 6.4節

```
SELECT ～ FROM ,テーブル1
    JOIN テーブル2
        ON テーブル1.列名A=テーブル2.列名a,
        AND テーブル1.列名B=テーブル2.列名b,
    JOIN テーブル3 ON …;
```

- **ON**の後に結合条件を指定、複数指定可能
- 条件が複数ある場合はANDかORを指定
- 値が一致するものを指定することが多いが（等価結合）、不一致や大小での指定も可能
  ⓂⓂⓅ 共通の列名がある場合、結合条件を**USING(**列名**)**で指定可能。等価結合のみ
  ⓂⓂⓅ NATURAL JOINで共通の列名すべてを使用した結合が可能。等価結合のみ



- WHERE句は結合条件の後に書く（JOINした結果を絞り込むことになる）

結合のバリエーション　➡6.4節

結合	意味
説明	
CROSS JOIN	総当たり
条件なしの結合、いわゆる総当たり表になる	
JOIN, **INNER JOIN**	内部結合
結合条件に合うデータのみ、一般的なJOINはこれ	
LEFT JOIN, **LEFT OUTER JOIN**	左外部結合
先に指定したテーブルは全件、後に指定したテーブルに該当データがない場合は**NULL**　➡3.3節	
RIGHT JOIN, **RIGHT OUTER JOIN**	右外部結合
後に指定したテーブルは全件、先に指定したテーブルに該当データがない場合は**NULL**	
FULL JOIN, **FULL OUTER JOIN**	完全外部結合ⓅⓈ
左外部結合＋右外部結合	

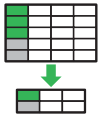

INNER、OUTERは省略可能

LEFT JOINとLEFT OUTER JOIN意味は同じ！


ⓂMySQLで使用可能　ⓂMariaDBで使用可能
ⓅPostgreSQLで使用可能　ⓈSQL Serverで使用可能
※記載がないものはⓂⓂⓅⓈ共通で使用可能

データの集約　GROUP BY　➡6.6節

```
SELECT ～ FROM テーブル JOIN ON ～ WHERE ～
    GROUP BY ～ HAVING ～;
```



- **SUM**や**COUNT**で集計できる、これらを集約関数という
- **SELECT**句では集約に使用した列名（**GROUP BY**で指定した列）または集約関数が使用可能

例
```
SELECT branch,COUNT(*)
    FROM students
    GROUP By branch;
```

- **NULL**は対象外だが**COUNT(*)**の場合は**NULL**の件数も含まれる
- **GROUP BY**は**WHERE**より後（**WHERE**で絞った結果を集約する）
- 集約関数の結果を使って絞り込みたい場合は**HAVING**を使用

おもな集約関数

関数	結果
AVG	平均
SUM	合計
MAX	最大
MIN	最小
COUNT	件数

並べ替え　ORDER BY　➡6.1節

```
SELECT ～ FROM テーブル JOIN ON ～
    WHERE ～
    ORDER BY ～;
```

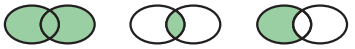
SELECTにない列でもOK!


- **SELECT**文を実行した結果のデータ順は不定、DBMSが条件やデータの件数等を鑑みて最適な方法で取得する
- 並び順を固定したい場合は**ORDER BY**で列名を指定
- 列名は複数指定可能
- 降順にしたい場合は列名の後に**DESC**（*descending order*）をつける。昇順を明示したい場合は**ASC**（*ascending order*）
- **NULL**の位置はDBMSによって異なる。ⓂⓂⓈは**ASC**の場合先頭、**DESC**の場合は末尾、Ⓟは逆だが**NULLS FIRST**または**NULLS LAST**で明示することが可能

テーブルを結合する　UNION　➡6.5節

```
SELECT ～ FROM テーブル ～ WHERE ～
UNION
SELECT ～ FROM テーブル ～ WHERE ～ ORDER BY ～ ;
```

- 各**SELECT**文で抽出する列の個数を揃える。ⓅⓈはデータ型も揃える必要がある
- **WHERE**句は各**SELECT**文で参照するテーブルや**JOIN**による結合、**WHERE**句による絞り込み等は自由
- 重複は除去される。重複を残したい場合は**UNION ALL**
- **ORDER BY**は最後に書く（UNIONした結果を並べ替える）。同様に、**SELECT**の結果同士で共通しているデータを求める**INTERSECT**と差を求める**EXCEPT**がある


和 UNION　共通 INTERSECT　差 EXCEPT　➡6.13節

取得件数を指定する　➡6.1節

```
ⓂⓂⓅ SELECT ～ FROM ～ WHERE ～ LIMIT 件数;
Ⓢ    SELECT TOP 件数 ～ FROM ～ WHERE ～;

開始位置（0からカウント）と件数を指定
ⓂⓂ   ～ LIMIT 開始 件数;
Ⓟ    ～ LIMIT 件数 OFFSET 開始

FETCHの使用
ⓂⓅ   SELECT ～ WHERE ～ FETCH FIRST 件数 ROWS ONLY;
ⓂⓅⓈ SELECT ～ WHERE ～
                FETCH OFFSET 開始 NEXT 件数 ROWS ONLY;
```

主キーと正規形

主キー：データを特定するのに使う

（複数の値の組み合わせによるキー）

この状態が大前提

……という中で以下状態を発見したら要注意！

部分関数従属
で が決まる
（複合キーの一部でデータが決まる）
解消できれば第2正規形

推移的関数従属　**非キー同士に注目**
で が決まる
（主キーではないデータで別のデータが決まる）
解消できれば第3正規形

推移的関数従属　**キーの一部に注目**
で や が決まる
で が決まる
（複合キーの一部やデータから別の複合キーの一部やデータが決まる）
解消できればBC正規形

複合キーについて

- 複数の値を組み合わせることでデータを特定できる
- 組み合わせの状態が他と重複してはいけない
  ※実用上扱いにくい場合は連番などで代替することがあるが（➡3.4節）正規化の際は複合キーで考える

第1正規形　➡4.2節

データをテーブルに格納できる形になっている

- **(001, タヌ吉)**, **(002, 亜由子)**のように同じ関係を持つ組み合わせの集合が「テーブル」、学生番号の列、名前の列のように関係を列で表した列×行の表の形を取ることができる
- 各列には同じかデータ型の値が1つだけ
- 導出項目（計算等で求められる値）を取り除く
- 繰り返し項目を取り除く

第2・第3正規形・ボイスコッド正規形　➡4.3節

主キーと主キーで決定できる値のみになっている

- ある値から別の値が決まる関係（IDから名前がわかる）が**関数従属**
- **第2正規形**　➡第1正規形かつ複合キーの一部から決まる値（部分関数従属）がない
- **第3正規形**　➡第2正規形かつ非キー同士で関数従属（推移的関数従属）がない
- **BC正規形**　➡第3正規形かつキー以外の項目や復号キーの一部からの関数従属がない

他の観点からの正規形もある。
- 第4正規形・第5正規形　➡4.4節
- ドメインキー正規形・第6正規形　➡4.5節


他のテーブルに登録されていなければならない値は参照制約（外部キー）で設定　➡2.3節
例 試験テーブルの学生番号は学生マスターにある
　注文テーブルの商品番号は商品マスターにある など

テーブルの定義やどんなテーブルがあるのかを確認する方法

テーブルの定義

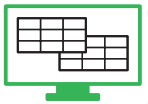
テーブルはCREATE TABLEで定義する（➡2.2節, 3章）。テーブルの構造は各DBMSのビジュアルツールでも確認可能。コマンドは以下。

```
ⓂⓂ SHOW CREATE TALBLE テーブル名;
Ⓟ  \d テーブル名   \は環境によって円マークかバックスラッシュで表示
Ⓢ  EXEC sp_help 'テーブル名';   テーブル名にはシングルクォーテーションが必要
```

テーブルの一覧

各データベースに情報管理用のテーブルがありSELECT文で取得可能。ⓂⓂⓅは専用コマンドがある。

```
ⓂⓂ SHOW TABLES;
SELECT table_name FROM information_schema.tables WHERE table_schema = 'データベース名';
Ⓟ  \dt
Ⓟ  SELECT tablename from pg_tables WHERE schemaname = 'public';
Ⓢ  SELECT name FROM sys.tables;
```



データベースの一覧

```
ⓂⓂ SHOW DATABASES;
ⓂⓂ SELECT schema_name FROM information_schema.
schemata;
Ⓟ  \l   \の後に小文字のエル
Ⓟ  SELECT datname FROM pg_database
        WHERE datistemplate = false;
Ⓢ  SELECT name FROM sys.databases;
```

データベースの切り替え

サーバー接続時にデータベース（sampledb等）を指定。ⓂⓂⓅは専用コマンドがある。

```
ⓂⓂⓈ use データベース名   末尾に;は不要だが付けてもOK
```

『標準SQL＋データベース入門』RDBとDB設計、基本の力』（西村めぐみ著、技術評論社）［特別収録］コンテンツ

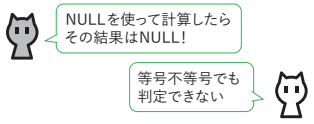
MySQL
MariaDB
PostgreSQL
SQL Server
対応

標準SQL &DB設計 Quickリファレンス

1/2

NULL ➡3.3節, 3.4節, 6.3節

不明、未定、適用外などの理由で「列に入れるべき値がない」場合はNULL（ヌル、ナル）という特別な値を使用する。NULLを大小比較や演算の対象にした場合、結果はすべてNULLになる。



- 列の値としてNULLを許容するかどうかはテーブル設計で指定
- NULLを可能としなければいけない場合、テーブル設計に問題があることが多い

比較演算子 ➡6.2節

演算子	説明
=, <>	等しい、等しくない※1
>, <	より大きい、より小さい
>=, <=	より大きいか等しい（以上）、より小さいか等しい（以下）
IS, IS NOT	NULLかどうか※2

※1　等しくない（<>）は「!=」も使用可能。
※2　Ⓜ Ⓟ 値 IS TRUEのような判定も可能。

- 数値や文字列は比較演算子で大小の比較や一致不一致を指定可能。条件が複数ある場合はANDとORで組み合わせる（()で優先順位を調整可能）
- NULLかどうかはIS NULL, IS NOT NULLでのみ判定可能 ➡6.3節

比較演算子のおもな使い方

意味	例
値1以上	SELECT ～ WHERE 列名 >= 値1;
値1以上かつ値2以下	SELECT ～ WHERE 列名 >= 値1 AND 列名 <= 値2;
（同上）	SELECT ～ WHERE 列名 BETWEEN 値1 AND 値2;
値1, 2, 3のいずれか	～ WHERE 列名 = 値1 OR 列名 = 値2 OR 列名 = 値3;
	～ WHERE 列名 IN (値1, 値2, 値3);
NULLではない	SELECT ～ WHERE 列名 IS NOT NULL;

数値や日時の計算 ➡6.2節

- +, −, *, /による四則演算のほか、ROUNDによる四捨五入などの関数がある。関数のサポートはDBMSによって異なる
- 現在の日時はⓂ Ⓟ NOW()やLOCALTIMEで取得可能。日付と時刻を別に取得したい場合はCURRENT_DATEとCURRENT_TIME、Ⓢ現在の日時をGETDATE()、日付や時刻はCAST(GETDATE() AS DATE)またはCONVERT(DATE, GETDATE())で変換
- 日時から日付や時刻を取り出す関数は標準SQLはⓂ Ⓟ EXTRACT()だが、Ⓜ DAY(), HOUR(), Ⓟ TO_CHAR()のような型変換も使える

よく使う数値や日時の計算

用途	MySQL/MariaDB	PostgreSQL	SQL Server
四捨五入	ROUND(数値), ROUND(数値, 桁数)	ROUND(数値), ROUND(数値, 桁数)	ROUND(数値, 桁数)
商（整数除算）	数値 DIV 数値	DIV(数値, 数値)	整数同士の場合整数除算
余り	数値 MOD 数値, 数値 % 数値	MOD(数値, 数値)	数値 % 数値
日時から「日」を取り出す	EXTRACT(DAY FROM 日時)	EXTRACT(DAY FROM 日時)	DATEPART(DAY, 日時)
経過日数	TIMESTAMPDIFF(単位, 開始日, 終了日) TIMESTAMPDIFF(DAY, 開始日, 終了日)	AGE(開始日, 終了日) EXTRACT(DAY FROM(AGE(開始日, 終了日))	DATEDIFF(単位, 開始日, 終了日) DATEDIFF(DAY, 開始日, 終了日)

文字列関連 ➡6.2節

- LIKEで任意の1文字「_」と任意の文字列「%」を使ったパターン検索が可能
- Ⓜ はREGEXPで、Ⓟ は~（チルダ記号）で正規表現による検索が可能、Ⓢ はLIKEで[]と[^]による範囲指定が可能
- 標準SQLでもSIMILAR TOで正規表現が定義されたがあまり使われていない
- 演算子以外にも各DMBSでパターン検索やパターンに基づく変換などを行う関数、全文検索を行う手段などが用意されている

例「A」から始まる5文字（=文字「A」+任意の4文字）

```
Ⓜ Ⓟ Ⓢ   SELECT ～ WHERE 列名 LIKE 'A____';
Ⓜ Ⓟ     SELECT ～ WHERE 列名 REGEXP '^A.{4}$';
Ⓟ       SELECT ～ WHERE 列名 ~ '^A.{4}$';
```

正規表現の^は先頭、$は末尾、.は任意の1文字
{回数}は直前パターンの繰り返しを表している

例「A～C」から始まる文字列

```
Ⓜ Ⓟ Ⓢ   SELECT ～ WHERE 列名 LIKE 'A%'
                    OR 列名 LIKE 'B%';
                    OR 列名 LIKE 'C%';
Ⓜ Ⓟ     SELECT ～ WHERE 列名 REGEXP '^[A-C].*$';
Ⓟ       SELECT ～ WHERE 列名 ~ '^A.{4}$';
Ⓢ       SELECT ～ WHERE 列名 LIKE '[A-C]%'
```

正規表現で使用できる記号 Ⓜ Ⓟ

記号	説明
.	任意の1文字
[…]	括弧内のいずれかの文字に一致。[^…]でその否定
*	直前のパターンの0回以上の繰り返し
+	直前のパターンの1回以上の繰り返し
?	直前のパターンの0回か1回の繰り返し
ab|cde|fg	「ab」または「cde」または「fg」という文字列
^	文字列の先頭
$	文字列の末尾
{n}	直前のパターンのn回の繰り返し
(abc){2,3}	文字列「abc」の2回以上3回未満の繰り返し

おもな文字列関数

用途	MySQL/MariaDB/PostgreSQL	SQL Server
文字列の長さ	CHAR_LENGTH(文字列)	LEN(文字列)
文字列の位置	POSITION(文字列1 IN 文字列2)	CHARINDEX(文字列1 文字列2)
文字列の一部を取り出す	SUBSTRING(文字列, 開始位置, 文字数)	（←と同様）
文字列の一部を置換	REPLACE(文字列, 対象文字列, 置換文字列)	（←と同様）
空白を取り除く	TRIM(文字列), LTRIM(文字列), RTRIM(文字列)	LTRIM(文字列), RTRIM(文字列)
文字列の連結	CONCAT(文字列1, 文字列2, 文字列3…) CONCAT_WS(区切り文字, 文字列1, 文字列2, 文字列3…)	（←と同様）

CASE式 ➡6.7節

おもに値を置き換えたい場合に使用する。
書き方は2種類。 例 ➡1.1節, 6.7節, 7.2節

列の値で判定する（単純CASE式）

```
CASE 列 WHEN 値1 THEN 値A
        WHEN 値2 THEN 値B …
        ELSE 値C END
```

式で判定する（検索CASE式）

```
CASE WHEN 式1 THEN 値A
     WHEN 式2 THEN 値B …
     ELSE 値C END
```

- SELECT, WHERE, ORDER BY, GROUP BY, HAVINGで使用可能
- ELSEは省略可能（該当がない場合はNULLになる）
- THENおよびELSEで指定する値の型を揃える（すべて数値など）

サブクエリ　副問い合わせ ➡6.8節

抽出した結果を使って別の抽出を行いたい時などに使用する。
例 ➡6.8節, 7.2節, 7.3節

```
SELECT * FROM bungu  例 平均価格より高い商品を取得
WHERE
  std_price > (SELECT AVG(std_price) FROM bungu);
```

- SELECT文とSELECT文を組み合わせる
- SELECT, FROM, WHERE, HAVING, JOINで使用可能

例 SELECT結果に値が含まれている/いない

```
WHERE 列 IN (サブクエリ)、NOT IN (サブクエリ)
```

例 SELECT結果すべての値と比較
▶列の値がサブクエリの結果すべてより大きい

```
WHERE 列 > ALL (サブクエリ)
```

▶列の値がサブクエリの結果いずれかより大きい

```
WHERE 列 > SOME (サブクエリ)  ※ANYでも可
```

例 SELECT結果があるか？

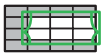
```
WHERE EXISTS (サブクエリ)
WHERE NOT EXISTS (サブクエリ)
```

ウィンドウ関数 ➡6.9節

データを区切って集計したり順位付けしたい場合などに使用する。
例 ➡6.8節, 7.2節, 7.3節

- 複数の区画を作りそれぞれで集計や順位をつける
- 区画はOVER()で定義、何に着目するかはPARTITION BYで指定、順番を求める場合はOVER()の中にORDER BYも入れる
- 各区画にWINDOW AS～で名前を付けることもできるが省略されることが多い（無名ウィンドウ）
- SELECT, ORDER BYで使用可能

Window!



```
SELECT
  order_code,
  ROW_NUMBER()
    OVER (PARTITION BY order_code
        ORDER BY order_code, bungu_code),
  bungu_code,
  act_price,
  qty
FROM hanbai;
```

order_code		bungu_code	act_price	qty
XX0011	1	S01	120	10
XX0011	2	S02	225	5
XX0012	1	S01	96	150
XX0012	2	S02	200	150
XX0012	3	S03	80	100
XX0013	1	S02	225	10
XX0013	2	S04	252	10

order_codeごとのPARTITION

PARTITION単位のROW_NUMVER

OVER()と組み合わせて使用する

集計・番号

関数	意味
COUNT(), AVG(), MAX(), MIN()	件数、平均、最大、最小
ROW_NUMBER()	行番号
RANK()	順位（1位、2位、2位の次は4位）
DENSE_RANK()	順位（1位、2位、2位の次は3位）
PERCENT_RANK()	相対順位

行の取得

関数	内容
LAG()	前の行（自分がどの行に対して遅れ=ラグがあるか）
LEAD()	後の行（自分がどの行に対してリードしているか）
FIRST_VALUE()	PARTITION BYで区切られた区間の最初の行
LAST_VALUE()	区間の、現在までの最後の行
NTH_VALUE()	区間の、これまで取得した中での順位を指定して取得。scoreの2位であればNTH_VALUE(score, 2)

MySQL
MariaDB
PostgreSQL
SQL Server
対応

標準SQL &DB設計 Quickリファレンス 2/2

NULLの変換 ➡6.2節

```
COALESCHE(値1, 値2, 値3…)
```

- 最初に登場したNULL以外の値を返す
- すべてNULLの場合はNULL

```
NULLIF(値1, 値2)
```

- 値1と値2が一致していたらNULLを返す
- 両方NULLの場合はNULL

データの登録と更新、削除

```
追加 INSERT INTO テーブル(列1, 列2, 列3)
          VALUES (値1,値2,値3), (値1,値2,値3)…;
追加 INSERT INTO テーブル(列1, 列2, 列3)
          SELECT ～;
```

- 列名を省略した場合はVALUESですべての列に対する値を指定する ➡2.4節
- 省略した列にはCREATE TABLEで設定したDEFAULTの値、DEFAULTがない場合はNULL（NOT NULLの場合はエラー）
- VALUESの代わりにSELECT文を書くことも可能 ➡6.11節

```
更新 UPDATE テーブル SET 列1=値1,列2=値2 ～ WHERE ～;
```

- WHEREに該当したデータが対象、WHEREがない場合は全件 ➡2.5節
- SETでCASE式やSELECT文を書くことも可能 ➡2.5節, 6.11節

```
削除 DELETE FROM テーブル WHERE ～;
```

- WHEREに該当したデータが対象、WHEREがない場合は全件 ➡2.6節

登録/更新/削除を練習するときはTRANSACTIONが便利 ➡6.11節