SQLコーディング規約（PostgreSQL）

Future Enterprise Coding Standards

Table of Contents

本コーディング規約は、世の中のシステム開発プロジェクトのために無償で提供致します。  
ただし、掲載内容および利用に際して発生した問題、それに伴う損害については、フューチャー株式会社は一切の責務を負わないものとします。  
また、掲載している情報は予告なく変更することがございますので、あらかじめご了承下さい。

# はじめに

## 前提条件

本書は、SQL コーディング規約についてまとめたものである。  
今回 RDBMS として採用する PostgreSQL での SQL の使用を前提に記述している。

# SQL コーティング規約（可読性・管理性）

本章では可読性・管理性を高めることを目的としたコーディング規約について記載する。

## 書式全般

書式全般についてのコーディング規約を下記に示す。

* 1 行につき、1 文のみを記述する。
* SQL 中のインデントは、Java コーディング規約にあわせて半角スペースではなくタブ文字とする。
* ヘボン式ローマ字を使用する。
* 外来語に関しては、原語の綴りを使用する。
* 横は 80 文字を目安に改行する。
* 定数を条件に用いる場合やインライン・ビューで取得したいデータなど開発者の意図はコメントにて記載する。

## 予約語

予約語に対しては、小文字を使用する。(例 ： select、insert、update、delete 等)

## 予約語以外

予約語以外に対しても、予約語と同様に小文字を使用する。(例 ： オブジェクト名、カラム名 等)

## 短縮名称

SQL 中に記述するエイリアス名など単語の短縮について示す。

1. 外来語に関しては、原語の短縮形を使用する。短縮形が存在しない場合には、母音を抜かして表記する。  
   例) corporation → corp / computer → cmptr
2. ローマ字の短縮は、単語の区切れの頭文字、または母音を抜かした子音字等を利用する。  
   例) nichijo → nchj

* カラムには必ずテーブルエイリアスを付与する
* テーブルのエイリアスは必ず付与すること。  
  必要ない場合（単一テーブルへの SELECT 等）も必ず付与すること  
  また、テーブルのエイリアス名は同 SQL 文の中で重複しないように命名すること。  
  （副問い合わせで利用したエイリアス名をメインの SQL 中のエイリアス名に利用しない。など）

## 文字コード

SQL ファイルの文字コード（エンコーディング）は Java ソースファイルと同じく『 Unicode UTF-8 』で保存する。

## 不要な空白文字（スペース）は除去する

不要な空白文字（スペース）は除去すること。

## SQL 文の整形

DML 文の節に対する予約語は左揃えにする。  
項目ごとに改行を入れ、項目の前にはインデントを挿入する。カンマは項目の前へ記入する。  
Java ソースファイルのようにファイルの先頭にコメント行を入れると DB 分析作業に支障があるため禁止とする。  
よって SQL ファイルの先頭は必ずselect、update、insert、deleteの何れかになる。

物理カラム名、テーブル名に対応する論理名を入れる場合、その後ろに単数行コメント（--）にて記述する。  
SQL 内に挿入する単数行コメントは、/\*（半角スペース）コメント本文（半角スペース）\*/ で行う。  
,（カンマ）とANDについては各行の先頭に記述する。（以下の例を参照のこと）  
SQL フレームワークで実行する SQL の場合、SQL ステートメントの終わりを示す;（セミコロン）は記述しない。

良い例：

select  
/\*  
 処理名  
\*/  
 tbl1.column1 as column1 -- カラム1  
, tbl1.column2 as column2 -- カラム2  
from  
 table1 tbl1 -- テーブル1

WHERE 句の=、!=、is（is nullのis）の縦位置は揃える。

良い例：

where  
 a.ten\_no = b.ten\_no  
and a.kamoku\_cd != '1'  
and a.anken\_no is null

### SELECT 文

良い例：

select  
 tbl1.column1 as column1 -- カラム1  
, tbl1.column2 as column2 -- カラム2  
from  
 table1 tbl1 -- テーブル1  
where  
 tbl1.column3 = 1  
order by  
 tbl1.column1  
, tbl1.column2

### INSERT 文

良い例：

insert  
into  
 table1 -- テーブル1  
(  
 column1 -- カラム1  
, column2 -- カラム2  
, column3 -- カラム3  
) values (  
 value1  
, value2  
, value3  
)

### UPDATE 文

良い例：

update  
 table1 tbl1 -- テーブル1  
set  
 tbl1.column2 = 100 -- カラム2  
, tbl1.column3 = 100 -- カラム3  
where  
 tbl1.column1 = 10

### DELETE 文

良い例：

delete  
from  
 table1 tbl1 -- テーブル1  
where  
 tbl1.value = 1

### AND（副問い合わせ）

良い例：

and xxx = (  
 select  
 tbl1.column1  
 , tbl1.column2  
 from  
 table1 tbl1  
 )

### CASE 式

CASE 式は下記のように記載すること。

良い例：

case  
 when  
 xxx.hoge = yyy.fuga  
 and xxx.fuga = yyy.fuga  
 then  
 1  
 else  
 0  
end

case、when、then、elseの後に改行を挿入すること。  
caseの後、endの前までは 1 インデント挿入すること。

### IN 句

カンマの後にスペースを１文字入れる。

### 比較演算子

比較演算子の前後にタブ（またはスペース）を１文字入れる。

### 改行位置

select 句、order by 句、group by 句等は最初に出現するカラムとカラムの区切りのカンマ前に改行を入れること。  
select の from 句の最初に出現するテーブルと結合テーブルの区切りのカンマ前に改行を入れること。  
where 句の on 句の各条件文の（and や or の）前に改行を入れること。  
命令句の後は、ヒント句が挿入できるように改行すること。

良い例：

select  
 t1.column1 as column1  
, t2.column2 as column2  
from  
 table1 t1  
, table2 t2  
where  
 t1.column3 = 1  
and t1.column4 = t2.column4  
order by  
 t1.column1  
, t2.column2

### WITH 句

with の前後に改行を挿入すること  
また、インデントは下記のように記述すること

良い例：

-- カラムエイリアスあり  
with  
 name1 (  
 col\_alias1 -- カラム１  
 , col\_alias2 -- カラム２  
 ) as (  
 select  
 ・・・  
 )  
, name2 (  
 col\_alias1 -- カラム１  
 , col\_alias2 -- カラム２  
 ) as (  
 select  
 ・・・  
 )  
  
-- カラムエイリアスなし  
with  
 name1 as (  
 select  
 ・・・  
 )  
, name2 as (  
 select  
 ・・・  
 )

### LIMIT 句 OFFSET 句

LIMIT、OFFSET の前に改行を挿入すること

良い例：

select  
 tbl1.column1 as column1  
from  
 table1 tbl1  
order by  
 tbl1.column2 desc  
limit 5  
offset 5

### HINT 句

HINT 句は独立した行で記載すること  
HINT 内容にはインデントを付けること

良い例：

select  
/\*+  
 ここにhintを記載  
\*/  
 tbl1.column1 as column1  
from  
 table1 tbl1  
where  
 tbl1.column2 = 1

## コメント

* 修正コメント
* （修正コメントが必要な場合、）  
  処理追加の際、追加行の 1 行目の前と最終行の次の行にコメントを入れる。単一行の場合は、同一行の最後にコメントをつける。
* 良い例：
* -- 2004/04/23 仕様変更管理番号 ADD(または、MOD、DEL) 変更者名 S ←　修正開始点コメント  
  (追加処理)  
  -- 2004/04/23仕様変更管理番号 ADD(または、MOD、DEL) 変更者名 E ←　修正終了点コメント
* 単数行コメント
* SQL 内で使用する単数行コメント（カラムコメントなど）には、「 -- 」を使用する。
* 複数行コメント
* 「 /\* \*/ 」を使用する。下記に例を示す。  
  なお、前述で触れたとおり、SQL ファイルの先頭にコメントを記述することは禁止とする。
* 良い例：
* /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  /\*  
   \* コメントを始めるスラッシュとアスタリスクは、それだけを１行に置く。  
   \* それから、コメント・ブロック内の各行は縦にアスタリスクを置き  
   \* コメントがあることを強調する。  
   \* 最後に、アスタリスクとスラッシュは、それだけを１行に置く。  
   \*/  
  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
* 複数行コメントアウト
* 複数行をコメントアウトする場合は、各行を「--」でコメントアウトする。  
  「 /\* \*/ 」を使用すると、その中に「 /\* \*/ 」が存在した場合、コメントアウトが途中で切れてしまう恐れがあるため、  
  使用しない。
* 論理名の記載
* select・insert・update・mergeのカラム名記述箇所には単数行コメントでカラムの論理名を記載する。  
  select・insert・update・delete・mergeのテーブル名記述箇所には単数行コメントでテーブルの論理名を記載する。  
  論理名は ERD 等で定義された論理名と必ず一致させること。

## 型変換

代入や WHERE 句の条件設定などでデータ・タイプが異なる場合、明示的にタイプ変換を行う。（暗黙の型変換は使用しない）

## 比較演算子

「等しくない」を示す演算子は「!=」を利用する。「<>」も動作するが統一の観点から利用しない。

## ORDER BY 句

order by 数字の構文は使用せずに、カラム名を記載する。

## GROUP BY 句

group by 数字の構文は使用せずに、カラム名を記載する。

集約関数を利用する場合は必ず記載すること。（省略可能であっても省略しない）

## EXISTS 句

EXISTS 句を記載する際、サブクエリになる SELECT 句の指定は定数「1」とする。  
「\*」（ワイルドカード）や「'X'」は統一の観点から利用しない。  
また「\*」（ワイルドカード）についてはパフォーマンスの観点からも禁止とする。

良い例：

where  
 exists(  
 select  
 1  
 from  
 foo f  
 where  
 f.col1 = m.key  
 )

## AS 句

トップレベルの SELECT 句には必ずas句を記載し別名を付ける。  
同一の名前であっても AS 句を付与する。  
また、「as」は省略可能であるが、省略はしないこと。

## WHERE 句

* 論理名の記載  
  WHERE 句でカラムと式を比較する際は左辺がカラムになるように記載すること。
* 良い例：
* where  
   tbl.column1 = 1  
  and tbl.amount2 > tbl.amount3 + tbl.amount4
* 条件式の順序  
  原則として、WHERE 句で条件式を列挙する際、下記の順序を守ること。
  1. テーブル単位にまとめて順番に記述する  
     この際、テーブルの順序は FROM 句に記述した順序に準ずること。
  2. 1.のテーブル単位の中で絞り込み条件をまず記述し、その後結合条件を記述する。
* 良い例：
* from  
   a\_table a -- a\_table  
  , b\_table b -- b\_table  
  , c\_table c -- c\_table  
  where  
  -- a\_tableの絞り込み  
   a.key1 = ?  
  and a.key2 = ?  
  -- b\_tableの絞り込み  
  and b.key1 = ?  
  and b.key2 = ?  
  -- b\_tableの結合条件  
  and b.col1 = a.col1  
  -- c\_tableの絞り込み  
  and c.key1 = ?  
  and c.key2 = ?  
  -- c\_tableの結合条件  
  and c.col1 = a.col1  
  and c.col2 = a.col2  
  and c.col3 = b.col3

## COUNT 文

レコード数を数える COUNT 文の記述はcount(\*)と記述する。  
count(1)・count('x') ・count(key1)という記載は NG。

## 文字列リテラル

### エスケープシーケンス

文字列リテラル中のシングルクォーテーションのエスケープは「 '' 」とシングルクォーテーションを二つつなげた記述をする。  
「 \' 」も同様の結果が得られるが円マーク（バックスラッシュ）によるエスケープは利用しない。  
円マークをエスケープせざるを得ないときのみ円マークによるエスケープ利用して良いが、  
円マークを文字列リテラルで表現する必要のある設計自体を避けること。

※PostgreSQL は設定によって円マーク（バックスラッシュ）によるエスケープを無効にできます。（デフォルト無効）  
円マークをエスケープするときは、自プロジェクトでどちらに設定されているか確認してください。

良い例：

update  
 table\_a  
set  
 text\_data = 'that''s right'

悪い例：

update  
 table\_a  
set  
 text\_data = 'that\'s too bad'

# SQL コーディング規約（パフォーマンス性）

本章ではパフォーマンス性を高めることを目的としたコーディング規約について記載する。

## 検索

検索処理におけるコーディング規約を下記に示す。

* 中間一致、後方一致検索はインデックスを利用できないため避ける
* 検索条件で=（等号）を使用できる場合は必ず使用する
* a=1 or a=2とする方がa>0 and a<3などと記述するのよりパフォーマンス上優位な場合が多い。  
  これは a にインデックスがある場合、=であれば、インデックスが有効に使われるためである。
* 可能な限り検索条件にパーティションキーの値を指定する
* 全列ワイルドカード「\*」の使用はせず、カラム名を明記する
* インデックスによる検索を指定したい場合、下記の記載を行わない
  + インデックスカラムを含む演算に対して条件指定
  + 悪い例：
  + tbl1.col1 + 1 > 100 /\* NGパターン 右辺で演算してください \*/
  + 良い例：
  + tbl1.col1 > 100 - 1
  + インデックスカラムに関数を通した値に対して条件指定
  + 悪い例：
  + /\* NGパターン 右辺に関数を通してください \*/  
    to\_char(tbl1.col1, 'YYYYMMDD') > '20151231'
  + 良い例：
  + tbl1.col1 >= to\_date('20160101', 'YYYYMMDD')
  + インデックスカラムをORで条件指定（禁止ではないが原則行わない）
  + 悪い例：
  + (  
     /\* NGパターンINDEXが利用されない場合があります。他の方法を検討してください \*/  
     tbl1.col1 = 1  
     or tbl1.col1 = 2  
     )

## 挿入

挿入処理におけるコーディング規約を下記に示す。

* 全列ワイルドカード「\*」の使用はせず、カラム名を明記する。

## 更新

更新処理におけるコーディング規約を下記に示す。

* 主キーの値の UPDATE は原則行わない。外部キーがあればエラーになる。  
  外部キーが無い場合でも、事実上、主キーの値を利用して、検索、更新する場合は、リンクが切れてしまう。
* パーティションキーの UPDATE は原則行わない。
* VIEW を使用するデータ更新は禁止。更新は実表に対して行う。

## 削除

削除処理におけるコーディング規約を下記に示す。

* 大量件数（数百万件レベル）の delete 文発行は避ける。

## DISTINCT 句

DISTINCT は、暗黙のソート処理が行われる可能性があるため性能劣化につながる。  
EXISTS 句の使用・代替を検討すること。

悪い例：

select  
 distinct  
 d.no as dept\_no  
, d.name as dept\_name  
from  
 department d  
, employee e  
where  
 d.no = e.dept\_no -- 社員が一人以上属している部門を取得

良い例：

select  
 d.no as dept\_no  
, d.name as dept\_name  
from  
 department d  
where  
 exists(  
 select  
 1  
 from  
 employee e  
 where  
 d.no = e.dept\_no -- 社員が一人以上属している部門を取得  
 )

## IN 句

IN 句は最大 1000 個まで指定できるが、200 個程度でも ORA エラーが発生するケースがある。  
また IN 句の少しだけ異なる SQL が大量に発行されると CPU 高騰やメモリ枯渇を招く。  
従って 100 を超えるような長い IN 句は使用せず、一時表を利用して in (select ・・・ from 一時表)のように書き換える。

## NOT IN 句

原則not in（select～）は使用せずに、not exists (select～)を使用する。  
NOT IN句は、内部的にソートマージの結合をすることでテーブルをフルスキャンする場合があるため、性能が悪化する可能性がある。

## UNION 句

unionはdistinct処理が含まれパフォーマンス上問題があるため、union allを使用する。

## SELECT FOR UPDATE

* select for updateで複数行にロックをかける場合、同時実行されるとデットロックを起こす可能性があるため、１件のロックでない場合はorder byを指定する。

## 分析関数

分析関数の使用は可能だが、性能悪化を招く恐れがあるため、特性を知らない方は DBA に相談する。

## インデックス

インデックスの必要性については DBA で最終判断するため、必要とする場合は DBA へ相談する。

## 外部結合

外部結合する際、内部表（駆動表）はなるべく想定件数の少ない表にする。

# License

[CC-By-4.0](https://creativecommons.org/licenses/by/4.0/deed.ja)