関係者外秘

○○PJ関係者限り

サンプルプロジェクト

DB設計標準

第１．０版

2018年8月24日

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | 版数 | 変更日 | 区分 | 変更箇所（項番等） | | 変更内容 | 担当者 |
| 1 | 1.0版 | 2018/8/24 | - | - | - | （新規作成） | TIS |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

[1. 本書について 1](#_Toc367281818)

[1.1. 本書の目的 1](#_Toc367281819)

[1.2. 想定読者 1](#_Toc367281820)

[1.3. 本書の対象範囲 1](#_Toc367281821)

[2. 前提および制約事項 2](#_Toc367281822)

[2.1. データベース方式設計 2](#_Toc367281823)

[2.2. Nablarchの採用 2](#_Toc367281824)

[2.3. DBMS製品 2](#_Toc367281825)

[2.4. データベース設計における一般知識 2](#_Toc367281826)

[3. 命名規約 3](#_Toc367281827)

[3.1. 基本方針・共通ルール 3](#_Toc367281828)

[3.1.1. 命名手順 3](#_Toc367281829)

[3.1.2. 論理名に関する共通ルール 3](#_Toc367281830)

[3.1.3. 物理名に関する共通ルール 3](#_Toc367281831)

[3.2. DBオブジェクト別の命名規約 4](#_Toc367281832)

[3.2.1. テーブル 4](#_Toc367281833)

[3.2.2. カラム（テーブル項目） 4](#_Toc367281834)

[3.2.3. シーケンス 5](#_Toc367281835)

[3.2.4. ビュー 5](#_Toc367281836)

[3.2.5. マテリアライズド・ビュー 6](#_Toc367281837)

[3.2.6. インデックス 6](#_Toc367281838)

[4. ドメイン定義 7](#_Toc367281839)

[4.1. ドメイン定義への準拠 7](#_Toc367281840)

[4.2. ドメインのデータ種別とDBデータ型 7](#_Toc367281841)

[5. カラム順 7](#_Toc367281842)

[5.1. 基本方針 7](#_Toc367281843)

[5.2. カラム追加時の対応 8](#_Toc367281844)

[6. 制約 8](#_Toc367281845)

[6.1. 主キー制約(PRIMARY KEY) 8](#_Toc367281846)

[6.2. 外部キー制約(FOREIGN KEY) 8](#_Toc367281847)

[6.3. 非NULL制約(NOT NULL) 8](#_Toc367281848)

[6.4. ユニーク制約（UNIQUE） 8](#_Toc367281849)

[6.5. 検査制約（CHECK） 9](#_Toc367281850)

[7. デフォルト値（DEFAULT） 9](#_Toc367281851)

[8. テーブル共通項目 10](#_Toc367281852)

[8.1. 排他制御用項目 10](#_Toc367281853)

[8.2. データ削除管理用項目 10](#_Toc367281854)

[8.3. データ有効期限管理用項目 10](#_Toc367281855)

[8.4. 処理済フラグ 10](#_Toc367281856)

[8.5. レコード登録･更新記録 11](#_Toc367281857)

[9. 項目暗号化 12](#_Toc367281858)

[10. インデックス設計 12](#_Toc367281859)

[11. 非正規化設計 12](#_Toc367281860)

[12. パーティションの使用 13](#_Toc367281861)

[13. ビュー、マテリアライズド・ビューの使用 13](#_Toc367281862)

[14. ストアドプロシージャの使用 13](#_Toc367281863)

# 本書について

本書では、本PJにおけるデータベース設計標準を以下の通り定義する。

## 本書の目的

本書は、データベース設計がシステム全体を通して画一的かつ最適な状態となることを目的とする。  
具体的には以下のような状態を想定している。

* データベース設計について、標準となる設計指針・考え方が明文化されていること
* どのチーム・担当者が設計を行った場合でも、同じような設計内容となる（例えばテーブル内の項目順や項目名など）
* データベース管理者（以降、DBA）、データベース設計者が明文化された設計指針により、円滑にデータベース設計を進めることができる

## 想定読者

* DBA（データベース設計を管理し、各チームのデータベース設計申請に対する承認権限を有するチーム・担当者）
* DB設計担当者（外部設計に伴うデータベース設計（主にテーブル・項目の定義）を行い、DBAに対して設計内容の申請を行うチーム・担当者）

## 本書の対象範囲

本書の対象範囲は、『想定読者であるDBAおよびDB設計担当者が設計責務をもつデータベース設計』とする。よって、この範囲に該当しない設計事項については本書の対象外となる。以下に、対象となる設計事項、および対象外の設計事項を例示する。

＜例：対象とする設計事項＞

* テーブル設計
* インデックス設計  
  ・・・など

＜例：対象外となる設計事項＞

* データベース・インスタンスのパラメータ設計（インフラチームの担当範囲）
* 表領域設計（インフラチームの担当範囲）
* スキーマ設計（インフラ・アプリ基盤チームの担当範囲）  
  ・・・など

なお、対象の設計作業において作成する成果物は以下の通り。

＜対象設計成果物＞

* テーブル定義書
* テーブル一覧
* 採番一覧
* ER図(EDMファイル)
* ドメイン定義書
* 単語辞書
* コード設計書

# 前提および制約事項

本書における前提事項・制約事項を以下に示す。

## データベース方式設計

本書で定義する設計標準は、上位の設計概念にあたるデータベースについての方式設計（インフラ観点、アプリ観点）を大前提としており、その方式設計で定義された内容に違反するような設計は許可しない。

## Nablarchの採用

Javaアプリケーション開発/実行基盤「Nablarch」（以降、Nablarch）を採用する前提としている。よって、後述の記載において、Nablarch特有の処理方式や設計思想が背景・経緯として存在するが、その詳細を本書では説明しない。必要に応じて、Nablarchの関連ドキュメントを参照すること。

## DBMS製品

本システムで採用するDBMS製品「PostgreSQL」を前提とした記載としており、PostgreSQLに関連する用語などは、対象製品のオンラインマニュアルに掲載されている用語を使用している。

## データベース設計における一般知識

データベース設計における一般知識（関連用語や設計・モデリング手法など）について、本書の読者が理解していることを前提としており、それら一般知識についての説明は割愛する。

## 命名規約

本章では、データベース設計における命名規約について定義する。

## 基本方針・共通ルール

### 命名手順

通常、データベース設計における命名規約には「論理名」と「物理名」が存在する。本PJにおいて、対象DBオブジェクトを命名する場合には、原則、以下の手順で命名を行う。

1. 論理名の命名
2. 物理名の命名

最初に、業務的観点から対象DBオブジェクトに対して、日本語をベースとした名前を論理名として命名する。

後続となる物理名の命名では、先に決定した論理名に対し、単語辞書（論理名と物理名の対応関係）を利用し、機械的に決定・命名する。

【補足事項】

上記は、最も効率的かつ画一的に命名作業を実施できる方法との判断により採用している。

（かつ、現状のシステム開発において比較的オーソドックスな方法と想定する）

### 論理名に関する共通ルール

* 単語辞書に登録･定義されている単語を使用すること  
  （単語辞書に存在しない単語を使用したい場合は、本PJで定めた登録手続きを行うこと）
* 対象を端的に表した名前となるように、意味のある単語のみで構成し、自明な情報・冗長となる単語は含めないこと  
  （例えば、テーブル名に「テーブル」という単語をわざわざ付与する必要はない）
* 原則、それ自体が意味を持たない連番は使用しないこと
* 永続性に欠ける単語は使用しない（例：特定の製品名やバージョンなど）

### 物理名に関する共通ルール

* 使用可能な文字は半角英数字とアンダースコアとする。英字は小文字のみを使用し、大文字は使用しない

半角数字 ： **1234567890**半角英字 ： **abcdefghijklmnopqrstuvwxyz**記号 ： **\_**

* 物理名を構成する単語と単語の間はアンダースコアでつなぐ
* 物理名の長さは、最大63Byteまでとする（PostgreSQLの製品仕様上の制約）  
  ただし、接頭辞、接尾辞、命名規約内の構成ルールの都合により、DBオブジェクトによっては63Byteより短くなるものがある  
  （後述するDBオブジェクトごとの命名規約において説明）
* 物理名の長さが定義された最大長を超過してしまう場合には、以下の方法により物理名を短縮する

**＜物理名の短縮ルール＞**

物理名に含まれる全ての単語について、単語辞書に登録されている略称に置き換える。  
※元の単語の一部だけを略称に置き換えるような対応は行わない。（部分的な略称への置換とした場合、担当者によって異なる箇所を略称に置き換えてしまうことにより、結果的に同一項目に対する複数の物理名が存在することを防止するため）

* 論理名と同様、単語辞書に登録･定義されている単語のみを使用すること

## DBオブジェクト別の命名規約

以降では、本システムで使用するDBオブジェクト単位での命名規約を定義する。

### テーブル

#### 論理名

* 2.6.2 論理名に関する共通ルールに従うこと
* テーブル間で論理名が重複しないこと

#### 物理名

* 2.6.3 物理名に関する共通ルールに従うこと
* 最大57Byteとすること（後述するインデックスの物理名についての制約上）

#### 命名事例

テーブルの命名事例を以下に示す。

**命名事例：テーブル**

■論理名

　　　　メール送信要求

■物理名

　　　　mail\_send\_request

### カラム（テーブル項目）

#### 論理名

* 2.6.2. 論理名に関する共通ルールに従うこと
* 原則、対象カラムが所属するドメインのドメイン名（論理）に完全一致した論理名とすること
* 同一テーブルに同じドメインに属する項目が存在する場合など、論理名をドメイン名に完全一致させることができない場合は、以下の方法により命名を行うこと  
    
  **＜カラム名をドメイン名に完全一致させることができない場合の対処方法＞**「対象ドメイン名を修飾する単語をドメイン名の前に追加する」

例）テーブル「ユーザマスタ」にドメイン『電話番号』に属するカラム“自宅の電話番号”と“勤務先の電話番号”を含める場合  
⇒カラム「自宅電話番号」、「勤務先電話番号」を追加する。

* 他のテーブルに同じ項目が存在する場合は、原則、論理名を統一すること

#### 物理名

* 2.6.3 物理名に関する共通ルールに従うこと
* カラムの論理名に対応した物理名とすること

#### 命名事例

カラム（テーブル項目）の命名事例を以下に示す。

**命名事例：カラム（テーブル項目）**

■論理名

　　　　住所コード

■物理名

　　　　address\_code

### シーケンス

#### 論理名

* 2.6.2. 論理名に関する共通ルールに従うこと
* 本システムのシーケンス内で一意に特定可能であること

#### 物理名

* シーケンスの物理名は下記構成とする

**s**

**q**

**\_**

1. 接頭辞　『sq\_』固定 ：　3桁

②シーケンス論理名に対応する物理名 ：　最大60桁

②

①

* 物理名の構成要素「シーケンス論理名に対応した物理名」は、2.6.3 物理名に関する共通ルールに従うこと
* ただし、SERIAL型を設定することで自動生成されるシーケンスの名前については例外とする

#### 命名事例

シーケンスの命名事例を以下に示す。

**命名事例：シーケンス**

■論理名

　　　　クレジット明細ID

■物理名

　　　　sq\_credit\_detail\_id

### ビュー

#### 論理名

* 2.6.2 論理名に関する共通ルールに従うこと
* 本システムのビュー内で一意に特定可能であること

#### 物理名

* ビューの物理名は下記構成とする

**v**

**w**

**\_**

1. 接頭辞　『vw\_』固定 ：　3桁

②ビュー論理名に対応する物理名 ：　最大60桁

②

①

* 物理名の構成要素「ビュー論理名に対応する物理名」は、2.6.3 物理名に関する共通ルールに従うこと

#### 命名事例

ビューの命名事例を以下に示す。

**命名事例：ビュー**

■論理名

　　　　月次売上高

■物理名

　　　　vw\_monthly\_sales\_amount

### マテリアライズド・ビュー

#### 論理名

* 2.6.2 論理名に関する共通ルールに従うこと
* 本システムのマテリアライズド・ビュー内で一意に特定可能であること

#### 物理名

* マテリアライズド・ビューの物理名は下記構成とする

**m**

**v**

**\_**

1. 接頭辞　『mv\_』固定 ：　3桁

②マテリアライズド・ビュー論理名に対応する物理名 ：　最大60桁

②

①

* 物理名の構成要素「マテリアライズド・ビュー論理名に対応する物理名」は、2.6.3 物理名に関する共通ルールに従うこと

#### 命名事例

マテリアライズド・ビューの命名事例を以下に示す。

**命名事例：マテリアライズド・ビュー**

■論理名

　　　　月次売上高

■物理名

　　　　mv\_monthly\_sales\_amount

### インデックス

#### 論理名

* 定義不要（インデックスを論理名で管理する必要がないため）

#### 物理名

* インデックスの物理名は下記構成とする

**i**

**x**

**\_**

1. 接頭辞　『ix\_』固定 ：　3桁

②インデックスを作成するテーブルの物理名 ：　最大57桁

③連番（ “\_”　+　01～99） ：　3桁

②

①

**9**

**\_**

**9**

③

#### 命名事例

インデックスの命名事例を以下に示す。

**命名事例：インデックス**

「クレジット明細」テーブルに定義した場合の例

■物理名

　　　　ix\_credit\_detail\_01

# ドメイン定義

## ドメイン定義への準拠

テーブル内のカラムは、ドメイン定義書に定義されたドメインに属さなければならない。よって、ドメイン定義書に該当するドメインが存在しないカラムを追加する場合には、最初にドメイン定義を行うこと。

【補足事項】

カラムに直接DBデータ型を定義する設計プロセスでは、各テーブルで対象カラムが発生する都度、設計者が個別にDBデータ型・データ長の設計（選択）を行うため、対応する設計者によっては他と異なるDBデータ型・データ長として設計してしまい、システム全体として設計の整合性が取れなくなることが発生しうる。

そこで、上記の状況を回避するため、抽象化したドメイン定義を設計し、そのドメインを経由したカラム設計を行うことで、同一ドメインに属するデータ項目は常に同じDBデータ型として設計されるため、設計の整合性を維持することができる。

## ドメインのデータ種別とDBデータ型

対象カラムがどのDBデータ型を使用するかは、前述の通り、そのカラムが属するドメインのデータ種別によって決定される。ここでは、本システムが取り扱うドメインについて、そのデータ種別を大別した上で、そのデータ種別に対応するDBデータ型を明示する。結果として、本システムではここに挙げたDBデータ型以外を使用することを許可しない。

| 対象ドメインのデータ種別（大別） | 使用するDBデータ型 | ドメイン定義例 |
| --- | --- | --- |
| 文字列 | VARCHAR | プロジェクト名、コード |
| 数値（整数） | SMALLINT  INTEGER  SERIAL  BIGINT | ユーザID、売上高 |
| 数値（小数） | NUMERIC | 重量、速度 |
| 日付 | DATE | プロジェクト開始日付、パスワード有効期限 |
| 日時 | TIMESTAMP | 登録日時、更新日時 |
| 真偽値 | BOOLEAN | PM職フラグ |
| バイナリ | BYTEA | セッションオブジェクト、アップロードファイル |

SERIALは、連番を自動採番させたい数値カラムに使用する。

またSMALLINT, INTEGER, BIGINTは、それぞれの型の範囲内にカラムの数値が収まるかどうかで型を選択する。

|  |  |
| --- | --- |
| 型 | 範囲 |
| SMALLINT | -32768から+32767 |
| INTEGER | -2147483648から+2147483647 |
| BIGINT | -9223372036854775808から+9223372036854775807 |

# カラム順

テーブル内のカラム順の設計方針について説明する。

## 基本方針

業務仕様、および対象データを理解しやすい状態とし、保守容易性の維持･向上が可能なカラム順とすること。  
具体的には下記の観点を漏れなく考慮すること。

* 同一テーブルに配置するカラム間で、業務観点から密接に関連するデータ項目は隣同士、またはその近辺にグルーピングされた状態として配置すること。
* 主キーに設定したカラムはテーブルの先頭とする。
* テーブル共通項目（主に、方式設計などに依存するシステム制御用のカラム）は最後尾に配置し、業務観点で設計したカラムは、全てテーブル共通項目よりも前に配置すること。
* 業務上、主要なデータは前方に、主要でないデータは後方に配置する。
* 候補キーや外部キーは前方に配置する。
* 更新頻度が低いカラムは前方に、頻繁に更新されるカラムは後方に配置する。
* 参照頻度が高いカラムは前方に、あまり参照されないカラムは後方に配置する。
* 固定長データは前方に、可変長データは後方へ配置する。

## カラム追加時の対応

システムリリース以降の保守および保守開発において、仕様変更を起因としたテーブル変更が発生した場合、基本方針を単純に適用することはできない。この場合、対象テーブルの特性によって以下のように対応方針が異なる。

* 既存のテーブルデータを保持する必要がある場合  
  トランザクションテーブルのように、既存レコードデータをDB上で維持しなければならない場合は、DDL変更をパッチ作業（alter table）によって対応するため、追加するカラムは既存のカラム順の最後尾に追加されることになる。
* 既存のテーブルデータを保持しなくてよい場合  
  ワークテーブルのように一時的なデータ格納用として用意されているテーブルについては、alter tableによる追加対応ではなく、テーブル再作成（drop/create）とすることで、上記の基本方針に準拠したカラム順とすることが可能である。

# 制約

各種制約の設計方針について標準を定義する。

## 主キー制約(PRIMARY KEY)

主キーに対して自動的に設定される制約であるため、主キー自体の設計方針が本制約の設計方針となる。

原則、自然キーを主キーとして採用すること。（代替キーは多用しない）

【補足事項】

一般的な主キー設計に従う。なお、上記の方針において自然キーを採用するのは、テーブルデータおよびテーブル間の関連を理解しやすい状態とするためである。代替キーはそれ自体では意味を持たないデータ（単純な連番など）であるため、データ構造やデータ間の関連を理解しづらくなる。

## 外部キー制約(FOREIGN KEY)

外部キー制約を使用する。（原則、ER図上で外部キーとして関連付けたものは、DDLに外部キー制約として反映する）

【補足事項】

アプリケーション側では事前にデータ間の整合性を精査することが保証されるが、アプリケーションを介さない障害発生時や仕様変更等によるパッチ作業等での対応ミスによるデータ整合性の損失にも対応可能とするため。アプリケーションの実行においては、アプリケーション側での精査とデータベース側での二重保証によりデータ整合性に対する堅牢性を保証する。

## 非NULL制約(NOT NULL)

業務上、NULLが発生し得ないカラムには、原則として非NULL制約を付与すること。

【補足事項】

業務ロジックを簡素化するために、NULL値に意味を持たせるような業務設計・データ設計は行わない。

## ユニーク制約（UNIQUE）

以下のケースにおいて、ユニーク制約を使用する。

* 主キー以外のカラムについて、システム制御の観点から絶対に一意性を保障しなければならない場合  
  （例えば、システム利用者を特定するためのユーザID、ログインIDがテーブル構成上、主キーとならない場合）
* 主キー以外のカラムを、外部キー制約の親カラム（参照元）とする場合

【補足事項】

上記は本システムのデータ整合性を維持することを目的としている。

## 検査制約（CHECK）

検査制約は使用しない。

【補足事項】

データベースへのデータ格納時にはアプリケーション側で事前に入力値精査を行うため、データベース側での精査は不要とする。（両者で精査処理を実施した場合、業務ロジックを一元管理できなくなり保守性が低下する）

# デフォルト値（DEFAULT）

原則、デフォルト値の指定は行わない。

ただし、SERIAL型を指定することによって自動採番される値については例外とする。

【補足事項】

データ生成・加工についての責務をアプリケーションで一貫して持たせることにより、システムの保守性・可読性を維持・向上を図る。

# テーブル共通項目

本章では、本システムのアプリケーション処理方式の統一に伴い、各テーブルのカラムとして共通的に定義・利用するデータ項目について説明する。

## 排他制御用項目

排他制御を必要とするテーブルについては、以下の通り排他制御用項目をカラムとして用意すること。

| カラム論理名 | カラム物理名 | DBデータ型 | NOT NULL |
| --- | --- | --- | --- |
| バージョン番号 | version\_no | NUMERIC (10) | ○ |

【補足事項】

アプリケーション方式設計（排他制御）に従う。詳細についてはアプリケーション方式設計書の該当記載を参照のこと。

## データ削除管理用項目

本システムでは個別の業務アプリケーションにはテーブルデータの物理削除は実施させない方針としている。そのため、業務観点でテーブルデータの削除が必要な場合、個別の業務アプリケーションは論理削除を実施する必要がある。（後述の【補足事項】を参照）  
このような場合には、以下の管理用項目を対象テーブルにカラムとして用意すること。

| カラム論理名 | カラム物理名 | DBデータ型 | NOT NULL |
| --- | --- | --- | --- |
| 論理削除日 | deleted\_date | DATE |  |

なお、論理削除されたデータについての物理削除は、システム内で汎用的に用意されたテーブルクリーニング機能によってのみ実施可能である。

【補足事項】

個別の業務アプリケーションに論理削除を要求する事由は、以下の通り。

* 業務上削除したデータについて参照する機能を実現するため（即時、物理削除とする処理方式では対応できない）
* ユーザオペレーションのミス、またはアプリケーション不具合によるデータ損失を回避するため

## データ有効期限管理用項目

マスタデータを取り扱うようなテーブルでは、対象レコードの有効期間を指定させることによって、該当日時における有効データを選択させるといったケースが想定される。このような場合には以下の項目を対象テーブルにカラムとして用意すること。

| カラム論理名 | カラム物理名 | DBデータ型 | NOT NULL |
| --- | --- | --- | --- |
| 適用開始日 | apply\_start\_date | DATE | ○ |
| 適用終了日 | apply\_end\_date | DATE | ○ |

【補足事項】

対象レコードデータに管理用項目を追加することにより、対象データの登録などにおいて予約登録が行えるようになり、システム運用上での利便性向上が見込める。

## 処理済フラグ

作業用の一時的なテーブルなどにおいて、各レコードデータに対する処理実施状態を管理したい場合には、以下の項目を対象テーブルのカラムとして用意すること。

| カラム論理名 | カラム物理名 | DBデータ型 | NOT NULL |
| --- | --- | --- | --- |
| 処理済フラグ | processed\_flg | BOOLEAN | ○ |

【補足事項】

障害発生時などに備え、対象テーブルデータについてのリジューム・処理スキップを実施可能とすることが本項目の主旨である。

## レコード登録･更新記録

対象レコードデータについて、システム制御観点での更新履歴を取得する場合には、以下の項目を対象テーブルにカラムとして用意すること。

| カラム論理名 | カラム物理名 | DBデータ型 | NOT NULL |
| --- | --- | --- | --- |
| 登録日時 | insert\_date\_time | TIMESTAMP | ○ |
| 登録ユーザID | insert\_user\_id | INTEGER | ○ |
| 更新日時 | update\_date\_time | TIMESTAMP | ○ |
| 更新ユーザID | update\_user\_id | INTEGER | ○ |

【補足事項】

最低限、登録時と最終更新時の情報を保持しておけば、障害調査等での対象レコードデータのトレース作業に対して必要十分との判断により、上記項目としている。

# 項目暗号化

セキュリティ要件で取り決めた暗号化対象データは、暗号化した状態でデータベースへ格納すること。

なお、暗号化対象データについては非機能要件定義書のセキュリティ要件を、暗号化の実現方式については、関連する方式設計（アプリケーション処理方式、インフラ方式）を参照すること。

【補足事項】

本システムのセキュリティ要件を遵守における必須事項である。

# インデックス設計

原則、ユニークインデックス（B-Treeインデックス）を使用すること。

その他のインデックス（以下参照）の使用を検討する場合には、アプリ基盤チーム・性能チームへ相談すること。

【補足事項】

通常取り扱う性能対策では、カーディナリティの高いカラムに対するインデックス設計が大半を占めるため、本システムで使用可能（テーブル定義書内で選択可能）な各インデックスの特性を考慮した場合、B-Treeインデックスの使用要否が主要な検討事項になることが想定される。それ以外のインデックスについては、B-Treeインデックスと比較して一般的に利用頻度が低く、デメリットを含めた効果測定を十分に実施する必要があるため、有識者への相談を前提とする。

# 非正規化設計

原則、テーブルの非正規化は行わない。

ただし、以下のようなケースでは、非正規化によるメリットを享受できる可能性があるため、適用を検討する際には、アプリ基盤チーム・性能チームと相談の上で適用可否を判断すること。

■非正規化が有効となるケース

* テーブル結合・複数テーブル参照の回避　（性能改善）  
  テーブル結合を必要とする正規化済みテーブルに対し、同一項目を両テーブルで持つように冗長化することにより、  
  テーブル結合が不要な単一表でのアクセスとする。
* 月次集計データ（断面）の取得　（機能実現・性能改善）  
  日次の履歴データが膨大に存在する場合に、日次集計データを事前に累計・集計しておくためのテーブルを追加する。
* 導出項目の項目化　（性能改善）  
  ⇒残高のような複数のデータ項目による演算を必要とするデータ項目を、テーブルのカラムとして追加定義（永続化）し、演算結果を事前に格納する。
* 連続データの同一レコード化　（保守性向上・性能改善）  
  ⇒複数レコードとして扱われる月別データ（1～12月）を同一レコード内の別項目として含める。

【補足事項】

一般的なデータベース設計セオリーである「一事実一箇所（1 fact in 1 place）」に従い、正規化された状態（第三正規形）を前提とする。これに反し、冗長的なテーブル･項目設計を行った場合、冗長化を行った対象データ間の整合性を維持する方法を考慮しなければならなくなり、設計が複雑化してしまうため、上記方針としている。

# パーティションの使用

原則、パーティションは使用しない。

性能リスクへの対応策など、パーティションの使用を検討する場合には、アプリ基盤チーム・性能チームと相談の上、使用可否や方法を検討すること。

【補足事項】

通常のテーブル設計と比べて設計の難易度が上がり、かつメンテナンスコストが増加するため、必要がない場合には使用させない。

# ビュー、マテリアライズド・ビューの使用

原則、ビュー、およびマテリアライズド・ビューについては使用しない。

これらの使用を検討する場合には、アプリ基盤チーム・性能チームと相談の上、使用可否や対応方法を検討すること。以下にビュー、マテリアライズド・ビューが有効となる対応例を示す。

■有効となるケース

* テーブル結合や複数のテーブル参照が発生し複雑化している業務ロジックについて、ビューを使用することにより業務ロジックを単純化することで、保守性を向上させることが可能となるケース
* マテリアライズド・ビューを使用することにより、検索に時間を要していた業務ロジック部分を事前に算出して格納することが可能となるため、性能対策として有効となるケース
* 他システムなどのスキーマやインスタンスを跨いだデータ連携において、ビュー・実体を持つビューを利用するケース

【補足事項】

以下に挙げたデメリットを考慮し、ビュー、マテリアライズド・ビューの使用を規制する。

* ビュー自体は検索クエリの集合でしかないため、多用することにより性能劣化を招く可能性がある。
* マテリアライズド･ビューでは実体をもつため、参照元テーブルとのデータ同期などのメンテナンスが必要となり、通常のテーブルへのデータ格納に比べて管理コストがかかる。

# ストアドプロシージャの使用

原則、ストアドプロシージャは使用しないこと。

ストアドプロシージャの使用を検討する場合、アプリ基盤チームへ相談をすること。なお、有効と想定されるケースは以下の通り。

■有効となるケース

* APサーバとDBサーバ間でのネットワーク負荷がボトルネックとなっている性能課題に対して、ストアドプロシージャの使用によりデータベースサーバ側に処理を寄せることでボトルネックの解消を図るケース
* 特定の業務機能に依存していないシステムメンテナンス処理など業務色の薄い機能について、ストアドプロシージャを使用した方が、実装は容易となり、なおかつ保守運用を含めたトータルで費用対効果が高いと判断できるケース

【補足事項】

ストアドプロシージャの使用によって、業務ロジックの実装箇所がアプリケーションサーバ（Java言語）とデータベースサーバ（ストアドプロシージャ）のそれぞれに分散してしまう。これにより、仕様変更時などに考慮すべき箇所が必然的に増え、考慮漏れなどが発生しやすくなり、システムの保守性低下につながることが懸念されるため、その使用を規制する。