Confidential

\_\_\_\_\_\_\_\_ PJ related persons only

Sample Project

DB Design Standards

Version No. 1.0

August 24, 2018

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | Version No. | Revision date | Classification | Changes (item numbers, etc.) | | Revision details | Contact person |
| 1 | Version 1.0 | 8/24/2018 | - | - | - | (New creation) | TIS |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

[1. Information on this document 1](#_Toc42859247)

[1.1. The purpose of this document 1](#_Toc42859248)

[1.2. Expected readers 1](#_Toc42859249)

[1.3. Scope of this document 1](#_Toc42859250)

[2. Assumptions and restrictions 2](#_Toc42859251)

[2.1. Database architecture design 2](#_Toc42859252)

[2.2. Adoption of Nablarch 2](#_Toc42859253)

[2.3. DBMS product 2](#_Toc42859254)

[2.4. General knowledge of database design 2](#_Toc42859255)

[3. Naming conventions 3](#_Toc42859256)

[3.1. Basic policy and common rules 3](#_Toc42859257)

[3.1.1. Naming procedure 3](#_Toc42859258)

[3.1.2. Common rules for logical names 3](#_Toc42859259)

[3.1.3. Common rules for physical names 3](#_Toc42859260)

[3.2. Naming convention for each DB object 4](#_Toc42859261)

[3.2.1. Table 4](#_Toc42859262)

[3.2.2. Column (table field) 4](#_Toc42859263)

[3.2.3. Sequence 5](#_Toc42859264)

[3.2.4. View 5](#_Toc42859265)

[3.2.5. Materialized views 6](#_Toc42859266)

[3.2.6. Primary key 6](#_Toc42859267)

[3.2.7. Unique key 7](#_Toc42859267)

[3.2.8. Index 7](#_Toc42859267)

[4. Domain definition 8](#_Toc42859268)

[4.1. Compliance with domain definitions 8](#_Toc42859269)

[4.2. Domain data type and DB data type 8](#_Toc42859270)

[5. Column order 9](#_Toc42859271)

[5.1. Basic policy 9](#_Toc42859272)

[5.2. Response when adding a column 9](#_Toc42859273)

[6. Constraints 10](#_Toc42859274)

[6.1. Primary key constraint (PRIMARY KEY) 10](#_Toc42859275)

[6.2. Foreign key constraint (FOREIGN KEY) 10](#_Toc42859276)

[6.3. NOT NULL constraint (NOT NULL) 10](#_Toc42859277)

[6.4. Unique constraint (UNIQUE) 10](#_Toc42859278)

[6.5. Check constraint 10](#_Toc42859279)

[7. Default 11](#_Toc42859280)

[8. Table common fields 12](#_Toc42859281)

[8.1. Fields for exclusive control 12](#_Toc42859282)

[8.2. Data deletion management fields 12](#_Toc42859283)

[8.3. Data expiration date management fields 1](#_Toc42859284)2

[8.4. Processed flag 1](#_Toc42859285)2

[8.5. Record registration/update 1](#_Toc42859286)3

[9. Item encryption 1](#_Toc42859287)4

[10. Index design 1](#_Toc42859288)4

[11. Denormalization 14](#_Toc42859289)

[12. Using partitions 15](#_Toc42859290)

[13. Using views, materialized views 15](#_Toc42859291)

[14. Using stored procedures 15](#_Toc42859292)

# Information on this document

In this document, the database design standard in this PJ is defined as follows.

## The purpose of this document

The purpose of this document is to ensure that the database design is in a uniform and optimal state throughout the system.   
Specifically, the following state is assumed.

* Standard design guidelines and concepts for database design must be clearly stated
* Regardless of the team or person in charge of the design, the design should be the same (for example, order of the field in the table and field names)
* Database administrators (hereinafter referred to as DBAs) and database designers can smoothly proceed with database design according to written design guidelines.

## Expected readers

* DBA (a team/person who manages database design and has the authority to approve the application of each team's database design)
* DB designer (a team/person who performs database design (mainly definition of tables and fields) with external design and submits the design contents to DBA)

## Scope of this document

The scope of this document is "Database design in which the DBA and DB designers, who are the intended readers, have design responsibilities". Therefore, design items that are not within this scope are not covered by this document. The following are examples of target and non-target design items.

<Example: Included design items>

* Table design
* Index design,  
  ・・・etc.

<Example: Excluded design items>

* Parameter setting of database instances (infrastructure team responsibility)
* Tablespace design (infrastructure team responsibility)
* Schema design (infrastructure/application platform team responsibility)  
    
   etc.

The deliverables created in the target design work are as follows.

<Target design deliverables>

* Table definition document
* Table list
* Numbering list
* ER diagram(EDM file)
* Domain definition document
* Word dictionary
* Code design document

# Assumptions and restrictions

The assumptions and restrictions in this document are as follows.

## Database architecture design

The design standards defined in this document is based on a general premise of a architecture design (infrastructure viewpoint, application viewpoint) for a database, which is a higher-level design concept and does not permit design that violates the contents defined in the architecture design.

## Adoption of Nablarch

It is assumed that the Java application development/execution platform "Nablarch" (hereinafter referred to as "Nablarch" in this document) will be adopted. Therefore, in the following description, the processing method and design concept unique to Nablarch are present as the background and history, but the details are not described in this document. Refer to the relevant Nablarch documentation as needed.

## DBMS product

The descriptions have been written based on the premise of DBMS product "PostgreSQL" that is used in this system, and terms used related to PostgreSQL are described in the online manual of the target product.

## General knowledge of database design

It is assumed that readers of this document have a general knowledge of database design (related terms, design/modeling techniques, etc.), and descriptions of such general knowledge are omitted.

# Naming conventions

This chapter defines the naming conventions for database design.

## Basic policy and common rules

### Naming procedure

The naming conventions used for database design generally consists of "logical names" and "physical names". In this PJ, use the following procedure in principle when naming the target DB object.

1. Naming logical names
2. Naming physical names

First, assign a logical name to the target DB object from a business viewpoint.

For naming the subsequent physical names, the word dictionary (correspondence between the logical name and physical name) is used for the logical name determined previously, and it is determined and named automatically.

[Supplementary note]

The above is adopted by judging that it is the most efficient and uniform method for the naming operation.

(And it is assumed to be a relatively orthodox method in the current system development)

### Common rules for logical names

* Use registered words defined in the word dictionary  
  (to use a word that does not exist in the word dictionary, follow the registration procedure specified in this PJ.)
* Use only meaningful words so that the target name is simply described, and do not include obvious or redundant words.  
  (For example, there is no need to add the word "table" to the table name.)
* In principle, do not use serial numbers that have no meaning by themselves
* Do not use non-persistent words (Example: Specific product names or versions)

### Common rules for physical names

* Available characters are alphanumeric characters and underscore. Use only lowercase letters and not uppercase letters

Half-width numbers: **1234567890**Alphabetic characters: **abcdefghijklmnopqrstuvwxyz**Symbols: **\_**

* Connect the words that make up the physical name with underscores
* The maximum length of the physical name is 63 bytes (restriction in PostgreSQL product specifications)  
  However, depending on the prefix, suffix, and configuration rules in the naming convention, some DB objects may be shorter than 63 bytes.  
  (Explained in the naming convention for each DB object described later)
* If the length of the physical name exceeds the defined maximum length, shorten the physical name by the following method

**<Shortening rules for physical name>**

All words included in the physical name are replaced with abbreviations registered in the word dictionary.   
\* Do not replace only a part of the original word with an abbreviation. (This is to prevent the existence of multiple physical names for the same fields due to replacement of different parts of the original word with partial abbreviations by different members)

* Similar to logical names, use only words registered and defined in the word dictionary

## Naming convention for each DB object

The naming conventions for each DB object used in this system are defined below.

### Table

#### Logical name

* Follow 2.6.2 Common rules for logical names
* Logical names must not be duplicated between tables

#### Physical name

* Follow 2.6.3 Common rules for physical names
* Maximum 57 bytes (due to restrictions on the physical name of the index described later)

#### Naming example

An example of naming the table is shown below.

**Naming example: Table**

■ Logical name

　　　　Request to send e-mail

■ Physical name

　　　　mail\_send\_request

### Column (table field)

#### Logical name

* Follow 2.6.2. Common rules for logical names
* In principle, use a logical name that exactly matches the domain name (logical) of the domain to which the target column belongs
* If the logical name cannot be completely matched with the domain name, such as when multiple fields belong to the same domain in the same table, use the following method for naming  
    
  **<Solution when the column name cannot be completely matched with the domain name>**"Add a word to qualify the target domain name before the domain name"

Example: When the columns "Home phone number" and "Work phone number" belonging to the domain "Phone number" in the table "User master"  
⇒Add the columns "Home phone number" and "Work phone number".

* If the same fields exist in another table, in principle, use the same logical name

#### Physical name

* Follow 2.6.3 Common rules for physical names
* Use a physical name corresponding to the logical name of the column

#### Naming example

An example of naming columns (table fields) is shown below.

**Naming example: Column (table fields)**

■ Logical name

　　　　Address code

■ Physical name

　　　　address\_code

### Sequence

#### Logical name

* Follow 2.6.2. Common rules for logical names
* Should be uniquely identifiable within the sequence of the system

#### Physical name

* The physical name of the sequence must be configured as follows

**s**

**q**

**\_**

1. Prefix "sq\_" fixed: 3 digits
2. Physical name that corresponds to the sequence logical name: Maximum 60 characters

(2)

(1)

* The configuration element "physical name corresponding to the sequence logical name" of the physical name must conform to 2.6.3 Common rules for physical names
* However, the name of the sequence automatically generated by setting the SERIAL type is an exception

#### Naming example

An example of naming the sequence is shown below.

**Naming example: Sequence**

■ Logical name

　　　　Credit detail ID

■ Physical name

　　　　sq\_credit\_detail\_id

### View

#### Logical name

* Follow 2.6.2 Common rules for logical names
* Should be uniquely identifiable within the view of the system

#### Physical name

* The physical name of the view must be configured as follows

**v**

**w**

**\_**

1. Prefix "vw\_" fixed: 3 digits
2. Physical name that corresponds to the view logical name: Maximum 60 characters

(2)

(1)

* The configuration element "physical name corresponding to the logical name of the view" of the physical name, must conform to 2.6.3 Common rules for physical names

#### Naming example

An example of naming the view is shown below.

**Naming example: View**

■ Logical name

　　　　Monthly sales

■ Physical name

　　　　vw\_monthly\_sales\_amount

### Materialized views

#### Logical name

* Follow 2.6.2 Common rules for logical names
* Should be uniquely identifiable within the materialized view of the system

#### Physical name

* The physical name of the materialized view must be configured as follows

**m**

**v**

**\_**

1. Prefix "mv\_" fixed: 3 digits
2. Physical name that corresponds to the materialized view logical name: Maximum 60 characters

(2)

(1)

* The configuration element "physical name corresponding to the logical name of the materialized view" of the physical name must conform to 2.6.3 Common rules for physical names

#### Naming example

An example of naming the materialized views is shown below.

**Naming example: Materialized view**

■ Logical name

　　　　Monthly sales

■ Physical name

　　　　mv\_monthly\_sales\_amount

### Primary key

#### Logical name

* No definition required (because there is no need to manage primary keys with logical names)

#### Physical name

* The physical name of the primary key must be configured as follows

**p**

**k**

**\_**

1. Prefix "uk\_" fixed: 3 digits
2. Physical name of the table for which the primary key is to be created: Maximum 60 characters

(2)

(1)

#### Naming example

An example of naming the primary key is shown below.

**Naming example: Primary key**

Example of primary key in "project" table

■ Physical name

　　　　pk\_project

### Unique key

#### Logical name

* No definition required (because there is no need to unique keys with logical names)

#### Physical name

* The physical name of the unique key must be configured as follows

**u**

**k**

**\_**

1. Prefix "uk\_" fixed: 3 digits
2. Physical name of the table for which the unique key is to be created: Maximum 57 characters
3. Serial number ("\_" + 01 to 99): 3 digits

(2)

(1)

**9**

**\_**

**9**

(3)

#### Naming example

An example of naming the unique key is shown below.

**Naming example: Unique key**

Example of unique key in "system account" table

■ Physical name

　　　　uk\_system\_acount\_01

### Index

#### Logical name

* No definition required (because there is no need to manage indexes with logical names)

#### Physical name

* The physical name of the index must be configured as follows

**i**

**x**

**\_**

1. Prefix "ix\_" fixed: 3 digits
2. Physical name of table to be indexed: Maximum 57 characters
3. Serial number ("\_" + 01 to 99): 3 digits

(2)

(1)

**9**

**\_**

**9**

(3)

#### Naming example

An example of naming the index is shown below.

**Naming example: Index**

Example of index in "credit detail" table

■ Physical name

　　　　ix\_credit\_detail\_01

# Domain definition

## Compliance with domain definitions

Columns in the table must belong to the domain defined in the domain definition document. Therefore, when adding a column for which no corresponding domain exists in the domain definition document, first define the domain.

[Supplementary note]

In the design process where the DB data type is directly defined in the column, the designers will individually design (select) the DB data type and data length every time a target column is defined in each table due to which the data type and data length of the DB will be different depending on the designer, and the system as a whole may not have a consistent design.

Therefore, to avoid the above situation, design an abstract domain definition and design columns through that domain so that data fields belonging to the same domain will always be designed with the same DB data type and they can be maintained easily.

## Domain data type and DB data type

The DB data type used by the target column is determined by the data type of the domain to which the column belongs, as described above. The data types are broadly classified for the domains handled by this system, and the corresponding DB data types are specified. As a result, the system does not allow the use of any data types other than those listed here.

| Data type of target domain | DB data type to use | Domain definition example |
| --- | --- | --- |
| String | VARCHAR | Project name, Code |
| Numerical value (integer) | SMALLINT  INTEGER  SERIAL  BIGINT | User ID, Sales |
| Numerical value (decimal) | NUMERIC | Weight, Speed |
| Date | DATE | Project start date, Password expiration date |
| Date and time | TIMESTAMP | Registration date and time, update date and time |
| Boolean | BOOLEAN | PM job flag |
| Binary | BYTEA | Session object, Uploaded file |

SERIAL type is used for numeric columns where you want to automatically assign serial numbers.

SMALLINT, INTEGER, and BIGINT select the type based on whether the numerical value of the column fits within the range of each type.

| Type | Range |
| --- | --- |
| SMALLINT | -32768 to +32767 |
| INTEGER | -2147483648 to +2147483647 |
| BIGINT | -9223372036854775808 to +9223372036854775807 |

# Column order

The design policy for the column order in the table will be described.

## Basic policy

Business specifications and target data should be in an easy-to-understand state and should be in column order that enables to retain and improve maintainability.   
Specifically, consider the following viewpoints without omission.

* The data fields, which are closely related from a business viewpoint between columns that are arranged in the same table, should be grouped next to each other or in the vicinity.
* The column set as the primary key should be at the beginning of the table.
* Table common fields (mainly, columns for system control that depend on architecture design, etc.) should be placed at the end of the table, and all columns designed from the business viewpoint should be placed before the table common items.
* For business purposes, key data is placed towards the beginning and non-key data is placed towards the end.
* Candidate keys and foreign key constraints (FOREIGN KEY) are placed towards the beginning.
* Columns that are updated less frequently are placed towards the beginning, and columns that are updated frequently are placed towards the end.
* Columns referenced frequently are placed towards the beginning, and columns that are not referenced often are placed towards the end.
* Fixed-length data is placed towards the beginning, and variable-length data is placed towards the end.

## Response when adding a column

If there is a change in the table due to a specification change during maintenance and maintenance development after the system release, the basic policy cannot be simply applied. The policy, in this case, differs depending on the characteristics of the target table as follows.

* When the existing table data has to be retained  
  When the existing record is required to be maintained in the DB, such as a transaction table, additional columns are added to the end of the existing column order because DDL changes will be handled by patchwork (alter table).
* When the existing table data need not be retained  
  In the case of tables that have been prepared for temporary data storage, such as worktables, the table can be recreated (drop/create) with the column order in accordance with the above basic policy instead of adding by the alter table.

# Constraints

Define standards for design policies for various constraints.

## Primary key constraint (PRIMARY KEY)

Since the constraint is automatically set for the primary key, the design policy of the primary key is the design policy for this constraint.

In principle, use the natural key as the primary key. (Do not frequently use alternate keys)

[Supplementary note]

Follow the general primary key design. The reason why natural keys are adopted in the above policy is to make it easier to understand the table data and relationship between the tables. Since substitute keys are data that have no meaning (such as a simple serial number), it is difficult to understand the data structure and the relationship between the data.

## Foreign key constraint (FOREIGN KEY)

Using Foreign key constraint (FOREIGN KEY) (In principle, items related as a foreign key constraint (FOREIGN KEY) on the ER diagram are incorporated in DDL as a foreign key constraint (FOREIGN KEY).)

[Supplementary note]

Although it is guaranteed that the application will scrutinize the consistency between data in advance, there is a possibility of loss in data integrity due to mistakes in patchwork, etc. due to failures not involving the application or changes in specifications, etc. A double guarantee is provided by scrutiny on the application side and database side for the robustness of data integrity during the execution of the application.

## NOT NULL constraint (NOT NULL)

In principle, non-NULL constraints should be assigned to columns that cannot generate NULLs.

[Supplementary note]

Business design and data design that make the NULL value meaningful are not performed to simplify business logic.

## Unique constraint (UNIQUE)

In the following cases, unique constraints are used.

* When columns other than the primary key must be absolutely unique from a system control perspective  
  (For example, when the user ID and login ID for specifying the system user are not the primary keys in the table configuration)
* When a column other than the primary key is used as the parent column (reference source) of the foreign key constraint (FOREIGN KEY)

[Supplementary note]

The above is intended to maintain the data integrity of the system.

## Check constraint

Check constraints are not used.

[Supplementary note]

When storing the data in the database, the application scrutinizes the input value in advance, so scrutiny in the database is not required. (business logic cannot be centrally managed if the application and database perform the scrutiny.)

# Default

In principle, do not specify default values.

However, the value automatically numbered by specifying SERIAL type is an exception.

[Supplementary note]

Improve the maintainability and readability of the system by making the application consistently responsible for data generation and processing.

# Table common fields

This chapter describes the data fields that are commonly defined and used as columns of each table with the unification of the application processing methods of this system.

## Fields for exclusive control

For tables that require exclusive control, prepare the columns for exclusive control as follows.

| Column logical name | Column physical name | DB data type | NOT NULL |
| --- | --- | --- | --- |
| Version number | version\_no | BIGINT | Y |

[Supplementary note]

Follow the application system design (exclusive control). For details, refer to the relevant description in the application system design document.

## Data deletion management fields

In this system, the policy of not allowing individual business applications to delete table data physically is followed. When it is necessary to delete the table data from a business viewpoint, individual business applications need to perform the logical deletion. (See [Supplementary note] below)  
For such cases, prepare the following management fields as columns in the target table.

| Column logical name | Column physical name | DB data type | NOT NULL |
| --- | --- | --- | --- |
| Delete flag | deleted\_flg | BOOLEAN | Y |
| Logically deleted date | deleted\_date | DATE |  |

The physical deletion of the logically deleted data can be performed only by a table cleaning function prepared in the system.

[Supplementary note]

The reasons for requesting logical deletion in individual business applications are as follows.

* For implementing a function to browse data that has been deleted by the business (such a function cannot be handled by a processing method that uses immediate and physical deletion)
* To avoid data loss due to user operation mistakes or application failures

## Data expiration date management fields

In the table handling master data, the case of selecting valid data corresponding to the date and time by specifying the validity period of the target record is assumed. For such cases, prepare the following fields as columns in the target table.

| Column logical name | Column physical name | DB data type | NOT NULL |
| --- | --- | --- | --- |
| Application start date | apply\_start\_date | DATE | Y |
| Application end date | apply\_end\_date | DATE | Y |

[Supplementary note]

By adding a management field to the target record, it becomes possible to perform reservation registration of the target data and is expected to improve the convenience of system operations.

## Processed flag

To manage the processing implementation status of each record in a temporary worktable, prepare the following fields as columns of the target table.

| Column logical name | Column physical name | DB data type | NOT NULL |
| --- | --- | --- | --- |
| Processed flag | processed\_flg | BOOLEAN | Y |

[Supplementary note]

The point of this item is to enable resuming and process skipping of the target table data in preparation when such a failure occurs.

## Record registration/update

To acquire the update history from system control viewpoint for the target record, prepare the following columns in the target table.

| Column logical name | Column physical name | DB data type | NOT NULL |
| --- | --- | --- | --- |
| Registered date | insert\_date\_time | TIMESTAMP | Y |
| Registered user ID | insert\_user\_id | INTEGER | Y |
| Update date and time | update\_date\_time | TIMESTAMP | Y |
| Update user ID | update\_user\_id | INTEGER | Y |

[Supplementary note]

At least, if the information of registration time and last update time is retained, the above items have retained as they are considered to be necessary and sufficient for the tracing work of the target record when investigating failures.

# Item encryption

The data to be encrypted which is determined by the security requirements should be encrypted and stored in the database.

Refer to the security requirements in the non-functional requirement definition document for the data to be encrypted, and refer to the relevant system design (application processing system, infrastructure system) for the encryption realization method.

[Supplementary note]

This is an essential item for complying with the security requirements of this system.

# Index design

In principle, use a unique index (B-Tree index).

When considering the use of other indices (see below), consult the App Platform Team/Performance Team.

[Supplementary note]

Since index design for columns with high cardinality occupies most of the performance measures usually handled, it is expected that the necessity of using a B-Tree index will be a major examination item if the characteristics of each index that can be used in this system (can be selected in the table definition document) is considered. Other indexes are generally used less often than B-Tree indexes, and consultation with experts is required since it is necessary to measure the effectiveness, including disadvantages sufficiently.

# Denormalized design

In principle, do not perform table denormalization.

In the following cases, there is a possibility that denormalization may be beneficial. Therefore, when considering the application, consult with the application platform team and performance team to determine whether to implement denormalization.

■ Case where denormalization is effective

* Avoid table joins and multiple table references (improves performance)  
  For a normalized table requiring a table join, by providing redundancy so that both tables have the same items, a single table is accessed so that a table join is not required.
* Acquisition of monthly tabulated data (cross-section) (function realization/performance improvement)  
  If the volume of daily history data is high, add a table for cumulating and totaling daily data in advance.
* Itemization of derived items (performance improvement)  
  ⇒ Additionally, define (permanently) data items that require operation using multiple data items, such as balances, like columns in a table, and store the operation results in advance.
* Conversion of continuous data to the same record (improvement of maintainability and performance)  
  ⇒ Include monthly data (January to December) handled as multiple records as separate items in the same record.

[Supplementary note]

A normalized state (third normal form) is assumed according to the general database design theory "1 fact in 1 place". On the other hand, the above policy is used since it is necessary to consider a method of maintaining consistency between the target data with redundancy, which makes the design complicated, for preparing redundant table/item design.

# Using partitions

In principle, partitions are not used.

When considering the use of partitions, such as measures against performance risks, consult the application platform team and performance team to determine the availability and method.

[Supplementary note]

Since the design is more complicated and the maintenance cost increases as compared with a normal table design, partitions are not used when it is unnecessary.

# Using views, materialized views

In principle, views and materialized views are not used.

When considering the use of these, consult the application platform team and performance team, and examine whether they can be used and how to deal with them. The following shows an example of how views and materialized views are valid.

■ Effective cases

* For business logic that has become complicated due to table joins and multiple table references, maintainability can be improved by simplifying the business logic by using views
* Since it is possible to calculate and store in advance the business logic that takes a long time to search by using materialized views, makes it effective as a performance measure
* Cases where views or materialized views are used to link data across schemas and instances of other systems

[Supplementary note]

Restrict the use of views and materialized views taking into account the following disadvantages.

* Since the view is only a collection of search queries, there is a possibility that overuse may lead to performance degradation.
* Since a materialized view has substance, maintenance such as data synchronization with the referenced table is required, and management costs are higher than storing data in a normal table.

# Using stored procedures

In principle, do not use stored procedures.

When considering the use of stored procedures, consult the application platform team. Stored procedures are considered to be effective in the following cases.

■ Effective cases

* For performance issues where the network load between the AP and DB servers is a bottleneck, the bottleneck can be eliminated by using the stored procedure to send the process to the database server
* Cases where using stored procedures make it easier to implement functions with low business consideration, such as system maintenance processing that does not depend on specific business functions and can be determined to have overall cost-effectiveness, including maintenance and operation.

[Supplementary note]

Business logic implementation locations are distributed between the application server (Java language) and database server (stored procedure) with the use of stored procedures. As a result, it is inevitable that more places have to be considered when changing the specifications, and it is likely that omissions will occur, leading to poor system maintainability due to which its use is limited.