DEVELOPMENT OF DYNAMIC ID GENERATOR EXTENSION ON POSTGRESQL

Muhamad Lukman Hakim[1], Alif Finandhita[2]
[1, 2] Teknik Informatika – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
E-mail : lukmanhakim@email.unikom.ac.id[1], alif.finandhita@email.unikom.ac.id[2]

ABSTRAK

A database requires an index number [1]. Use index numbers to distinguish one data from another data. The goal is to make it easier to find [2]. The form of index numbers can also vary in the form of rows of numbers, or a combination of numbers and letters that represent certain meanings. For examples numbering which has a prefix or prefix as in the accounting account numbering account. The auto increment feature in the database cannot provide prefixes that have prefixes automatically.

Based on this we need a function that can handle numbering that has prefixes or suffixes or prefixes and suffixes that can be run automatically. In multi-tenant architecture that must be flexible [3] and has several applications in it. The numbering function is experiencing code redundancy. The redundancy of the code requires great attention because on one hand it can be beneficial and on the other hand it can be dangerous [4]. Extension id dynamic generator is a software concept extension that was built to be the solution of the previous exposure. The dynamic id generator extension will use high-performance databases such as PostgreSQL with a speed of 7.6 times faster than MySQL [5] and users increase by an average of around 16.5% per year (2016-2019) [6].

From the test results it can be concluded that the dynamic ID generator extension on PostgreSQL can make automatic numbering that has prefixes or suffixes or prefixes and suffixes without code redundancy.

Keywords: Index, Prefix, Suffix, Extension, Multi Tenant.

1. INTRODUCTION

This section will explain the background, purpose and objectives, and theoretical basis.

1.1. Background

In order to speed up and simplify searching for information stored in a database an index number [1] is needed. Giving index numbers for data rows is unique to distinguish one from another. The form of index numbers can be in the form of rows of numbers, or a combination of numbers with letters that represent a meaning and can also have certain patterns. For examples Chart of Account numbering which has an XYYYZZ structure where X represents the asset account code, Y represents the group code and Z represents the account details. In making numbering for account details, the form X and Y will not change while Z will change according to the order of making in other words X and Y will be the prefix for Z. Seeing the complexity of the index number form, we need a function that handles number assigning The index can automatically add prefixes or suffixes or a combination of prefixes and suffixes.

Multi tenant software is software designed to serve many customers with different needs. According to S. Aulbach Multi tenant software must be able to meet diverse customer needs. To meet different needs, multi-tenant software must be built with a flexible architecture both in the software architecture and in the data scheme [2].

The problem that arises when a function is made in an application in software with multi-tenant architecture is code redundancy. According to Bayu Priyambadha Code redundancy or code duplication is a code with the same function in software without or with changes. The redundancy of the code requires great attention because on the one hand it can be beneficial but on the other hand it can be dangerous. For examples if an error occurs in a block of code but the code has been cloned to several parts of a file or application, then repairs must be made to all the same code blocks. Some code blocks need to be adapted to the logic flow where the code blocks are placed. Uncodified code blocks will cause process errors [3].

In the database there is an automatic index numbering feature. Because this function is attached to the database, if implemented in multi-tenant software there will be no function redundancy. However, a good performance database is needed. PostgreSQL which performs better than other open-source databases can process 100,000 data with the same type PostgreSQL superior to 7.6 times faster than MySQL. Whereas the delete process is almost 2 times faster [4]. In 2017 and 2018 PostgreSQL was named the "DBMS of the Year". According to a survey conducted by DBEngineRank, PostgreSQL showed an average user growth of around 16.5% per year in the period January 2016 to January 2019. In 2019 PostgreSQL users rose 15.5% from the previous year while MySQL users fell by around 11.2% [5].

Automatic index numbering which can be handled by the PostgreSQL database is in the form of auto-increment whereas based on previous exposure the index numbering form has prefixes or may have suffixes or have prefixes and suffixes. Then it is necessary to make an additional function in the database to handle numbering by adding prefixes or suffixes or prefixes and suffixes.

Based on the problems that arise in giving index numbers automatically having prefixes or suffixes or prefixes and suffixes, "Dynamic ID Generator Extension Development in PostgreSQL" will be carried out which can adjust the needs.

1.2. Purpose and Objectives

The purpose of this research is to build a dynamic id generator extension in postgresql with the aim of:

1. Create functions that can add prefixes or suffixes or prefixes and suffixes to numbering automatically.
2. Replace the numbering functions that exist in each application in a multi-tenant architecture with one extension installed in the database.

1.3. PostgreSQL Extension

PostgreSQL is the most sophisticated open-source database. Developed at the University of Berkeley led by Michael Stonebraker in 1986-1994. Open-source software means no company. With a system like this, developers have the possibility to contribute ideas to solve problems [7].

Extensions allow the expansion of functionality and special content outside the application and make it available to users when interacting with the application.

The purposes why the application supports extensions are:

1. Allow third-party developers to improve and develop application capabilities.
2. To support the ease of adding new features.
3. To reduce the size of the application.
4. To separate the source code from the application due to incompatible software licenses.

In PostgreSQL there are already many functions, operators, data types and aggregations, but sometimes users still need to create their own functions so that their needs can be met in the form of extensions [8].

1.4. Procedural Language Functions

PostgreSQL allows user-defined functions to be written in languages other than SQL and C. These other languages are generally referred to as procedural language (PL).

Functions written using procedural language cannot be directly read by the PostreSQL database. But the task can be forwarded to a special handler who knows the details of the language. The handler used will do all the work of parsing, syntax analysis, execution and so on or it can also be a "glue" between PostgreSQL and the existing implementation of the programming language used. Handler is a C language that is compiled into shared objects and loaded on demand, just like functions written in other C languages [8].

At present there is a special procedural language for the GO language under the name PLGO. PLGO is a tool that can be used to create PostgreSQL extensions using the GO language. PLGO will wrap the code in GO language so that it can be read and run by PostgreSQL.

1.5. GO

Code less, compile quicker, execute faster => have more fun !. The sentence that states in its entirety regarding the GO language.

Some Google employees feel frustrated when developing a software using C ++. It takes a long time to compile and the language is "old". Many tools and ideas have developed in the last few decades but do not have the opportunity to be influenced using the C ++ language. What they need is language that can solve problems:

1. Software must be built quickly.
2. Language must be able to work well on many platforms.
3. Language must be able to run well on computer networks.
4. Language must be easier to use.

GO was born with a dynamic language like Python or Ruby, but it has performance and security like C or Java. Built by developing from its predecessor language C / Java / C #.

GO is a new language that introduces many interesting features that make programming easier. GO utilizes a simple memory model to automatically manage multiple executions, which can be far easier than the manual approach carried out by C.

Because the initial design of GO was simple, actually GO could not be one level with C. For compile time, GO could outperform C but in the performance of a large software, C was better. GO's goal is to have a performance approach with C and make software engineering easier [9].

1.6. JSON

The JSON data type is used to store JSON (JavaScript Object Notation) data, as defined in RFC 7159. Such data can be saved in text format, but the JSON data type has advantages in its use because the data stored is valid according to JSON rules.

There are two types of JSON, json and jsonb. Both have almost identical sets of values as input. The difference is in terms of efficiency. The json data type stores an exact copy of the text input, whose processing function must be repeated at every execution, while jsonb is stored in a decomposed binary format which makes it a bit drawable to be input because of the added conversion overhead, but it is significantly faster to process, because no need to repair. Jsonb also supports indexing which is a significant advantage [8].

2. RESEARCH CONTENTS

2.1. System Analysis and Design

In this section explained the analysis and design of the system to be built.

2.1.1. Problem Analysis

Based on the previous presentation, there are two problems, namely:

1. The automatic numbering feature in the PostgreSQL database cannot add prefixes or suffixes or prefixes and suffixes. The current state of the automatic numbering feature in the PostgreSQL database is auto-increment.
2. Redundancy of code blocks used to create numbering functions in software with multi-tenant architectures can cause process errors if the initial code block has errors before being copied to applications that are on multi-tenant architectures. The current condition is that the code block for numbering is copied into two applications, for example in an accounting application and an employee management application.

Table 1. The code used for numbering in accounting applications.

```
public function numberGenerator($params = null)
{
   // get last increment value
   $LastValueQuery = "
      SELECT id
      FROM accounting
      ORDER BY id DESC
      LIMIT 1
   ";

   $getLastValue                                  =
DB::SELECT(DB::RAW($LastValueQuery))-
>get()[0];

   $incrementValue = $getLastValue + 1;

   // assign prefix and suffix value
   if($params->prefix_type == 'constant') {
      $prefix = 'ACC';
   } else {
      $prefix   =   $params->prefix   ?   $params-
>prefix : ";
   }

   if($request->suffix_type == 'constant') {
      $suffix = 'ACC';
   } else {
      $suffix = $params->suffix ? $params->suffix
: ";
   }

   // assign number pattern
   $number = $prefix . $incrementValue . $suffix;

   return $number;
}
```

Table 2. Code used for numbering in employee management applications.

```
public function numberGenerator($params = null)
{
   // get last increment value
   $LastValueQuery = "
      SELECT id
      FROM work_contract
      ORDER BY id DESC
      LIMIT 1
   ";

   $getLastValue                                  =
DB::SELECT(DB::RAW($LastValueQuery))-
>get()[0];

   $incrementValue = $getLastValue + 1;

   // assign prefix and suffix value
   if($params->prefix_type == 'constant') {
      $prefix = 'SPKWT';
   } else {
      $prefix   =   $params->prefix   ?   $params-
>prefix : ";
   }

   if($request->suffix_type == 'constant') {
      $suffix = 'PRG';
   } else {
      $suffix = $params->suffix ? $params->suffix
: ";
   }

   // assign number pattern
   $number = $prefix . $incrementValue . $suffix;

   return $number;
}
```

2.1.2. Multi Tenant Architecture Analysis

Multi-tenant is a principle of software architecture, where a software that runs on a server serves many users / tenants. The multi-tenant scheme used is multi-tenant with a single-database with details in Figure 1:

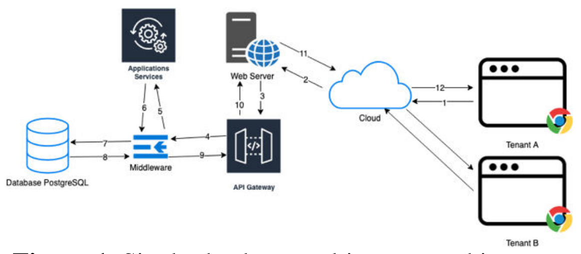


Figure 1. Single-database multi-tenant architecture.

The workflow of a single multi-tenant database architecture is as follows:

1. Tenants or users make requests for services that are in the Application Services. The tenant request contains information about what service to use.
2. Requests from tenants are channeled using cloud media or an internet connection to the web server.
3. The web server accepts requests from tenants, and forwards them to the Gateway API.
4. Gateway API forwards requests from tenants to middleware.
5. Middleware as a controller will check to where the request from the tenant will be forwarded then the middleware forwards the request to the application services.
6. In the application services there are several applications for example accounting and employee management systems. Applications services will process tenant requests based on the information available on the tenant request and the results are returned to the middleware.
7. Middleware will read the results of the process of applications services and direct the results of the process to the database.
8. After the results of the process are obtained, the database manipulates data from the results of the process either stored, deleted or changed. Then the database informs the middleware that the data has been manipulated.
9. Middleware forwards data and information on the Gateway API.
10. The Gateway API will display information and be retrieved by the web server.
11. Web server will return tenant requests via cloud or internet.
12. Tenant gets the results of requests made.

This process will run exactly the same even if the tenants who make requests are different. With architecture like this, the process in it needs to be made as dynamic as possible. According to S. Aulbach Multi-tenant software must be able to meet diverse customer needs. To meet different needs, multi-tenant software must be built with a flexible architecture both in the software architecture and in the data schema [2].

2.1.3. Numbering Analysis

Numbering is not just a series of numbers that distinguishes one information from other information. Numbering can have a certain pattern form that stores additional information. Numbering that has a pattern can be broken down into several sections.

The front of a number that doesn't change and has a certain meaning is called a prefix. Prefixes usually represent a certain group or section so the numbering used can be grouped. Whereas the back of a number that doesn't change and has a certain meaning is called a suffix. Suffixes can act like prefixes when numbering does not have a prefix or can be a detailed part of something when numbering has a combination of prefixes and suffixes.

The following sample data list of estimated accounts obtained from PT Mabra Technology Solutions can be seen in table 3.

Table 3. List of chart of accounts from PT Mabra Technology Solution.

Chart of Account			**Account Name**
1			**Aset**
	1101		Kas & Bank
		110101	Kas Kecil
		110102	Bank
	1102		**Piutang Usaha**
		110201	Piutang Usaha
	1103		Persediaan
		110301	Persediaan
	1104		**Aset Tetap**
		110401	Inventaris Kantor

Chart of account account numbering has the following rules:

1. First digit is the account classification.
2. Second to Fourth Digits (three digits) are groups of accounts.
3. The Fifth and Sixth Digits (two digits) are account details.

The numbering rules that are set in the chart of account numbering standard are only the first digits as a classification, while there are no standard rules for numbering account groups and account details.

The next data sample is the numbering of employee contracts obtained from PT Mabra Technology Solution can be seen in table 4.

Table 4. Sample data on employee contract numbering from PT Mabra Technology Solution.

Contract Number	**First Date**	**Last Date**	**Employee code**
PKWTT/ 001/CTO	26 Jul 2017	26 Nov 2055	9607001
PKWTT/ 002/MIT	26 Jul 2017	3 Mar 2056	9607002
SPKWT/ 003/PRG	5 Jan 2018	20 Jun 2018	9607003
SPKWT/ 004/PRG	21 Jun 2018	1 Jan 2019	9607003
SPKWT/ 005/PRG	21 Agu 2018	20 Okt 2018	9607004

SPKWT /006/PRG	21 Feb 2019	20 Mei 2019	9607005

Contract numbering has the following rules:
1. The first segment describes the status of the employee. PKWTT for permanent employees and SPKWT for temporary employees.
2. The second segment is the increment of the contract release sequence.
3. The third segment is the position code.

With the forms and rules for making approximate account numbering from data samples, it can be concluded that the general form of forecasting account numbering is as follows:

Prefix + Increment

The general form obtained from the contract numbering analysis from existing data samples is as follows.

Prefix + Increment + Suffix

In addition to the different numbers, the number of digits also has a difference. For examples for classification (Assets) numbering only uses one digit, while for account grouping (Cash & Bank) uses three digits and account details (Petty Cash) uses two digits.

Data needed to generate ID numbers based on case examples that are pulled in the general form are:
1. Prefix.
2. Suffix.
3. Number of digits used in increment.
4. The character used as a filler in the length of the digit increment (padding).

2.1.4. Analysis of Numbering

Based on the numbering analysis that has been done, the results of how the numbering process works are as follows:
1. Checking and taking the last increment value.
2. Check the prefix information obtained, whether there is a constant value or not.
3. Getting input data in the form of prefix and suffix information, as well as prefix and suffix forms.
4. If there is a constant value in the prefix information input data, then the prefix value will be entered with the value written in the code block.
5. If there is no constant value, then the prefix value will be entered with the prefix input value.
6. If there is a constant value in the suffix information input data, then the suffix value will be entered with the value already written in the code block.
7. If there is no constant value, then the suffix value will be entered with the suffix input value.
8. The final process is the merging of prefixes, increments and suffixes.

Based on the numbering process that has been described, a flowchart can be made that illustrates the numbering process in Figure 2.

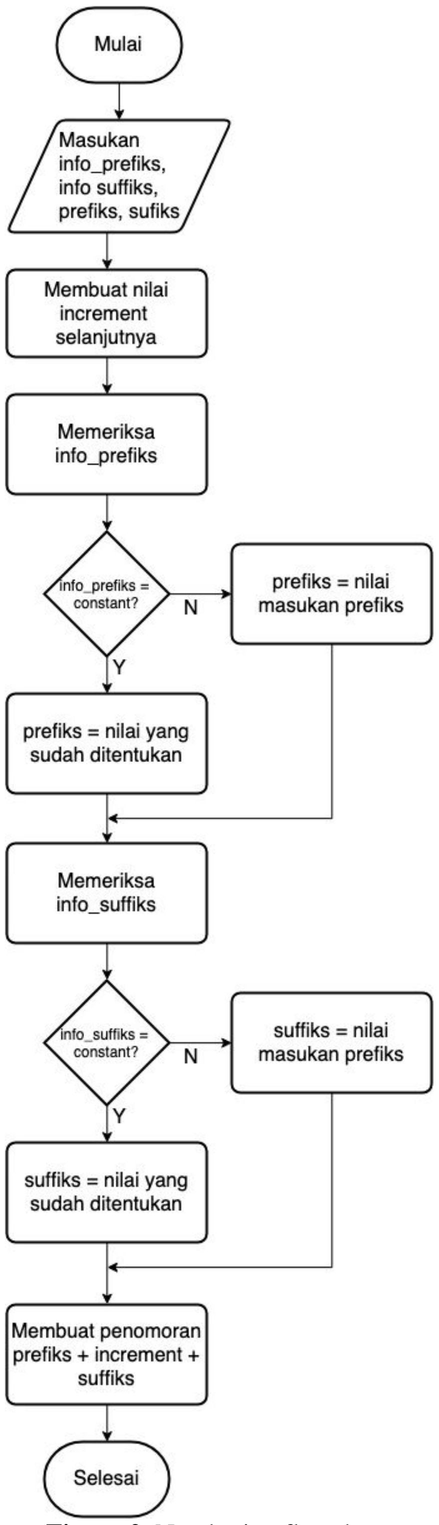

Mulai
Masukan info_prefiks, info suffiks, prefiks, sufiks
Membuat nilai increment selanjutnya
Memeriksa info_prefiks
info_prefiks = constant?
N
prefiks = nilai masukan prefiks
Y
prefiks = nilai yang sudah ditentukan
Memeriksa info_suffiks
info_suffiks = constant?
N
suffiks = nilai masukan prefiks
Y
suffiks = nilai yang sudah ditentukan
Membuat penomoran prefiks + increment + suffiks
Selesai

Figure 2. Numbering flowchart.

Based on the results of the analysis of the code block numbering functions before in the analysis of the problem as well as the numbering form in the numbering analysis can be made pseudocode which is the main basis of the logic of the numbering function.

Table 5. Pseudocode numbering making.

```
Start
   input(info_prefiks, info_suffiks, b, c)
   a = getLastIndex() + 1

   if (info_prefix == 'constant') {
      prefix = Constant Value;
   } else {
      prefix = b
   }

   if (info_suffix == 'constant') {
      suffix = Constant Value
   } else {
      suffix = c
   }

   d = prefix + a + suffix

   return d
End
```

2.1.5. Data Analysis

Analysis of the data used in this research is ORM (Object Relational Mapping) data analysis. ORM does the mapping of database tables that are owned by an entity class in an object-oriented programming language [10].

To be able to do this, a link in the form of JSON is needed to connect the ORM with the existing tables in the database. JSON format structure is divided into two, namely data objects and data arrays [10].

1. Object Data

JSON structure data object is data that is received by the database is a single data or in the form of an object.

Table 6. Object Data.

```
{
   "key": "value",
}
```

2. Array Data

JSON structure data array is data received by the database is a single data or in the form of an object with additional information in the form of an array.

Table 7. Array Data.

```
{
   "key": "value",
   "items": [
      {
         "key": "value",
      },
   ]
}
```

Numbering data storage in the form of JSON is divided into two, the first form of numbering and the second use of numbering.

Table 8. Number data storage form.

```
{
  "prefix": {
        "type" : "(constant / system)",
        "constant": "prefix"
  },
  "suffix": {
         "type": "none"
  },
  "increments": {[
         {
               "type" : "numeric",
               "length" : "length of number",
               "padchr" : "0",
               "padded" : true
         },
  ]},
}
```

Table 9. Number usage data storage.

```
"field": {
     "target  column"
  },
  "increments": {
     "value" : "next increment value",
  },
}
```

2.1.6. Functional Requirements Analysis

The following is a use case diagram of the software built.

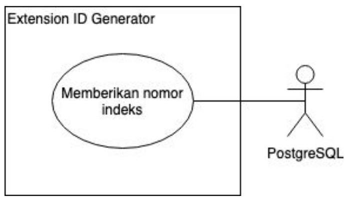


Figure 3. Use Case

2.1.7. Database Design

Database design is the stage to map the conceptual model into the database model that will be used, along with the relation scheme used by the extension id generator.

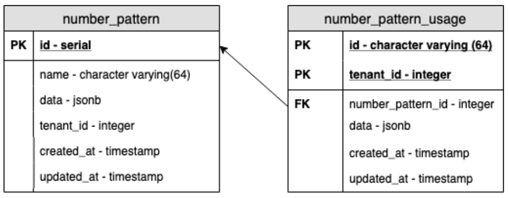


Figure 4. The relation scheme used by the extension.

2.2. System Implementation and Testing

In this section, an implementation environment is described, which is a description of the hardware and software specifications in which the built extension will be implemented. Furthermore, testing is performed on the function of the extension that has been built.

2.2.1. Implementation Environment

The implementation environment is a description of the specification in which the extension will be implemented.

1. Hardware Environment.

Following are the software specifications used for implementing the extension.

Table 10. Hardware Environment Implementation.

No.	Kebutuhan	Spesifikasi
1	Prosesor	Intel Core i5 2.7 GHz
2	Memori	8 GB
3	Penyimpanan	SSD 125 GB

2. Software Environment.

The following software specifications are used for implementing extensions.

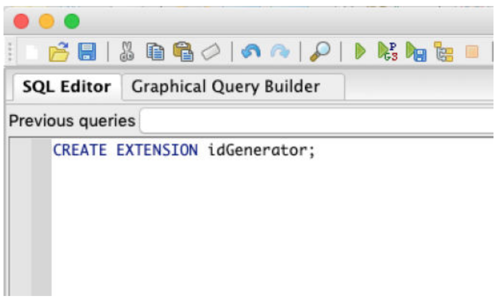
Table 11. Implementation of Software Environment.

No	Name	Sources	Info
1	PostgreSQL	PostgreSQL 9.6	Used as an operational database.
2	PGAdmin3	PGAdmin	Used to access the postgresql database.
3	PL/GO	Github: microo8/plgo	Tools used to create extensions with stored procedures and triggers in GO.
4	GO	Google	Open-source programming language. Used to create, retrieve resources and compile extensions.

2.2.2. Implement extensions

Following is the implementation of the extension in postgresql:

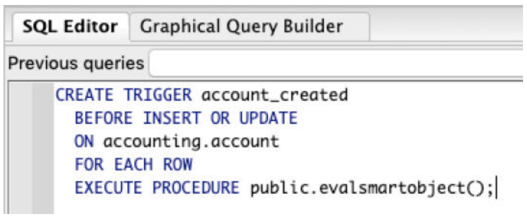
1. Use the 'CREATE EXTENSION idGenerator' command.



```
CREATE EXTENSION idGenerator;
```

Figure 5. The generator id extension for PostgreSQL uses pgAdmin.

2. Utilization of extensions in the database table is called by using query trigger before insert.



```
CREATE TRIGGER account_created
  BEFORE INSERT OR UPDATE
  ON accounting.account
  FOR EACH ROW
  EXECUTE PROCEDURE public.evalsmartobject();
```

Figure 6. Utilization of the id generator extension in the account table.

3. Information that needs to be added after the insert query in table 12.

Table 12. Information for extensions.

```
{
   "@number": {
      "prefix": {"value":" Prefix"},
      “suffix” : {“value”:”suffix”},
      "increments": [null],
      "id": "Target Column",
      "tenant_id": “tenant id”
   }
}
```

2.2.3. Testing

Testing is done on software using the black-box method focusing on functional requirements.

1. Testing Plan

The functional testing plan that will be performed on extensions uses the black-box method. Testing focuses on the requirements needed by the functional.

2. Testing Scenarios

Testing is done by trying all the possibilities that occur and testing is done repeatedly, if the test found an error will be carried out a search or repair to correct errors that occur. The test plan that will be carried out on the extension built can be seen in table 13.

Table 13. Test plan.

Process Name	Test Point	Test Method	Test Detail
Number	Insert data without prefiks	Black Box	Equivalence Partitioning
	Insert data with prefiks	Black Box	Equivalence Partitioning
	Insert data without sufiks	Black Box	Equivalence Partitioning
	Insert data with sufiks	Black Box	Equivalence Partitioning
	Insert data without prefiks and sufiks	Black Box	Equivalence Partitioning
	Insert data with prefiks and sufiks	Black Box	Equivalence Partitioning

3. Testing Evaluation

After testing the functionality, there are results of the test and it can be concluded that:

1. In testing the functionality of the dynamic id generator extension id software on postgresql, the built extension has been running as expected. Extensions can make numbering by adding prefixes or suffixes or prefixes and suffixes.
2. Previous numbering functions that exist in some applications in software with multi-tenant architecture can be removed and replaced by extensions in the database. Thus the code redundancy does not occur which can cause a problem in the future.

In the testing phase of building the dynamic id generator extension at postgresql the initial objective of building this extension has been achieved. The objectives achieved based on the test results are:

1. Create functions that can add prefixes or suffixes or prefixes and suffixes to numbering automatically.
2. Replace the numbering functions that exist in each application in a multi-tenant architecture with one number extension installed in the database.

3. CONCLUSION

Based on the Dynamic ID Generator Extension test results, the following conclusions are obtained:

1. The need for index numbering automatically by adding prefixes or suffixes or prefixes and suffixes can be fulfilled by the built function.
2. The numbering functions that exist in every application in the software with multi-tenant architecture can be replaced by the dynamic ID generator extension installed in the database. Thus the code redundancy can be eliminated.

The suggestions that can be used as a reference for developing extensions to a better direction are as follows:

1. Adds a function to increment by adding more than one increment value.
2. Save the value that has been generated but deleted so it can be used again.

REFERENCES

[1] K. C. C. G.G. Liversidge J.F. Bishop, D.A. Czekai, “United States Patent (19) 54,” vol. 96, no. 19, pp. 62–66, 1980.

[2] I. Afrianto, A. Heryandi, A. Finandhita, and S. Atin, “E-Document Autentification With Digital Signature For Smart City : Reference Model,” vol. 407, pp. 1–6, 2018.

[3] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, “Multi-tenant databases for software as a service: schema-mapping techniques,” *Proc. 2008 ACM SIGMOD Int. Conf. Manag. data - SIGMOD ’08*, p. 1195, 2008.

[4] B. Priyambadha and S. Rochimah, “Kuantifikasi Pengaruh Kloning Dan Kompleksitas Kode Terhadap Cacat Pada Evolusi Perangkat Lunak,” *JUTI J. Ilm. Teknol. Inf.*, vol. 11, no. 2, p. 19, 2014.

[5] C.-O. Truică, A. Boicea, and F. Rădulescu, “Asynchronous Replication in Microsoft SQL Server, PostgreSQL and MySQL,” *The 2013 International Conference on Cyber Science and Engineering (CYBERSE 2013).* pp. 50–55, 2013.

[6] Solid IT, “DB-Engine Ranking,” 2019. [Online]. Available: https://db-engines.com/en/ranking.

[7] B. Momjian, “PostgreSQL Introduction and Concepts,” *Read*, p. 462, 2002.

[8] T. Postgresql and G. Development, *PostgreSQL 9.6.13 Documentation*. .

[9] I. Balbaert, *The Way To Go: A Thorough Introduction To The Go Programming Language*. 2012.

[10] A. M. Bachtiar and I. I. Sukirman, “Pembangunan Perangkat Lunak Extension Browser Pada Aplikasi Pengawasan Penggunaan Internet Anak ‘ Dodo Kids Browser ’ Teknik Informatika – Universitas Komputer Indonesia Jurnal Ilmiah Komputer dan Informatika (KOMPUTA),” vol. 6, pp. 45–50, 2015.