GSoC 2022 PostgreSQL Project Proposal

Improve PostgreSQL Regression Test Coverage

1. Basic Information

- Name: DongWook Lee
- Email: sh95119@gmail.com
- Location: Seoul, South Korea (UTC+ 09:00)
- LinkedIn: https://www.linkedin.com/in/michael--lee96/
- Tech Blog: https://dongwooklee96.github.io/
- Github: https://github.com/dongwooklee96
- Available time: (6 pm - 12 pm)

2. About me

I'm DongWook Lee, a 27-years-old who is interested in database management systems. It's been a year since I graduated in computer science. I have been interested in databases since I was a student, and these interests remain the same today. I was interested in database internal, but now I'm focusing more on the aspects that I use well.

I'm currently working at CRScube which deals with clinical management as a web application developer. I usually use Flask to develop the backend, and sometimes use React.js to make the front-end and I often use PostgreSQL and MariaDB to store data. I prefer PostgreSQL because that has various extensions and communities to solve problems.

I'm familiar with C, Python, and Java and I've learned even Perl, but I'm not that used to it and forgot a lot. However I think Python and Pearl have a lot in common (internal philosophy is quite different.), and I can get accustomed to them quickly. and also I'm used to working in a Linux environment and I can use tools like Vim, GDB, strace, perf.

Over the past year, I've had almost constant time off work, therefore I can participate in the project steadily.

after work, I usually read books or write technical blogs in my free time. If I participate in GSoC and get used to writing test codes I think it will be my new hobby. And it would be a pleasure to add test codes written by others or to review the test codes written by others.

3. Why this project is important to me

I had a lot of interest in databases before, and I also wanted to contribute to open-source projects. Therefore, I tried to contribute and contributed several times.

- PostgreSQL - Add some basic regression tests for pg_freespacemap
- PostgreSQL - Improve references to term “FSM” in pageinspect and pgfreespacemap
- PostgreSQL - pg_stat_statements: Remove unnecessary call to GetUserId()
- PostgreSQL - Add link from pg_dump –encoding to supported encodings
- PostgreSQL - Add more TAP tests for pg_dump options with range checks
- Arcus - ENHANCE: optimize do_item_replace()

But what I could only contribute to was so basic or simple things. So, I applied to GSoC 2022 to have the ability to contribute one step further and it’s very interesting to have a lot of colleagues with common interests. In addition, working with a mentor who can help me is very good.

Writing test code is like understanding the internal logic of the application. It’s very important to me because I want to study PostgreSQL’s internal movements deeply.

Many functions are newly developed and modified, So I want to make PostgreSQL a more stable DBMS by creating a test code. Furthermore, I hope to help many people using PostgreSQL.

4. Project Abstract

Directory	Line Coverage		Functions	
contrib/pg_freespacemap	0.0 %	0 / 11	0.0 %	0 / 3
contrib/auth_delay	0.0 %	0 / 13	0.0 %	0 / 3
contrib/old_snapshot	0.0 %	0 / 44	0.0 %	0 / 6
contrib/tcn	0.0 %	0 / 67	0.0 %	0 / 4
contrib/pg_buffercache	0.0 %	0 / 83	0.0 %	0 / 3
contrib/pgrowlocks	0.0 %	0 / 120	0.0 %	0 / 3
contrib/pg_prewarm	0.0 %	0 / 357	0.0 %	0 / 17
src/backend/snowball/libstemmer	3.0 %	533 / 17818	5.4 %	36 / 666
src/bin/pg_test_fsync	13.7 %	31 / 227	15.4 %	2 / 13
contrib/oid2name	14.9 %	38 / 255	27.3 %	3 / 11
contrib/vacuumlo	15.3 %	43 / 281	66.7 %	2 / 3
src/backend/access/rmgrdesc	15.8 %	210 / 1329	22.2 %	14 / 63
src/bin/pg_waldump	16.6 %	392 / 2358	23.8 %	24 / 101
contrib/fuzzystrmatch	25.5 %	235 / 921	82.9 %	29 / 35
src/test/modules/test_ddl_deparse	27.0 %	63 / 233	100.0 %	7 / 7
src/interfaces/ecpg/preproc	28.8 %	3925 / 13620	87.8 %	79 / 90
src/bin/pg_test_timing	35.7 %	30 / 84	50.0 %	2 / 4
contrib/basic_archive	40.2 %	39 / 97	87.5 %	7 / 8
src/backend/utils/mb/conversion_procs/euc2004_sjis2004	40.4 %	74 / 183	100.0 %	8 / 8
src/backend/main	43.0 %	43 / 100	80.0 %	4 / 5
src/backend/utils/activity	45.4 %	404 / 890	90.9 %	30 / 33
contrib/adminpack	45.7 %	79 / 173	70.8 %	17 / 24
src/backend/utils/mb/conversion_procs/euc_jp_and_sjis	46.5 %	167 / 359	100.0 %	19 / 19
contrib/pgstattuple	50.6 %	259 / 512	73.5 %	36 / 49
contrib/xml2	51.4 %	150 / 292	69.6 %	16 / 23
contrib/sslinfo	52.2 %	72 / 138	91.3 %	21 / 23
src/interfaces/libpq	54.6 %	4534 / 8299	76.6 %	315 / 411
src/bin/initdb	55.5 %	1151 / 2075	80.0 %	80 / 100
src/bin/pg_upgrade	57.2 %	1342 / 2345	80.2 %	89 / 111
src/backend/libpq	57.3 %	2463 / 4302	84.8 %	195 / 230
src/backend/utils/mb/conversion_procs/euc_tw_and_big5	58.3 %	204 / 350	100.0 %	22 / 22
src/interfaces/ecpg/pgtypeslib	58.9 %	1994 / 3383	87.1 %	88 / 101
src/bin/psql	59.1 %	6288 / 10645	55.0 %	278 / 505
src/backend/utils/mb/conversion_procs/euc_kr_and_mic	59.2 %	42 / 71	100.0 %	7 / 7
src/backend/utils/mb/conversion_procs/euc_cn_and_mic	60.3 %	41 / 68	100.0 %	7 / 7
contrib/spi	60.9 %	204 / 335	60.0 %	9 / 15
src/timezone	63.6 %	1933 / 3040	86.4 %	102 / 118
src/interfaces/ecpg/ecpglib	64.7 %	2323 / 3590	93.8 %	121 / 129
src/bin/pg_ctl	65.4 %	464 / 709	93.3 %	28 / 30

PostgreSQL is a very actively developed project and there are so many existing codes and so many new ones are added. I think there are still many functions that have not been tested. Test codes that various cases increase development stability and other developers can make changes safely. I will write a test code for the module that has not been tested yet.

5. Features To be implemented

These are ideas that I want to do on a project.

- **Write test codes for untested parts**

When I checked the current test coverage, many modules are not covered. Therefore, first of all, I will write a test focusing on the parts that are not covered.

First, I will write a test for a module that has not been tested at all. When I checked the coverage, many extensions remained untested. So I will focus on writing test codes to increase test coverage.

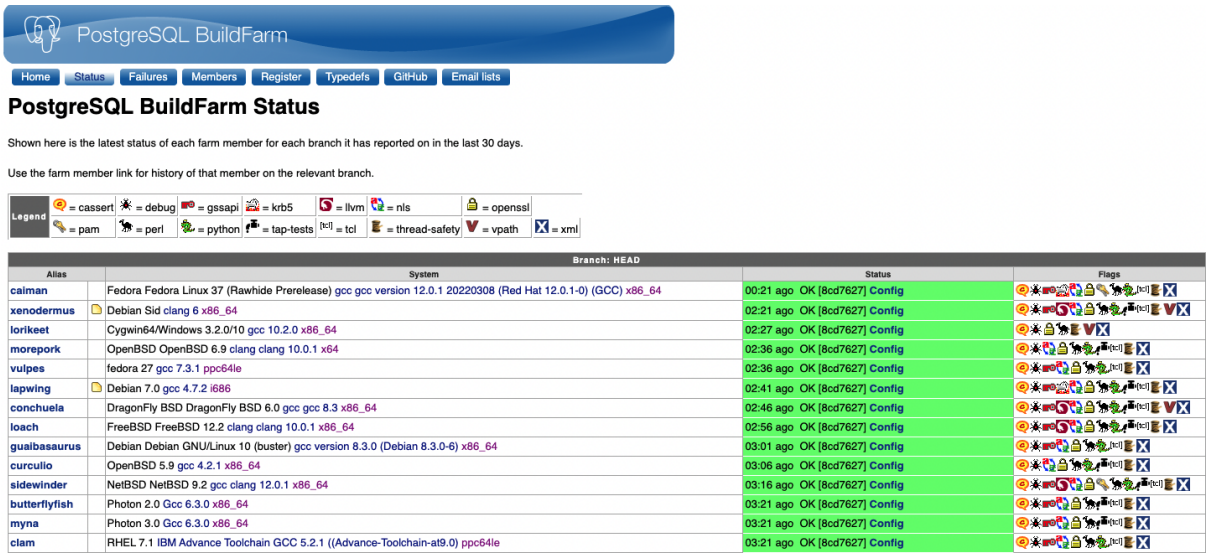
- **List of modules to test**

I made a list of modules to be tested based on modules with personal interest and less test coverage.

module	description	document
pg_dump	extract a PostgreSQL database into a script file or other archive file	https://www.postgresql.org/docs/devel/app-pgdump.html
psql	PostgreSQL interactive terminal	https://www.postgresql.org/docs/devel/app-psql.html
pg_ctl	initialize, start, stop, or control a PostgreSQL server	https://www.postgresql.org/docs/devel/app-pg-ctl.html
initdb	create a new PostgreSQL database cluster	https://www.postgresql.org/docs/devel/app-initdb.html
pg_stat_statments	It's for tracking planning and execution statistics of all SQL statements executed by a server.	https://www.postgresql.org/docs/devel/pgstatstatements.html
pg_buffercache	It's for examining what's happening in the shared buffer cache in real-time.	https://www.postgresql.org/docs/devel/pgbuffercache.html
pg_prewarm	It provides a convenient way to load realation data into either the operating system buffer cache or the PostgreSQL buffer cache.	https://www.postgresql.org/docs/devel/pgprewarm.html
pgrowlocks	It provides a function to show row locking information for a specified table.	https://www.postgresql.org/docs/devel/pgrowlocks.html
pgstattuple	It provides various functions to obtain tuple-level statistics.	https://www.postgresql.org/docs/devel/pgstattuple.html
pg_waldump	Display a human-readable rendering of the write-ahead log of a PostgreSQL database cluster.	https://www.postgresql.org/docs/devel/pgwaldump.html

pg_basebackup	Takes a base backup of a PostgreSQL cluster.	https://www.postgresql.org/docs/devel/app-pgbasebackup.html

- **Refactor the test code**

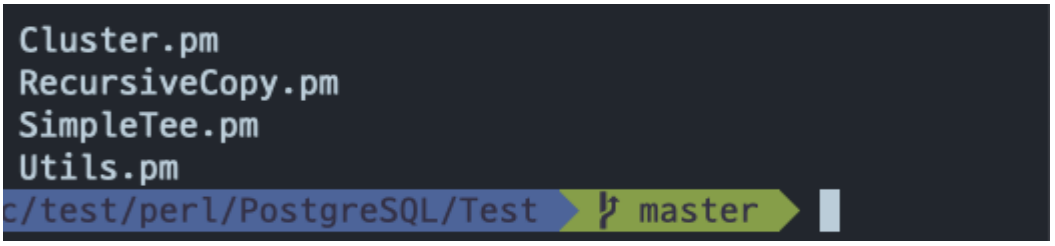


Among the existing tests, I will find the part that is being tested repeatedly and remove it. If I can improve test code performance, I will improve it. I will focus on reducing the test execution time through refactoring. Also, I will check that all tests work without errors in the PostgreSQL Build-farm.

- **Add test codes reflecting bug reports**

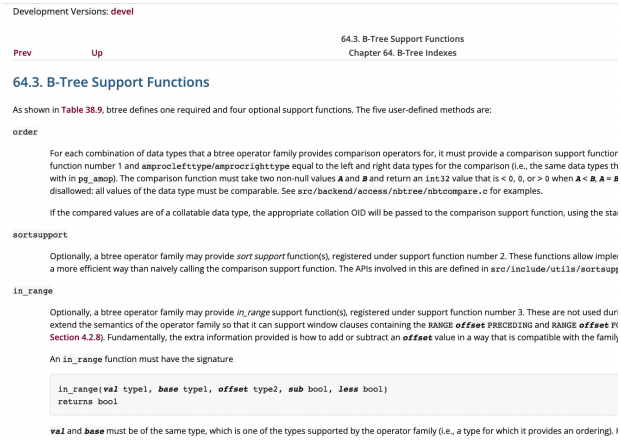
When creating or modifying a test code, search the mailing list for related parts and add them if there is a part to be tested.

- **Add function to Perl PostgreSQL module**



If Cluster.pm, RecursiveCopy.pm, SimpleTee.pm, and Utils.pm needs necessary functions, I will create or modify submodules.

- **Documentation**



Development Versions: devel

Prev Up 64.3. B-Tree Support Functions Chapter 64. B-Tree Indexes

64.3. B-Tree Support Functions

As shown in Table 38.9, btree defines one required and four optional support functions. The five user-defined methods are:

order

For each combination of data types that a btree operator family provides comparison operators for, it must provide a comparison support functior function number 1 and amprocelefttype/amprocrighttype equal to the left and right data types for the comparison (i.e., the same data types th with in pg_amop). The comparison function must take two non-null values A and B and return an int32 value that is < 0, 0, or > 0 when A < B, A = B disallowed: all values of the data type must be comparable. See src/backend/access/nbtree/nbtcompare.c for examples.

If the compared values are of a collatable data type, the appropriate collation OID will be passed to the comparison support function, using the sta

sortsupport

Optionally, a btree operator family may provide *sort support* function(s), registered under support function number 2. These functions allow imple a more efficient way than naively calling the comparison support function. The APIs involved in this are defined in src/include/utils/sortsupp

in_range

Optionally, a btree operator family may provide *in_range* support function(s), registered under support function number 3. These are not used dur extend the semantics of the operator family so that it can support window clauses containing the RANGE *offset* PRECEDING and RANGE *offset* F Section 4.2.8). Fundamentally, the extra information provided is how to add or subtract an *offset* value in a way that is compatible with the famil

An in_range function must have the signature

```
in_range(val type1, base type1, offset type2, sub bool, less bool)
returns bool
```

val and *base* must be of the same type, which is one of the types supported by the operator family (i.e., a type for which it provides an ordering). I

As I write the test, I'll learn a lot of new things. So, I will write down the new facts and contents in the document, writing a test code. Plus I can find missing content in the document.

6. Schedule

- **May 20**
    - GSoC announcement.
- **May 20 - June 12 (3 weeks)**
    - Get to know the team better.
    - Ask something to write the correct test code.
    - Start discussing how to write correct test codes.
    - Read the documents of to-be-tested modules and learn about the functions I don't know.
    - Read the already written test code and see how it is written.

- **June 13 - June 27 (2 weeks)**
    - Coding officially starts.
    - Research the module to be tested in detail.
    - Write a test code for *pg_dump*, *psql*.
- **June 27 - July 11 (2 weeks)**
    - Write a test code for *pg_ctl*, *pg_initdb.*
    - Write documents if there are to add.
- **June 11 - July 25 (2 weeks)**
    - Write a test code for *pg_stat_statments*.
    - Review the test.
    - Refactor the test code, and write documents if there are to add.
- **July 25 - July 29**
    - First evaluation between mentors and students.
- **August 1 - August 15 (2 weeks)**
    - Write a test code for *pg_buffercache*, *pg_prewarm*.
    - Look for improvements in existing test codes.
- **August 15 - August 29 (2 weeks)**
    - Add additional tests to be added by referring to documents and mailing lists.
    - Write a test code for *pgrowlocks*, *pgstattuple*.
    - Remove unnecessary or overlapping parts.

- **August 29 - September 12 (2 weeks)**
    - Write a test code for *pg_waldump*, *pg_basebackup*.
    - Review the test.
    - Refactor the test code, and write documents if there are to add.
- **September 12 - September 19**
    - Submit code and final evaluation.