
python-pgextras Documentation

Release 0.1.1

Scott Woodall

May 06, 2014

Contents:

python-pgextras

Unofficial Python port of [Heroku's pgextras](#) that provides various statistics for a Postgres instance. The biggest difference of this module is that your Postgres database can be anywhere, not just hosted with Heroku.

- [BSD license](#)
- [Documentation](#)

1.1 Features

Note: pgextras does not format the output as seen in all the examples below. Instead it returns an iterable that contains namedtuples. For example, the “Total Indexes Size” method returns the following:

```
[
    Record(table='pgbench_accounts', index_size='2208 kB'),
    Record(table='pgbench_tellers', index_size='16 kB'),
    Record(table='pgbench_branches', index_size='16 kB'),
    Record(table='pgbench_history', index_size='0 bytes')
]
```

It's up to you on how to format and present the data. The results below are formatted to make it easier to understand the data available to you.

1.1.1 Cache Hit

Calculates your cache hit rate (effective databases are at 99% and up):

name	ratio
index hit rate	0.999888
table hit rate	0.999696

1.1.2 Index Usage

Calculates your index hit rate (effective databases are at 99% and up):

relname	percent_of_times_index_used	rows_in_table
pgbench_history		149985
pgbench_accounts	99	100000

pgbench_tellers	96	10
pgbench_branches	93	1

1.1.3 Calls

Show 10 most frequently called queries:

qry	exec_time	prop_exec_time	ncalls	sync_io_time
BEGIN;	0:00:00.140968	0.0%	414,000	0:00:00
INSERT INTO pgbench_history (tid, bid..	0:00:03.788899	0.0%	414,000	0:00:00
SELECT abalance FROM pgbench_accounts..	0:00:04.471754	0.0%	414,000	0:00:00
UPDATE pgbench_accounts SET abalance ..	0:00:21.798029	0.2%	414,000	0:00:00
END;	0:00:00.126220	0.0%	414,000	0:00:00
UPDATE pgbench_branches SET bbalance ..	0:30:00.544749	15.9%	414,000	0:00:00
UPDATE pgbench_tellers SET tbalance =..	2:38:45.396566	83.9%	414,000	0:00:00
BEGIN	0:00:00.002141	0.0%	212	0:00:00
SELECT pid, application_name AS sourc..	0:00:00.039576	0.0%	77	0:00:00
SELECT exists(SELECT ? FROM pg_exten..	0:00:00.001085	0.0%	43	0:00:00

1.1.4 Blocking

Display queries holding locks other queries are waiting to be released:

blocked_pid	blocking_statement	blocking_du
1688	UPDATE pgbench_tellers SET tbalance = tbalance + -2309 WHERE tid = 9;	0:00:00.018

1.1.5 Outliers

Show 10 queries that have longest execution time in aggregate:

qry	exec_time	prop_exec_time	ncalls	sync_io_time
UPDATE pgbench_tellers SET tbalance =..	2:59:30.137916	83.9%	467,897	0:00:00
UPDATE pgbench_branches SET bbalance ..	0:33:53.945889	15.8%	467,856	0:00:00
UPDATE pgbench_accounts SET abalance ..	0:00:25.384166	0.2%	467,897	0:00:00
SELECT abalance FROM pgbench_accounts..	0:00:05.086917	0.0%	467,897	0:00:00
INSERT INTO pgbench_history (tid, bid..	0:00:04.356031	0.0%	467,848	0:00:00
vacuum pgbench_branches	0:00:00.336647	0.0%	17	0:00:00
select count(*) from pgbench_accounts ;	0:00:00.294740	0.0%	1	0:00:00
BEGIN;	0:00:00.160855	0.0%	467,897	0:00:00
END;	0:00:00.142983	0.0%	467,848	0:00:00
SELECT relname, CASE idx_scan WHEN ? ..	0:00:00.110683	0.0%	6	0:00:00

1.1.6 Vacuum Stats

Show dead rows and whether an automatic vacuum is expected to be triggered:

schema	table	last_vacuum	last_autovacuum	rowcount	dead_rowcount
public	pgbench_tellers	2014-04-24 20:02	2014-04-24 20:03	10	0

public	pgbench_branches	2014-04-24 20:02	2014-04-24 20:03	1	0
public	pgbench_history	2014-04-23 20:45		15,000	0
public	pgbench_accounts	2014-04-23 20:45		100,000	17,581

1.1.7 Bloat

Table and index bloat in your database ordered by most wasteful:

type	schemaname	object_name	bloat	waste	
-----+-----+-----+-----+-----+					
table	public	pgbench_accounts	1.3	3768 kB	
table	public	pgbench_tellers	19	144 kB	
table	public	pgbench_branches	8	56 kB	

1.1.8 Long Running Queries

Show all queries running longer than five minutes by descending duration:

pid	duration	query	
-----+-----+-----+-----+			
19578	02:29:11.200129	EXPLAIN SELECT "students".* FROM "students" WHERE "students"."id" = 145	
19465	02:26:05.542653	EXPLAIN SELECT "students".* FROM "students" WHERE "students"."id" = 188	
19632	02:24:46.962818	EXPLAIN SELECT "students".* FROM "students" WHERE "students"."id" = 158	

1.1.9 Sequence Scans

Show the count of sequential scans by table descending by order:

name	count	
-----+-----+		
pgbench_branches	57086	
pgbench_tellers	15595	
pgbench_accounts	2	
pgbench_history	0	

1.1.10 Unused Indexes

Show unused and almost unused indexes, ordered by their size relative to the number of index scans. Exclude indexes of very small tables (less than 5 pages), where the planner will almost invariably select a sequential scan, but may not in the future as the table grows:

table	index	index_size	index_scans	
-----+-----+-----+-----+				
public.grade_levels	index_placement_attempts_on_grade_level_id	97 MB	0	
public.observations	observations_attrs_grade_resources	33 MB	0	
public.messages	user_resource_id_idx	12 MB	0	

1.1.11 Total Table Size

Show the size of the tables (including indexes), descending by size:

name	size
pgbench_accounts	18 MB
pgbench_history	2904 kB
pgbench_tellers	272 kB
pgbench_branches	256 kB

1.1.12 Total Indexes Size

Show the total size of all the indexes on each table, descending by size:

table	index_size
pgbench_accounts	2208 kB
pgbench_tellers	16 kB
pgbench_branches	16 kB
pgbench_history	0 bytes

1.1.13 Table Size

Show the size of the tables (excluding indexes), descending by size:

name	size
pgbench_accounts	16 MB
pgbench_history	2904 kB
pgbench_tellers	256 kB
pgbench_branches	240 kB

1.1.14 Index Size

Show the size of indexes, descending by size:

name	size
pgbench_accounts_pkey	2208 kB
pgbench_tellers_pkey	16 kB
pgbench_branches_pkey	16 kB

1.1.15 Total Index Size

Show the total size of all indexes:

size
2240 kB

1.1.16 Locks

Display queries with active locks:

procpid	relname	transactionid	granted	query_snippet	age
31776			t	<IDLE> in transaction	00:19:29.837898
31776		1294	t	<IDLE> in transaction	00:19:29.837898
31912			t	select * from hello;	00:19:17.94259
3443			t		00:00:00
				select	
				pg_stat_activi	

1.1.17 Table Indexes Size

Show the total size of all the indexes on each table, descending by size:

table	index_size
pgbench_accounts	2208 kB
pgbench_tellers	16 kB
pgbench_branches	16 kB
pgbench_history	0 bytes

1.1.18 PS

View active queries with execution time:

pid	source	running_for	waiting	query
28023	pgbench	0:00:00.107013	0	UPDATE pgbench_accounts SET abalance = aba
28018	pgbench	0:00:00.017257	0	END;
28015	pgbench	0:00:00.001055	1	UPDATE pgbench_branches SET kbalance = bba

1.1.19 Version

Get the Postgres server version:

version
PostgreSQL 9.3.3 on x86_64-apple-darwin13.0.0, compiled by Apple LLVM version 5.0 (clang-500.2.79)

Installation

Dependencies

- `psycopg2`: `python-pgextras` will not install this for you. I do not feel comfortable forcing a specific version of your database driver module. If you come across a bug with your version of `psycopg2` please open an [issue](#).

At the command line:

```
$ pip install pgextras
```

Usage

Your “dsn” value will be specific to the postgres database you want to connect to. See the [postgres documentation](#) for more information on configuring connection strings:

```
>>> from pgextras import PgExtras
>>> with PgExtras(dsn='dbname=testing') as pg:
...     results = pg.bloat()
...     for row in results:
...         print(row)
...
Record(type='index', schemaname='public', object_name='addresses_to_geocode::addresses_to_geocode_ad
Record(type='table', schemaname='public', object_name='addresses_to_geocode', bloat=Decimal('1.2'), v
Record(type='table', schemaname='pg_catalog', object_name='pg_attribute', bloat=Decimal('2.5'), waste
```

3.1 Class Methods

3.1.1 .bloat()

```
def bloat(self):
    """
    Table and index bloat in your database ordered by most wasteful.

    Record(
        type='index',
        schemaname='public',
        object_name='pgbench_accounts::pgbench_accounts_pkey',
        bloat=Decimal('0.2'),
        waste='0 bytes'
    )

    :returns: list of Records
    """

    return self.execute(sql.BLOAT)
```

3.1.2 .blocking()

```
def blocking(self):
    """
```

Display queries holding locks other queries are waiting to be released.

```
Record(
    pid=40821,
    source='',
    running_for=datetime.timedelta(0, 0, 2857),
    waiting=False,
    query='SELECT pg_sleep(10);'
)

:returns: list of Records
"""

return self.execute(
    sql.BLOCKING.format(
        query_column=self.query_column,
        pid_column=self.pid_column
    )
)
```

3.1.3 .cache_hit()

```
def cache_hit(self):
    """
    Calculates your cache hit rate (effective databases are at 99% and up).

    Record(
        name='index hit rate',
        ratio=Decimal('0.99994503346970922117')
    )

    :returns: list of Records
    """

    return self.execute(sql.CACHE_HIT)
```

3.1.4 .calls(truncate=False)

```
def calls(self, truncate=False):
    """
    Show 10 most frequently called queries. Requires the pg_stat_statements
    Postgres module to be installed.

    Record(
        query='BEGIN;',
        exec_time=datetime.timedelta(0, 0, 288174),
        prop_exec_time='0.0%',
        ncalls='845,590',
        sync_io_time=datetime.timedelta(0)
    )

    :param truncate: trim the Record.query output if greater than 40 chars
    :returns: list of Records
    """
```



```
if self.pg_stat_statement():
    if truncate:
        select = """
            SELECT CASE
                WHEN length(query) < 40
                THEN query
                ELSE substr(query, 0, 38) || '..'
            END AS qry,
        """
    else:
        select = 'SELECT query,'

    return self.execute(sql.CALLS.format(select=select))
```

3.1.5 .index_usage()

```
def index_usage(self):
    """
    Calculates your index hit rate (effective databases are at 99% and up).

    Record(
        relname='pgbench_history',
        percent_of_times_index_used=None,
        rows_in_table=249976
    )

    :returns: list of Records
    """

    return self.execute(sql.INDEX_USAGE)
```

3.1.6 .locks()

```
def locks(self):
    """
    Display queries with active locks.

    Record(
        procpid=31776,
        relname=None,
        transactionid=None,
        granted=True,
        query_snippet='select * from hello;',
        age=datetime.timedelta(0, 0, 288174),
    )

    :returns: list of Records
    """

    return self.execute(
        sql.LOCKS.format(
            pid_column=self.pid_column,
            query_column=self.query_column
```

```
)  
)
```

3.1.7 .long_running_queries()

```
def long_running_queries(self):  
    """  
    Show all queries longer than five minutes by descending duration.  
  
    Record(  
        pid=19578,  
        duration=datetime.timedelta(0, 19944, 993099),  
        query='SELECT * FROM students'  
    )  
  
    :returns: list of Records  
    """  
  
    if self.is_pg_at_least_nine_two():  
        idle = "AND state <> 'idle'"  
    else:  
        idle = "AND current_query <> '<IDLE>'"  
  
    return self.execute(  
        sql.LONG_RUNNING_QUERIES.format(  
            pid_column=self.pid_column,  
            query_column=self.query_column,  
            idle=idle  
        )  
    )
```

3.1.8 .outliers()

```
def outliers(self, truncate=False):  
    """  
    Show 10 queries that have longest execution time in aggregate. Requires  
    the pg_stat_statements Postgres module to be installed.  
  
    Record(  
        qry='UPDATE pgbench_tellers SET tbalance = tbalance + ?;',  
        exec_time=datetime.timedelta(0, 19944, 993099),  
        prop_exec_time='67.1%',  
        ncalls='845,589',  
        sync_io_time=datetime.timedelta(0)  
    )  
  
    :param truncate: trim the Record.qry output if greater than 40 chars  
    :returns: list of Records  
    """  
  
    if self.pg_stat_statement():  
        if truncate:  
            query = """  
                CASE WHEN length(query) < 40  
                THEN query
```

```
        ELSE substr(query, 0, 38) || '...'
    END
    """
    else:
        query = 'query'

    return self.execute(sql.OUTLIERS.format(query=query))
```

3.1.9 .ps()

```
def ps(self):
    """
    View active queries with execution time.

    Record(
        pid=28023,
        source='pgbench',
        running_for=datetime.timedelta(0, 0, 288174),
        waiting=0,
        query='UPDATE pgbench_accounts SET abalance = abalance + 423;'
    )

    :returns: list of Records
    """

    if self.is_pg_at_least_nine_two():
        idle = "AND state <> 'idle'"
    else:
        idle = "AND current_query <> '<IDLE>'"

    return self.execute(
        sql.PS.format(
            pid_column=self.pid_column,
            query_column=self.query_column,
            idle=idle
        )
    )
```

3.1.10 .seq_scans()

```
def seq_scans(self):
    """
    Show the count of sequential scans by table descending by order.

    Record(
        name='pgbench_branches',
        count=237
    )

    :returns: list of Records
    """

    return self.execute(sql.SEQ_SCANS)
```

3.1.11 .total_table_size()

```
def total_table_size(self):
    """
    Show the size of the tables (including indexes), descending by size.

    Record(
        name='pgbench_accounts',
        size='15 MB'
    )

    :returns: list of Records
    """

    return self.execute(sql.TOTAL_TABLE_SIZE)
```

3.1.12 .unused_indexes()

```
def unused_indexes(self):
    """
    Show unused and almost unused indexes, ordered by their size relative
    to the number of index scans. Exclude indexes of very small tables
    (less than 5 pages), where the planner will almost invariably select
    a sequential scan, but may not in the future as the table grows.

    Record(
        table='public.grade_levels',
        index='index_placement_attempts_on_grade_level_id',
        index_size='97 MB',
        index_scans=0
    )

    :returns: list of Records
    """

    return self.execute(sql.UNUSED_INDEXES)
```

3.1.13 .vacuum_stats()

```
def vacuum_stats(self):
    """
    Show dead rows and whether an automatic vacuum is expected to be
    triggered.

    Record(
        schema='public',
        table='pgbench_tellers',
        last_vacuum='2014-04-29 14:45',
        last_autovacuum='2014-04-29 14:45',
        rowcount='10',
        dead_rowcount='0',
        autovacuum_threshold='52',
        expect_autovacuum=None
    )
```

```
:returns: list of Records
"""

return self.execute(sql.VACUUM_STATS)
```

3.1.14 .version()

```
def version(self):
    """
    Get the Postgres server version.

    Record(
        version='PostgreSQL 9.3.3 on x86_64-apple-darwin13.0.0'
    )

    :returns: list of Records
    """

    return self.execute(sql.VERSION)
```

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at <https://github.com/scottwoodall/pgextras/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

python-pgextras could always use more documentation, whether as part of the official python-pgextras docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/scottwoodall/pgextras/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *pgextras* for local development.

1. Fork the *pgextras* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pgextras.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pgextras
$ cd pgextras/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pgextras tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/scottwoodall/pgextras/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pgextras
```

Credits

5.1 Development Lead

- Scott Woodall <scott.woodall@gmail.com>

5.2 Contributors

None yet. Why not be the first?

History

6.1 0.1.1 (2014-05-01)

- updated documentation
- increased test coverage

6.2 0.1.0 (2014-04-25)

- First release on PyPI.

Indices and tables

- *genindex*
- *modindex*
- *search*