pg_bloat_check

pg_bloat_check is a script to provide a bloat report for PostgreSQL tables and/or indexes. It requires at least Python 2.6 (it is Python 3 compatible) and the `pgstattuple` contrib module - https://www.postgresql.org/docs/current/static/pgstattuple.html

Note that using `pgstattuple` to check for bloat can be extremely expensive on very large databases or those with many tables. The script first requires running `--create_stats_table` to create a table for storing the bloat statistics. This makes it easier for reviewing the bloat statistics or running a regular monitoring interval without having to rescan the database again. The expense of this method of bloat checking means this script is not meant to be run often to provide any sort of real-time bloat monitoring. At most, it's recommended to run this once a week or once a month during off-peak hours to search for objects that need major maintenance. Continuous runnning of this script could cause performance issues since it can cause higher priority data in shared buffers to be flushed out if run too frequently.

Output

- a simple text listing, ordered by wasted space. Good for email reports.
- a JSON blob that provides more detail and can be used by other tools that require a structured format
- a python dictionary with the same details as JSON, but can be more easily used with other python scripts

Filters

Filters are available for bloat percentage, wasted size and object size. Object size allows reporting on only those objects of a designated size or larger. The bloat percentage & wasted size reported are a combination of dead tuples and free space per object, while also accounting for their fillfactor setting. The simple output format automatically takes into account an object's fillfactor setting when calculating the wasted space & percentage values it gives. Use the JSON or python dictionary output to see the distinction between dead tuples and free space for a more accurate picture of the bloat situation. If dead tuples is high, this means autovacuum is likely not able to run frequently enough on the given table or index. If dead tuples is low but free space is high, this indicates a vacuum full or reindex is likely required to clear the bloat and return the disk space to the system. Note that free space may never be completely empty due to fillfactor settings, so both that setting and the estimated number of pages for that object are also included. By default, tables have very little reserved space (fillfactor=100) so it shouldn't affect their free space values much. Indexes by default have 10% reserved space (fillfactor=90), so that should be taken into account when looking at the raw free percent and free space values.

The `--exclude_object_file` (-e) option can make the monitoring of bloat much

more fine-grained for certain objects. The -s, -z and -p options are filters that are applied against all objects scanned. However, sometimes very large objects will have a very high amount of wasted space, but it's a very low percentage. Additionally, other relatively small objects always have a very high amount of bloat. Seeing them in the report every time isn't ideal, but you likely don't want to completely ignore these objects because if their bloat size got out of hand you would never know. To help create clearer bloat reports of things that need immediate attention, each line in the file for the -e option is a comma separated value entry of the following format:

```
objectname,bytes_wasted,percent_wasted
```

The bytes_wasted and percent_wasted are additional filters on top of -s, -z and -p that tell the exclude option to ignore the given object unless these additional filter values are exceeded as well.

Examples

First, the setup.

```
admin@mydb=# CREATE EXTENSION pgstattuple;

pg_bloat_check.py -c dbname=mydb --create_stats_table

pg_bloat_check.py -c "host=192.168.1.201 dbname=mydb user=admin" --create_stats_table --bloa
```

The first example above installs the stats tables to the default schema in your search path. You only have to run that once per database and if you run it again, it just drops the table if it exists and recreates it. If you want it in a different schema, `--bloat_schema` lets you set that, but you must then use that option every time you run the script or add that schema to your search path. The second example shows that as well as connecting to a remote system.

```
pg_bloat_check.py -c dbname=mydb -z 10485760 -p 45 -s 5242880

1. pg_catalog.pg_attribute.....................................................(83.76%) 4697
```

The above finds any tables or indexes that are at least 5MB in size with either 10MB of bloat space or are at least 45% bloated. Since it's over 45%, this table is returned in the report.

Next, an example using the -e option with a file containing the following:

```
pg_catalog.pg_attribute,5000000,85
```

```
pg_bloat_check.py -c host=localhost -z 10485760 -p 45 -s 5242880 -e filterfile

No bloat found for given parameters
```

First, the -s, -z and -p filters are applied. The exclude file is processed and the pg_attribute table is excluded if its bloat is either over 5 million bytes or

exceeds 85%. Looking above, you can see none of these are true, so the report comes back empty. This now keeps the bloat report clear until we're sure specific conditions for this table are met. A more realistic example condition would be:

```
pg_bloat_check.py -c "dbname=prod" --min_wasted_size=5368709120 -p 45 -s 1073741824

1. public.exclusions.....................................................(24.31%) 8866 MB wa
```

Here, we're looking for objects that are at least 1GB in size and have either at least 5GB of bloat or are 45% bloated. The table returned has 8GB of bloat, but it's quite a low percentage of the whole table. This is likely a good candidate to be excluded from the regular report, but since it's such a large table, we wouldn't want to ignore it if it got out of hand. So, an exclude filter line that watches for it to be at least 55% bloated or have over 20GB of bloat could work here:

```
public.exclusions,21474836480,55
```

If an object needs to be excluded entirely, no matter its bloat, either just list the object name by itself or set the additional parameters to zero.

```
pg_catalog.pg_attribute

OR

pg_catalog.pg_attribute,0,0
```

See `--help` for more information.

NOTE: The 1.x version of this script used the bloat query found in check_postgres.pl. While that query runs much faster than using `pgstattuple`, it can be inaccurate, at times missing large amounts of table and index bloat. Using `pgstattuple` provides the best method known to obtain the most accurate bloat statistics. Version 2.x of this script is not a drop-in replacement for 1.x. Please review the options and update any existing jobs accordingly.