Postgrurl

Authors:

- Luis Alfredo Leon - luis.leon.villapun@ulb.be
- Maren Hoschek - maren.hoschek@ulb.be
- Satria Bagus - satria.wicaksono@ulb.be
- Sayyor Yusupov - sayyor.yusupov@ulb.be

Repository for the final project for the Databases Architecture course of the ULB. The aim of the project is to create a Postgresql extension to handle URL data, emulating Java's .net.URL class and implementing those functionalities into postgres.

Requirements

- Postgresql >= 10.0 (Recommended 14).
- gcc >= 9.0 (Recommended).
    - Please ensure that you are using the recommended gcc version. CLang has proven to be unstable on OSX with M1 or M2 Chips when using this extension.

Assumptions

- Again, we are assuming the extension is compiled under a valid gcc, not clang or other compiler, since it will probably throw errors either in compilation time or behave incorrectly when implementing the extension if the compilation was successful (especially when evaluating regex expressions). If you are trying the project on OSX with M1 or M2 chip, the recommendation is to try this on a virtual machine on the cloud.
- We expect the input URL to have the following parts in this order:
    - Protocol`://` (optional), required if host is not defined.
    - User Info`@` (optional), in the form `user:password`.
    - Host (optional), in the form: `subdomain.domain.domain_extension`, can contain several subdomains separated by `.`, subdomain and domain parts cumullatively contain up to 256 characters and domain extension contains up to 63 characters, required if protocol is not defined.
    - `:`Port (optional), contains numeric characters.
    - `/`Path, in the form: `/path_segment/filename` (optional), can contain several path segments separated by `/`, if path exists, filename is optional.
    - `?`Query (optional) .
    - #Reference (optional).
- If the input does not follow the format above with that order, the error `ERROR:  not a valid URL` will be returned.
- A defaut port is only assigned when a protocol is present in the URL but no port (i.e. http://example.com/file -> default Port = 80 ; www.example.com/file -> default Port stays 0 because no protocol is defined).
- We are considering the file part of the URL as the `/path/to/file.txt`, without including the query part or any other part of the URL as part of the file.
- In the `URL(url context, string spec)` constructor, the assumption is that we will not process `.` and `..` in the path and instead assume it it part of the file. I.e `http://www.host.com/path/` combined with `/subpath/./..` would be `http://www.host.com/path/subpath/./..` without normalizing the URL.

- The hierarchy of comparisons in our URL implementation is as follows (important to understand this for the btree implementation):
    - Host vs host:
        - If equal, proceed to compare file section.
        - If host A is < host B (i.e `abc.com` vs `url.com`), returns -3.
        - If host A is > host B (i.e `url.com` vs `abc.com`)returns 3.
    - File vs file:
        - If hosts were equal, then we compare file section. If equal, proceed to compare raw string version of the URL.
        - If file A < file B (i.e `abc.com/about/us.txt` vs `abc.com/philosophy/values.txt`), returns -2.
        - If file A > file B (i.e `abc.com/philosophy/values.txt` vs abc.com/about/us.txt vs), returns 2.
    - Raw string vs Raw string.
        - If hosts were equal and file equal, then we perform a full raw string comparison on the urls. If exactly equal (i.e: `https:test.com:8088` vs `https:test.com:8088`), returns 0.
        - If raw string A < raw string B (i.e `https:test.com:8087/about/` vs `https:test.com:8088/about`), returns -1.
        - If raw string A > raw string B (i.e `https:test.com:8088/about` vs `https:test.com:8087/about/`), returns 1.

Note how the port part won't be analysed until the very end, if and only if both host and file part of the URL were the same. In the example above, they have the same host (`test.com`), same file (`/about`), but different raw strings (`https:test.com:8087/about/`). This returns a number closer to 0 because they share both attributes host and file and therefore belong to the same "family". This is useful for the btree index cmp function implementation, since it will sort the URLs in a proper manner.

How to install

Download the repository and run the following command on a terminal inside said folder:

```
git clone https://github.com/Action52/Postgrurl
cd Postgrurl
make clean && make && make install
```

This will compile the code inside the repository and generate the appropiate executables.

Please note that running `make installcheck` will most probably mark an error on the test. This is expected since the test file calls for `EXPLAIN` commands that will be different on each computer.

To verify that the tests are working, please run the `make installcheck` and manually verify that the output is correct in the `results/postgrurl_test.out` file that is generated after running the previous command.

The other option is to run the tests manually by copy-pasting them into the compiler.

To better understand the usage, we recommend following manually the steps below to create the extension and try some of the tests from the test file manually.

Usage

Log into your postgres installation and run the command

```
CREATE EXTENSION postgrurl;
```

This will create the extension on the database of your choice. Now you can use the postgrurl data type as desired.

Once you have implemented the extension, you will have access to the data type, constructors, functions, operators optimized for URL handling. We can then also create a btree index for this data type.

Some examples for implementation:

```
CREATE TABLE testurl(id int, purl postgrurl);
INSERT INTO testurl(id, purl) VALUES(1, 'test');
INSERT INTO testurl(id, purl) VALUES(2, 'http://test.com');
INSERT INTO testurl(id, purl) VALUES(3,
'http://test.com/file.txt');
INSERT INTO testurl(id, purl) VALUES(4,
'http://test.com/random/path/');
INSERT INTO testurl(id, purl) VALUES(5, 'http://test.com:8162');
INSERT INTO testurl(id, purl) VALUES(6, URL('http', 'test.com', 10,
'about/file.txt'));
INSERT INTO testurl(id, purl) VALUES(7, URL('http', 'test.com',
'about/file.txt'));
INSERT INTO testurl(id, purl) VALUES(8, URL('http', 'facebook.com',
'about/file.txt'));
INSERT INTO testurl(id, purl) VALUES(9, URL('http', 'facebook.com',
'profile'));
INSERT INTO testurl(id, purl) VALUES(10,
'https://facebook.com:8888/random/path');
```

If you do a `SELECT *` on the table you will get something like:

```
postgres=# SELECT * FROM testurl;
 id |                 purl
----+---------------------------------------
  1 | test
  2 | http://test.com
  3 | http://test.com/file.txt
  4 | http://test.com/random/path/
  5 | http://test.com:8162
  6 | http://test.com:10/about/file.txt
  7 | http://test.com/about/file.txt
  8 | http://facebook.com/about/file.txt
  9 | http://facebook.com/profile
 10 | https://facebook.com:8888/random/path
(10 rows)
```

To test the index usage, let's turn seq scan off for a while to enforce postgres query planner to pick the index. We have defined 3 functions to be index supported: equals(postgrurl, postgrurl), sameHost(postgrurl, postgrurl) and sameFile(postgrurl, postgrurl).

```
-- Create a table with an ID and postgrurl data type, using some of
the available constructors and the default in function
-- which will try to parse a string and change it into a postgrurl
data type.
```

```
CREATE INDEX postgrurl_idx ON testurl(purl);
SET enable_seqscan TO off;
EXPLAIN ANALYSE SELECT purl FROM testurl WHERE sameFile(purl,
'http://test.com/file.txt'::postgrurl);
SELECT purl FROM testurl WHERE sameFile(purl,
'http://test.com/file.txt'::postgrurl);
EXPLAIN ANALYSE SELECT purl FROM testurl WHERE sameHost(purl,
'http://test.com/file.txt'::postgrurl);
SELECT purl FROM testurl WHERE sameHost(purl,
'http://test.com/file.txt'::postgrurl);
EXPLAIN ANALYSE SELECT purl FROM testurl WHERE equals(purl,
'http://test.com/file.txt'::postgrurl);
SELECT purl FROM testurl WHERE equals(purl,
'http://test.com/file.txt'::postgrurl);
EXPLAIN ANALYSE SELECT * FROM testurl WHERE purl = URL('http',
'test.com', 'about/file.txt');
SELECT * FROM testurl WHERE purl = URL('http', 'test.com',
'about/file.txt');
EXPLAIN ANALYSE SELECT * FROM testurl WHERE purl >=
'http://test.com'::postgrurl AND purl <=
'https://test.com/random/path/about/'::postgrurl;
SELECT * FROM testurl WHERE purl >= 'http://test.com'::postgrurl
AND purl <= 'https://test.com/random/path/about/'::postgrurl;
EXPLAIN ANALYSE SELECT purl FROM testurl WHERE sameHost(purl,
'http://test.com/file.txt'::postgrurl) OR sameFile(purl,
'http://facebook.com/file.txt'::postgrurl);
SELECT purl FROM testurl WHERE sameHost(purl,
'http://test.com/file.txt'::postgrurl) OR sameFile(purl,
'http://facebook.com/file.txt'::postgrurl);
SET enable_seqscan TO on;
```

This will return:

```
                                                        QUERY PLAN


-------------------------------------------------------------------------------------
 Bitmap Heap Scan on testurl  (cost=16.31..20.91 rows=3 width=1024)
(actual time=0.645..0.714 rows=1 loops=1)
   Filter: samefile(purl, 'http://test.com/file.txt'::postgrurl)
   Rows Removed by Filter: 9
   Heap Blocks: exact=2
   ->  Bitmap Index Scan on postgrurl_idx  (cost=0.00..16.31
rows=10 width=0) (actual time=0.125..0.125 rows=10 loops=1)
 Planning Time: 2.248 ms
 Execution Time: 0.797 ms
(7 rows)
           purl
--------------------------
 http://test.com/file.txt
(1 row)
                                                        QUERY PLAN
```

```
-------------------------------------------------------------------------------------------
 Bitmap Heap Scan on testurl  (cost=16.31..20.91 rows=3 width=1024)
(actual time=0.174..0.200 rows=4 loops=1)
   Filter: samehost(purl, 'http://test.com/file.txt'::postgrurl)
   Rows Removed by Filter: 6
   Heap Blocks: exact=2
   ->  Bitmap Index Scan on postgrurl_idx  (cost=0.00..16.31
rows=10 width=0) (actual time=0.007..0.007 rows=10 loops=1)
 Planning Time: 0.135 ms
 Execution Time: 0.217 ms
(7 rows)
               purl
-----------------------------------
 http://test.com/file.txt
 http://test.com/random/path/
 http://test.com:10/about/file.txt
 http://test.com/about/file.txt
(4 rows)
                                                       QUERY PLAN


-------------------------------------------------------------------------------------------
 Bitmap Heap Scan on testurl  (cost=16.31..20.91 rows=3 width=1024)
(actual time=0.121..0.145 rows=1 loops=1)
   Filter: equals(purl, 'http://test.com/file.txt'::postgrurl)
   Rows Removed by Filter: 9
   Heap Blocks: exact=2
   ->  Bitmap Index Scan on postgrurl_idx  (cost=0.00..16.31
rows=10 width=0) (actual time=0.015..0.015 rows=10 loops=1)
 Planning Time: 0.116 ms
 Execution Time: 0.159 ms
(7 rows)
           purl
--------------------------
 http://test.com/file.txt
(1 row)
                                                        QUERY PLAN


-------------------------------------------------------------------------------------------
 Index Scan using postgrurl_idx on testurl  (cost=0.26..8.28 rows=1
width=1028) (actual time=0.021..0.022 rows=1 loops=1)
   Index Cond: (purl = 'http://test.com/about/file.txt'::postgrurl)
 Planning Time: 0.266 ms
 Execution Time: 0.043 ms
(4 rows)
 id |             purl
----+--------------------------------
  7 | http://test.com/about/file.txt
(1 row)
                                                        QUERY PLAN


-------------------------------------------------------------------------------------------
```

```
 Index Scan using postgrurl_idx on testurl  (cost=0.26..8.28 rows=1
width=1028) (actual time=0.175..0.179 rows=4 loops=1)
   Index Cond: ((purl >= 'http://test.com'::postgrurl) AND (purl <=
'https://test.com/random/path/about/'::postgrurl))
 Planning Time: 0.065 ms
 Execution Time: 0.289 ms
(4 rows)
 id |             purl
----+-----------------------------
  2 | http://test.com
  5 | http://test.com:8162
  3 | http://test.com/file.txt
  4 | http://test.com/random/path/
(4 rows)
                                                           QUERY
PLAN

--------------------------------------------------------------------------------
 Bitmap Heap Scan on testurl  (cost=16.31..23.41 rows=6 width=1024)
(actual time=0.108..0.158 rows=4 loops=1)
   Filter: (samehost(purl, 'http://test.com/file.txt'::postgrurl)
OR samefile(purl, 'http://facebook.com/file.txt'::postgrurl))
   Rows Removed by Filter: 6
   Heap Blocks: exact=2
   ->  Bitmap Index Scan on postgrurl_idx  (cost=0.00..16.31
rows=10 width=0) (actual time=0.007..0.007 rows=10 loops=1)
 Planning Time: 0.069 ms
 Execution Time: 0.173 ms
(7 rows)
               purl
-----------------------------------
 http://test.com/file.txt
 http://test.com/random/path/
 http://test.com:10/about/file.txt
 http://test.com/about/file.txt
(4 rows)
```

We can also use the `COPY` command to insert in bulk, for example:

```
COPY testurl TO '/tmp/testurl.in' WITH BINARY;
DELETE FROM testurl;
COPY testurl FROM '/tmp/testurl.in' WITH BINARY;
SELECT * FROM testurl;

-- SELECT * after repopulating the table with the COPY command.
 id |                 purl
----+------------------------------------
  1 | test
  2 | http://test.com
  3 | http://test.com/file.txt
  4 | http://test.com/random/path/
  5 | http://test.com:8162
  6 | http://test.com:10/about/file.txt
  7 | http://test.com/about/file.txt
  8 | http://facebook.com/about/file.txt
  9 | http://facebook.com/profile
 10 | https://facebook.com:8888/random/path
(10 rows)
```

We also defined helper functions which return a particular part of the inputted URL. If the part is not in the URL, error will be returned.

```
SELECT getHost('http://test.com:8239/hola?query=6'::postgrurl);
SELECT getPort('http://test.com:8239/hola?query=6'::postgrurl);
SELECT getDefaultPort('sftp://test.com/hola?query=6'::postgrurl);
SELECT getProtocol('http://test.com:8239/hola?query=6'::postgrurl);
SELECT getQuery('http://test.com:8239/hola?query=6'::postgrurl);
SELECT getRef('http://test.com:8239/hola#somewhere'::postgrurl);
```

Finally delete the test data if necessary.

```
DROP INDEX IF EXISTS postgrurl_idx;
DROP TABLE testurl;
DROP EXTENSION postgrurl;
```

You can find more usage and testing examples on the test file included in the repository.