Ass1

this assignment want us add a new data type to postgreSQL

it is quite interesting and can extremely excercise your ability to code engineer project

totally we have several procedure to finish this assignment:

- cd to /home/your_userName/postgreSQL-xx/src/tutorial
    - copy complex.c and complex.source to create pname.c and pname.source
    - and then modify Makefile used to compile pname.c and pname.source into pname.sql as a script which can be called in a database
- then create a new database, and create a tables to test the new datatype you define

some instructions

https://cgi.cse.unsw.edu.au/~cs9315/20T1/postgresql/documentation/xtypes.html

1. Initial preparation

first, copy complex.c and complex.source to create pname.c and pname.source, these two files is what we manage to complete in this assignment.

and then, modifying your Makefile, like code shown below:

```
#-------------------------------------------------------------------------
#
# Makefile--
#    Makefile for tutorial
#
# By default, this builds against an existing PostgreSQL installation
# (the one identified by whichever pg_config is first in your path).
# Within a configured source tree, you can say "make NO_PGXS=1 all"
# to build using the surrounding source tree.
#
# IDENTIFICATION
#    src/tutorial/Makefile
#
#-------------------------------------------------------------------------

# MODULES = complex funcs
MODULES = complex funcs pname
# DATA_built = advanced.sql basics.sql complex.sql funcs.sql syscat.sql
DATA_built = advanced.sql basics.sql complex.sql funcs.sql syscat.sql pname.sql

ifdef NO_PGXS
subdir = src/tutorial
```

```
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/src/makefiles/pgxs.mk
else
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
endif

%.sql: %.source
  rm -f $@; \
  C=`pwd`; \
  sed -e "s:_OBJWD_:$$C:g" < $< > $@
```

Note:

- PostgreSQL installation provides a **build infrastructure for extensions**, called **PGXS**, so that simple extension modules can be built simply against an already-installed server.
- pg_config: retrieve information about the installed version of PostgreSQL

1.1 coding pname.c

in pname.c, we need to define some I/O functions, which receive the input of users and parse it into ouput

```
/*****************************************************************************
 * some check functions supported by regex
 *****************************************************************************/
PG_MODULE_MAGIC;

typedef struct
{
  int32 length;
  char name[FLEXIBLE_ARRAY_MEMBER];
} PersonName;

// function used to compile the regrex and execute it
// only beging visible for file pname.c
static int regexMatch(char * str, char * regexPattern)

// function to check family name by regex
static int check_family_name(char * s)

// function to check given name by regex
static int check_given_name(char * s)

// function to check the name is valid
static int valid_name(char *str)

// function to delete all the ' ' in the left of a string
// eg "  a b " -> "a b "
static char *lltrim(char *str)

// function to compare the person name
static int pname_cmp(PersonName *p1,PersonName *p2)
```

and then we can define some I/O functions and some judge functinos:

```
/*****************************************************************************
 * Input/Output functions
 *****************************************************************************/

// function for input
// in PostgreSQL, the C declaration is always Datum funcname(PG_FUNCTION_ARGS)
// User-defined functions can be written in C, such function will be compiled into dynamically loadable object
// and can be loaded by the server on demand.
// we can use marco PG_FUNCTION_INFO_V1 to process it.
PG_FUNCTION_INFO_V1(pname_in);

Datum
pname_in(PG_FUNCTION_ARGS)
{
  char *str = PG_GETARG_CSTRING(0);
  int len;
  PersonName *result;
  char *family = NULL;
  char *given = NULL;
  char *temp1=(char *)palloc(strlen(str)+1);
  char *temp2=(char *)palloc(strlen(str)+1);
  // copy the name
  char *temp=(char *)palloc(strlen(str)+1);
  snprintf(temp, strlen(str)+1, "%s", str);

  // check the name
  if(!valid_name(temp)){
    ereport(ERROR,
    (errcode(ERRCODE_INVALID_TEXT_REPRESENTATION),
     errmsg("invalid input syntax for type %s: \"%s\"",
        "PersonName", str)));
  }
  snprintf(temp1, strlen(str)+1, "%s", str);
  snprintf(temp2, strlen(str)+1, "%s", str);
  // the name is valid, then split them and format them
  family=strtok_r(temp1,",",&given);
  given=lltrim(given);
  // regenerate the name and remove the space if there exists in the begining of given name
  sprintf(temp2,"%s,%s",family,given);
  len=strlen(temp2)+1;
  result = (PersonName *) palloc(VARHDRSZ+len);
  SET_VARSIZE(result,VARHDRSZ+len);
  snprintf(result->name,len,"%s",temp2);
  PG_RETURN_POINTER(result);
}

// function for output
PG_FUNCTION_INFO_V1(pname_out);

// function to show the family name
PG_FUNCTION_INFO_V1(family);

// function to show the given name
PG_FUNCTION_INFO_V1(given);

// function to show the name
PG_FUNCTION_INFO_V1(show);

// function to judge one name larger than the other
```

```
PG_FUNCTION_INFO_V1(pname_bigger);

// function to judge one name larger and equal than the other
PG_FUNCTION_INFO_V1(pname_bigger_equal);

// function to judge one name less than the other
PG_FUNCTION_INFO_V1(pname_less);

// function to judge one name less and equal than the other
PG_FUNCTION_INFO_V1(pname_less_equal);

// function to judge one name equal to the other
PG_FUNCTION_INFO_V1(pname_equal);

// function to judge one name not equal to the other
PG_FUNCTION_INFO_V1(pname_not_equal);

// function for btree
PG_FUNCTION_INFO_V1(pname_abs_cmp);

// function for hash
PG_FUNCTION_INFO_V1(pname_own_hash);
```

1.2 coding pname.source

Once we have write I/O functions and compile then into a shared library, we can define the “pname” type in SQL.

this file is a sql-like script, function is replace the “_OBJWD_/pname” with the actually pname file path

```
CREATE FUNCTION pname_in(cstring)
   RETURNS PersonName
   AS '_OBJWD_/pname'
   LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION pname_out(PersonName)
   RETURNS cstring
   AS '_OBJWD_/pname'
   LANGUAGE C IMMUTABLE STRICT;


CREATE TYPE PersonName (
   input = pname_in,
   output = pname_out
);

CREATE FUNCTION show(PersonName)
   RETURNS text
   AS '_OBJWD_/pname'
   LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION family(PersonName)
   RETURNS text
   AS '_OBJWD_/pname'
   LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION given(PersonName)
   RETURNS text
   AS '_OBJWD_/pname'
   LANGUAGE C IMMUTABLE STRICT;
```

```
CREATE FUNCTION pname_bigger(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_bigger_equal(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_less(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_less_equal(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_equal(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_not_equal(PersonName, PersonName) RETURNS bool
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;


CREATE OPERATOR < (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_less,
   commutator = > , negator = >= ,
   restrict = scalarltsel, join = scalarltjoinsel
);
CREATE OPERATOR <= (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_less_equal,
   commutator = >= , negator = > ,
   restrict = scalarlesel, join = scalarlejoinsel
);
CREATE OPERATOR = (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_equal,
   commutator = = ,
   -- leave out negator since we didn't create <> operator
   negator = <> ,
   restrict = eqsel, join = eqjoinsel
);
CREATE OPERATOR >= (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_bigger_equal,
   commutator = <= , negator = < ,
   restrict = scalargesel, join = scalargejoinsel
);
CREATE OPERATOR > (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_bigger,
   commutator = < , negator = <= ,
   restrict = scalargtsel, join = scalargtjoinsel
);
CREATE OPERATOR <> (
   leftarg = PersonName, rightarg = PersonName, procedure = pname_not_equal,
   commutator = <> ,
   -- leave out negator since we didn't create <> operator
   negator = = ,
   restrict = neqsel, join = neqjoinsel
);

CREATE FUNCTION pname_abs_cmp(PersonName, PersonName) RETURNS int4
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;
CREATE FUNCTION pname_own_hash(PersonName) RETURNS int4
   AS '_OBJWD_/pname' LANGUAGE C IMMUTABLE STRICT;

CREATE OPERATOR CLASS PersonName_btree_ops
    DEFAULT FOR TYPE PersonName USING btree AS
        OPERATOR        1       < ,
        OPERATOR        2       <= ,
        OPERATOR        3       = ,
        OPERATOR        4       >= ,
        OPERATOR        5       > ,
        FUNCTION        1       pname_abs_cmp(PersonName, PersonName);
```

```
CREATE OPERATOR CLASS PersonName_hash_ops
    DEFAULT FOR TYPE PersonName USING hash AS
        OPERATOR        1       =,
        FUNCTION        1       pname_own_hash(PersonName);
```

2. test our work

After completing file pname.c and pname.source, we can compile then into file pname.sql by Makefile.

We can compile it by command **“make NO_PGXS=1 all”**

and then we get script “pname.sql”, we can insert it in our database.

the test operation is shown below:

```
create table Students (
  zid integer primary key,
  name PersonName not null,
  degree text,
  -- etc. etc.
);
insert into Students(zid,name,degree) values
(9300035,'Shepherd, John Andrew', 'BSc(Computer Science)'),
(5012345,'Smith, Stephen', 'BE(Hons)(Software Engineering)');

create index on Students using hash (name);

select a.zid, a.name, b.zid
from Students a join Students b on (a.name = b.name);

select family(name), given(name), show(name)
from Students;

select name,count(*)
from Students
group by name;
```

the result is shown below:

```
myNewDB=#
insert into Students(zid, name, degree) values
(9300035,'Shepherd, John Andrew', 'BSc(Computer Science)'),
(5012345,'Smith, Stephen', 'BE(Hons)(Software Engineering)');
INSERT 0 2
myNewDB=# select * from Students;
   zid   |         name         |             degree
---------+----------------------+--------------------------------
 9300035 | Shepherd,John Andrew | BSc(Computer Science)
 5012345 | Smith,Stephen        | BE(Hons)(Software Engineering)
(2 rows)

myNewDB=# create index on Students using hash(name);
CREATE INDEX
myNewDB=# select a.zid, a.name, b.zid
myNewDB-# from Students a join Students b on (a.name = b.name);
   zid   |         name         |   zid
---------+----------------------+---------
 9300035 | Shepherd,John Andrew | 9300035
 5012345 | Smith,Stephen        | 5012345
(2 rows)

myNewDB=# select family(name), given(name), show(name)
myNewDB-# from Students;
  family  |    given    |     show
----------+-------------+---------------
 Shepherd | John Andrew | John Shepherd
 Smith    | Stephen     | Stephen Smith
(2 rows)

myNewDB=# select name,count(*)
myNewDB-# from Students
myNewDB-# group by name;
         name         | count
----------------------+-------
 Smith,Stephen        |     1
 Shepherd,John Andrew |     1
(2 rows)

myNewDB=# explain analyze select * from Students where name='Smith,John';
                                              QUERY PLAN

-----------------------------------------------------------------------------------------
---------------
 Seq Scan on students  (cost=0.00..1.02 rows=1 width=68) (actual time=0.013..0.014
ows=0 loops=1)
   Filter: (name = 'Smith,John'::personname)
   Rows Removed by Filter: 2
 Planning Time: 0.080 ms
 Execution Time: 0.031 ms
(5 rows)

myNewDB=#
```