PostgreSQL Extensions - A Deeper Dive

Jignesh Shah

Amazon RDS for PostgreSQL

Postgres Conference 2019

PostgreSQL Core

Robust feature sets

Multi-Version Concurrency Control (MVCC), point in time recovery, granular access controls, tablespaces, asynchronous replication, nested transactions, online/hot backups, a refined query planner/optimizer, and write ahead logging

Supports international character sets, multi-byte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting

Reliable

High fault tolerance, ACID compliance, and full support for foreign keys, joins, views, triggers, and stored procedures

Standards-compliant

Includes most SQL:2008 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. Supports storage of binary large objects, including pictures, sounds, or video

What are PostgreSQL Extensions?

Extend beyond Core PostgreSQL functionality

Customize existing functionality

Loadable set of functions

Adding new features to PostgreSQL core

- New datatype, new operator, new index operator

Common PostgreSQL Extensions

Extensions	Description
pg_stat_statements	Statistics about executed queries
postgis	Spatial Datatypes support
postgres_fdw	Foreign Data Wrapper for PostgreSQL
plv8	Procedural Language in Java Script using v8
uuid-ossp	Generate universally unique identifiers (UUIDs)

Supported PostgreSQL Extensions

```
testdb=# SELECT * FROM pg_available_extensions;
             name             | default_version | installed_version |
comment
------------------------------+-----------------+-------------------+------------
--------------------------------------------------------------------------------
------------------------
 pg_stat_statements           | 1.7             |                   | track
execution statistics of all SQL statements executed
 plpgsql                      | 1.0             | 1.0               | PL/pgSQL
procedural language
 uuid-ossp                    | 1.1             |                   | generate
universally unique identifiers (UUIDs)
 postgres_fdw                 | 1.0             |                   | foreign-
data wrapper for remote PostgreSQL servers
```

Using PostgreSQL Extension

```
testdb=# CREATE EXTENSION pg_stat_statements;
…
testdb=> select total_time, query from pg_stat_statements ORDER BY total_time
DESC LIMIT 3;
     total_time     |                                           query
--------------------+----------------------------------------------------------
---------------------------------
12021.36151499999 | UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE
bid = $2
4912.441237999993 | SELECT count(*) FROM information_schema.tables WHERE
table_catalog = $1 and table_name = $2
 2569.502414999663 | UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE
tid = $2
(3 rows)
```

Listing Used PostgreSQL Extensions

```
testdb=> SELECT * FROM pg_extension;
  oid  |              extname              | extowner | extnamespace | extrelocatable
| extversion  |          extconfig          |
extcondition

-------+-------------------------------+----------+--------------+----------------
+-------------+-----------------------------+-------------------------------------
-------------------------------------

 14299 | plpgsql                       |       10 |           11 | f
| 1.0         |                             |
73505 | pg_stat_statements            |       10 |         2200 | t                 |
1.7         |                             |
```

Upgrade and Removing PostgreSQL Extensions

Upgrade extension

- `ALTER EXTENSION name UPDATE;`
- `ALTER EXTENSION name UPDATE TO ‘2.5.2’;`

Remove extension

Unsafe

- `DROP EXTENSION name CASCADE;`

Safe

- `DROP EXTENSION name;`

PostgreSQL Loadable Modules

Similar to extensions but only need to be loaded

C Functions are automatically loaded

LOAD is needed that modifies server behavior with hooks

- Non-superusers can only apply LOAD to library files located in `$libdir/plugins/`

Example: `auto_explain`

PostgreSQL Plugins

Logical Decoding Output Plugins

- Loaded dynamically shared library

Examples:

- `test_decoder`
- `decoder_raw`
- `wal2json`

Developing PostgreSQL Extension

Requirements of an Extension

Minimum extension requirements

- Requires a control file
- Requires a minimum of a script file of SQL Commands with function definitions
- If required loadable modules

SQL Only Demo Extension

- `demo--0.9.sql`

```
CREATE OR REPLACE FUNCTION demo_version() RETURNS TEXT
    AS $$ SELECT  'Demo Extension 0.9' $$
LANGUAGE SQL;
```

- `demo.control`

```
# demo extension
comment = 'Demo Extension for Postgres Conference'
default_version = ‘0.9'
relocatable = true
# requires = ‘pg_stat_statements’
# directory = 'extension'
```

Copy these files to `${PGSQL}/share/extension`

Execution of SQL Only Demo Extension

```
psql (12beta3)
Type "help" for help.
testdb=# CREATE EXTENSION demo;
CREATE EXTENSION

testdb=# select demo_version();
    demo_version
--------------------
 Demo Extension 0.9
(1 row)
```

Demo Extension using C Language

- demo--1.0.sql

```
CREATE OR REPLACE FUNCTION demo_version() RETURNS TEXT  AS
'MODULE_PATHNAME','demo'
LANGUAGE C STRICT;
```

- demo.control

```
# demo extension
comment = 'Demo Extension for Postgres Conference'
default_version = '1.0'
module_pathname = '$libdir/demo'
relocatable = true
```

Demo Extension using C Language

- demo.c

```
#include "postgres.h"
#include "fmgr.h"

PG_MODULE_MAGIC;
PG_FUNCTION_INFO_V1(demo);

#define DEMO_VERSION_STR  "Demo Extension 1.0"
Datum
demo(PG_FUNCTION_ARGS)
{
  PG_RETURN_TEXT_P(cstring_to_text(DEMO_VERSION_STR));
}
```

Demo Extension Using C Language

- Makefile (USE_PGXS=1)

```
MODULES = demo
EXTENSION = demo
DATA = demo--0.9.sql demo--1.0.sql
PGFILEDESC = "demo extensions - examples of using extensions”
# Following is stub common in many extensions based on C
PG_CPPFLAGS = -DREFINT_VERBOSE
ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/demo
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

Execution of Demo Extension Using C Language

```
testdb=# DROP EXTENSION demo;
DROP EXTENSION

testdb=# CREATE EXTENSION demo;
CREATE EXTENSION

testdb=# SELECT demo_version();
    demo_version
--------------------
 Demo Extension 1.0
(1 row)
```

Extension to support new Foreign Data Wrapper

```
CREATE FUNCTION demofdw_handler() RETURNS fdw_handler
AS 'MODULE_PATHNAME’ LANGUAGE C STRICT;

CREATE FUNCTION demofdw_validator(text[], oid)
RETURNS void AS 'MODULE_PATHNAME’ LANGUAGE C STRICT;

CREATE FOREIGN DATA WRAPPER demofdw
  HANDLER demofdw_handler VALIDATOR demofdw_validator;
```

Extension to support new Procedural Language

```
CREATE OR REPLACE FUNCTION demolang_call_handler() RETURNS language_handler  AS
'$libdir/demolang.so' LANGUAGE C STRICT;

CREATE OR REPLACE FUNCTION demolang_inline_handler(internal) RETURNS
language_handler  AS '$libdir/demolang.so' LANGUAGE C STRICT;

CREATE OR REPLACE FUNCTION demolang_validator(oid) RETURNS language_handler  AS
'$libdir/demolang.so' LANGUAGE C STRICT;
```

```
CREATE PROCEDURAL LANGUAGE demolang HANDLER
demolang_call_handler INLINE demolang_inline_handler VALIDATOR
demolang_validator ;
```

```
COMMENT ON PROCEDURAL LANGUAGE demolang IS 'Demo Language for PostgreSQL
Conference';
```

Upgrading PostgreSQL Extensions

Demo Extension – 1.1

- demo.c

```
#include "postgres.h"
#include "fmgr.h"
PG_MODULE_MAGIC;
PG_FUNCTION_INFO_V1(demo);
PG_FUNCTION_INFO_V1(demo11);
#define DEMO_VERSION_STR  "Demo Extension 1.0"
#define DEMO11_VERSION_STR  "Demo Extension 1.1"
Datum
demo(PG_FUNCTION_ARGS)
{
  PG_RETURN_TEXT_P(cstring_to_text(DEMO_VERSION_STR));
}
Datum
demo11(PG_FUNCTION_ARGS)
{
  PG_RETURN_TEXT_P(cstring_to_text(DEMO11_VERSION_STR));
}
```

Demo Extension 1.1

- `demo--1.0–1.1.sql`

```
CREATE OR REPLACE FUNCTION demo_version() RETURNS TEXT  AS
'MODULE_PATHNAME','demo11’ LANGUAGE C STRICT;
```

- `Makefile` (assumes to be in contrib/$name)

```
MODULES = demo
EXTENSION = demo
DATA = demo--0.9.sql demo--0.9--1.0.sql \
       demo--1.0.sql demo--1.0--1.1.sql
PGFILEDESC = "demo extensions - examples of using extensions"
```

If your library module version changes, you need both the original library and the new library in order for upgrade to work.

Demo Extension 1.1

```
testdb=# select * from pg_extension;
  oid  | extname | extowner | extnamespace | extrelocatable | extversion | extconfig | extcondition
-------+---------+----------+--------------+----------------+------------+-----------+--------------
40963 | demo    |       10 |         2200 | t              | 1.0        |           |

testdb=# ALTER EXTENSION demo UPDATE TO '1.1';
ALTER EXTENSION
testdb=# select demo_version();
    demo_version
--------------------
 Demo Extension 1.1

testdb=# select * from pg_extension;
  oid  | extname | extowner | extnamespace | extrelocatable | extversion | extconfig | extcondition
-------+---------+----------+--------------+----------------+------------+-----------+--------------
40963 | demo    |       10 |         2200 | t              | 1.1        |           |
```

Extension Upgrade Paths

```
testdb=# SELECT * FROM pg_extension_update_paths('demo');
 source | target |      path
--------+--------+-----------------
 0.9    | 1.0    | 0.9--1.0
 0.9    | 1.1    | 0.9--1.0--1.1
 1.0    | 0.9    |
 1.0    | 1.1    | 1.0--1.1
 1.1    | 0.9    |
 1.1    | 1.0    |
```

GUCs for PostgreSQL Extensions

Demo GUCs in Extension

- demo.c

```
#include "funcapi.h"
#include "utils/builtins.h"
#include "utils/guc.h"
…
void  _PG_init(void)
{
        DefineCustomIntVariable("demo.param1",
               "Demo Parameter to show GUC in extensions.",
               NULL, &demo_param1, 0, 0, INT_MAX, PGC_USERSET,
               0, NULL, NULL, NULL);
}
```

Execution of Demo Extension

Postgresql.conf: `shared_preload_library = ‘demo’`

Restart PostgreSQL server

```
testdb=# show demo.param1;
 demo.param1
-------------
 0
(1 row)
```

Server Programming Interface

Server Programming Interface

- Interface Functions
  - `SPI_connect, SPI_exec, …`
- Interface Support Functions
  - `SPI_fname, SPI_getvalue, …`
- Memory Management
  - `SPI_palloc, SPI_copytuple, …`
- Transaction Management
  - `SPI_commit, SPI_rollback, SPI_start_transaction`

Documentation: https://www.postgresql.org/docs/12/spi.html

SPI Demo Extension

- demo.c

```
#include "executor/spi.h”
…
Datum demo11(PG_FUNCTION_ARGS)
{ char *sql = "SELECT version();";
  int ret;
  SPI_connect();
  ret = SPI_execute(sql, true, 1);
  SPI_processed;
  char buf[256];
  if (ret > 0 && SPI_tuptable != NULL)
  {     snprintf(buf,256, "%s with %s", DEMO11_VERSION_STR,
                SPI_getvalue(SPI_tuptable->vals[0], SPI_tuptable->tupdesc, 1));
  }
  else snprintf(buf,256, "%s", DEMO11_VERSION_STR);
  SPI_finish();
  PG_RETURN_TEXT_P(cstring_to_text(buf));
}
```

SPI Demo Extension

```
psql (12beta3)
Type "help" for help.

testdb=# SELECT demo_version();
                                                           demo_version
-------------------------------------------------------------------------
-----------------------------------------------------------
 Demo Extension 1.1 with PostgreSQL 12beta3 on x86_64-pc-linux-gnu,
compiled by gcc (GCC) 7.3.1 20180303 (Red Hat 7.3.1-5), 64-bit
(1 row)
```

Advanced Reading

Shared Memory Access

PostgreSQL Hooks

Trigger Data

Summary

Extensions help extend core PostgreSQL functionality and are easy to use and develop

Thank you!