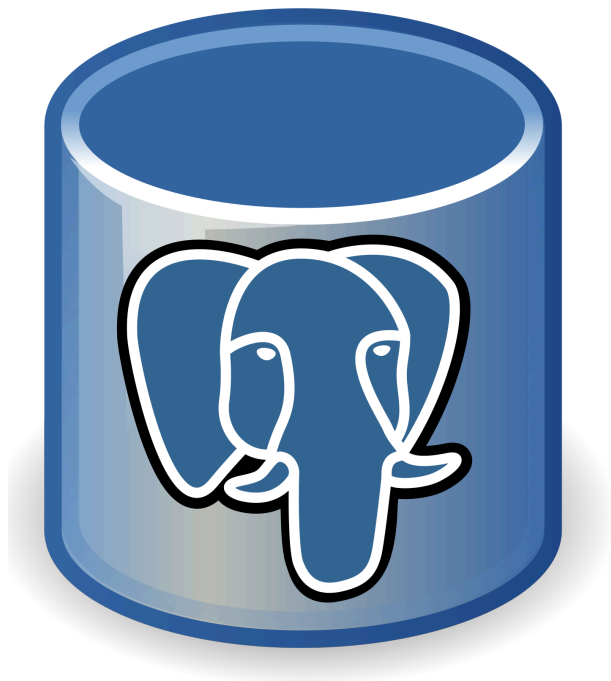
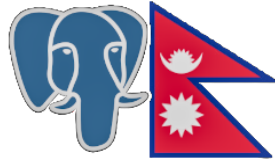


# Extensions in PostgreSQL, and how to develop one



- *By, Swastik Gurung*

Hands On exercises for the workshop at  
**Third PostgreSQL Conference, Nepal**

## **Table of Contents**

<b>Prerequisite.....</b>	<b>3</b>
<b>Layout of an Extension.....</b>	<b>4</b>
Required Files.....	4
my_extension.control.....	4
my_extension--1.0.sql.....	4
Optional Files.....	5
my_extension--1.0--1.1.sql.....	5
my_extension.c.....	5
Makefile.....	5
README.md.....	5
<b>Examples.....</b>	<b>6</b>
Calculate Factorial of a given number.....	6
Reversing a String.....	9
Log DML Events.....	12
<b>Docker Commands.....</b>	<b>15</b>
<b>References.....</b>	<b>16</b>

# Prerequisite

Make sure you have a Database Client Tool, such as **psql** or **PgAdmin**.

From a terminal, execute following to connect postgres:

```
psql -h <host> -p <port> -U <user> <database>
```

View the directory where your extensions are installed:

```
SELECT * FROM pg_config WHERE name = 'SHAREDIR';
```

Or, from a terminal:

```
pg_config --sharedir
```

List available extensions:

```
SELECT * FROM pg_available_extensions ORDER BY name;
```

List Installed Extensions:

```
SELECT * FROM pg_extension;  
\dx      -- list extensions  
\dx+     -- list extensions and its objects
```

To create an extension in postgres:

```
CREATE EXTENSION pg_stat_statements;
```

To update an extension:

```
ALTER EXTENSION pg_stat_statements UPDATE;
```

To drop an extension:

```
DROP EXTENSION pg_stat_statements;
```

# Layout of an Extension

```
my_extension/  
|  
+-- my_extension.control  
|  
+-- my_extension--1.0.sql  
|  
+-- my_extension--1.0--1.1.sql  
|  
+-- my_extension.c  
|  
+-- Makefile  
|  
+-- README.md
```

## Required Files

### my\_extension.control

Control file of the extension, which describes it. Content of this file are:

- **directory** = Directory containing SQL script files, SHAREDIR used if not specified
- **comment** = Description of the extension
- **default\_version** = Current Version of the Extension
- **module\_pathname** = Library Directory (*only required if using C*)
- **relocatable** = true for allowing extension to move to a different schema, false to disallow
- **encoding** = The character set encoding used by the script file(s)
- **requires** = List of dependent extensions
- **superuser** = Defaults to true, allowing only superusers to install the extension
- **trusted** = Defaults to false, disallowing non-superusers to install the extension. This parameter is irrelevant if superuser is false

### my\_extension--1.0.sql

SQL script to create the extension, which contains SQL functions, types, operators, etc. This file gets executed with **CREATE EXTENSION ...**

An extension's SQL script files can contain any SQL commands, except for BEGIN, COMMIT, VACUUM, etc. as script files are implicitly executed within a transaction block.

All the objects declared inside the file are managed together.

Follows the format of *name--version.sql*

## Optional Files

### my\_extension--1.0--1.1.sql

This file contains upgraded SQL scripts between versions, which can be updated using **ALTER EXTENSION ... UPDATE**

**ALTER EXTENSION** is able to execute sequences of update script files to achieve a requested update.

This is an optional file.

Follows the format of *name--from\_version--to\_version.sql*

### my\_extension.c

Added C code for Advanced Functions using low-level programming.

This can be used when an extension contains performance-critical logic or needs hooks into PostgreSQL internals.

## Makefile

Controls the build process using PostgreSQL's PGXS infrastructure. In the makefile, we need to set some variables and include the global PGXS makefile.

Required when compiling with C. Content of this file are:

- **EXTENSION** = Name of the extension, must provide an *extension.control* file
- **DATA** = SQL file defining the extension
- **PG\_CONFIG** = Locates the PostgreSQL configuration, build environment
- **PGXS** = Ensures that PostgreSQL's build infrastructure is used
- **include \$(PGXS)** = Includes PostgreSQL's make rules for extensions

Need to run **make** to compile, and then **make install** to install the module.

## README.md

Documentation for users on how to install, use, and upgrade the extension.

# Examples

## Calculate Factorial of a given number

### Layout of the Extension

```
factorial_extension/  
|  
+-- factorial_extension.control  
|  
+-- factorial_extension--1.0.sql
```

Create the control file for the extension:

```
nano factorial_extension.control
```

**NOTE:** *nano* or *vim* used for UNIX-like operating systems.

Content of the control file:

```
# factorial_extension.control  
comment = 'Extension to calculate factorial'  
default_version = '1.0'  
relocatable = false
```

Create the SQL file of the extension:

```
nano factorial_extension--1.0.sql
```

Content of the SQL file:

```
-- factorial_extension--1.0.sql

-- Create a new schema for the extension
CREATE SCHEMA factorial_extension;

-- Create the factorial function
CREATE OR REPLACE FUNCTION factorial_extension.factorial(n INTEGER)
RETURNS BIGINT AS $$
DECLARE
    result BIGINT := 1;
BEGIN
    IF n < 0 THEN
        RAISE EXCEPTION 'Factorial is not defined for negative numbers';
    ELSIF n > 1 THEN
        FOR i IN 2..n LOOP
            result := result * i;
        END LOOP;
    END IF;
    RETURN result;
END;
$$ LANGUAGE plpgsql;

-- Grant execute permission to public (change to appropriate roles if needed)
GRANT EXECUTE ON FUNCTION factorial_extension.factorial(INTEGER) TO PUBLIC;
```

Copy these files, into the extension directory:

```
sudo cp factorial_extension.control /usr/share/postgresql/17/extension
sudo cp factorial_extension--1.0.sql /usr/share/postgresql/17/extension
```

**NOTE:** The extension directory used above */usr/share/postgresql/17/extension* is for debian/ubuntu based installation, adjust it accordingly.

Restart PostgreSQL Server:

```
sudo service postgresql restart
```

Connect to the database:

```
psql -h <host> -p <port> -U <user> <database>
```

Verify if the extension exists:

```
SELECT * FROM pg_available_extensions WHERE name = 'factorial_extension';
```

Install the extension:

```
CREATE EXTENSION factorial_extension;
```

Test the extension:

```
SELECT factorial_extension.factorial(5);
```



## Reversing a String

**NOTE:** Available in PostgreSQL, using `reverse()` function.

### Layout of the Extension

```
reverse_string/  
|  
+-- reverse_string.control  
|  
+-- reverse_string--1.0.sql  
|  
+-- Makefile
```

Create a separate folder to group files together:

```
mkdir reverse_string  
cd reverse_string/
```

Create the control file for the extension:

```
nano reverse_string.control
```

**NOTE:** *nano* or *vim* used for UNIX-like operating systems.

Content of the control file:

```
# reverse_string.control  
comment = 'Extension to reverse string'  
default_version = '1.0'  
relocatable = true
```

Create the SQL file of the extension:

```
nano reverse_string--1.0.sql
```

Content of the SQL file:

```
-- reverse_string--1.0.sql

-- Create the reverse string function
-- IMMUTABLE : always returns same output for same input
-- STRICT : avoids executing when input is NULL
CREATE FUNCTION reverse_string(para text)
RETURNS text
LANGUAGE plpgsql IMMUTABLE STRICT
AS $$
DECLARE
    ret text;
BEGIN
    ret := reverse(para); -- Using PostgreSQL's built-in reverse function
    RETURN ret;
END;
$$;
```

Create the Makefile:

```
nano Makefile
```

Content of the Makefile:

```
EXTENSION = reverse_string
DATA = reverse_string--1.0.sql
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

Compile and Install the Extension:

```
cd reverse_string/
make
make install
```

This process installs the SQL and control files into PostgreSQL's extension directory.

Connect to the database:

```
psql -h <host> -p <port> -U <user> <database>
```

Verify if the extension exists:

```
SELECT * FROM pg_available_extensions WHERE name = 'reverse_string';
```

Install the extension:

```
CREATE EXTENSION reverse_string;
```

Test the extension:

```
SELECT reverse_string('hello world');  
SELECT reverse_string('Welcome to the Third PostgreSQL Conference, Nepal  
2025');
```

# Log DML Events

## Layout of the Extension

```
audit_log/  
|  
+-- audit_log.control  
|  
+-- audit_log--1.0.sql
```

Create the control file for the extension:

```
nano audit_log.control
```

**NOTE:** *nano* or *vim* used for UNIX-like operating systems.

Content of the control file:

```
# audit_log.control  
comment = 'Extension to Log DML Events'  
default_version = '1.0'  
relocatable = false
```

Create the SQL file of the extension:

```
nano audit_log--1.0.sql
```

Content of the SQL file:

```
-- audit_log--1.0.sql

-- Create a new schema for the extension
CREATE SCHEMA logging;

-- Create a log table to store events
CREATE TABLE logging.audit_logs (
    id serial,
    log_time timestamp DEFAULT now(),
    schemaname text,
    tablename text,
    operation text,
    username text DEFAULT current_user,
    new_val json,
    old_val json
);

-- Create the audit log function
-- RETURNS trigger used to hook the function to a trigger
CREATE FUNCTION logging.log_to_table() RETURNS trigger AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO logging.audit_logs(schemaname, tablename, operation, new_val) VALUES
(TG_TABLE_SCHEMA, TG_RELNAME, TG_OP, row_to_json(NEW));
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO logging.audit_logs(schemaname, tablename, operation, new_val, old_val) VALUES
(TG_TABLE_SCHEMA, TG_RELNAME, TG_OP, row_to_json(NEW), row_to_json(OLD));
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO logging.audit_logs(schemaname, tablename, operation, old_val) VALUES
(TG_TABLE_SCHEMA, TG_RELNAME, TG_OP, row_to_json(OLD));
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE 'plpgsql' SECURITY DEFINER;
-- SECURITY DEFINER used to succeed the function, even with normal user

-- Create table to test out the function
CREATE TABLE public.employee (
    id serial,
    employee_name text,
    employee_dob date,
    address text,
    department text,
    salary numeric );

-- Create the trigger
CREATE TRIGGER audit_log_employee BEFORE INSERT OR UPDATE OR DELETE ON public.employee FOR
EACH ROW EXECUTE PROCEDURE logging.log_to_table();
```

Copy these files, into the extension directory:

```
sudo cp audit_log.control /usr/share/postgresql/17/extension
sudo cp audit_log--1.0.sql /usr/share/postgresql/17/extension
```

**NOTE:** The extension directory used above */usr/share/postgresql/17/extension* is for debian/ubuntu based installation, adjust it accordingly.

Restart PostgreSQL Server:

```
sudo service postgresql restart
```

Connect to the database:

```
psql -h <host> -p <port> -U <user> <database>
```

Verify if the extension exists:

```
SELECT * FROM pg_available_extensions WHERE name = 'audit_log';
```

Install the extension:

```
CREATE EXTENSION audit_log;
```

Test the extension:

```
SELECT * FROM employee;

INSERT INTO employee VALUES (1, 'Michael', '1956-10-01', 'California',
'Development', 25000.00);

SELECT * FROM logging.audit_logs;
```

**NOTE:** While running the command `\dx+ audit_log`, we can observe additional types created are *composite* type **employee**, *array* type **employee[]** of composite, *composite* type **logging.audit\_logs** and *array* type **logging.audit\_logs[]** of composite.

These are created automatically as PostgreSQL treats each table as a row type.

# Docker Commands

Connect to postgres:

```
docker exec -it <container> psql -h <host> -p <port> -U <user> <database>
```

If port is exposed on Host machine, then simply:

```
psql -h <host> -p <port> -U <user> <database>
```

View share directory:

```
docker exec -it <container> pg_config --sharedir
```

Copy files inside docker:

```
sudo docker cp my_extension.control container:/path/to/extension  
sudo docker cp my_extension--1.0.sql container:/path/to/extension
```

Restart postgres container:

```
sudo docker stop <container>  
sudo docker start <container>
```

Verify Extension:

```
docker exec -it <container> psql -h <host> -p <port> -U <user> <database>  
-c "SELECT * FROM pg_available_extensions WHERE name =  
'factorial_extension'"
```

# References

1. Calculate Factorial of a given number [\[Link\]](#)
2. Reversing a String [\[Link\]](#)
3. Log DML Events [\[Link\]](#)