
Extending PostgreSQL
EurOmnis 2025
Alex Clay - Suran Systems, Inc.

About Me

- Alex Clay
- Omnis developer since 2000
- CEO of Suran since 2019
- Used PostgreSQL since 2010
- Thousands of deployments
- Original turn-key installer for on-prem PG installs
- Now all hosted on Suran’s managed servers

About you

Some Factors at Suran

- Multi-tenant by individual databases
- Multiples tenants per server
- Multi-platform (Omnis, Rails, Native Mobile)
- Driving logic server-side
- 3rd party integrations
- Single source of truth

PostgreSQL Glue

- dblink
- Foreign Data Wrappers
- Extensions for HTTP calls
- Bulk loading (COPY)
- Shelling out (run programs on the server)
- Bonus :)

Extensions

PostgreSQL Extensions

- Extend PostgreSQL with new functionality
- Many built-in extensions: https://www.postgresql.org/docs/current/contrib.html
- Install additional extensions:
  - OS package manager
  - Compile
  - PGXN
- Some require loading shared libraries
- Special consideration for managed cloud platforms, like RDS

Creating an extension

```
CREATE EXTENSION IF NOT EXISTS [extension] SCHEMA [schema]
CASCADE;

euromnis_demo=# create extension unaccent;
CREATE EXTENSION

euromnis_demo=# create schema unaccent;
CREATE SCHEMA
euromnis_demo=# alter extension unaccent set schema unaccent;
ALTER EXTENSION
euromnis_demo=# drop extension unaccent;
DROP EXTENSION
```

Today's Examples

Example Sources

- https://github.com/barkingfoodog/euromnis2025-courses
- Docker
  - Docker Desktop
  - Orbstack
  - colima

Use Cases

Use Case: suran_database_manager

Client DB 1

Client DB 2

Client DB 3

suran_database_manager

Table of client databases

- Description
- Version
- User ID

How to keep data in suran_database_manager up-to-date?

Use Case: suran_database_manager

dblink

- Used to connect to another PostgreSQL database
- Built into PostgreSQL
- Can run one-off commands
- Can establish a persistent connection
- Best for:
  - Functions
  - One-off SQL commands
- Predecessor to PostgreSQL Foreign Database Wrapper (FDW)

dblink Connection Types

Type	Advantage	Disadvantage
One-off	+ Least code	- Only runs one SQL command then disconnects
Unnamed	+ Less code + Persistent	- Not explicit - Could cause bugs if using multiple unnamed connections
Named	+ Persistent + Can have multiple, concurrent	- A little more setup - Need to manage your connection

dblink One-off

```
# select * from dblink.dblink(
   'dbname=geodata',
   'SELECT geo.country_distance(''US'', ''NL'')'
  ) as t(
   distance float
  );

     distance
--------------------
 7514531.024600981
(1 row)
```

dblink Unnamed

```
euromnis_demo=# select dblink.dblink_connect('dbname=geodata');
 dblink_connect
----------------
 OK
(1 row)

euromnis_demo=# select * from dblink.dblink('SELECT geo.country_distance(''US'', ''NL'')') as
t(distance float);
     distance
------------------
 7514531.024600981
(1 row)

euromnis_demo=# select dblink.dblink_disconnect();
 dblink_disconnect
-------------------
 OK
(1 row)
```

dblink Named

```
euromnis_demo=# select dblink.dblink_connect('geo', 'dbname=geodata');
 dblink_connect
----------------
 OK
(1 row)

euromnis_demo=# select * from dblink.dblink('geo', 'SELECT geo.country_distance(''US'', ''NL'')')
as t(distance float);
     distance
-------------------
 7514531.024600981
(1 row)

euromnis_demo=# select dblink.dblink_disconnect('geo');
 dblink_disconnect
-------------------
 OK
(1 row)
```

Using dblink

- dblink() to query and return results
  - MUST specify return structure
- dblink_exec() to send a command without results
  - void functions
  - INSERTS
- dblink_send_query() sends asynchronous connection
- Cursor functions are available as well

Using dblink

- Connection uses libpq, same as Omnis PG DAM
- Can pass configuration values, sslmode, etc.
- Authentication via .pgpass or foreign server/user mapping (more on that later)
- https://www.postgresql.org/docs/current/dblink.html
- https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-CONNSTRING

Demo

geo.country_distance(source_county, destination_country)

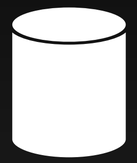
database_utilities.update_fgdb_info()

Use Case: Processor Data Exchange

Client DB 1

Client DB 2

Client DB 3



pde

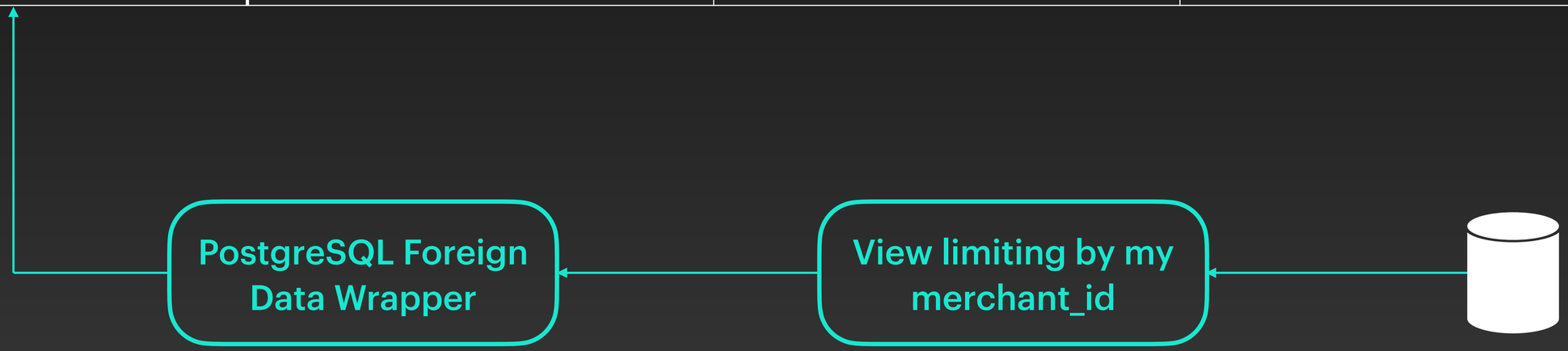
Transaction data for all clients

Organized by merchant_key

How to access filtered data and run repeated queries?

Use Case: Processor Data Exchange

merchant_key	payment_id	date	amount
16	523	2025-10-01	$102.34
16	578	2025-10-02	$56.98
23	580	2025-10-02	$230.21



Foreign Data Wrapper (FDW)

- SQL/MED standard
- Exposes an external or foreign data source via SQL
- Use standard CRUD (SELECT/INSERT/UPDATE/DELETE)
- Built-in:
  - postgres_fdw (Access other PostgreSQL or even your own!)
  - File (Query a file or program, alternative to COPY)
- MANY other options!
- https://wiki.postgresql.org/wiki/Foreign_data_wrappers

FDW Components

EXTENSION

Which FDW?

SERVER

How to get to the data—host, database, file path, URL, etc.

USER MAPPING

Maps local PG roles (users) to credentials for the server

FOREIGN TABLE/SCHEMA

Local relation (table) representing the remote data; Can import a full schema for convenience

PostgreSQL FDW

- Maintained with the core PostgreSQL code
- Improvements with each release, especially for push-down
- Supports push-down where possible
  - Normally, queries/joins pull the full data set from the foreign server, then join or reduce results locally
  - If possible, postgres_fdw asks the remote server to query/join (plan and execute) then only transfers necessary data
  - Big performance benefit
  - Some SQL is too complex for this

Materialized Views

- GREAT way to manage FDW performance, especially over a remote connection for read-only operations
- Strategy:
  - Set up foreign table
  - Create a materialized view for that table
  - Query/join work with materialized view
  - Refresh if/when needed

Materialized Views

```
-- Foreign Table pde.tempe_transaction_auth
# select * from pde.tempe_transaction_auth where
merchant_key = '33551';
Time: 806.358 ms

-- Materialized View pde.transaction_auth
# select * from pde.transaction_auth where merchant_key =
'33551';
Time: 179.144 ms
```

PostgreSQL FDW

- Transactions are created on the remote server to mirror the local connection
- This solves a major challenge with distributed data updates
- Connection management is normally pretty automatic
  - Connect when needed, once per session/server/user
  - Closed when your session terminates
- Can manage with keep_connections or specific functions (see docs)

Compare with dblink

- No function calls with FDW unless mapped to a view
- FDW can be MUCH more performant
- Easier to join and work with foreign tables
- Importing a foreign schema avoids needing to maintain table definition in two places (though PG 18 adds a LIKE option)
- FDW has more setup than dblink, but it persistent and self-healing
- Transaction management is much better with FDW

Demo

geo.country

pde

ucc-data-hub-import

Use Case: API integration from PG

Client DB

Suran API

Running on same host as PG

How to call API endpoints from database, e.g. triggers?

Use Case: API integration from PG

Client DB

Atlassian Jira

Stores courtesy contact issues

Used by Suran to follow up with clients we haven’t engaged with recently

How to automatically create new courtesy contacts?

HTTP from PostgreSQL

- pgsql-http extension
- Wraps libcurl for HTTP calls
- Basic http() function takes a request and returns a response
- Various convenience wrapper functions with methods and content types
- Handy urlencode()
- Talk to any web service, including Omnis!

Example HTTP Call

```
euromnis_demo=# select * from http.http_get('https://api.ipify.org?
format=json');
-[ RECORD 1 ]
+-----------------------------------------------------------------
-----------------------------------------------------------------
-----------------------------------------------------------------
status       | 200
content_type | application/json
headers      | {"(date,\"Sun, 05 Oct 2025 07:28:15 GMT\")","(content-
type,application/json)","(server,cloudflare)","(vary,Origin)","(cf-
cache-status,DYNAMIC)","(content-encoding,br)","(cf-
ray,989b3381199d8cb0-AMS)"}
content      | {"ip":"83.167.209.169"}
```

Example HTTP Call

```
# select trim(both '"' from (content::jsonb ->
'ip')::text) as public_ip from http.http_get('https://
api.ipify.org?format=json');
   public_ip
_________________
 83.167.209.169
(1 row)
```

Demo

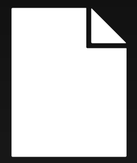
IP Address

Currency exchange

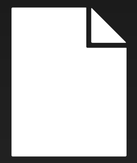
populate_courtesy_contacts

Use Case: Bulk Loading Data

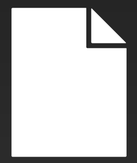
Client DB



CSV Data



CSV Data



CSV Data

How to easily and quickly load data into PostgreSQL?

COPY

- Use COPY to move large amounts of data into and out of PostgreSQL
- Default is a plain text format
- Can also read/write CSV, binary or PROGRAM! (More to come)
- Options for handling NULLs, quoting, error handling
- Can target data to load in out (COPY TO query; add WHERE to COPY FROM)
- Expose as table using file_fdw

COPY vs. \copy

Type	Example	Use
COPY	**COPY geo.country TO country.csv**	• Works with a server-side file • Must be R/W by the postgres process
psql	**\copy geo.country TO country.csv**	• Accesses files by psql client • More versatile for permissions, remote access

Timings

Method	Time 1	Time 2	Time 3	Average
COPY (2m rows)	1443ms	1441ms	1464ms	1449ms
\copy (2m rows)	1494ms	1460ms	1466ms	1473ms
Omnis (1k rows)	5340ms	6900ms	7080ms	6440ms

file_fdw

- Expose any file readable by COPY as a SQL table
- Useful for querying without loading into a table
- No indexes or optimizations; use a table if you need better performance
- Define a single server; specify the file on the FOREIGN TABLE

Demo

people data

doc import people_and_roles

Use Case: Distributing Schemas

Client DB 1

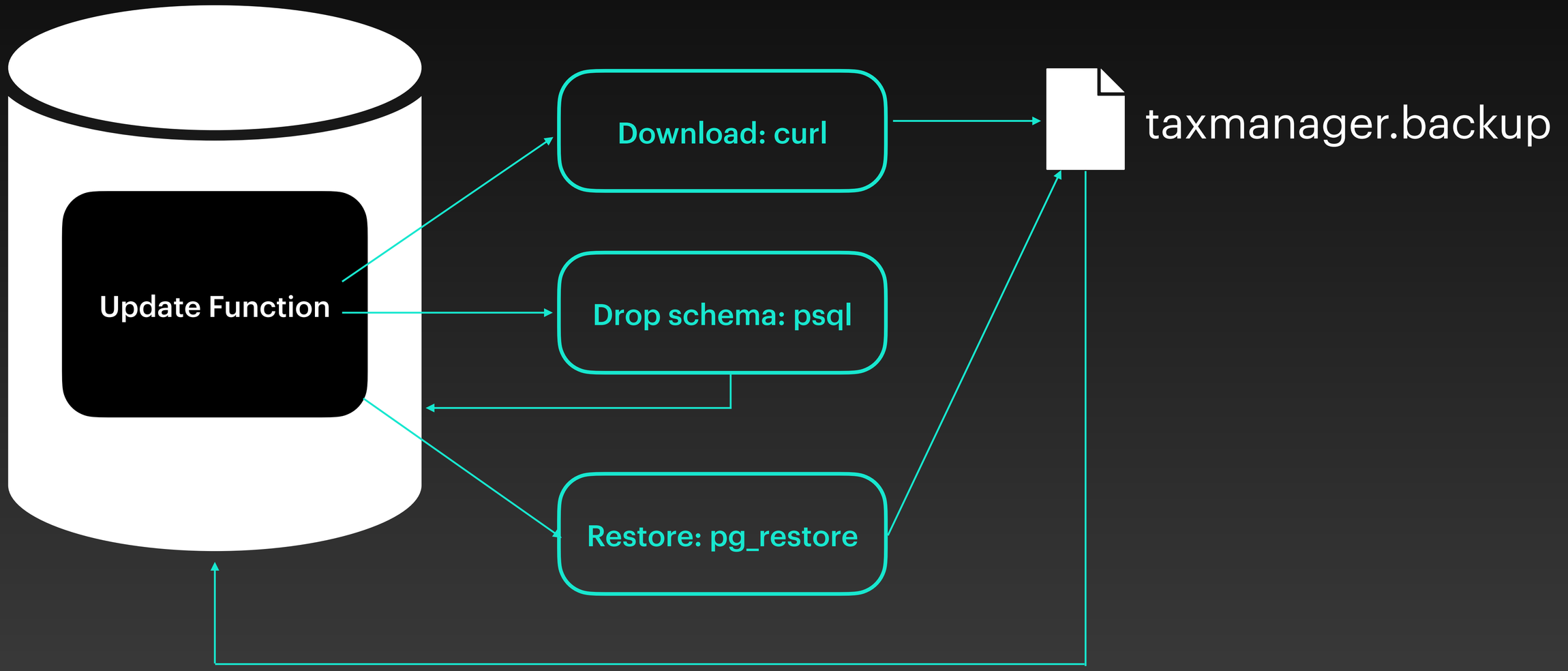
Client DB 2

Client DB 3

Static data and functions stored in a single schema

How to ship updated data and functions to all users on demand?

Use Case: Distributing Schemas



COPY FROM/TO PROGRAM

- Launches a program server-side as the user running postgres
- Consider working directory, permissions of that user
- Returns stdout for COPY FROM
- Sends stdin for COPY TO
- Can add stderr and return code with a little manipulation
- Can expose as a foreign table using file_fdw

Example COPY FROM PROGRAM

```
euromnis_demo=# CREATE TEMPORARY TABLE shell_output (output text);
CREATE TABLE
euromnis_demo=# COPY shell_output FROM PROGRAM 'whoami';
COPY 1
euromnis_demo=# TABLE shell_output;
  output
----------
 postgres
(1 row)

euromnis_demo=# DROP TABLE shell_output;
DROP TABLE
```

Example COPY FROM PROGRAM

file_fdw variant

```
euromnis_demo=# CREATE FOREIGN TABLE shell_output (output text)
SERVER file_fdw OPTIONS (program 'whoami');
CREATE FOREIGN TABLE
euromnis_demo=# TABLE shell_output;
  output
----------
 postgres
(1 row)
```

Example COPY TO PROGRAM

```
euromnis_demo=# copy (table geo.country) to program 'gzip >
country.csv.gz' with
 csv header;
COPY 245

ea5f13a40fa7:/# gunzip $PGDATA/country.csv.gz
ea5f13a40fa7:/# head -n 3 $PGDATA/country.csv
country,latitude,longitude,name
AD,42.546245,1.601554,Andorra
AE,23.424076,53.847818,United Arab Emirates
```

Demo

run_shell_command

update_taxmanager

Bonus

Fun topics

- pg_curl: https://github.com/RekGRpth/pg_curl
- pg_background: https://github.com/vibhorkum/pg_background
- Logical Replication: https://www.postgresql.org/docs/current/logical-replication-subscription.html#LOGICAL-REPLICATION-SUBSCRIPTION-EXAMPLES
- pgAgent
- plperlu… 👎