PostgreSQL Anonymizer and the battle for privac

Table des matières

Bonjour

- Damien Clochard
- Co-founder of Dalibo
- Active member of the French PostgreSQL community

My Journey

- In 2018, someone asked me to anonymize a database
- I thought it would be quick and easy
- I started up writing a bunch of pl/pgsql helper functions…
- I called those functions PostgreSQL Anonymizer[1]
- … and somehow 7 years later I'm still working on it
- This is my story :)

Menu

- The Paradox of Privacy
- 6 principles to protect Data Privacy
- How to implement them with PostgreSQL Anonymizer

---

[1] https://gitlab.com/dalibo/postgresql_anonymizer

THE PARADOX OF PRIVACY

« What is privacy ? »

---

200 years ago

---

50 years ago

This is just the beginning

As our everyday life is getting more and more digitalized,

We want more and more privacy.

Eric Schmidt does not agree

> If you have something that you don't want anyone to know, maybe you shouldn't be doing it in the first place.

Eric Schmidt, Google CEO, 2009

Big Tech wants to redefine the meaning of Privacy

Collecting and selling your personnal info is their business.

The Personal Data is blooming

The data broker industry is worth $200 billion a year.

That's the GDP of Hungary

But it's a shady business

Do you any know of these companies ?

Top 5 Data Brokers	Annual Revenue
Experian	$9.7 billion
Equifax	$5.1 billion
Epsilon	$2.9 billion
Acxiom	$2.7 billion
CoreLogic	$1 billion

This is just the top of this iceberg

- Those 5 brokers combined are bigger than Red Hat
- There are actually 200+ data brokers[2]
- In theory, you can send a letter asking them to remove your data
- … and do it again next month
- Of course, some companies will write these letters for you for 10$/month !

So the black market is rising

This market is absolutely NOT regulated

… and the number of data leaks is exploding

In 2024, 1.7 billion individual notifications[3] of data breach.

That's +300% compared to 2023

This a battle

People want more Privacy

Tech Companies want more data

---

[2] https://onerep.com/blog/largest-data-brokers-in-america
[3] https://www.cnet.com/tech/services-and-software/near-record-number-of-data-breaches-reported-in-2024-report-says/

As a DBA, you're right in the middle

6 BASIC PRINCIPLES

- Privacy By Design
- Attack Surface Reduction
- Role separation
- Data minimisation
- Risk Evaluation
- Privacy By Default

Privacy By Design

The masking policy should be written by the application developpers

---

Role Separation

Define different roles with different masking rules

Attack Surface Reduction (ASR)

Minimizing the risk of data leaks by reducing the places where the data is stored

---

Data minimisation

The type and amount of personal information you collect must be limited to what is directly relevant and necessary to accomplish a specified purpose.

Risk Evaluation

Frequent impact assessment to evaluate the origin, nature, particularity and severity of the risk of leaking personal data

---

Privacy By Default

If you're not sure wether or not a column contains personal information,

you should treat it like it contains personal information.

What is this ?

- An open-source Postgres extension publish by DALIBO
- In production since 2020
- Written in Rust + pl/pgsql
- Install it via RPM / DEB / Docker / Ansible
- Available on Google Cloud SQL, Azure, Crunchy Bridge, Neon, …
- Many outside contributors : DGFIP, ANR, …

A masking Engine

5 different masking methods

- Static Masking
- Dynamic Masking
- Anonymized Dumps
- Maskign Data Wrappers
- Masking Views

A masking toolbox

10 types of masking functions

- Destruction
- Noise Addition
- Shuffling / Permutation
- Randomization
- Faking / Synthetizing
- Pseudonymization
- Hashing
- Partial Scrambling
- Generalization
- Image alteration

Load and Create the extension

```
ALTER DATABASE foo SET session_preload_libraries = 'anon';

\connect foo

CREATE EXTENSION anon;
```

Privacy By Design

- The extension has a declarative approach of anonymization.
- Declare **masking rules** within the database model
- Using the `SECURITY LABEL` syntax

Privacy By Design : example

```
SELECT * FROM people;

   id    | firstname | lastname |   phone
---------+-----------+----------+------------
 153478 | Sarah     | Conor    | 0609110911
```

Privacy By Design : masking with fake data

```
SECURITY LABEL FOR anon ON COLUMN people.lastname
  IS 'MASKED WITH FUNCTION anon.dummy_last_name()';
```

Privacy By Design : destruction

```
SECURITY LABEL FOR anon ON COLUMN people.id
  IS 'MASKED WITH VALUE NULL';
```

Privacy By Design : partial destruction

```
SECURITY LABEL FOR anon ON COLUMN people.phone
  IS 'MASKED WITH FUNCTION anon.partial(phone,2,$$******$$,2)';
```

Privacy By Design : Applying the rules

```
SELECT anon.anonymize_table('people');
```

Privacy By Design : Static Masking

```
SELECT * FROM people;

   id   | firstname | lastname  |   phone
--------+-----------+-----------+------------
        | Sarah     | Stranahan | 06******11
```

Role Separation

- Declare that a role is MASKED
- The masking rules will be automatically applied to this role
- By definition a MASKED role is read-only
- Other roles can still read and write the real data

Role Separation : Dynamic Masking

```
CREATE ROLE skynet LOGIN;

ALTER ROLE skynet SET anon.transparent_dynamic_masking TO true;

SECURITY LABEL FOR anon ON ROLE skynet IS 'MASKED';

GRANT pg_read_all_data to skynet;
```

Role Separation : Dynamic Masking

```
SET ROLE skynet;

SELECT * FROM people;

 id | firstname | lastname  |   phone
----+-----------+-----------+------------
    | Sarah     | Donahue   | 06******11
```

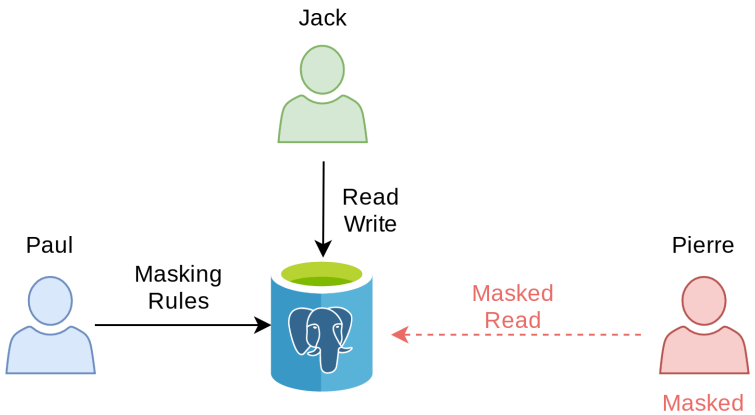
Role Separation : Dynamic Masking

```
SET ROLE postgres;

SELECT * FROM people;

 id     | firstname | lastname |   phone
--------+-----------+----------+------------
 153478 | Sarah     | Conor    | 0609110911
```

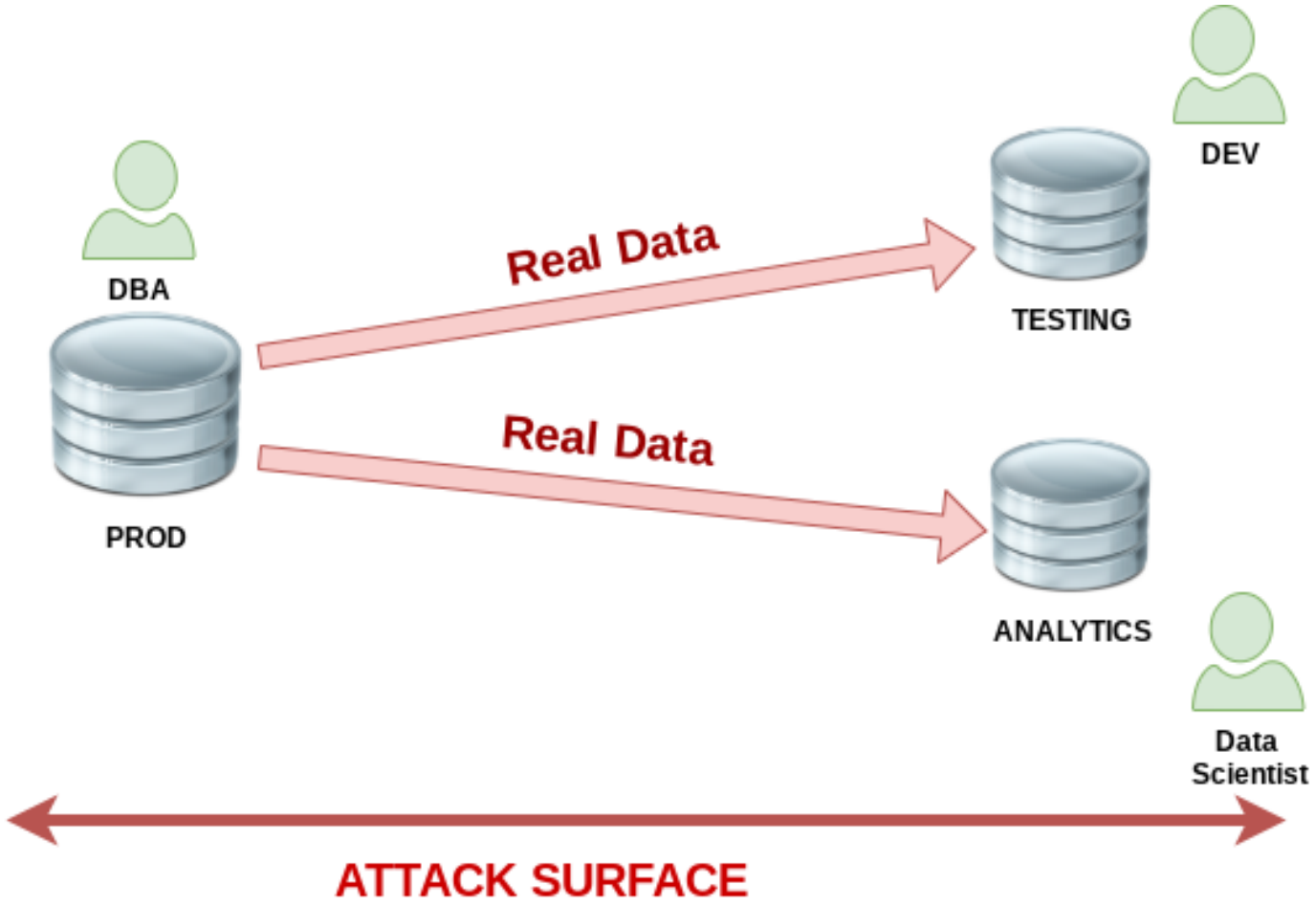
Role Separation : Overview



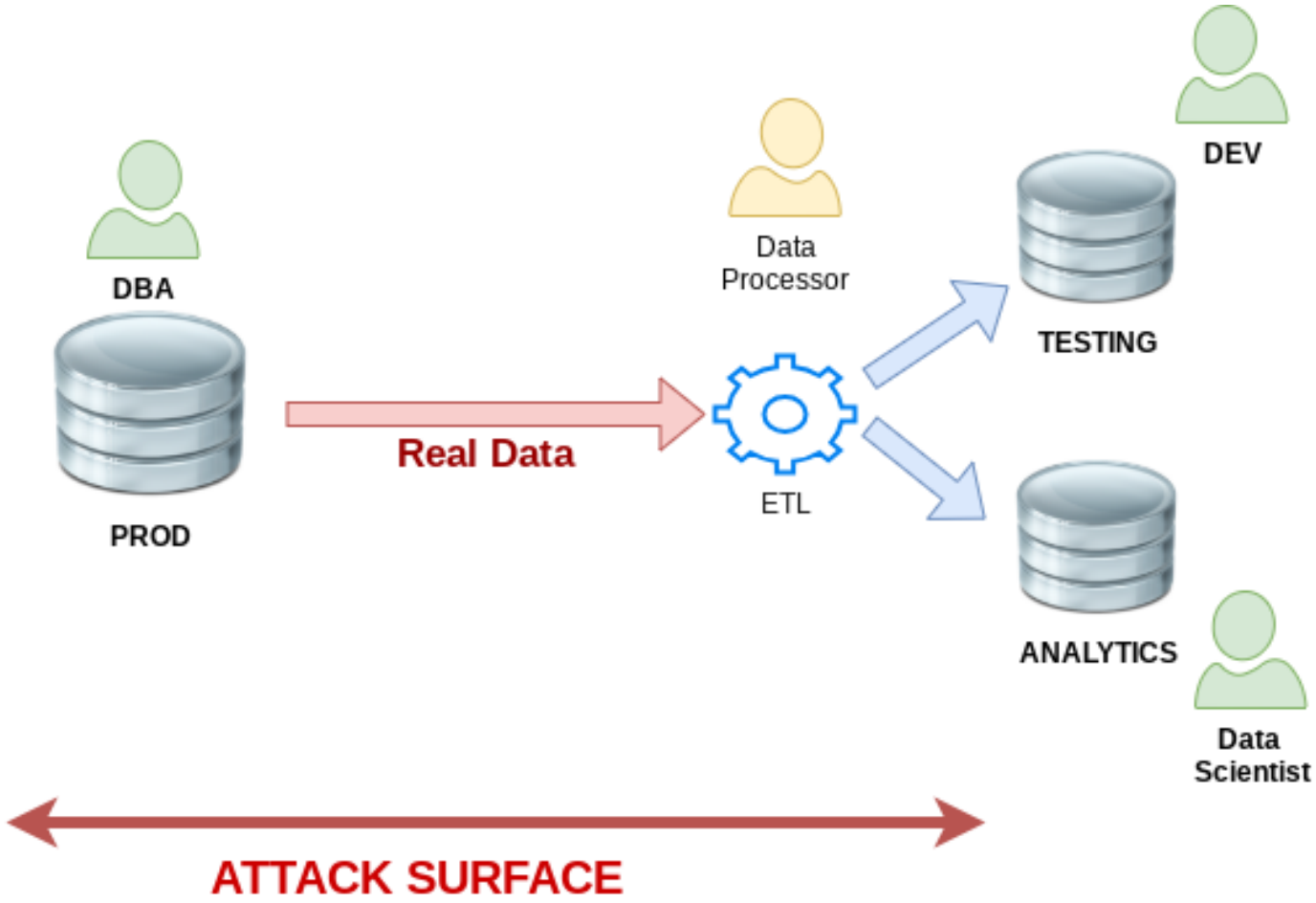
Attack Surface Reduction (ASR)

ASR : Basic Example

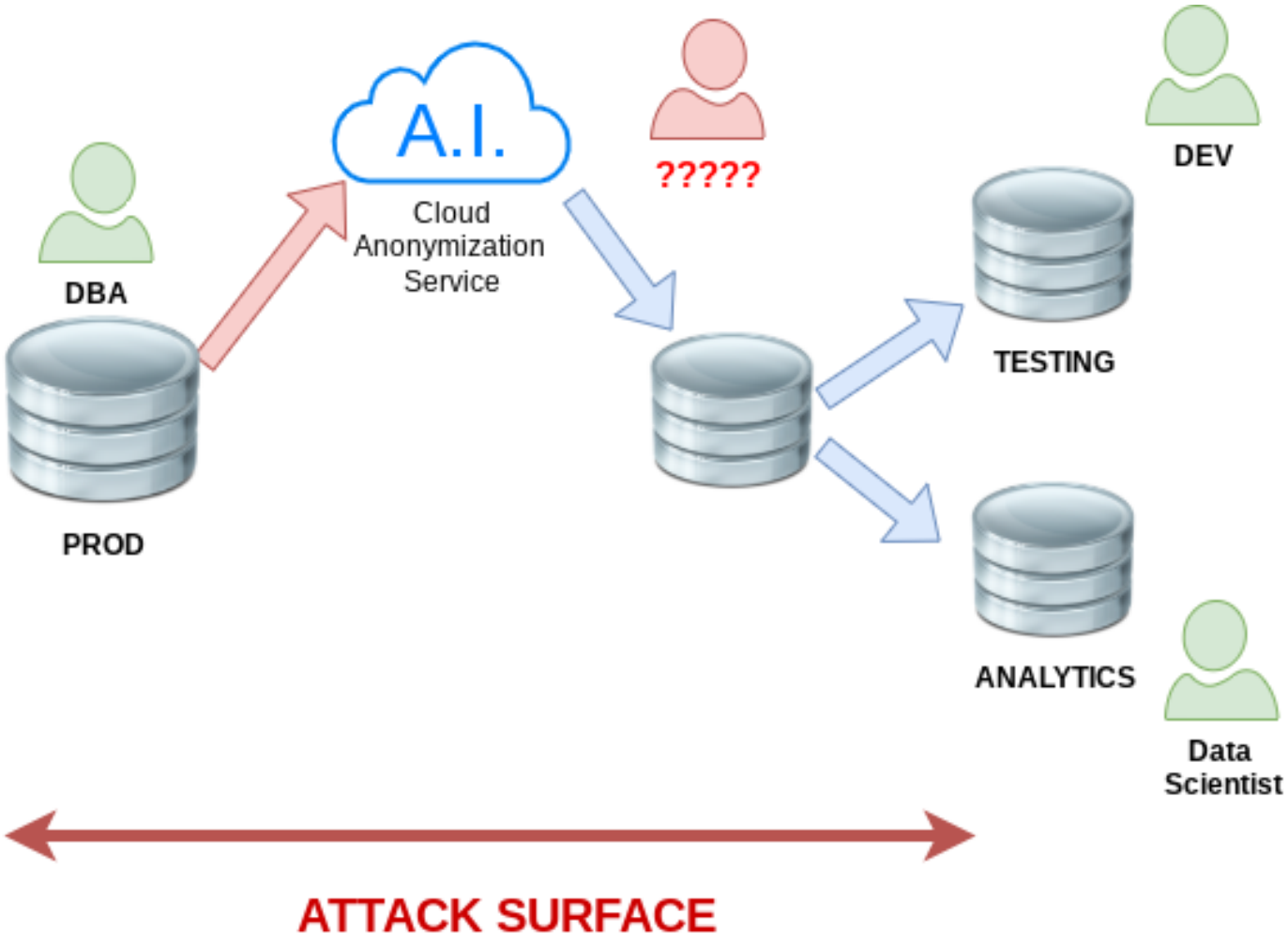
ASR : Worst Case Scenario



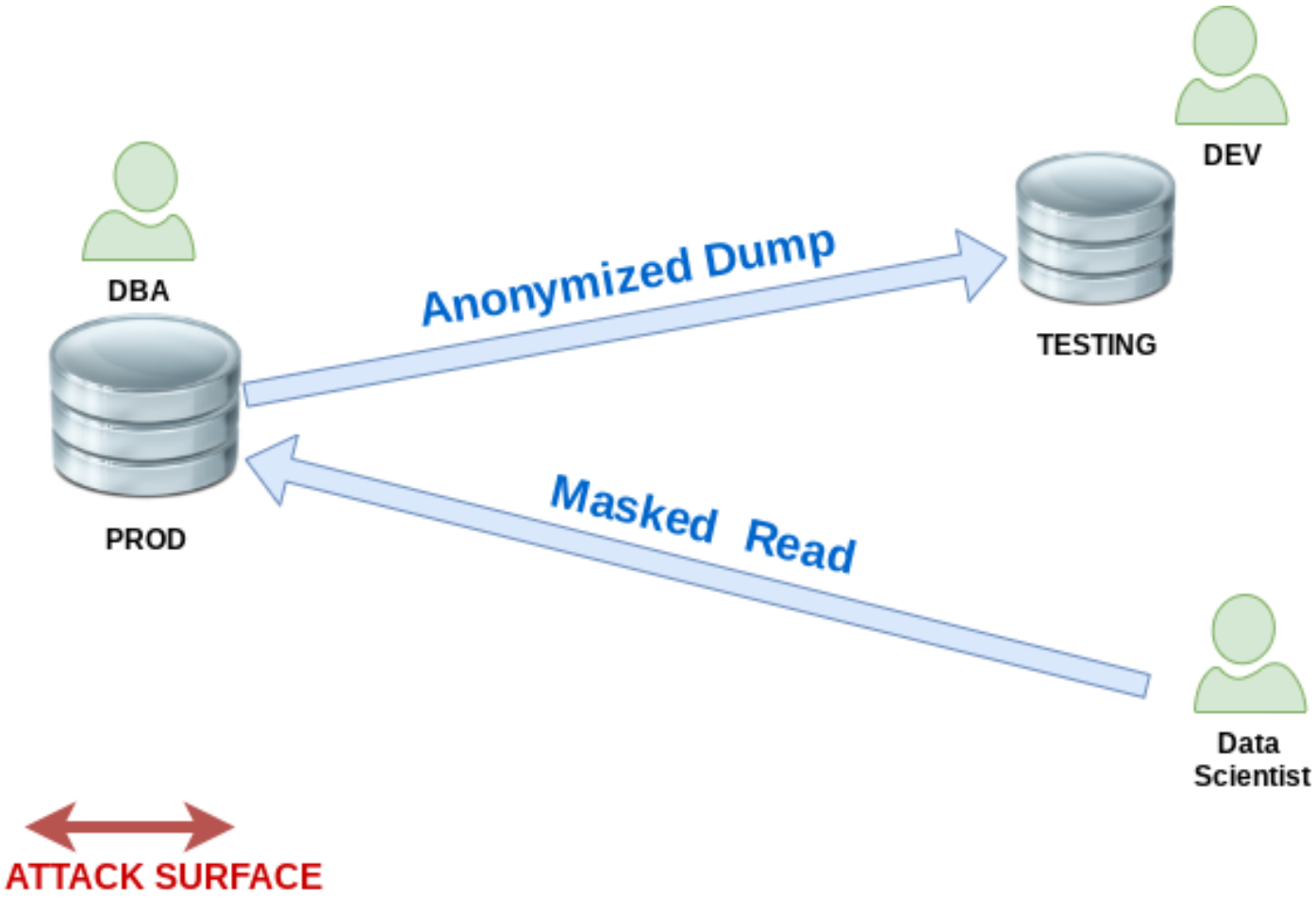
ASR : ETL


DBA
PROD
Real Data
Data
Processor
ETL
DEV
TESTING
ANALYTICS
Data
Scientist
ATTACK SURFACE

ASR : WTF


A.I.
Cloud
Anonymization
Service
?????
DBA
PROD
DEV
TESTING
ANALYTICS
Data
Scientist
ATTACK SURFACE

ASR : PostgreSQL Anonymizer



ASR : Anonymous Dumps

```
CREATE ROLE anon_dumper LOGIN PASSWORD 'CHANGEME';

ALTER ROLE anon_dumper SET anon.transparent_dynamic_masking = True;

SECURITY LABEL FOR anon ON ROLE anon_dumper IS 'MASKED';

GRANT pg_read_all_data to anon_dumper;
```

ASR : Anonymous Dumps

```
$ pg_dump -h localhost -U dump_anon [...] foo > anonymous_dump.sql
```

You can now share and reload the `anonymous_dump.sql` file anywhere you want

ASR : Anonymous Dumps

- Use the anonymous dump to refresh development environments
- This is the regular pg_dump
- `--format=custom` is supported

Data minimisation

- In most use case, the masked role do not need to access **ALL** the data
- A sample is sufficient for tests, analytics, demo, training data
- This is called sampling
- Did you know that PostgreSQL already has a `TABLESAMPLE` clause ?

Sampling with TABLESAMPLE

Let's say you have a huge amounts of http logs stored in a table. You want to remove the ip addresses and extract only 10% of the table:

```
CREATE TABLE http_logs (
  id integer NOT NULL,
  date_opened DATE,
  ip_address INET,
  url TEXT
);
```

Sampling with TABLESAMPLE

You want to remove the ip addresses and extract only 10% of the table:

```
SECURITY LABEL FOR anon ON COLUMN http_logs.ip_address
  IS 'MASKED WITH VALUE NULL';

SECURITY LABEL FOR anon ON TABLE http_logs
  IS 'TABLESAMPLE BERNOULLI(10)';
```

Sampling with RLS policies

Another approach for sampling is to use Row Level Security Policies[4], also known as RLS or `Row Security Policies`.

Sampling with RLS policies

Let's use the same example as a above, this time we want to define a limit so the mask users can only see the logs of the last 6 months.

```
ALTER TABLE http_logs ENABLE ROW LEVEL SECURITY;

CREATE POLICY http_logs_sampling_for_masked_users
  ON http_logs
  USING (
    NOT anon.hasmask(CURRENT_USER::REGROLE)
    OR date_opened >= now() - '6 months'::INTERVAL
  );
```

Risk Evaluation

- The extension provides an implementation of the `k-anonymity`
- A detection function to find columns that should have a mask

---

[4] https://www.postgresql.org/docs/current/ddl-rowsecurity.html

Risk Evaluation : k-anonymity

- k-anonymity is an industry-standard term used to describe a property of an anonymized dataset.
- Any anonymized individual cannot be distinguished from at least k-1 other individuals inside the dataset

---

Risk Evaluation : k-anonymity

```
SELECT anon.k_anonymity('generalized_patient')

 k_anonymity
--------------
 3
```

The higher the value, the better…

Risk Evaluation : detection missing masking rules

```
# SELECT anon.detect('en_US');

 table_name |  column_name   | identifiers_category | direct
------------+----------------+----------------------+--------
 customer   | CreditCard     | creditcard           | t
 vendor     | Firstname      | firstname            | t
 customer   | firstname      | firstname            | t
 customer   | id             | account_id           | t
```

Privacy By Default

Privacy By Default : Example

Imagine a database named `access_logs` with a basic table containing HTTP logs:

```
CREATE TABLE http_logs (
    date_opened DATE,
    ip_address INET,
    url TEXT
    browser_agent DEFAULT 'unknown'
  );
```

Privacy By Default

```
SELECT * FROM access_logs LIMIT 1;

  date_opened  |     ip_addr     |    url     |    browser_agent
---------------+-----------------+------------+-----------------------
 2009-01-08    | 192.168.100.128 | /home.html | Mozilla/5.0 (Windows)
```

Privacy By Default

Now let's activate privacy by default:

```
ALTER DATABASE foo SET anon.privacy_by_default = True;
```

Privacy By Default : Unmask columns

```
SECURITY LABEL FOR anon ON COLUMN access_logs.url
  IS 'NOT MASKED';
```

```
SECURITY LABEL FOR anon ON COLUMN access_logs.date_opened
  IS $$ MASKED WITH FUNCTION pg_catalog.date_trunc('year',birth) $$;
```

Privacy By Default

We can now anonymize the table without writing any masking rule.

```
SELECT anon.anonymize_database();
 anonymize_database
--------------------
 t
```

Privacy By Default

```
SELECT * FROM access_logs LIMIT 1;

 date_open  | ip_addr | url            | browser_agent
------------+---------+--------------+---------------
 2009-01-01 |         | /home.html     | unknown
```

IN A NUTSHELL

- The Battle for Privacy is happening now..
- Wether you want it or not : you’re in it

Gear up Step by step

- Privacy by Design is the key concept
- then Role Saparation and ASR
- Finally : Data Minimisation, Risk Evaluation, Privacy By Default

---

Do not fight alone

This is a team effort

- Application developpers
- Sysadmins
- DPO
- etc.

Different strategies for different use cases

- Anonymous Dumps : Simply export the masked data into an SQL file
- Static Masking : Permanently remove the personal info
- Dynamic Masking : Hide personal info only for the masked users

---

Thanks !

Project

https://gitlab.com/dalibo/postgresql_anonymizer/

Tutorial

https://dali.bo/anon_tuto

Contact

damien.clochard@dalibo.com[5]

PostgreSQL Anonymizer 2025 Survey

https://dali.bo/anon_survey_2025

---

[5]

QUESTIONS ?