

# Mandatory Access Control in PostgreSQL - giving users ownership of their data

Leon du Toit

2019-01-15

# Outline

- ▶ why take data ownership seriously?
- ▶ why Mandatory Access Control?
- ▶ a brief introduction to the `pg-need-to-know` module
- ▶ a use case to demonstrate features:
  - ▶ For users: ownership, insight and consent-based usage
  - ▶ For administrators: fine-grained access control, audit information
  - ▶ For developers: a rich REST API, with a built-in authorization model
- ▶ optionally: a look at some implementation details

# Why take data ownership seriously?

- ▶ Regulations of the GDPR
  - ▶ increased focus on data privacy and protection
  - ▶ right to access
  - ▶ right to be forgotten
  - ▶ data portability
  - ▶ consent-based data usage
  - ▶ increased demand for audit information
- ▶ To counter surveillance capitalism
  - ▶ you (and your data) are the product
  - ▶ building applications to fight this trend

# What Mandatory Access Control?

- ▶ *enforcible* policies, in contrast to Discretionary Access Control
- ▶ enables consent-based data access
- ▶ supports granular access needs

## pg-need-to-know

- ▶ PostgreSQL “module” - really just a set of tables, views, and functions
- ▶ implements Mandatory Access Control
- ▶ more limited approach than SEPostgreSQL
- ▶ source: <https://github.com/leondutoit/pg-need-to-know>
- ▶ written in PL/pgSQL
  - ▶ procedural language, extending SQL with control structures
  - ▶ used to create functions
  - ▶ ~1000 sloc, another ~1500 for tests
- ▶ uses Row-Level Security policies to implement MAC
- ▶ designed to be used via a REST API

# Use case

Key terms:

- ▶ data owner: provides data about themselves
- ▶ data user: analyses data about others
- ▶ admin: creates and implements access control policies

# Use case

Assume the following setup:

- ▶ data owners: A, B, C, D, E, F
- ▶ data users: X, Y, Z
- ▶ tables: `spending_habits`, `personal_details`, containing data from all data owners

# Use case

Now suppose we need to set up the following access control rules in our DB:

- ▶ data users X, and Y should only have access to data in tables spending\_habits and only data from owners A, B, C, D
- ▶ data user Z should have access to all data - i.e. tables spending\_habits, personal\_details



# Use case

A hypothetical sequence of events:

1. admin creates tables
2. data owners and data users register themselves, data is collected
3. admin creates groups, adds members, adds table grants
4. data is analysed
5. users manage their own data
6. admins get audit insights
7. developers create applications using composing these features

# Table creation

```
set role admin_user;
SET
select table_create(
  '{"table_name": "spending_habits",
    "columns": [
      {"name": "spending", "type": "int",
        "description": "Amount spent in NOK"},
      {"name": "item_type", "type": "text",
        "description": "Type of item purchased"},
      {"name": "purchase_date", "type": "date",
        "description": "Year-Month-Day on which purchase occurred"} ],
    "description": "data about spending habits"}'::json,
  'mac');
table_create
-----
Success
(1 row)
```

Figure 1: Creating a new table

# User registration

- ▶ can require consent before user registration
- ▶ data collection not possible without registration

# Group setup, table grants

- ▶ can link consent(s) to groups via group metadata
- ▶ group1
  - ▶ members: ((X, Y), (A, B, C, D))
  - ▶ select table access grant: (spending\_habits)
- ▶ group2
  - ▶ members: ((Z), (A, B, C, D, E, F))
  - ▶ select table access grants: (spending\_habits, personal\_details)

# Data analysis

```
set session "request.jwt.claim.user" = 'user_X';
SET
select current_setting('request.jwt.claim.user');
current_setting
-----
user_X
(1 row)

select * from spending_habits;
      row_id      | row_owner | row_originator | spending | item_type | purchase_date
-----+-----+-----+-----+-----+-----
4ad3b11e-32ff-42a1-850c-aff1f93f190e | owner_A  | owner_A        |      140 | food      | 2019-01-02
975f2758-5749-4915-bac1-48530f703062 | owner_A  | owner_A        |      100 | drink     | 2019-01-03
899efca4-a935-4e0d-ba25-29c4413d7c2a | owner_B  | owner_B        |       60 | drink     | 2019-01-02
7ef73351-1e7d-4f26-989b-d55fa0f1bfa5 | owner_B  | owner_B        |       78 | drink     | 2019-01-04
1be698b3-e1c0-4b98-a236-f273883f67dc | owner_C  | owner_C        |     1020 | travel    | 2019-01-04
c225db92-2171-4d21-9b7a-67c4ef0ad942 | owner_C  | owner_C        |      101 | food      | 2019-01-04
123f6322-130a-4b13-8f2a-2dbe0f0a9523 | owner_D  | owner_D        |      230 | travel    | 2019-01-05
ca97c462-7fea-49ce-8bde-fcef08a910ab | owner_D  | owner_D        |     448 | travel    | 2019-01-06
(8 rows)

select * from personal_details;
psql:./src/11-user-X-data-access.sql:7: ERROR:  access denied to table
CONTEXT:  PL/pgSQL function ntk.data_user_group_membership_with_correct_privileges(uuid,text,text) line 18 at
RAISE
```

Figure 2: User X's data access

# Data analysis

```
set session "request.jwt.claim.user" = 'user_Z';
```

```
SET
```

```
select * from spending_habits;
```

row_id	row_owner	row_originator	spending	item_type	purchase_date
4ad3b11e-32ff-42a1-850c-aff1f93f190e	owner_A	owner_A	140	food	2019-01-02
975f2758-5749-4915-bac1-48530f703062	owner_A	owner_A	100	drink	2019-01-03
899efca4-a935-4e0d-ba25-29c4413d7c2a	owner_B	owner_B	60	drink	2019-01-02
7ef73351-1e7d-4f26-989b-d55fa0f1bfa5	owner_B	owner_B	78	drink	2019-01-04
1be698b3-e1c0-4b98-a236-f273883f67dc	owner_C	owner_C	1020	travel	2019-01-04
c225db92-2171-4d21-9b7a-67c4ef0ad942	owner_C	owner_C	101	food	2019-01-04
123f6322-130a-4b13-8f2a-2dbe0f0a9523	owner_D	owner_D	230	travel	2019-01-05
ca97c462-7fea-49ce-8bde-fcef08a910ab	owner_D	owner_D	448	travel	2019-01-06
fc56af9e-b361-4f9d-814a-3cab834730fd	owner_E	owner_E	10230	housing	2019-01-01
d2f1e45f-e3c0-4fae-8c3d-1470fc4fb75e	owner_F	owner_F	209	food	2019-01-06

```
(10 rows)
```

```
select * from personal_details;
```

row_id	row_owner	row_originator	name	age
5a2a949e-89e5-413d-b268-27516e4924b4	owner_A	owner_A	James Martin	44
336d4202-394c-4abc-9231-3127431df3e8	owner_B	owner_B	Sandra Fourie	18
ce67a250-dc92-48b1-882f-ef68c9ba9687	owner_C	owner_C	Willem White	11
e0ab7e50-3f81-4180-9c5e-b0423e8e17af	owner_D	owner_D	Lee Simpson	84
0b8cb4f0-78a0-448d-8ad5-173e94e1c488	owner_E	owner_E	Gerhard du Preez	23
0a7280e6-e19c-457f-82db-d2b40190ef7d	owner_F	owner_F	Hannah Furgeson	33

Figure 3: User Z's data access

# Data ownership

- ▶ right to access
- ▶ data portability
- ▶ right to be forgotten

# Right to access

```
set role data_owner;
SET
set session "request.jwt.claim.user" = 'owner_A';
SET
select * from spending_habits;
```

row_id	row_owner	row_originator	spending	item_type	purchase_date
4ad3b11e-32ff-42a1-850c-aff1f93f190e	owner_A	owner_A	140	food	2019-01-02
975f2758-5749-4915-bac1-48530f703062	owner_A	owner_A	100	drink	2019-01-03

(2 rows)

```
select * from personal_details;
```

row_id	row_owner	row_originator	name	age
5a2a949e-89e5-413d-b268-27516e4924b4	owner_A	owner_A	James Martin	44

(1 row)

Press any key to execute: (ans/14 owner\_A group remove sel

Figure 4:Owner A's data access



# Data portability

- ▶ owner A can simply download their data

# Right to be forgotten

Owner B deletes their data

# Audit insights

- ▶ data access
- ▶ access control changes
- ▶ user initiated group removals
- ▶ user initiated data deletions
- ▶ data updates

# Audit: data access

# Audit: access control changes

# Audit: user initiated group removals

# Audit: user initiated data deletions

# Audit: data updates



# Application development

Architecture:

`client -> webapp -> REST -> (pg-need-to-know, PostgreSQL)`

- ▶ developers can focus on business logic
- ▶ authorization taken care of