# Data Masking in PostgreSQL

## What is Data Masking?

Data masking involves altering data to protect sensitive information while retaining its usability for specific purposes. Unlike data encryption, which transforms data into an unreadable format, data masking obscures the data in such a way that it remains useful for development and testing but is not identifiable or exploitable.

## Why Use Dynamic Data Masking?

Dynamic data masking allows organizations to secure sensitive information by masking it in real-time as users access it. This method ensures that the actual data remains unchanged in the database but appears obscured to unauthorized users or in less secure environments. This is particularly useful for:

- **Compliance:** Meeting regulatory requirements such as GDPR, HIPAA, and CCPA.
- **Development and Testing:** Providing realistic data for development and testing without exposing real sensitive information.
- **User Privacy:** Ensuring that sensitive information is protected when accessed by users with limited privileges.

## Setting Up Dynamic Data Masking in PostgreSQL

To demonstrate how to implement dynamic data masking, we'll use the `anon` extension in PostgreSQL. Here's a step-by-step guide to setting up data masking:

### 1. Install the Required Packages

First, ensure you have the necessary packages installed. For PostgreSQL 14, you need `ddlx_14`, `python3-faker`, and `postgresql_anonymizer_14`. Install them using the following commands:

```
rpm -iv ddlx_14-0.27-1PGDG.rhel9.noarch.rpm
rpm -iv python3-faker-13.3.3-1.el9.noarch.rpm
rpm -iv postgresql_anonymizer_14-1.1.0-1.rhel9.x86_64.rpm
```

### 2. Configure the Database

Create a new database and configure it to use the `anon` extension:

```
CREATE DATABASE test;
ALTER DATABASE test SET session_preload_libraries = 'anon';  -- Apply this setting to
enable the anon extension
\c test; -- connect test db with psql
CREATE EXTENSION anon CASCADE;
SELECT anon.init();
```

### 3. Create and Populate Tables

Next, create a table and insert some sample data:

```
CREATE DATABASE employee;
\c employee;

CREATE TABLE people (
    id INT,
    firstname VARCHAR(10),
    lastname VARCHAR(10),
    phone VARCHAR(15)
);
INSERT INTO people (id, firstname, lastname, phone) VALUES (1, 'Kemal', 'Oz',
'9012345678');
```

### 4. Set Up Roles and Permissions

Create roles and grant the necessary permissions:

```
CREATE ROLE hr LOGIN;
GRANT SELECT ON people TO hr;
```

### 5. Apply Data Masking

Enable dynamic data masking and define masking policies:

```
CREATE EXTENSION IF NOT EXISTS anon CASCADE;
SELECT anon.start dynamic masking();

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

### 6. Test Data Masking

Now, when you query the `people` table as the `hr` role, you'll see the masked data:

```
\c employee;
SELECT * FROM people;
 id | firstname |  lastname  |   phone
----+-----------+------------+------------
  1 | Kemal     | Kshlerin   | 90******78
(1 row)
```

## Conclusion

Dynamic data masking in PostgreSQL using the `anon` extension is a robust solution for protecting sensitive information while maintaining its usefulness for various purposes. By following the steps outlined above, you can ensure that your data remains secure and compliant with regulations, all while providing realistic data for development, testing, and other use cases. Implementing data masking is a crucial step in safeguarding your organization's data. With the right tools and techniques, you can enhance your data security posture and maintain the trust of your users and stakeholders. For more detailed and technical articles like this, keep following our blog on Medium. If you have any questions or need further assistance, feel free to reach out in the comments below and directly.