» postgresql_database

The `postgresql_database` resource creates and manages database objects within a PostgreSQL server instance.

» Usage

```
resource "postgresql_database" "my_db" {
  name              = "my_db"
  owner             = "my_role"
  template          = "template0"
  lc_collate        = "C"
  connection_limit  = -1
  allow_connections = true
}
```

» Argument Reference

- `name` - (Required) The name of the database. Must be unique on the PostgreSQL server instance where it is configured.
- `owner` - (Optional) The role name of the user who will own the database, or `DEFAULT` to use the default (namely, the user executing the command). To create a database owned by another role or to change the owner of an existing database, you must be a direct or indirect member of the specified role, or the username in the provider is a superuser.
- `tablespace_name` - (Optional) The name of the tablespace that will be associated with the database, or `DEFAULT` to use the template database's tablespace. This tablespace will be the default tablespace used for objects created in this database.
- `connection_limit` - (Optional) How many concurrent connections can be established to this database. `-1` (the default) means no limit.
- `allow_connections` - (Optional) If `false` then no one can connect to this database. The default is `true`, allowing connections (except as restricted by other mechanisms, such as `GRANT` or `REVOKE CONNECT`).
- `is_template` - (Optional) If `true`, then this database can be cloned by any user with `CREATEDB` privileges; if `false` (the default), then only superusers or the owner of the database can clone it.
- `template` - (Optional) The name of the template database from which to create the database, or `DEFAULT` to use the default template (`template0`). NOTE: the default in Terraform is `template0`, not `template1`. Changing

this value will force the creation of a new resource as this value can only be changed when a database is created.

- `encoding` - (Optional) Character set encoding to use in the database. Specify a string constant (e.g. `UTF8` or `SQL_ASCII`), or an integer encoding number. If unset or set to an empty string the default encoding is set to `UTF8`. If set to `DEFAULT` Terraform will use the same encoding as the template database. Changing this value will force the creation of a new resource as this value can only be changed when a database is created.
- `lc_collate` - (Optional) Collation order (`LC_COLLATE`) to use in the database. This affects the sort order applied to strings, e.g. in queries with `ORDER BY`, as well as the order used in indexes on text columns. If unset or set to an empty string the default collation is set to `C`. If set to `DEFAULT` Terraform will use the same collation order as the specified `template` database. Changing this value will force the creation of a new resource as this value can only be changed when a database is created.
- `lc_ctype` - (Optional) Character classification (`LC_CTYPE`) to use in the database. This affects the categorization of characters, e.g. lower, upper and digit. If unset or set to an empty string the default character classification is set to `C`. If set to `DEFAULT` Terraform will use the character classification of the specified `template` database. Changing this value will force the creation of a new resource as this value can only be changed when a database is created.

» Import Example

`postgresql_database` supports importing resources. Supposing the following Terraform:

```
provider "postgresql" {
  alias = "admindb"
}

resource "postgresql_database" "db1" {
  provider = "postgresql.admindb"

  name = "testdb1"
}
```

It is possible to import a `postgresql_database` resource with the following command:

```
$ terraform import postgresql_database.db1 testdb1
```

Where `testdb1` is the name of the database to import and `postgresql_database.db1` is the name of the resource whose state will be populated as a result of the

command.

» postgresql_extension

The `postgresql_extension` resource creates and manages an extension on a PostgreSQL server.

» Usage

```
resource "postgresql_extension" "my_extension" {
  name = "pg_trgm"
}
```

» Argument Reference

- `name` - (Required) The name of the extension.
- `schema` - (Optional) Sets the schema of an extension.
- `version` - (Optional) Sets the version number of the extension.

» postgresql_role

The `postgresql_role` resource creates and manages a role on a PostgreSQL server.

When a `postgresql_role` resource is removed, the PostgreSQL ROLE will automatically run a `REASSIGN OWNED` and `DROP OWNED` to the `CURRENT_USER` (normally the connected user for the provider). If the specified PostgreSQL ROLE owns objects in multiple PostgreSQL databases in the same PostgreSQL Cluster, one PostgreSQL provider per database must be created and all but the final `postgresql_role` must specify a `skip_drop_role`.

Note: All arguments including role name and password will be stored in the raw state as plain-text. Read more about sensitive data in state.

» Usage

```
resource "postgresql_role" "my_role" {
  name     = "my_role"
  login    = true
  password = "mypass"
}
```

```
resource "postgresql_role" "my_replication_role" {
  name             = "replication_role"
  replication      = true
  login            = true
  connection_limit = 5
  password         = "md5c98cbfeb6a347a47eb8e96cfb4c4b890"
}
```

» Argument Reference

- `name` - (Required) The name of the role. Must be unique on the PostgreSQL server instance where it is configured.
- `superuser` - (Optional) Defines whether the role is a "superuser", and therefore can override all access restrictions within the database. Default value is `false`.
- `create_database` - (Optional) Defines a role's ability to execute `CREATE DATABASE`. Default value is `false`.
- `create_role` - (Optional) Defines a role's ability to execute `CREATE ROLE`. A role with this privilege can also alter and drop other roles. Default value is `false`.
- `inherit` - (Optional) Defines whether a role "inherits" the privileges of roles it is a member of. Default value is `true`.
- `login` - (Optional) Defines whether role is allowed to log in. Roles without this attribute are useful for managing database privileges, but are not users in the usual sense of the word. Default value is `false`.
- `replication` - (Optional) Defines whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. Default value is `false`
- `bypass_row_level_security` - (Optional) Defines whether a role bypasses every row-level security (RLS) policy. Default value is `false`.
- `connection_limit` - (Optional) If this role can log in, this specifies how many concurrent connections the role can establish. `-1` (the default) means no limit.
- `encrypted_password` - (Optional) Defines whether the password is stored encrypted in the system catalogs. Default value is `true`. NOTE: this value is always set (to the conservative and safe value), but may interfere with the behavior of PostgreSQL's `password_encryption` setting.
- `password` - (Optional) Sets the role's password. (A password is only of use for roles having the `login` attribute set to true, but you can nonetheless

define one for roles without it.) Roles without a password explicitly set are left alone. If the password is set to the magic value `NULL`, the password will be always be cleared.

- `valid_until` - (Optional) Defines the date and time after which the role's password is no longer valid. Established connections past this `valid_time` will have to be manually terminated. This value corresponds to a PostgreSQL datetime. If omitted or the magic value `NULL` is used, `valid_until` will be set to `infinity`. Default is `NULL`, therefore `infinity`.
- `skip_drop_role` - (Optional) When a PostgreSQL ROLE exists in multiple databases and the ROLE is dropped, the cleanup of ownership of objects in each of the respective databases must occur before the ROLE can be dropped from the catalog. Set this option to true when there are multiple databases in a PostgreSQL cluster using the same PostgreSQL ROLE for object ownership. This is the third and final step taken when removing a ROLE from a database.
- `skip_reassign_owned` - (Optional) When a PostgreSQL ROLE exists in multiple databases and the ROLE is dropped, a `REASSIGN OWNED` in must be executed on each of the respective databases before the `DROP ROLE` can be executed to dropped the ROLE from the catalog. This is the first and second steps taken when removing a ROLE from a database (the second step being an implicit `DROP OWNED`).

» Import Example

`postgresql_role` supports importing resources. Supposing the following Terraform:

```
provider "postgresql" {
  alias = "admindb"
}

resource "postgresql_role" "replication_role" {
  provider = "postgresql.admindb"

  name = "replication_name"
}
```

It is possible to import a `postgresql_role` resource with the following command:

```
$ terraform import postgresql_role.replication_role replication_name
```

Where `replication_name` is the name of the role to import and `postgresql_role.replication_role` is the name of the resource whose state will be populated as a result of the command.

» postgresql_schema

The `postgresql_schema` resource creates and manages schema objects within a PostgreSQL database.

» Usage

```
resource "postgresql_role" "app_www" {
  name = "app_www"
}

resource "postgresql_role" "app_dba" {
  name = "app_dba"
}

resource "postgresql_role" "app_releng" {
  name = "app_releng"
}

resource "postgresql_schema" "my_schema" {
  name  = "my_schema"
  owner = "postgres"

  policy {
    usage = true
    role  = "${postgresql_role.app_www.name}"
  }

  # app_releng can create new objects in the schema.  This is the role that
  # migrations are executed as.
  policy {
    create = true
    usage  = true
    role   = "${postgresql_role.app_releng.name}"
  }

  policy {
    create_with_grant = true
    usage_with_grant  = true
    role              = "${postgresql_role.app_dba.name}"
  }
}
```

» Argument Reference

- `name` - (Required) The name of the schema. Must be unique in the PostgreSQL database instance where it is configured.
- `owner` - (Optional) The ROLE who owns the schema.
- `if_not_exists` - (Optional) When true, use the existing schema if it exists. (Default: true)
- `policy` - (Optional) Can be specified multiple times for each policy. Each policy block supports fields documented below.

The `policy` block supports:

- `create` - (Optional) Should the specified ROLE have CREATE privileges to the specified SCHEMA.
- `create_with_grant` - (Optional) Should the specified ROLE have CREATE privileges to the specified SCHEMA and the ability to GRANT the CREATE privilege to other ROLEs.
- `role` - (Optional) The ROLE who is receiving the policy. If this value is empty or not specified it implies the policy is referring to the `PUBLIC` role.
- `usage` - (Optional) Should the specified ROLE have USAGE privileges to the specified SCHEMA.
- `usage_with_grant` - (Optional) Should the specified ROLE have USAGE privileges to the specified SCHEMA and the ability to GRANT the USAGE privilege to other ROLEs.

NOTE on `policy`: The permissions of a role specified in multiple policy blocks is cumulative. For example, if the same role is specified in two different `policy` each with different permissions (e.g. `create` and `usage_with_grant`, respectively), then the specified role with have both `create` and `usage_with_grant` privileges.

» Import Example

`postgresql_schema` supports importing resources. Supposing the following Terraform:

```
resource "postgresql_schema" "public" {
  name = "public"
}

resource "postgresql_schema" "schema_foo" {
  name  = "my_schema"
  owner = "postgres"

  policy {
    usage = true
  }
```

```
}
```

It is possible to import a `postgresql_schema` resource with the following command:

```
$ terraform import postgresql_schema.schema_foo my_schema
```

Where `my_schema` is the name of the schema in the PostgreSQL database and `postgresql_schema.schema_foo` is the name of the resource whose state will be populated as a result of the command.