» mysql_database

The `mysql_database` resource creates and manages a database on a MySQL server.

Caution: The `mysql_database` resource can completely delete your database just as easily as it can create it. To avoid costly accidents, consider setting `prevent_destroy` on your database resources as an extra safety measure.

» Example Usage

```
resource "mysql_database" "app" {
  name = "my_awesome_app"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the database. This must be unique within a given MySQL server and may or may not be case-sensitive depending on the operating system on which the MySQL server is running.
- `default_character_set` - (Optional) The default character set to use when a table is created without specifying an explicit character set. Defaults to "utf8".
- `default_collation` - (Optional) The default collation to use when a table is created without specifying an explicit collation. Defaults to `utf8_general_ci`. Each character set has its own set of collations, so changing the character set requires also changing the collation.

Note that the defaults for character set and collation above do not respect any defaults set on the MySQL server, so that the configuration can be set appropriately even though Terraform cannot see the server-level defaults. If you wish to use the server's defaults you must consult the server's configuration and then set the `default_character_set` and `default_collation` to match.

» Attributes Reference

The following attributes are exported:

- `name` - The name of the database.
- `id` - The id of the database.
- `default_character_set` - The default_character_set of the database.
- `default_collation` - The default_collation of the database.

» Import

Databases can be imported using their name, e.g.

```
$ terraform import mysql_database.example my-example-database
```

» mysql_grant

The `mysql_grant` resource creates and manages privileges given to a user on a MySQL server.

» Granting Privileges to a User

```
resource "mysql_user" "jdoe" {
  user               = "jdoe"
  host               = "example.com"
  plaintext_password = "password"
}

resource "mysql_grant" "jdoe" {
  user       = "${mysql_user.jdoe.user}"
  host       = "${mysql_user.jdoe.host}"
  database   = "app"
  privileges = ["SELECT", "UPDATE"]
}
```

» Granting Privileges to a Role

```
resource "mysql_role" "developer" {
  name = "developer"
}

resource "mysql_grant" "developer" {
  role       = "${mysql_role.developer.name}"
  database   = "app"
  privileges = ["SELECT", "UPDATE"]
}
```

» Adding a Role to a User

```
resource "mysql_user" "jdoe" {
  user               = "jdoe"
```

```
  host               = "example.com"
  plaintext_password = "password"
}

resource "mysql_role" "developer" {
  name = "developer"
}

resource "mysql_grant" "developer" {
  user     = "${mysql_user.jdoe.user}"
  host     = "${mysql_user.jdoe.host}"
  database = "app"
  roles    = ["${mysql_role.developer.name}"]
}
```

» Argument Reference

Note: MySQL removed the `REQUIRE` option from `GRANT` in version 8. `tls_option` is ignored in MySQL 8 and above.

Note: Attributes `role` and `roles` are only supported in MySQL 8 and above.

The following arguments are supported:

- `user` - (Optional) The name of the user. Conflicts with `role`.
- `host` - (Optional) The source host of the user. Defaults to "localhost". Conflicts with `role`.
- `role` - (Optional) The role to grant `privileges` to. Conflicts with `user` and `host`.
- `database` - (Required) The database to grant privileges on.
- `table` - (Optional) Which table to grant `privileges` on. Defaults to `*`, which is all tables.
- `privileges` - (Optional) A list of privileges to grant to the user. Refer to a list of privileges (such as here) for applicable privileges. Conflicts with `roles`.
- `roles` - (Optional) A list of rols to grant to the user. Conflicts with `privileges`.
- `tls_option` - (Optional) An TLS-Option for the `GRANT` statement. The value is suffixed to `REQUIRE`. A value of 'SSL' will generate a `GRANT ... REQUIRE SSL` statement. See the MYSQL `GRANT` documentation for more. Ignored if MySQL version is under 5.7.0.
- `grant` - (Optional) Whether to also give the user privileges to grant the same privileges to other users.

» Attributes Reference

No further attributes are exported.

» mysql_role

The `mysql_role` resource creates and manages a user on a MySQL server.

Note: MySQL introduced roles in version 8. They do not work on MySQL 5 and lower.

» Example Usage

```
resource "mysql_role" "developer" {
  name = "developer"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the role.

» Attributes Reference

No further attributes are exported.

» mysql_user

The `mysql_user` resource creates and manages a user on a MySQL server.

Note: The password for the user is provided in plain text, and is obscured by an unsalted hash in the state Read more about sensitive data in state. Care is required when using this resource, to avoid disclosing the password.

» Example Usage

```
resource "mysql_user" "jdoe" {
  user               = "jdoe"
  host               = "example.com"
```

```
  plaintext_password = "password"
}
```

» Example Usage with an Authentication Plugin

```
resource "mysql_user" "nologin" {
  user               = "nologin"
  host               = "example.com"
  auth_plugin        = "mysql_no_login"
}
```

» Argument Reference

The following arguments are supported:

- `user` - (Required) The name of the user.
- `host` - (Optional) The source host of the user. Defaults to "localhost".
- `plaintext_password` - (Optional) The password for the user. This must be provided in plain text, so the data source for it must be secured. An *unsalted* hash of the provided password is stored in state. Conflicts with `auth_plugin`.
- `password` - (Optional) Deprecated alias of `plaintext_password`, whose value is *stored as plaintext in state*. Prefer to use `plaintext_password` instead, which stores the password as an unsalted hash. Conflicts with `auth_plugin`.
- `auth_plugin` - (Optional) Use an authentication plugin to authenticate the user instead of using password authentication. Description of the fields allowed in the block below. Conflicts with `password` and `plaintext_password`.

- `tls_option` - (Optional) An TLS-Option for the `CREATE USER` or `ALTER USER` statement. The value is suffixed to `REQUIRE`. A value of 'SSL' will generate a `CREATE USER ... REQUIRE SSL` statement. See the MYSQL `CREATE USER` documentation for more. Ignored if MySQL version is under 5.7.0.

The `auth_plugin` value supports:

- `AWSAuthenticationPlugin` - Allows the use of IAM authentication with Amazon Aurora. For more details on how to use IAM auth with Aurora, see here.

- `mysql_no_login` - Uses the MySQL No-Login Authentication Plugin. The No-Login Authentication Plugin must be active in MySQL. For more information, see here.

» Attributes Reference

The following attributes are exported:

- `user` - The name of the user.
- `password` - The password of the user.
- `id` - The id of the user created, composed as "username@host".
- `host` - The host where the user was created.

» Attributes Reference

No further attributes are exported.

» mysql_user_password

The `mysql_user_password` resource sets and manages a password for a given user on a MySQL server.

NOTE on MySQL Passwords: This resource conflicts with the `password` argument for `mysql_user`. This resource uses PGP encryption to avoid storing unencrypted passwords in Terraform state.

NOTE on How Passwords are Created: This resource **automatically** generates a **random** password. The password will be a random UUID.

» Example Usage

```
resource "mysql_user" "jdoe" {
  user = "jdoe"
}
 resource "mysql_user_password" "jdoe" {
  user    = "${mysql_user.jdoe.user}"
  pgp_key = "keybase:joestump"
}
```

You can rotate passwords by running `terraform taint mysql_user_password.jdoe`. The next time Terraform applies a new password will be generated and the user's password will be updated accordingly.

» Argument Reference

The following arguments are supported:

- `user` - (Required) The IAM user to associate with this access key.

- `pgp_key` - (Required) Either a base-64 encoded PGP public key, or a keybase username in the form `keybase:some_person_that_exists`.
- `host` - (Optional) The source host of the user. Defaults to `localhost`.

» Attributes Reference

The following additional attributes are exported:

- `key_fingerprint` - The fingerprint of the PGP key used to encrypt the password
- `encrypted_password` - The encrypted password, base64 encoded.

NOTE: The encrypted password may be decrypted using the command line, for example: `terraform output encrypted_password | base64 --decode | keybase pgp decrypt`.