How to Create Users, Grant Them Privileges, and Remove Them in Oracle Database

July 30, 2018 | 8 minute read

Chris Saxon
Developer Advocate

So, you've got your shiny, brand new Oracle Database up and running. It's time to start creating users!

But how do you do this?

Ryan McGuire Gratisography

First you'll need login as system or sys. Once you're in, the basic create user command is:

Copy code snippet

```
create user <username> identified by "<password>";
```

So to create the user data_owner with the password Supersecurepassword!, use:

Copy code snippet

```
create user data_owner identified by "Supersecurepassword!";
```

Now you've got your user. The next step is to connect to it. But try to do so and you'll hit:

Copy code snippet

```
conn data_owner/Supersecurepassword!

ORA-01045: user DATA_OWNER lacks CREATE SESSION privilege; logon denied
```

What's going on?

The problem is you haven't given the user any permissions! By default a database user has no privileges. Not even to connect.

Granting User Privileges

You give permissions with the grant command. For system privileges this takes the form:

Copy code snippet

```
grant <privilege> to <user>
```

To allow your user to login, you need to give it the create session privilege. Let's do that:

Copy code snippet

```
grant create session to data_owner;
```

There are a whole raft of other permissions you can give your users. And some rather powerful roles that grant them all.

So what should you enable?

At this point, keen to get developing, you may be tempted to give your user a bucket of powerful permissions.

Before you do, remember a key security concept:

The Principle of Least Privilege.

Only give your users the smallest set of privileges they need to do their job. For a basic data schema that's simply create table:

Copy code snippet

```
grant create table to data_owner;
```

This allows you to make tables. As well as indexes and constraints on them. But critically, not store data in them!

Which is could lead to embarrassing errors when deploy your brand new application:

Copy code snippet

```
conn data_owner/Supersecurepassword!

create table customers (
  customer_id integer not null primary key,
  customer_name varchar2(100) not null
);

insert into customers values ( 1, 'First customer!' );

ORA-01950: no privileges on tablespace 'USERS'
```

To avoid this, you need to give your user a tablespace quota. You'll want to do this on their default tablespace. Which you can find with:

Copy code snippet

```
select default_tablespace from dba_users
where  username = 'DATA_OWNER';

DEFAULT_TABLESPACE
USERS
```

Assign the quota by altering the user, like so:

Copy code snippet

```
alter user data_owner quota unlimited on users;
```

These privileges will get you far. But to build an application there are a few other privileges you're likely to need:

create view – Allows you to create views

create procedure – Gives the ability to create procedures, functions and packages

create sequence – The ability to make sequences

You can give many system privileges in one go. Grant these to data_owner by chaining them together like so:

Copy code snippet

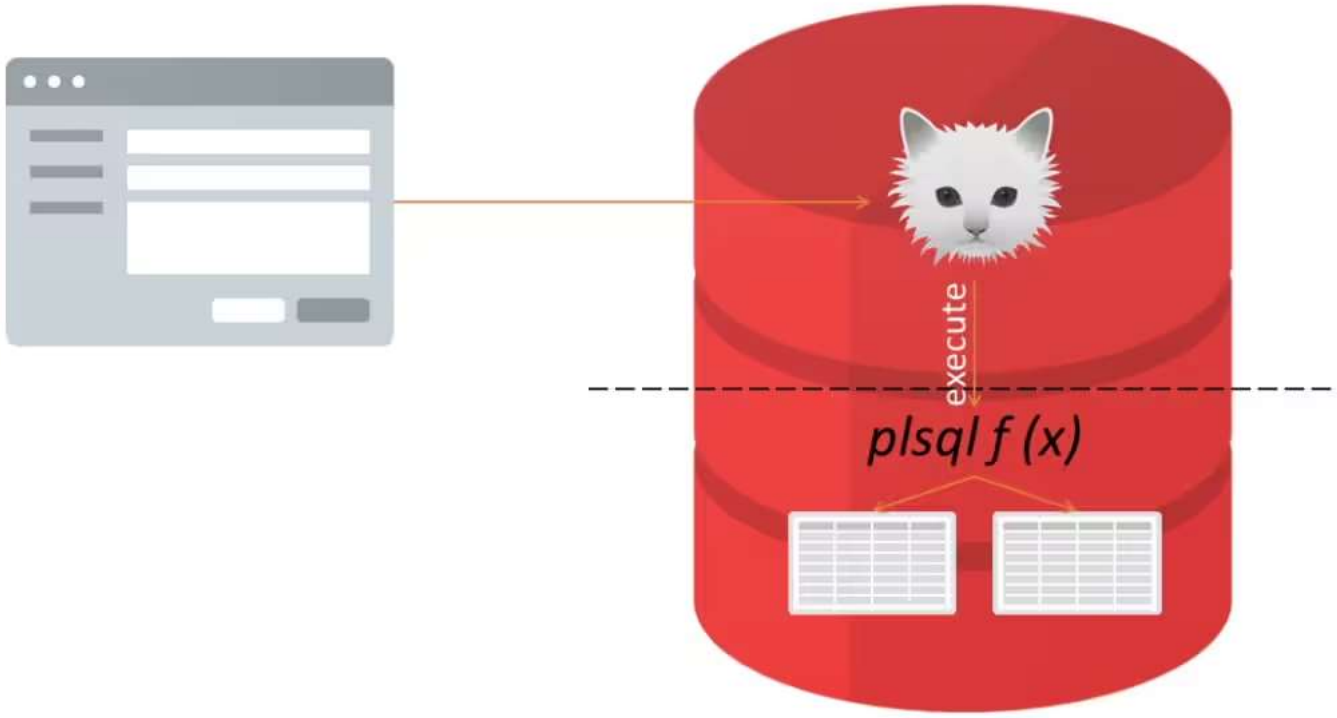
```
grant create view, create procedure, create sequence to data_owner;
```

Notice the lack of “drop <object type>” access. That’s because database users always have full privileges on their own objects. Meaning you can run any queries against your own tables. And insert, update, and delete rows however you like. And drop them!

Which brings a possible security loophole.

If your application connects to the database as the user which owns the tables, if you have any SQL injection vulnerabilities you’re in trouble!

To avoid this, separate the connection user and the data schema. Ideally with a PL/SQL API between your tables and the users.



To learn more about protecting your database behind a PL/SQL API, head to the SmartDB resource center.

So to secure your data, you need to create another user. The only system privilege you should give it is create session.

Great, another two statements you're thinking.

Luckily there's a shortcut. You can create a user and grant it system privileges in one go!

Just add the identified by clause to grant:

Copy code snippet

```
grant create session to app_user identified by "theawesomeeststrongestpasswor
```

If the user already exists this will grant the privileges. And reset the password. So take care when running this, or you may change their password!

Password Management

A brief note on password rules. By default the password will expire every 180 days. Which can lead to ORA-28002 errors on login.

Not only is this kinda annoying, it goes against current password guidelines. You can get around this by changing the password_life_time for the user's profile.

While you're at it, you probably want to stop people picking short, easy to crack passwords. You can define a password complexity function to do this.
So you've created your application user.

But you still need to assign it permissions on data_owner's objects. For table level access, you can give access to query and change the rows with:

Copy code snippet

```
grant select, insert, update, delete on data_owner.customers to app_user;
```

There is a "grant all" option for tables. But before you reach for this, be aware that not only does it include the DML permissions above, it also gives:

- alter
- debug
- flashback
- index
- on commit refresh
- query rewrite
- read
- references

Ouch!

Remember: only give out the exact permissions users need. No more!

If you have done the good thing and protected your data behind a PL/SQL API, grant execute to allow app_user to call it. Like so:

Copy code snippet

```
grant execute on data_owner.customers_api to app_user;
```

You can only grant permissions on one object at a time. So you'll need to repeat this for each thing app_user needs access to.

To give these object privileges, you need to either:

- Own the object in question
- Have the *grant any object privilege* privilege
- Have been granted the permission using the *with grant option*

As a rule you should avoid giving out "any" privileges. So in most cases you should only grant object privileges when connected as the object owner.

But you may want to have a low-level admin user. You'll use this to grant permissions to other users. Such as the ability to query some of data_owner's tables for reporting. If you're feeling lazy, grant allows you to create many users in one go:

Copy code snippet

```
grant create session to reporting_admin, report_user_1
  identified by "theadminpassword", "theuserpassword";
```

Now, to allow reporting_admin to give query privileges on data_owner's objects to report_user_1, you can:

- Connect to data_owner
- Grant query permissions with grant option
- Connect to reporting_admin to pass these permissions onto others

Like so:

Copy code snippet

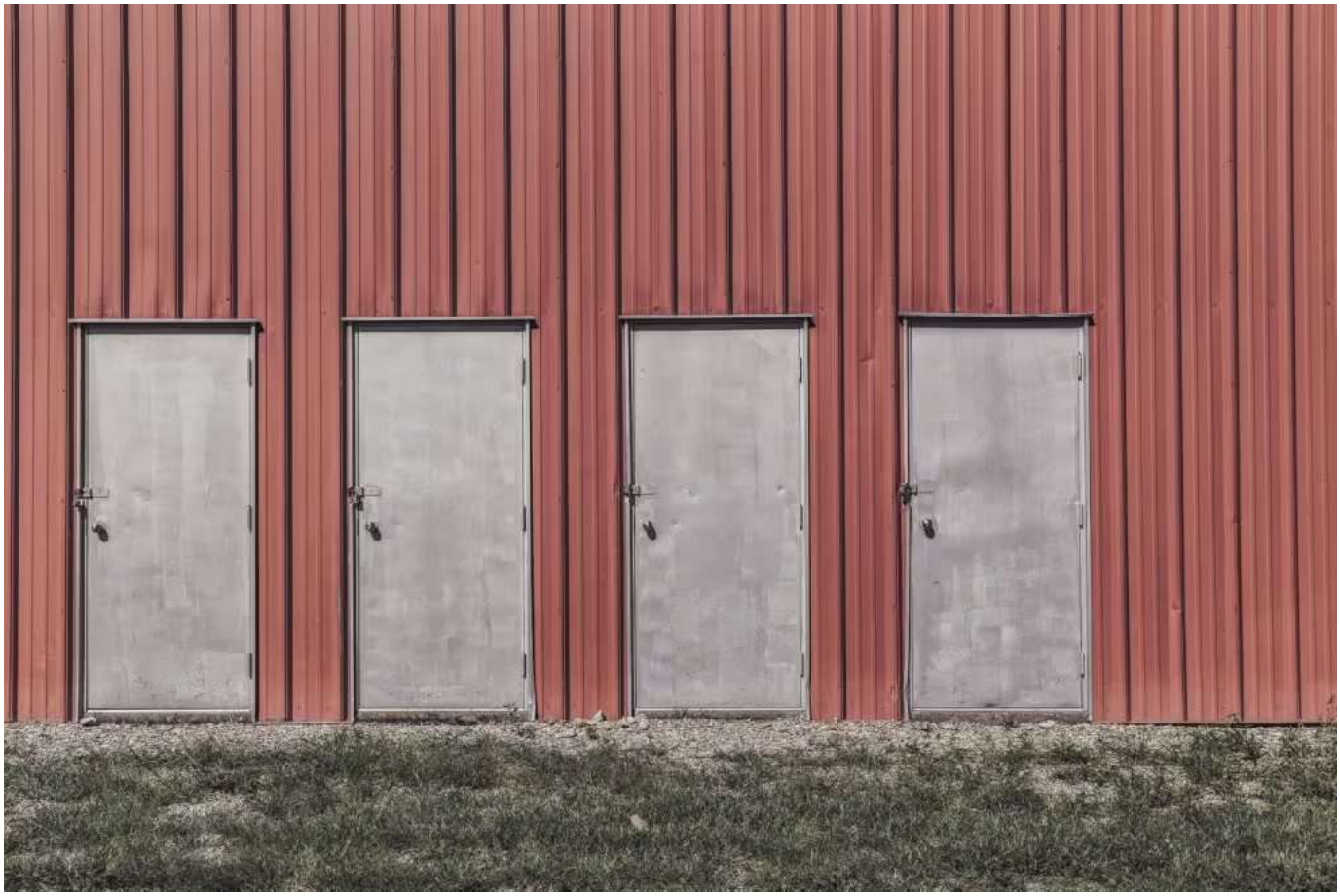
```
conn data_owner/Supersecurepassword!

grant read on customers to reporting_admin with grant option;

conn reporting_admin/theadminpassword
```

```
grant read on data_owner.customers to report_user_1;
```

Note the grant of read instead of select. This is a new privilege in Oracle Database 12c. Granting select allows users to lock tables. Read doesn't. So you should give this privilege to read-only users instead of select.

So you've given your application users the smallest set of privileges they need.

You've locked the front door. But there's still a backdoor!



Ryan McGuire Gratisography

Anyone with access to your network can connect as data_owner. At which point they're free to wreak havoc in your database.

This is a tricky problem to avoid. You can stop people getting in by locking the account with:

Copy code snippet

```
alter user data_owner account lock;
```

But this brings a couple of issues.

First up, it's easy to overlook this step. If you want to connect to data_owner, say to release some changes, you'll need to unlock it. And remember to lock it again afterwards! A step easily forgotten when dealing

you'll need to unlock it. And remember to lock it again afterwards. A step easily forgotten when dealing with emergency releases.

But there's another problem. It allows hackers to easily discover the names of your database users. When you try and connect to a locked account, you'll get the following message:

Copy code snippet

```
conn data_owner/random_password

ORA-28000: The account is locked.
```

If I'm phishing around your database, I now know it contains the user data_owner. Even though I don't know the password!

Now, hopefully(!), your network security is good enough that hackers can't scan through possible usernames to find the names of your accounts.

But this trick is a quick way for them to see if your database has Oracle supplied users installed. Things like Oracle Text or Oracle Spatial. If you have, this increases the options for a hacker to get in.

So what do you do?

Luckily Oracle Database 18c offers another way around this problem: schema-only accounts!

Schema vs. User

At this point it's worth noting the difference between schemas and users. Officially a schema is a collection of tables. Whereas a user is an account you use to connect to the database. Some databases allow you to make a distinction between these with separate create schema and create user commands.

But in Oracle Database, there's no difference between a schema and a user. All tables belong to one user.

While the create schema command exists, you can only use it to create tables within an existing user.

So "schema-only" accounts are users which have no password. To create one, use the no authentication clause instead of identified by:

Copy code snippet

```
create user data_owner no authentication;
```

Now there is literally no way to login to this account. Any attempts to do so will hit:

Copy code snippet

```
conn data_owner/random_password

ORA-01017: invalid username/password; logon denied
```

So you no longer know if data_owner is a valid account.

Is the user missing? Or are they present, but you've got the password wrong? You don't know.

So you've stopped hackers learning about your database. Great. But.

You're probably thinking:

How do *I* connect to data_owner?

From time-to-time it's likely you'll want to connect to do things like run release scripts.

Sure, you can assign a temporary password with:

Copy code snippet

```
alter user data_owner identified by "Supersecurepassword!";
```

And remove it again when you're done with:

Copy code snippet

```
alter user data_owner no authentication;
```

But this is a repeat of the lock problem again. What if you forget to remove authentication when you're done?

Luckily, there's a solution: proxy users.

Ryan McGuire Gratisography

Proxy Users

Proxy users are low privilege accounts. With the ability to connect to higher powered users.

To use them, you need to create the user. And give it the power to connect through another account:

Copy code snippet

```
grant create session to proxy_user identified by "proxy_user_password";
alter user data_owner grant connect through proxy_user;
```

With this in place, you can now connect to proxy_user. But run with the privileges of data_owner. Do so with:

Copy code snippet

```
conn proxy_user[data_owner]/proxy_user_password
```

Using this method, you can leave your schema-only accounts with no password.

Removing Access

Over time applications get decommissioned. Or rewritten to access different information. But usually the data remains.

Leaving the user with access to unneeded data is a security risk. Stay on top of this and remove access when it's no longer needed.

To do this, use the revoke command. This states what you're removing from who. For system privileges this is:

Copy code snippet

```
revoke create table from data_owner;
```

For object privileges, include the thing you're removing access from:

Copy code snippet

```
revoke select on data_owner.important_stuff from app_user;
```

Remember: if your release scripts have grants for existing objects you'll need to undo these if you have to rollback. So ensure you include the corresponding revoke in your rollback scripts!

Dropping Users

Getting rid of unwanted users is easy. Drop them with:

Copy code snippet

```
drop user <username>;
```

You can only do this if the user is not connected to the database. So ensure you clear up any sessions it has before you do so.

And there's another step you need to watch for. Run this for data_owner and you're likely to hit this error:

Copy code snippet

```
drop user data_owner;

ORA-01922: CASCADE must be specified to drop 'DATA_OWNER'
```

Why?

You can't remove users that own objects!

So you need to go in and drop all its tables, views, etc. Or do it in one shot with:

Copy code snippet

```
drop user data_owner cascade;
```

This is an easy way to wipe *all* your data. So use with care!

Want to know more?

Read up on create user, drop user, grant, and revoke in the documentation.

Learn the basics of SQL in Databases for Developers: Foundations.