User Management and Access Control in PostgreSQL

Estimated time needed: 15 minutes

For much of the routine tasks involved with interacting with a database, such as reading the content of a table or adding new entries, the postgres superuser may not be appropriate as it bypasses all permission checks, which carries inherent risk. Furthermore, as a database administrator, you will almost certainly not be the only one who will need to access the database in some capacity. For this reason, you will need a way to add new users to the database and give them the proper privileges that is appropriate for their use cases.

Objectives

After completing this lab, you will be able to:

- Create roles in a database and grant them select permissions
- Create new users in the database and assign them the appropriate role
- Revoke and deny access to the database from a user

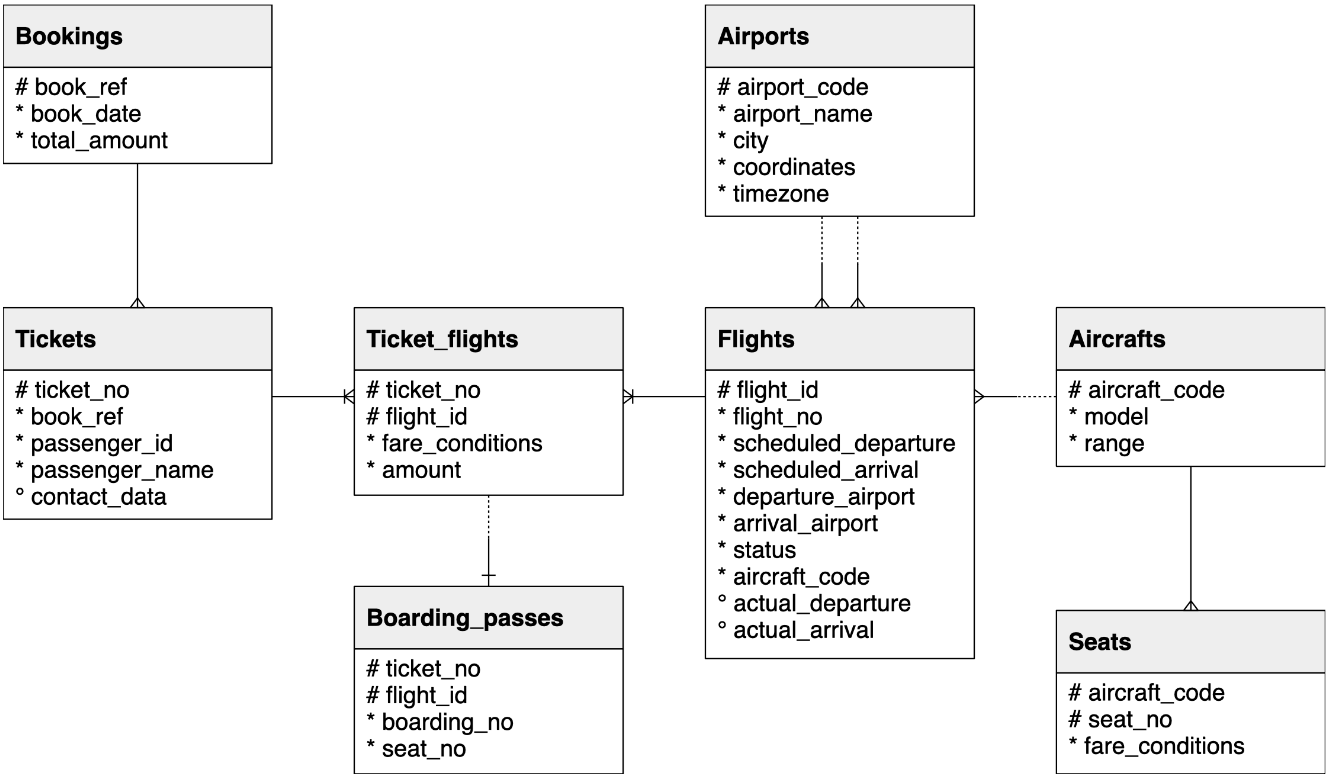
Software used in this Lab

In this lab, you will be using PostgreSQL. It is a popular open-source object Relational Database Management System (RDBMS) capable of performing a wealth of database administration tasks, such as storing, manipulating, retrieving, and archiving data.

To complete this lab, you will be accessing the PostgreSQL service through the IBM Skills Network (SN) Cloud IDE, which is a virtual development environnement you will utilize throughout this course.

Database used in this Lab

In this lab, you will use a database from https://postgrespro.com/education/demodb distributed under the PostgreSQL licence. It stores a month of data about airline flights in Russia and is organized according to the following schema:



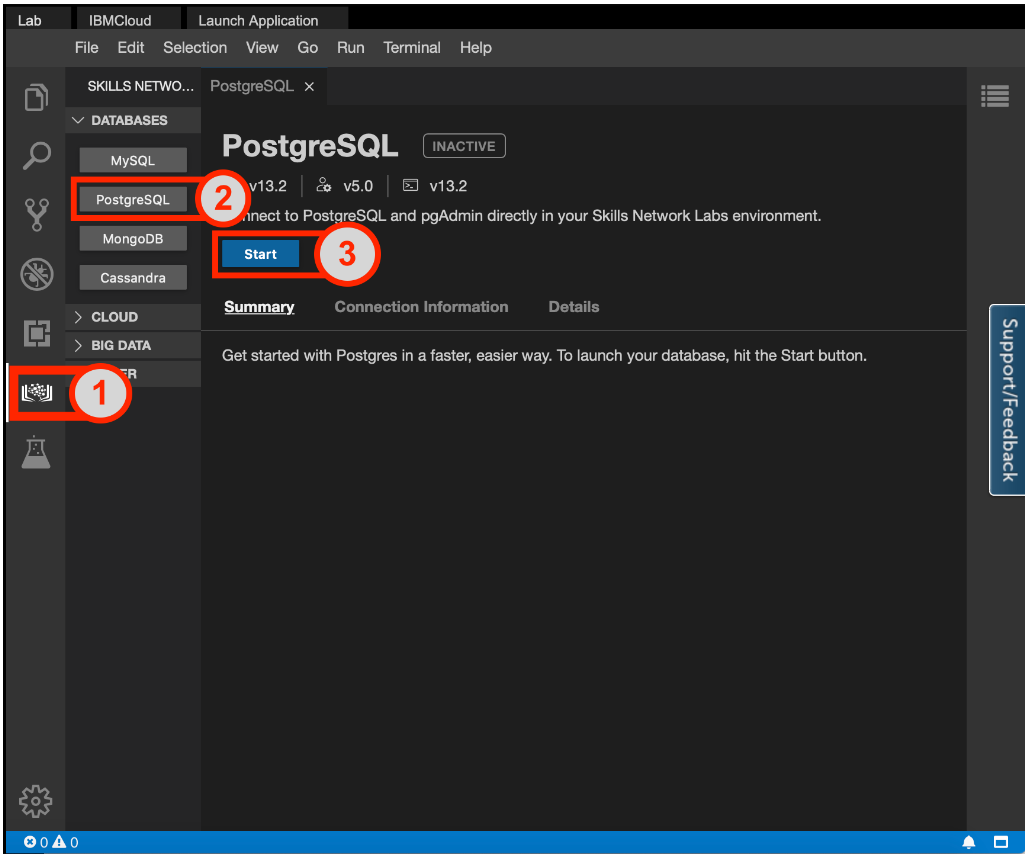
Launching PostgreSQL in Cloud IDE

To get started with this lab, launch PostgreSQL using the Cloud IDE. You can do this by following these steps:

1. Click on the **Skills Network extension** button on the left side of the window.

2. Open the **DATABASES** drop-down menu and click on **PostgreSQL**.

3. Click on the **Start** button. PostgreSQL may take a few moments to start.



PostgreSQL INACTIVE

v13.2 | v5.0 | v13.2

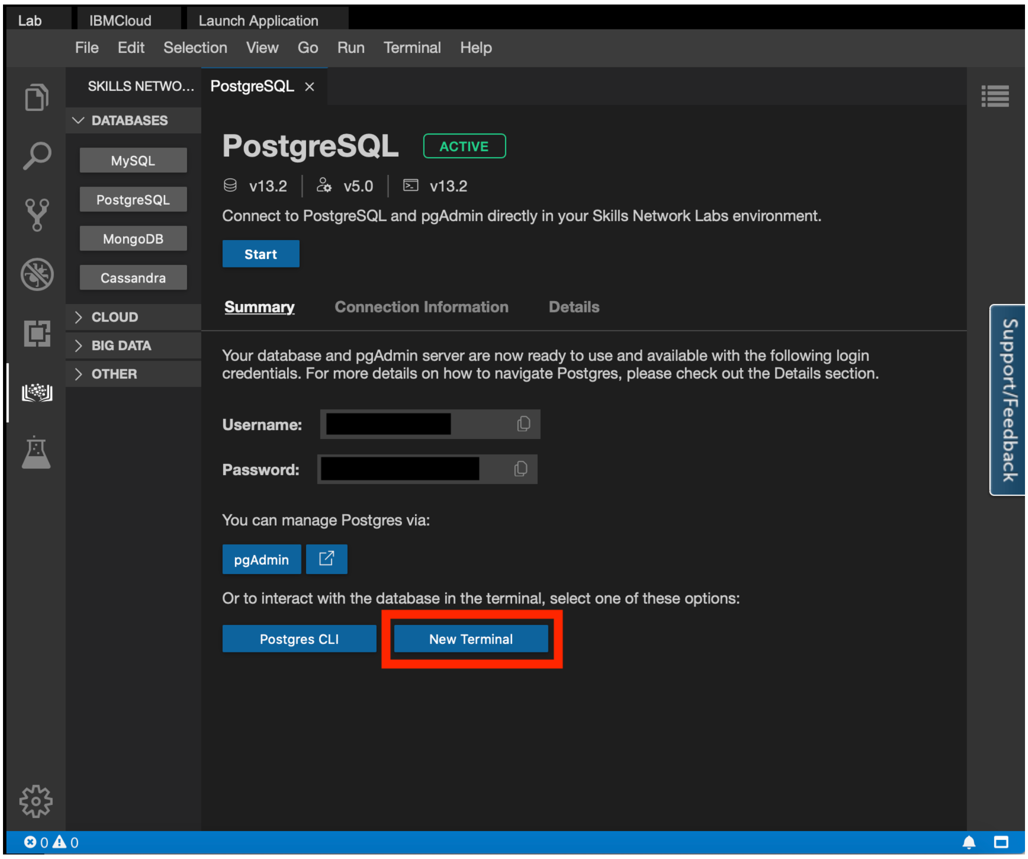
nnect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Get started with Postgres in a faster, easier way. To launch your database, hit the Start button.

Downloading and Creating the Database

First, we will need to download the database.

1. Open a new terminal by clicking on the **New Terminal** button near the bottom of the interface.



PostgreSQL ACTIVE

v13.2 | v5.0 | v13.2

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate Postgres, please check out the Details section.

Username: [redacted]

Password: [redacted]

You can manage Postgres via:

Or to interact with the database in the terminal, select one of these options:

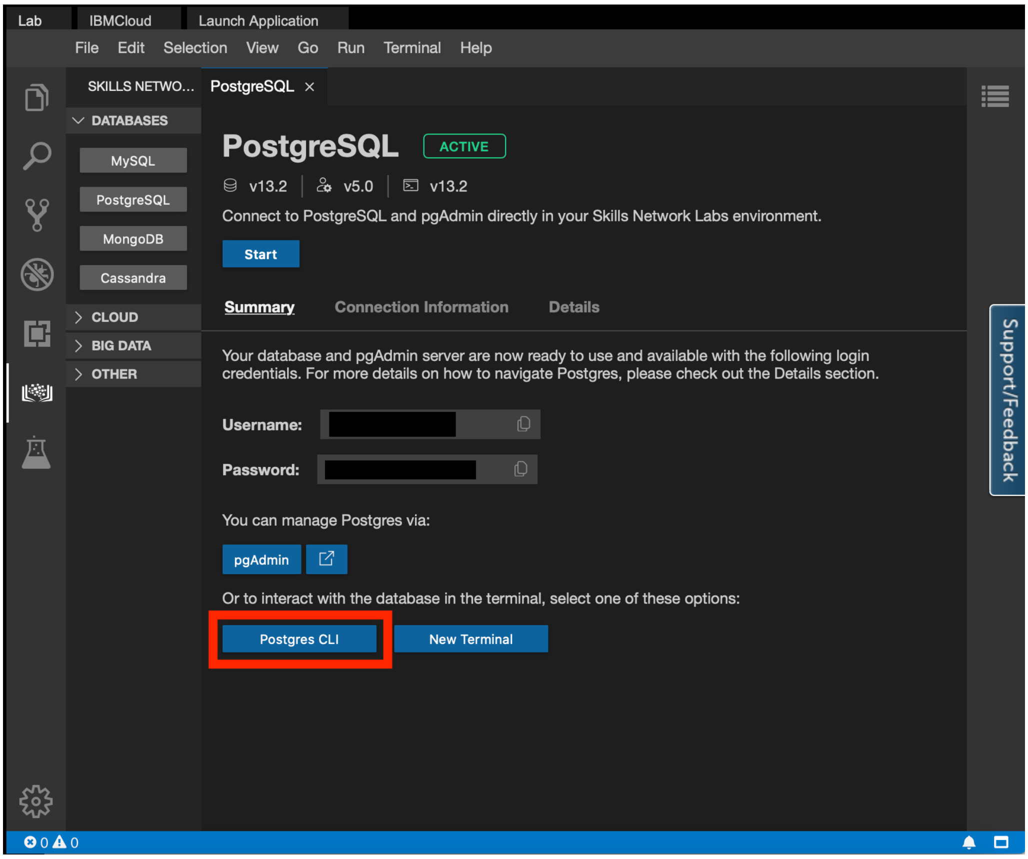
2. Run the following command in the terminal.

```
1. 1
1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/example-guided-project/flights_RUSSIA_small.sql
```

Copied!

The file which you downloaded is a full database backup of a month of flight data in Russia. Now, you can perform a full restoration of the dataset by first opening the PostgreSQL CLI.

3. Near the bottom of the window, click on the **PostgreSQL CLI** button to launch the Command Line Interface.



PostgreSQL

Connect to PostgreSQL and pgAdmin directly in your Skills Network Labs environment.

Your database and pgAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate Postgres, please check out the Details section.

Username: [redacted]

Password: [redacted]

You can manage Postgres via:

Or to interact with the database in the terminal, select one of these options:

4. In the PostgreSQL CLI, type in the command `\i <file_name>`. In your case, the filename will be the name of the file you downloaded, `flights_RUSSIA_small.sql`. This will restore the data into a new database called `demo`.

```
1. 1
1. \i flights_RUSSIA_small.sql
```

Copied!

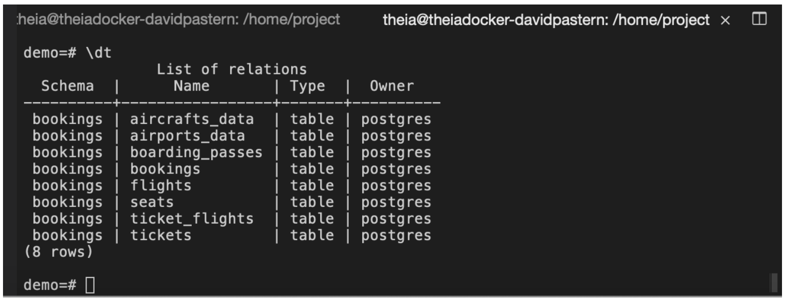
The restorations may take a few moments to complete.

5. Verify that the database was properly created by entering the following command:

```
1. 1
1. \dt
```

Copied!

You should see the following output showing all the tables that are part of the `bookings` schema in the `demo` database.


theia@theiadocker-davidpastern: /home/project
theia@theiadocker-davidpastern: /home/project ×

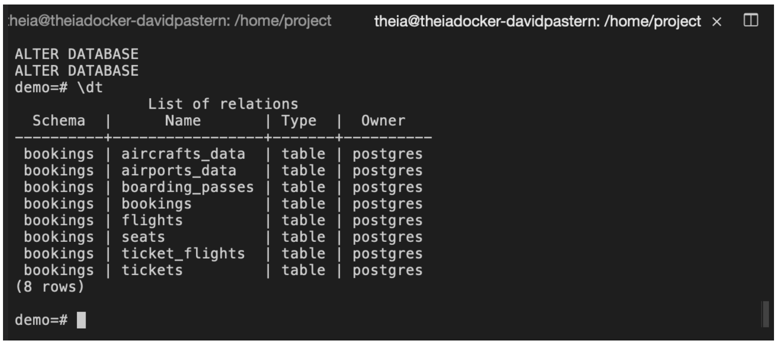
```
demo=# \dt
              List of relations
  Schema  |      Name       | Type  |  Owner
----------+-----------------+-------+----------
 bookings | aircrafts_data  | table | postgres
 bookings | airports_data   | table | postgres
 bookings | boarding_passes | table | postgres
 bookings | bookings        | table | postgres
 bookings | flights         | table | postgres
 bookings | seats           | table | postgres
 bookings | ticket_flights  | table | postgres
 bookings | tickets         | table | postgres
(8 rows)

demo=# 
```

Exercise 1: Create New Roles and Grant them Relevant Privileges

In PostgreSQL, users, groups, and roles are all the same entity, with the difference being that users can log in by default.

In this exercise, you will create two new roles: `read_only` and `read_write`, then grant them the relevant privileges.

To begin, ensure that you have the PostgreSQL Command Line Interface open and connected to the `demo` database, as such:


theia@theiadocker-davidpastern: /home/project
theia@theiadocker-davidpastern: /home/project ×

```
ALTER DATABASE
ALTER DATABASE
demo=# \dt
              List of relations
  Schema  |      Name       | Type  |  Owner
----------+-----------------+-------+----------
 bookings | aircrafts_data  | table | postgres
 bookings | airports_data   | table | postgres
 bookings | boarding_passes | table | postgres
 bookings | bookings        | table | postgres
 bookings | flights         | table | postgres
 bookings | seats           | table | postgres
 bookings | ticket_flights  | table | postgres
 bookings | tickets         | table | postgres
(8 rows)

demo=# █
```

Task A: Create a `read_only` role and grant it privileges

1. To create a new role named `read_only`, enter the following command into the CLI:

```
1. 1
1. CREATE ROLE read_only;
```

Copied!

2. First, this role needs the privilege to connect to the `demo` database itself. To grant this privilege, enter the following command into the CLI:

```
1. 1
1. GRANT CONNECT ON DATABASE demo TO read_only;
```

Copied!

3. Next, the role needs to be able to use the schema in use in this database. In our example, this is the `bookings` schema. Grant the privilege for the `read_only` role to use the schema by entering the following:

```
1. 1
1. GRANT USAGE ON SCHEMA bookings TO read_only;
```

Copied!

4. To access the information in tables in a database, the `SELECT` command is used. For the `read_only` role, we want it to be able to access the contents of the database but not to edit or alter it. So for this role, only the `SELECT` privilege is needed. To grant this privilege, enter the following command:

```
1. 1
1. GRANT SELECT ON ALL TABLES IN SCHEMA bookings TO read_only;
```

Copied!

This allows the `read_only` role to execute the `SELECT` command on **all** tables in the bookings schema.

Task B: Create a `read_write` role and grant it privileges

1. Similarly, create a new role called `read_write` with the following command in the PostgreSQL CLI:

   ```
   CREATE ROLE read_write;
   ```
   Copied!

2. As in Task A, this role should first be given the privileges to connect to the `demo` database. Grant this privilege by entering the following command:

   ```
   GRANT CONNECT ON DATABASE demo TO read_write;
   ```
   Copied!

3. Give the role the privileges to use the `bookings` schema that is used in the `demo` database with the following:

   ```
   GRANT USAGE ON SCHEMA bookings TO read_write;
   ```
   Copied!

4. So far the commands for the `read_write` role have been essentially the same as for the `read_only` role. However, the `read_write` role should have the privileges to not only access the contents of the database, but also to create, delete, and modify entries. The corresponding commands for these actions are `SELECT`, `INSERT`, `DELETE`, and `UPDATE`, respectively. Grant this role these privileges by entering the following command into the CLI:

   ```
   GRANT SELECT, INSERT, DELETE, UPDATE ON ALL TABLES IN SCHEMA bookings TO read_write;
   ```
   Copied!

Exercise 2: Add a New User and Assign them a Relevant Role

In this exercise, you will create a new user for the database and assign them the one of the roles you created in Exercise 1. This method streamlines the process of adding new users to the database since you don't have to go through the process of granting custom privileges to each one. Instead, you can assign them a role and the user inherits the privileges of that role.

Suppose you wish to add a new user, `user_a`, for use by an information and help desk at an airport. In this case, assume that there is no need for this user to modify the contents of the database. As you may have guessed, the appropriate role to assign is the `read_only` role.

1. To create a new user named `user_a`, enter the following command into the PostgreSQL CLI:

   ```
   CREATE USER user_a WITH PASSWORD 'user_a_password';
   ```
   Copied!

   In practice, you would enter a secure password in place of 'user_a_password', which will be used to access the database through this user.

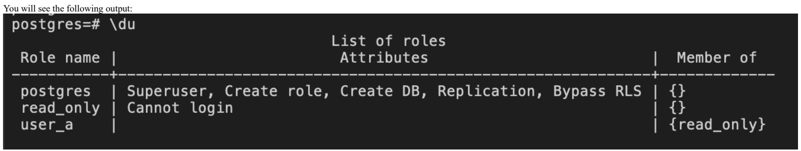
2. Next, assign `user_a` the `read_only` role by executing the following command in the CLI:

   ```
   GRANT read_only TO user_a;
   ```
   Copied!

3. You can list all the roles and users by typing the following command:

   ```
   \du
   ```
   Copied!

   You will see the following output:

   ```
   postgres=# \du
                                      List of roles
    Role name |                         Attributes                         |  Member of
   -----------+------------------------------------------------------------+-------------
    postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
    read_only | Cannot login                                               | {}
    user_a    |                                                            | {read_only}
   ```

   Notice that `user_a` was successfully created and that it is a member of `read_only`.

Exercise 3: Revoke and Deny Access

In this exercise, you will learn how to revoke a user's privilege to access specific tables in a database.

Suppose there is no need for the information and help desk at the airport to access information stored in the `aircrafts_data` table. In this exercise, you will revoke the `SELECT` privilege on the `aircrafts_data` table in the `demo` database from `user_a`.

1. You can use the `REVOKE` command in the Command Line Interface to remove specific privileges from a role or user in PostgreSQL. Enter the following command into the PostgreSQL CLI to remove the privileges to access the `aircrafts_data` table from `user_a`:

   ```
   REVOKE SELECT ON aircrafts_data FROM user_a;
   ```
   Copied!

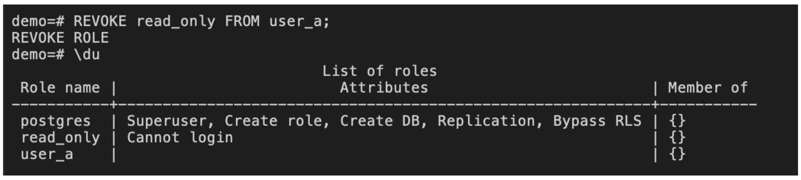
2. Now suppose `user_a` is transferred departments within the airport and no longer needs to be able to access the `demo` database at all. You can remove all their `SELECT` privileges by simply revoking the `read_only` role you assigned to them earlier. You can do this by entering the following command in the CLI:

   ```
   REVOKE read_only FROM user_a;
   ```
   Copied!

3. Now you can check all the users and their roles again to see that the `read_only` role was successfully revoked from `user_a` by entering the following command again:

   ```
   \du
   ```
   Copied!

   You will see the following output:

   ```
   demo=# REVOKE read_only FROM user_a;
   REVOKE ROLE
   demo=# \du
                                     List of roles
    Role name |                         Attributes                         | Member of
   -----------+------------------------------------------------------------+-----------
    postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
    read_only | Cannot login                                               | {}
    user_a    |                                                            | {}
   ```

   Notice that `user_a` is still present but it is no longer a member of the `read_only` role.

Practice Exercise

Now it's time to implement some of what you learned! In this practice exercise, you will use what you learned in the previous exercises to create a new user and assign them a relevant role.

Scenario: *Suppose there is a new employee at the airline in which you are the database administrator for. They interact directly with clients to create new bookings for flights. As such, they will need to not only access the information in the database, but also to create new bookings.*

To complete this exercise, create a new user called `user_b` and grant it the privileges to both read and write to the `demo` database.

▸ Hint (Click Here)

▾ Solution (Click Here)

1. First, you can create a new user using the following command:

   ```
   CREATE USER user_b WITH PASSWORD 'user_b_password';
   ```
   Copied!

2. Next, you can grant the user you just created the `read_write` role by entering the following command into the CLI:

   ```
   GRANT read_write TO user_b;
   ```
   Copied!

Conclusion

Congratulations on completing this lab on user management and access control in PostgreSQL. You now have some foundational knowledge on how to create new roles for your database, add new users, and assign those users relevant roles. In addition, you also have the capability to revoke privileges from users in the database.

Author(s)

David Pasternak

Other Contributor(s)

Sandip Saha Joy, Rav Ahuja

Changelog

Date	Version	Changed by	Change Description
2021-07-14	0.1	David Pasternak	Initial version created
2021-10-8	0.2	Steve Hord	Copy Edit
2022-07-27	0.3	Lakshmi Holla	Updated HTML tag
2023-05-05	0.4	Jaskomal Natt	Updated copyright date and deleted empty cell