# Setting up logins and users

## Adding a superuser login

1. Log in to CQL shell (cqlsh) with the Cassandra super user default user:

cqlsh -u cassandra -p cassandra

1. Create a new superuser account with password stored in the CQL database:

**CREATE** **ROLE** ***root\_user*** **with** **SUPERUSER** = true **AND** **LOGIN** = true **and** **PASSWORD** = '***root***';

**T: (\*1)**

1. Exit cqlsh:

**EXIT**;

1. In order to disable or drop the cassandra role, log in with the new role created in the previous step:

cqlsh -u ***root\_user***

Enter the password at the prompt.

1. Verify that the role was created as a superuser using [LIST ROLES](https://docs.datastax.com/en/dse/6.0/cql/cql/cql_reference/cql_commands/cqlListRoles.html):

**LIST** **ROLES**;

role | **super** | login | options

---------------------+-------+-------+---------

root\_user | True | True | {}

cassandra | True | True | {}

(2 rows)

1. Drop or update the cassandra account:
   * Drop the cassandra account:

**DROP** **ROLE** cassandra;

* + Update the cassandra role by disabling superuser and changing the password:

**ALTER** **ROLE** cassandra **WITH** **SUPERUSER** = false **AND** **LOGIN** = false **AND** **password**='new\_secret\_pw';

## Adding database users:

1. Create a role with login enabled and an internally stored password:

**CREATE** **ROLE** ***role\_name***

**WITH** **LOGIN** = true

**AND** **PASSWORD** = '***password\_string***';

where

* + *role\_name* - The user name for authentication. Enclose the role names that include uppercase or special characters in double quotes.
  + LOGIN = true - Allows the role to access the database.
  + PASSWORD = '***default\_password***' - Stored internally for database managed accounts.
  + (Optional) superuser = true - Gives full access to all database objects to the user.

1. To allow the role to be used for authentication when [scheme\_permissions](https://docs.datastax.com/en/dse/6.0/dse-admin/datastax_enterprise/config/configDseYaml.html" \l "configDseYaml__scheme_permissions" \t "_blank) is true, bind the role to an authentication scheme:

**GRANT** **EXECUTE**

**ON** INTERNAL SCHEME

**TO** ***role\_name***;

1. To allow another role to manage the new role:

**GRANT** **AUTHORIZE** **FOR** **ALTER**, **DROP**

**ON** ***new\_role\_name***

**TO** ***management\_role***;

**Tip:** All superusers have authorize permissions on all roles. And the role that created the role is granted all permissions on the role.

1. Each user can change their own password with the [ALTER ROLE](https://docs.datastax.com/en/dse/6.0/cql/cql/cql_reference/cql_commands/cqlAlterRole.html) command.
   * User logs in with their role name:

cqlsh -u ***role\_name*** -p ***default\_password***

* + Changes the password:

**ALTER** **ROLE** ***role\_name***

**WITH** **password** = '***newpassword***';

# Troubleshooting:

1. When new user or role with super user access is not getting created

Change

authenticator: AllowAllAuthenticator

To

authenticator: PasswordAuthenticator

in cassandra.yamlconfiguration file and restart Cassandra.

CREATE USER root\_user WITH PASSWORD 'root' SUPERUSER;

cassandra@cqlsh:system\_auth> desc tables;

resource\_role\_permissons\_index role\_permissions role\_members roles

cassandra@cqlsh:system\_auth> select \* from roles;

role | can\_login | is\_superuser | member\_of | salted\_hash

-----------+-----------+--------------+-----------+--------------------------------------------------------------

cassandra | True | True | null | $2a$10$A0a4ZneMGj4sqCLQGsNWwe4FBsMsg8h54i3PclVRM1LnR3fKEzfgi

root\_user | True | True | null | $2a$10$uf/bGcPu/IgzVy7lj02qyO1cKkKgmo455Pc6bVVRDqgY8Hozjm1FC

1. ServerError: java.lang.UnsupportedOperationException: GRANT operation is not supported by AllowAllAuthorizer Cassandra

authorizer: CassandraAuthorizer

# [Role Based Access Control (RBAC)](http://en.wikipedia.org/wiki/Role-based_access_control).

Under this new scheme, permissions are granted to a role just as they were previously granted to a user, the key difference is that roles can also be granted to each other. So in this context we can think of them as groups, rather than individuals. This greatly simplifies permissions management for administrators by allowing related privileges to be bundled together by granting them to roles, which can in turn then be assigned to specific database users. Some new constructs have been added to the CQL syntax to support this. For example, a simple scenario looks something like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | CREATE KEYSPACE warehouse WITH REPLICATION = {'class':'SimpleStrategy', 'replication\_factor':1};  USE warehouse;  CREATE TABLE addresses (  customer\_id bigint,  address\_id int,  address text,  PRIMARY KEY (customer\_id, address\_id)  );  CREATE TABLE orders (  customer\_id bigint,  order\_id timeuuid,  product\_id uuid,  product\_description text,  PRIMARY KEY (customer\_id, order\_id, product\_id)  );  CREATE ROLE supervisor;  GRANT MODIFY ON warehouse.orders TO supervisor;  GRANT SELECT ON warehouse.addresses TO supervisor; |

So now we have a Role, supervisor, with the necessary permissions to read and write from the two tables. When we have a new database user that we want to be able to act as a supervisor, we just grant them that Role.

|  |  |
| --- | --- |
| 1  2 | CREATE ROLE pam WITH PASSWORD = 'password' AND LOGIN = true;  GRANT supervisor TO pam; |

Let's examine those last two statements. The first creates another role, named pam and sets its LOGIN attribute to true. As you might expect, this is what enables a database user to actually identify as this role when logging in via a client such as cqlsh. We also assigned Pam a password as we're using Cassandra's internal password authentication mechanism. There's actually one other attribute we could set when creating a new role. We specify superuser status at the role level, which we would do by adding AND SUPERUSER = true to the CREATE ROLE statement. Finally, note that anything that can be set in CREATE ROLE can be modified later using ALTER ROLE (so we could retrospectively make Pam a superuser if we choose). Pam now is permitted to do all the things a supervisor can do:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | LIST ALL PERMISSIONS OF pam;     role       | username   | resource                    | permission  ------------+------------+-----------------------------+------------   supervisor | supervisor | <table warehouse.addresses> |     SELECT   supervisor | supervisor |    <table warehouse.orders> |     MODIFY |

(the username column is simply to provide backward compatibility with the results of LIST PERMISSIONS in previous releases).

If we were to add a new table to which supervisors require access, we would simply grant the necessary permissions on it to the supervisor role and Pam, along with all other users assigned the role, would automatically acquire them.

We can go further though, let's create another role and grant it some permissions all of the tables in another keyspace. Then, we'll assign our new role to Pam.

|  |  |
| --- | --- |
| 1  2  3  4 | CREATE ROLE office\_admin;  GRANT SELECT ON KEYSPACE office TO office\_admin;  GRANT MODIFY ON KEYSPACE office TO office\_admin;  GRANT office\_admin TO pam; |

And if we list Pam's permissions, we'll see they represent the aggregate of those granted to her roles.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | LIST ALL PERMISSIONS OF pam;     role         | username     | resource                    | permission  --------------+--------------+-----------------------------+------------   office\_admin | office\_admin |           <keyspace office> |     SELECT   office\_admin | office\_admin |           <keyspace office> |     MODIFY     supervisor |   supervisor | <table warehouse.addresses> |     SELECT     supervisor |   supervisor |    <table warehouse.orders> |     MODIFY |

Likewise, we can ask which roles Pam has been assigned.

|  |  |
| --- | --- |
| 1  2  3  4  5 | LIST ROLES OF pam;     role         | super | login | options  --------------+-------+-------+---------   office\_admin | False | False |        {}            pam | False |  True |        {}     supervisor | False | False |        {} |

# References:

<https://www.datastax.com/blog/2015/03/role-based-access-control-cassandra>

<https://docs.datastax.com/en/security/6.0/security/Auth/secCreateRootAccount.html>