### Accessing PostgreSQL Database in a Docker Container: A Step-by-Step Guide

Here’s a consolidated process for accessing a PostgreSQL database running inside a Docker container, including how to gather the necessary details (like environment variables) and successfully log in.

#### ****Step 1: Check Running Containers****

To begin, you need to find the name or ID of the running PostgreSQL container. Run the following command:

docker ps

This will show all running containers. For example:

CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES

Ba91f7430001 | docker\_test-web | "bash -c './wait-for…" | 2 hours ago | Up 2 hours | 0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp | docker\_test-web-1

Aa91f7430000 | postgres:13 | "docker-entrypoint.s…" | 2 hours ago | Up 2 hours | 5432/tcp | docker\_test-db-1

Here, the container name is docker\_test-db-1.

#### ****Step 2: Retrieve Container's Environment Variables****

If you don’t already know the credentials (e.g., POSTGRES\_USER, POSTGRES\_DB), you can retrieve the environment variables that were set when the container was created. This will allow you to find the values of these variables, which are crucial for logging in.

To check the environment variables inside the running container:

docker exec -it docker\_test-db-1 printenv

Sample output:

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/postgresql/13/bin

HOSTNAME=Aa91f7430000

TERM=xterm

POSTGRES\_PASSWORD=svc@1234

POSTGRES\_DB=docker\_tester\_login

POSTGRES\_USER=MyPostgres

GOSU\_VERSION=1.17

LANG=en\_US.utf8

PG\_MAJOR=13

PG\_VERSION=13.20-1.pgdg120+1

PGDATA=/var/lib/postgresql/data

HOME=/root

Here are the important variables:

* POSTGRES\_USER: The username to connect to PostgreSQL (in this case, MyPostgres).
* POSTGRES\_DB: The database name (in this case, docker\_tester\_login).
* POSTGRES\_PASSWORD: The password for the POSTGRES\_USER (in this case, svc@1234).

#### ****Step 3: Connect to PostgreSQL Using**** psql

Now that you have the necessary details, you can access the PostgreSQL database using the psql command-line tool.

To log in, run the following command from your local terminal:

docker exec -it docker\_test-db-1 psql -U MyPostgres -d docker\_tester\_login

* docker exec -it docker\_test-db-1: This runs the command inside the running container (docker\_test-db-1).
* psql -U MyPostgres -d docker\_tester\_login: This tells PostgreSQL to log in as user MyPostgres and connect to the database docker\_tester\_login.

#### ****Step 4: Execute SQL Commands****

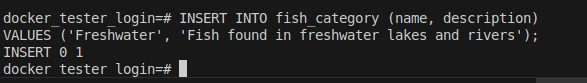
#### ****Insert Data into the**** Category ****Table****

To insert a new category, use the INSERT INTO SQL command. In this example, we'll insert a "Freshwater" category.

INSERT INTO yourapp\_category (name, description)

VALUES ('Freshwater', 'Fish found in freshwater lakes and rivers');

Note:

* Replace yourapp\_category with the actual table name in your database (Django will typically create tables using the lowercase model name, i.e., category).
* If your Category model includes an image field, it is handled via Django's ImageField, but if inserting manually, you'd typically need to specify a file path for images, which can be tricky via psql. For simplicity, we won’t include the image field in this example.

#### ****Insert Data into the**** Fish ****Table****

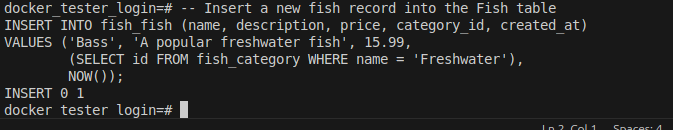
After inserting a category, insert a fish record and associate it with the "Freshwater" category. To do this, use the following SQL command:

-- Insert a new fish into the Fish table, associated with the Freshwater category

INSERT INTO yourapp\_fish (name, description, price, category\_id)

VALUES ('Bass', 'A popular freshwater fish', 15.99,

(SELECT id FROM yourapp\_category WHERE name = 'Freshwater'));



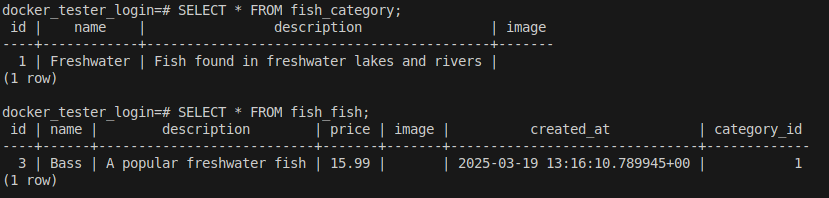
#### ****Verify the Insertions****

**Once inside the PostgreSQL interactive shell (**psql**), you can run SQL commands. For example:**

SELECT \* FROM your\_table;

Once you've run the INSERT commands, you can check if the data was inserted successfully.

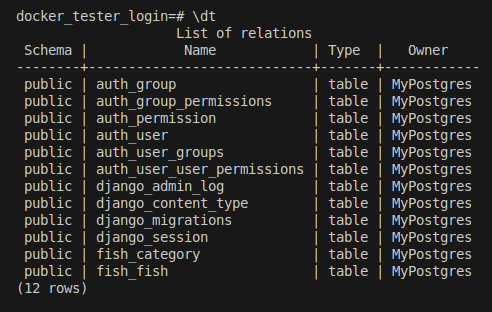
To see the data in the Category and fish table:

**To exit the psql shell:**

\q

**To list available tables:**

\dt



**To exit the psql shell:**

\q

### ****Key Points to Remember****

* **Container Name:** Use docker ps to find the name or ID of your running PostgreSQL container.
* **Environment Variables:** Use docker exec -it <container\_name> printenv to check the PostgreSQL username, database, and password.
* **Logging In:** Use the correct psql command with the user and database details to log into the container and interact with the database.

**POSTGRESQL COMMAND**

**1. Database Management**

* **Create a database:**

**CREATE DATABASE database\_name;**

* **List all databases:**

**\l**

* **Connect to a database:**

**\c database\_name**

* **Delete (drop) a database:**

**DROP DATABASE database\_name;**

### ****2. Table Management****

* **Create a table:**

**CREATE TABLE table\_name (**

**column\_name1 column\_type [constraints],**

**column\_name2 column\_type [constraints],**

**...**

**);**

* **List all tables:**

**\dt**

* **Describe a table (show structure):**

**\d table\_name**

* **Delete (drop) a table:**

**DROP TABLE table\_name;**

**3. Inserting Data**

* **Insert a single row:**

**INSERT INTO table\_name (column1, column2, column3, ...)**

**VALUES (value1, value2, value3, ...);**

* **Insert multiple rows:**

**INSERT INTO employees (name, age, department)**

**VALUES**

**('Jane Smith', 28, 'Marketing'),**

**('Tom Lee', 35, 'Sales');**

### ****4. Querying Data (SELECT)****

* **Select all columns from a table:**

**SELECT \* FROM table\_name;**

* **Select specific columns:**

**SELECT column1, column2 FROM table\_name;**

* **Filter results with** WHERE **clause:**

**SELECT \* FROM table\_name WHERE column\_name = value;**

* **Order results:**

**SELECT \* FROM table\_name ORDER BY column\_name [ASC|DESC];**

* **Limit the number of rows returned:**

**SELECT \* FROM table\_name LIMIT number;**

5. Updating Data

* **Update an existing row:**

**UPDATE table\_name**

**SET column1 = value1, column2 = value2**

**WHERE condition;**

**6. Deleting Data**

* **Delete a single row:**

**DELETE FROM table\_name WHERE condition;**

* **Delete all rows in a table:**

**DELETE FROM table\_name;**

**7. Table Constraints and Modifications**

* **Add a column to an existing table:**

**ALTER TABLE table\_name**

**ADD COLUMN column\_name column\_type;**

* **Change a column data type:**

**ALTER TABLE table\_name**

**ALTER COLUMN column\_name TYPE new\_data\_type;**

* **Rename a column:**

**ALTER TABLE table\_name**

**RENAME COLUMN old\_column\_name TO new\_column\_name;**

* **Drop a column:**

**ALTER TABLE employees**

**DROP COLUMN salary;**

### ****8. Join Tables****

* **Inner Join**: Combines rows from two tables based on a related column.

SELECT \*

FROM table1

INNER JOIN table2

ON table1.common\_column = table2.common\_column;

* **Left Join**: Returns all rows from the left table, even if there is no match in the right table.

SELECT \*

FROM table1

LEFT JOIN table2

ON table1.common\_column = table2.common\_column;

9. Aggregate Functions

* **Count rows:**

SELECT COUNT(\*) FROM table\_name;

* **Find the maximum value:**

SELECT MAX(column\_name) FROM table\_name;

* **Find the minimum value:**

**SELECT MIN(column\_name) FROM table\_name;**

* **Average of a column:**

**SELECT AVG(column\_name) FROM table\_name;**

****10. Create Index****

* **Create an index on a column for faster searching:**

**CREATE INDEX index\_name**

**ON table\_name (column\_name);**

**eg:**

**CREATE INDEX idx\_employee\_name**

**ON employees (name);**

**11. Transactions**

* **Start a transaction:**

**BEGIN;**

* **Commit a transaction:**

**COMMIT;**

* **Rollback a transaction:**

**ROLLBACK;**

**12. Other Useful Commands**

* **List all schemas:**

**\dn**

* **List all users:**

**\du**

* **Exit** psql**:**

**\q**