***Azure SQL Database*** is a fully managed relational database service provided by Microsoft on the Azure cloud platform. It's designed for SQL Server databases and offers scalability, high availability, security, and a range of deployment and configuration options to meet various needs.

***Problem faced:-***

***1.While loging to query editor error come:-***

The error you're seeing occurs when **Deny Public Network Access** is set to **Yes** on your Azure SQL server, meaning that it only accepts connections through **Private Endpoint** from within your Virtual Network (VNet). Here’s how to resolve this issue:

In environment install dependency

source venv/bin/activate

***2.This suggests that Django is trying to insert a NULL value into a column (name) in the django\_content\_type table, which does not allow NULL values.***

To fix this:

* Check if your migrations are up to date.
* You may need to manually ensure that required fields are populated or investigate which migrations failed to apply correctly.

***3. Broken Package Dependencies:***

* **Problem**: Encountered dpkg errors, indicating broken dependencies during the installation of msodbcsql17.
* **Reason**: Previous installation attempts left partial or broken packages.
* **Solution**: Resolved using sudo apt --fix-broken install.

***4. Pre-Installation Script Failure:***

* **Problem**: Installation failed, ODBC Driver 17 for SQL Server detected! when trying to install msodbcsql17.
* **Reason**: A conflicting or incomplete installation of an older driver version was detected.
* **Solution**: Removed existing ODBC driver and ensured no conflicting files were present.

**1. Set Up Azure SQL Database**

**Step 1.1: Create an Azure SQL Database**

1. **Log in to Azure Portal**: [Azure Portal](https://portal.azure.com).
2. Navigate to **Azure SQL** -> **Create SQL Database**.
3. Fill in the details:
   * **Resource Group**: Select or create a new one.
   * **Database Name**: Provide your database name.
   * **Server**: Click **Create new** and:
     + Provide a server name.
     + Set the admin username and password.
     + Choose a region.
   * **Pricing Tier**: Select a suitable performance tier (e.g., Basic for testing).
4. Click **Review + Create** and **Create**.

**Step 1.2: Configure Server Firewall**

1. Go to the **SQL Server** created.
2. Navigate to **Firewall settings**.
3. Add your **client IP** to the list of allowed IP addresses.
4. Save the changes.

**2. Install Required Packages**

**Step 2.1: Django Project Requirements**

Install necessary Python packages in your virtual environment:

bash

pip install django

pip install pyodbc

pip install django-pyodbc-azure

**1. Update Django Settings for ODBC Driver 18**

Make sure your settings.py file looks like this:

DATABASES = {

'default': {

'ENGINE': 'sql\_server.pyodbc',

'NAME': os.getenv('DB\_NAME', 'Mydatabase'), # Replace with your database name

'USER': os.getenv('DB\_USER', 'sysadmin'), # Replace with your DB username

'PASSWORD': os.getenv('DB\_PASSWORD', 'Himanshu@2001'), # Replace with your DB password

'HOST': os.getenv('DB\_HOST', 'mysqlserve1.database.windows.net'), # Your Azure SQL Server

'PORT': os.getenv('DB\_PORT', '1433'),

'OPTIONS': {

'driver': 'ODBC Driver 18 for SQL Server', # Correct ODBC Driver version

},

}

}

**2. Set Environment Variables**

You can use environment variables to securely store and access your database credentials. Here's how to set them on your **Azure VM**:

**Step 2.1: Set Environment Variables in Your Bash Profile**

1. Open your bash profile (or .bashrc if you use that):

bash

nano ~/.bashrc

1. Add the following lines at the end of the file to export your environment variables:

bash

export DB\_NAME="Mydatabase"

export DB\_USER="sysadmin"

export DB\_PASSWORD="Himanshu@2001"

export DB\_HOST="mysqlserve1.database.windows.net"

export DB\_PORT="1433"

1. Save the file and exit (press Ctrl + X, then Y, then Enter).

**Step 2.2: Apply the Changes**

To make the environment variables active, run:

source ~/.bashrc

This will set the environment variables, and you can use os.getenv() in your Django settings to access them.

**3. Install ODBC Driver 18**

Ensure you have the **ODBC Driver 18** installed on your Azure VM. If you haven’t done it already, use the following commands based on your Linux distribution:

**For Ubuntu:**

bash

sudo su

curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -

curl https://packages.microsoft.com/config/ubuntu/$(lsb\_release -rs)/prod.list > /etc/apt/sources.list.d/mssql-release.list

exit

sudo apt-get update

sudo ACCEPT\_EULA=Y apt-get install -y msodbcsql18 unixodbc-dev

**4. Install Python Dependencies**

If you haven't installed the necessary Python dependencies yet, do it now:

bash

pip install django pyodbc django-pyodbc-azure

**5. Verify Database Connection**

After configuring your settings and ensuring the environment variables are set, check if Django can connect to the Azure SQL Database.

**Step 5.1: Make Migrations**

bash

python manage.py makemigrations

**Step 5.2: Apply Migrations**

bash

python manage.py migrate

If migrations complete without any issues, your Django app should now be properly connected to your Azure SQL Database using **ODBC Driver 18**.

***Steps to Apply:***

**Deamon service:**

[Unit]

Description=Django Application Service

After=network.target

[Service]

User=sysadmin

Group=sysadmin

WorkingDirectory=/new\_chatapp/fundoo

ExecStart=/new\_chatapp/venv/bin/gunicorn --bind 0.0.0.0:8000 --timeout 120 fundoo.wsgi:application

Restart=always

# Specify environment variables directly

Environment="DB\_NAME=Mydatabase"

Environment="DB\_USER=sysadmin"

Environment="DB\_PASSWORD=Himanshu@2001"

Environment="DB\_HOST=mysqlserve1.database.windows.net"

Environment="DB\_PORT=1433"

Environment="DB\_DRIVER=ODBC Driver 18 for SQL Server"

[Install]

WantedBy=multi-user.target

1. **Reload systemd**: After you’ve made the changes to the systemd service file, you need to reload the systemd configuration to apply those changes.

bash

sudo systemctl daemon-reload

2. **Restart your Django service**: Once the service file is reloaded, restart your service:

bash

sudo systemctl restart django\_app.service

3. **Check the status of the service**: To confirm if the service has started correctly, check the service status:

bash

sudo systemctl status django\_app.service

This will show you any potential errors related to the service start.

  
