Django 适配 GaussDB 指导(Python)

V2.0

华为技术有限公司

目录

一、 介绍

该练习是用 Python 开发语言，用 Python 的 ROM 组件 Django，连接 GaussDB 数据库

全过程的操作演示，针对典型的 Python 应用中连接 GaussDB 数据源，操作 GaussDB 数据库的适配，改造过程总结说明。案例中 Demo 项目采用的是 Python3 版本作为开发语言，以下是 Django 连接 GaussDB 的步骤总结。

Django 连接 GaussDB 全过程：

- 环境准备

环境准备：准备开发环境和数据源，构建代码编译运行环境和数据库实例。

- 连接&操作数据库

1） 添加 psycopg2 驱动

2） 安装 Django 组件

3） 创建 Django 项目和应用

4） 初始化 GaussDB 数据库和用户

5） 配置数据库连接

- 验证

运行 Django 项目，在对应的网页查看 Django 页面，并登陆 GassDB 目标数据库，查看创建的数据。

二、 环境准备

1. 开发环境

开发语言：python3

2. 安装组件和数据库对应驱动

安装组件：django。

安装数据库 python 驱动：psycopg2。

3. 数据库

创建 GaussDB 实例：华为云 GaussDB、购买 GaussDB 实例（如果只是学习/体验 GaussDB，建议使用按需计费购买）

三、 连接步骤与 Django 项目与应用创建与使用

1. 创建项目虚拟环境

因为业务场景的 Python 开发，多数都是构建一个大型应用程序，并且不希望各种组件的各种版本之间相互冲突，所以需要设置一个虚拟环境。

```
pip3 install virtualenv          #安装 virtualenv
python3 -m venv myenv            #创建虚拟环境
source myenv/bin/activate        #激活环境
```

环境激活后，用户名前会有（myenv）字样。如下云服务器，Centos 终端用户名 hlv 界面

```
[hlv@ecs-django-test ~]$ source myenv/bin/activate
(myenv) [hlv@ecs-django-test ~]$
```

（注意：文档中使用的是 X86 的 CentOS7.9 的服务器，IP 为 192.168.0.52，文档中使用的数据库主节点 IP 为 192.168.1.120，个人实操时请换作个人环境的 IP 地址。）

2. 安装 Django 组件

用 python3 自带的安装工具 pip3，安装组件。

```
pip3 install django
```

或者指定版本号安装

```
pip3 install django==3.2
```

可通过以下命令检测 Django 是否安装成功

```
python
import django
django.__version__

quit()
```

3. 安装 GaussDB 数据库对应的 Python 驱动

GaussDB 数据库对应的 Python 驱动为 psycopg2。即 Django 组件允许通过 psycopg2 驱动，连接 GaussDB 数据库，并操作数据对象。

注意：在安装 psycopg2 时，环境会从 python 源下载 psycopg2 的源码去编译安装，会调用 pg_config，获取 pg 的一些编译依赖的函数和目录。而在编译 psycopg2 时，会调用 python.h 头文件中一些函数声明，所以需要在安装 psycopg2 之前，确保已经安装了 postgresql-devel 和 python3-devel 依赖。

```
yum install -y postgresql-devel python3-devel
```

确保依赖安装成功后，安装 psycopg2：

```
pip3 install psycopg2
```

4. 创建 Django 的项目

这里通过 Django 组件，创建一个自己的项目，项目名称为 myproj。

```
mkdir ~/myprojdir

django-admin startproject myproj ~/myprojdir
```

此时，项目 myproj 在当前用户名路径下，/home/用户名。项目中目录，应该包含以下内容：

```
#一个 Django 项目管理脚本。

 ~/myprojdir/manage.py

#Django 项目包 myproj 里应该会包含__init__.py、settings.py、urls.py、asgi.py 和 wsgi.py 文件。

 ls ~/myprojdir/myproj/
```

```
(myenv) [hlv@ecs-django-test ~]$ ls ~/myprojdir/myproj/
asgi.py  __init__.py  settings.py  urls.py  wsgi.py
(myenv) [hlv@ecs-django-test ~]$
```

5. 创建 Django 项目的应用

在 myproj 的项目目录里，创建一个应用，例如本例中创建一个 djangoapp 的应用

```
cd myprojdir

django-admin startapp djangoapp
```

如果需要自创建登录 web 界面，可指定—template=路径，指定应用模板，可在其中编写 html 脚本。此时 myprojdir 里会生成一个 djangoapp 应用目录，如下：

```
(myenv) [hlv@ecs-django-test myprojdir]$ ls
djangoapp  manage.py  myproj
```

进入 djangoapp 应用目录，初始内容如下：

```
(myenv) [hlv@ecs-django-test myprojdir]$ cd djangoapp/
(myenv) [hlv@ecs-django-test djangoapp]$ ls
admin.py  apps.py  __init__.py  migrations  models.py  tests.py  views.py
(myenv) [hlv@ecs-django-test djangoapp]$
```

其中 **models.py** 就是 **myproj** 项目里 djangoapp 要创建的表结构程序编写处，表示应用需要存储在数据库上的表。

```
from django.db import models

# Create your models here.
~
~
```

views.py 是视图程序编码处。表示应用 djangoapp 要使用的视图结构。

```
from django.shortcuts import render

# Create your views here.
~
~
```

test.py 是 djangoapp 应用需要自添加的测试用例，需要开发者自己编写测试用例。

```
from django.test import TestCase

# Create your tests here.
~
~
```

admin.py 是应用 djangoapp 的登录注册程序编码处。

```python
from django.contrib import admin

# Register your models here.
~
~
~
```

app.py 是 djangoapp 应用的主程序编写处。

```python
from django.apps import AppConfig


class DjangoappConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'djangoapp'
```

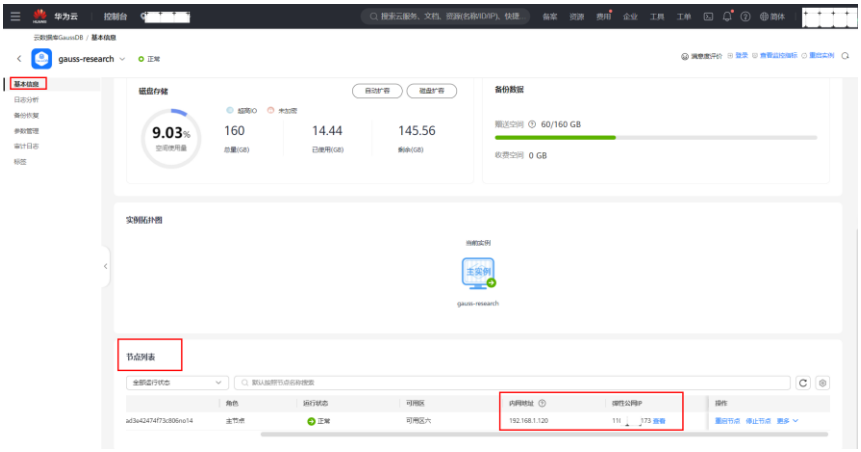
myproj 目录下的 urls.py 是该项目的路由配置文件。可以添加自己想要的路由。

```python
"""myproj URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

6. 数据库 GaussDB 端操作

1) 创建连接用户并赋权限

在购买华为云 GuassDB 数据库实例后，可以在控制台查看实例节点信息，通过基本信息页面右上角“登录“跳转到数据管理服务 DAS 登录数据库。



登录后创建新用户，并赋给新用户权限，后续操作均使用新用户。

```
create user myuser with sysadmin password ‘GaussDB@123’;
```

批注 [g(1]: 引号问题

2) 配置数据库 Ip 地址访问权限

a) 查看 GaussDB 所在的服务器 ip 地址

GaussDB 数据库节点 IP 地址可以在数据库实例基本信息页面查看，文档中使用的 GaussDB 数据库主节点 IP 如下，个人实验时，以个人创建的 GaussDB 数据库实例信息为准。

b）配置 GaussDB 数据库安全组

场景 1：创建 Django 项目所在的服务器跟 GaussDB 数据库在同一个虚拟私有云 VPC 下，走内网访问数据库，可参考下图配置 GaussDB 数据库的安全组，将源地址网段改为您的 VPC 网段即可。

场景 2：创建 Django 项目所在的服务器跟 GaussDB 数据库在同一个虚拟私有云 VPC 下，需要通过公网访问数据库，需要获取 Django 服务器公网 IP，参考下图，将源地址网段换成 Django 服务器公网 IP，例如：10.23.10.52/32。

本例中使用的 Django 服务器和 GaussDB 数据库在同一个 VPC 下，所以使用内网地址（192.168.1.120）进行连接即可。个人进行实验时，根据具体情况判断使用内网 IP 还是公网 IP 进行连接。

GaussDB 默认监听端口是 8000。

c）gsql 远程登录测试

测试 Django 服务器能否远程登录数据库。

需要在 Django 服务器安装 gsql 客户端工具。执行以下命令下载并安装 gsql 客户端工具：

```
source /etc/profile

wget https://sandbox-expriment-files.obs.cn-north-
```

```
1.myhuaweicloud.com/20220525/GaussDB_opengauss_client_tools.zip

unzip GaussDB_opengauss_client_tools.zip

cd GaussDB_opengauss_client_tools/Euler2.5_X86_64/

tar -xvf GaussDB-Kernel-V500R001C20-EULER-64bit-gsql.tar.gz

source gsql_env.sh
```

用 gsql 工具测试远程连接是否成功。

```
gsql -h192.168.1.120 -U myuser -W GaussDB@123 -d postgres -p 8000
```

```
(myenv) [hlv@ecs-django-test ~]$ gsql -h 192.168.1.120 -U myuser -W GaussDB@123 -d postgres -p 8000 -r
gsql ((GaussDB Kernel V500R001C20 build 2a554812) compiled at 2021-12-21 21:38:36 commit 1094 last mr 7042 )
SSL connection (cipher: DHE-RSA-AES128-GCM-SHA256, bits: 128)
Type "help" for help.

postgres=>
```

注意：这里使用的用户是上面在 GaussDB 中创建新的登录用户。

7. Django 项目端配置 setting.py 操作

进入创建的项目目录~/myprojdir/myproj 里，其中 setting.py 就是 Django 项目的配置文件。

```
(myenv) [hlv@ecs-django-test ~]$ cd ~/myprojdir/myproj/
(myenv) [hlv@ecs-django-test myproj]$ ls
asgi.py  __init__.py  settings.py  urls.py  wsgi.py
(myenv) [hlv@ecs-django-test myproj]$
```

setting.py 用来设置 python 应用的一些属性参数。如下：

```
DATABASES = {
    'default' : {
        'ATOMIC_REQUESTS' : 'True',
        'ENGINE' : 'django.db.backends.postgresql_psycopg2',
```

批注 [g(2]: 引号中文-》英文

```
        ‘NAME’ : ‘postgres’,            #数据库名
        ‘USER’ : ‘myuser’,              #用户名
        ‘PASSWORD’ : ‘GaussDB@123’,     #密码
        ‘HOST’ : ‘192.168.1.120’,       #数据库主节点 ip
        ‘PORT’ : ‘8000’,                #GaussDB 数据库端口
    }
}
```

```
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ATOMIC_REQUESTS': 'True',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'myuser',
        'PASSWORD': 'GaussDB@123',
        'HOST': '192.168.1.120',
        'PORT': '8000',
    }
}
```

除了 DATABASES 的配置项，还要在 ALLOWED_HOSTS 列表中，添加 Django 所在服务器的 IPv4 的地址。

```
ALLOWED_HOSTS = ['192.168.0.52']
```

8. 注册 APP ---- djangoapp

在 settings.py 中，INSTALLED_APPS 数组，添加 app 名称。如下：

```
vim ~/myprojdir/myproj/settings.py
```

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'djangoapp',
]
```

9. 运行 Django 服务器，连接 GaussDB 数据库

```
cd ~/myprojdir

python3 manage.py migrate

python3 manage.py runserver 192.168.0.52:8000
```

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(myenv) [hlv@ecs-django-test myprojdir]$
```

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py runserver 192.168.0.52:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 12, 2024 - 08:08:40
Django version 3.2.25, using settings 'myproj.settings'
Starting development server at http://192.168.0.52:8000/
Quit the server with CONTROL-C.
```

启动后，会显示开发服务器在 **http**://192.168.0.52:8000/

注：192.168.0.52 是 Django 项目所在的服务器的 ipv4 地址，不是数据库的地址。

(停止 Django 服务快捷键：CTRL+C)

创建 Django superuser 用户并登录 Django 管理后台：

1. 进入 **Django shell** 模式

```
python manage.py shell
```

2. 导入 **User** 模型类

```
from django.contrib.auth.models import User
```

3. 创建用户对象

```
user = User(username='test',email='test@qq.com')
```

4. 为用户对象属性赋值

```
user.first_name='test'

user.last_name='test'

user.is_superuser = True

user.is_active = True

user.is_staff = True

user.password = 'test123'
```

5. 将用户信息存入数据库，并退出 Django shell 模式

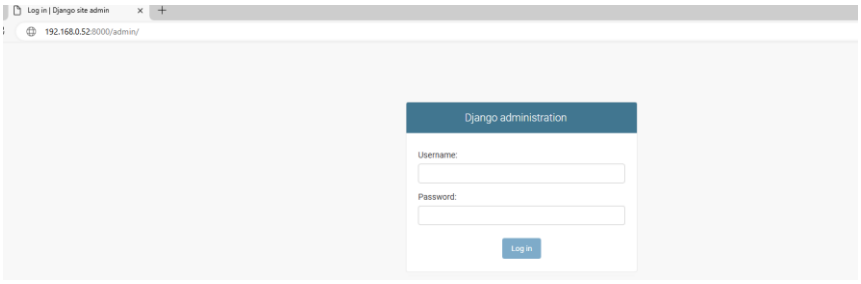
```
user.save()

quit()
```

6. 更改用户密码

```
python manage.py changepassword test
```

用户创建之后，重新启动下 Django 服务器，访问 Django 管理后台

http://192.168.0.52:8000/admin



10.创建表 Question 和 Choice

在项目 myprojdir 中 djangoapp 应用里的 models.py 里，添加表 Question 和 Choice。

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length = 200)
    pub_date = models.DateTimeField("date published")

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete = models.CASCADE)
    choice_text = models.CharField(max_length = 200)
    votes = models.IntegerField(default = 0)

# Create your models here.
```

```
class Question(models.Model):

        question_text = models.CharField(max_length = 200)

        pub_date = models.DateTimeField("date published")



class Choice(models.Model):

        question = models.ForeignKey(Question,on_delete = models.CASCADE)

        choice_text = models.CharField(max_length = 200)

        votes = models.IntegerField(default = 0)
```

回到项目根目录 myprojdir 再执行 `python3 manage.py makemigrations djangoapp`，

是对表的修改创建 migrations。

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py makemigrations djangoapp
Migrations for 'djangoapp':
  djangoapp/migrations/0001_initial.py
    - Create model Question
    - Create model Choice
(myenv) [hlv@ecs-django-test myprojdir]$
```

会看见 django 会迁移 model Question 和 Choice 到数据库中。

执行 `python3 manage.py sqlmigrate djangoapp 0001`，从日志中可以看见创建表的执行事务。

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py sqlmigrate djangoapp 0001
BEGIN;
--
-- Create model Question
--
CREATE TABLE "djangoapp_question" ("id" bigserial NOT NULL PRIMARY KEY, "question_text" varchar(200) NOT NULL, "pub_date" timestamp with time zone NOT
 NULL);
--
-- Create model Choice
--
CREATE TABLE "djangoapp_choice" ("id" bigserial NOT NULL PRIMARY KEY, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" big
int NOT NULL);
ALTER TABLE "djangoapp_choice" ADD CONSTRAINT "djangoapp_choice_question_id_8588053a_fk_djangoapp_question_id" FOREIGN KEY ("question_id") REFERENCES
"djangoapp_question" ("id") DEFERRABLE INITIALLY DEFERRED;
CREATE INDEX "djangoapp_choice_question_id_8588053a" ON "djangoapp_choice" ("question_id");
COMMIT;
(myenv) [hlv@ecs-django-test myprojdir]$
```

如果其中报了错误，可以执行 `python3 manage.py check` 来检查 Error

```
(myenv) [hlv@ecs-hlv-tidb-1 myprojdir]$ python3 manage.py check
System check identified no issues (0 silenced).
(myenv) [hlv@ecs-hlv-tidb-1 myprojdir]$
```

如果没有 Error，则执行 `python3 manage.py migrate`，告诉数据库同步我对表的修改操作。把这些修改应用于绑定的数据库中。

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, djangoapp, sessions
Running migrations:
  Applying djangoapp.0001_initial... OK
(myenv) [hlv@ecs-django-test myprojdir]$
```

原来的模板表是：

```
openGauss=> \d
                              List of relations
 Schema |              Name                |   Type   | Owner |             Storage
--------+----------------------------------+----------+-------+---------------------------------
 lvhui  | auth_group                       | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_group_id_seq                | sequence | lvhui |
 lvhui  | auth_group_permissions           | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_group_permissions_id_seq    | sequence | lvhui |
 lvhui  | auth_permission                  | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_permission_id_seq           | sequence | lvhui |
 lvhui  | auth_user                        | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_user_groups                 | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_user_groups_id_seq          | sequence | lvhui |
 lvhui  | auth_user_id_seq                 | sequence | lvhui |
 lvhui  | auth_user_user_permissions       | table    | lvhui | {orientation=row,compression=no}
 lvhui  | auth_user_user_permissions_id_seq | sequence | lvhui |
 lvhui  | django_admin_log                 | table    | lvhui | {orientation=row,compression=no}
 lvhui  | django_admin_log_id_seq          | sequence | lvhui |
 lvhui  | django_content_type              | table    | lvhui | {orientation=row,compression=no}
 lvhui  | django_content_type_id_seq       | sequence | lvhui |
 lvhui  | django_migrations                | table    | lvhui | {orientation=row,compression=no}
 lvhui  | django_migrations_id_seq         | sequence | lvhui |
 lvhui  | django_session                   | table    | lvhui | {orientation=row,compression=no}
(19 rows)
```

通过修改 **models.py**，添加了两张表 **Question** 和 **Choice**，现在的目标库表是

```
postgres=> \d
                              List of relations
 Schema |              Name                |   Type   |  Owner   |             Storage
--------+----------------------------------+----------+----------+---------------------------------
 public | auth_group                       | table    | myuser | {orientation=row,compression=no}
 public | auth_group_id_seq                | sequence | myuser |
 public | auth_group_permissions           | table    | myuser | {orientation=row,compression=no}
 public | auth_group_permissions_id_seq    | sequence | myuser |
 public | auth_permission                  | table    | myuser | {orientation=row,compression=no}
 public | auth_permission_id_seq           | sequence | myuser |
 public | auth_user                        | table    | myuser | {orientation=row,compression=no}
 public | auth_user_groups                 | table    | myuser | {orientation=row,compression=no}
 public | auth_user_groups_id_seq          | sequence | myuser |
 public | auth_user_id_seq                 | sequence | myuser |
 public | auth_user_user_permissions       | table    | myuser | {orientation=row,compression=no}
 public | auth_user_user_permissions_id_seq | sequence | myuser |
 public | django_admin_log                 | table    | myuser | {orientation=row,compression=no}
 public | django_admin_log_id_seq          | sequence | myuser |
 public | django_content_type              | table    | myuser | {orientation=row,compression=no}
 public | django_content_type_id_seq       | sequence | myuser |
 public | django_migrations                | table    | myuser | {orientation=row,compression=no}
 public | django_migrations_id_seq         | sequence | myuser |
 public | django_session                   | table    | myuser | {orientation=row,compression=no}
 public | djangoapp_choice                 | table    | myuser | {orientation=row,compression=no}
 public | djangoapp_choice_id_seq          | sequence | myuser |
 public | djangoapp_question               | table    | myuser | {orientation=row,compression=no}
 public | djangoapp_question_id_seq        | sequence | myuser |
 public | userdate                         | table    | root   | {orientation=row,compression=no}
(24 rows)
```

红色框里即是修改 **models.py**，添加的表关系 **Question** 和 **Choice**，在库里是应用名 **djangoapp** 前缀加表名。表结构如下图，他会自动在每个表加一列 **id**。

```
postgres=> \d djangoapp_choice
                                  Table "public.djangoapp_choice"
   Column    |          Type          |                          Modifiers
-------------+------------------------+-------------------------------------------------------------
 id          | bigint                 | not null default nextval('djangoapp_choice_id_seq'::regclass)
 choice_text | character varying(200) | not null
 votes       | integer                | not null
 question_id | bigint                 | not null
Indexes:
    "djangoapp_choice_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
    "djangoapp_choice_question_id_8588053a" btree (question_id) TABLESPACE pg_default
Foreign-key constraints:
    "djangoapp_choice_question_id_8588053a_fk_djangoapp_question_id" FOREIGN KEY (question_id) REFERENCES djangoapp_question(id) DEFERRABLE INITIALLY
DEFERRED

postgres=> \d djangoapp_question
                                  Table "public.djangoapp_question"
    Column     |           Type           |                          Modifiers
---------------+--------------------------+-------------------------------------------------------------
 id            | bigint                   | not null default nextval('djangoapp_question_id_seq'::regclass)
 question_text | character varying(200)   | not null
 pub_date      | timestamp with time zone | not null
Indexes:
    "djangoapp_question_pkey" PRIMARY KEY, btree (id) TABLESPACE pg_default
Referenced by:
    TABLE "djangoapp_choice" CONSTRAINT "djangoapp_choice_question_id_8588053a_fk_djangoapp_question_id" FOREIGN KEY (question_id) REFERENCES djangoap
p_question(id) DEFERRABLE INITIALLY DEFERRED

postgres=>
```

也可以通过 python3 manage.py shell，来直接写数据和查询。

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py shell
Python 3.6.8 (default, Nov 14 2023, 16:29:52)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from djangoapp.models import Choice, Question
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
>>> q.save()
>>> q.id
1
>>>
```

```
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2024, 10, 12, 8, 40, 35, 755697, tzinfo=<UTC>)
>>> q.question_text = "What is up?"
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
>>> quit()
(myenv) [hlv@ecs-django-test myprojdir]$
```

11. 创建视图

1.在应用 djangoapp 里的 views.py 文件中编码使用下面用例。

```
#三种方式创建视图函数

from django.shortcuts import render

from django.http import HttpResponse



def DETAIL(request):

        return HttpResponse("view DETAIL.")



def RESULTS(request):

        response = "view RESULTS."

        return HttpResponse(response)



def VOTE(request):

        return HttpResponse("view VOTE.")
```

2.在 djangoapp 应用目录下，打开 urls.py 文件，如果没有新建一个。

```
(myenv) [root@ecs-2778-dcoker-db54 djangoapp]# pwd
/root/myprojdir/djangoapp
(myenv) [root@ecs-2778-dcoker-db54 djangoapp]# ls
admin.py  apps.py  __init__.py  migrations  models.py  __pycache__  tests.py  urls.py  views.py
(myenv) [root@ecs-2778-dcoker-db54 djangoapp]#
```

3.在 django 应用目录下的 urls.py 文件中导入视图函数。内容如下：

```
from.views import DETAIL

from.views import RESULTS

from.views import VOTE
```

```
from django.urls import path

urlpatterns = [

    path('results/',RESULTS),

    path('vote/',VOTE),

    path('detail/',DETAIL),

]
```

4.在项目 myproj 目录下的 urls.py 文件中添加应用的 URL。内容如下：

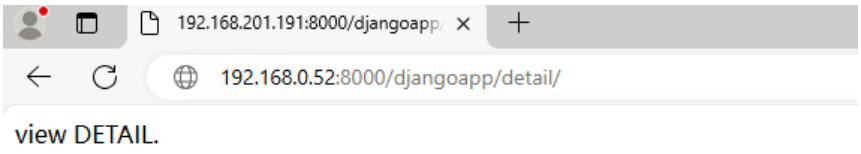
```
(myenv) [root@ecs-2778-dcoker-db54 myproj]# pwd
/root/myprojdir/myproj
(myenv) [root@ecs-2778-dcoker-db54 myproj]# ls
asgi.py  __init__.py  __pycache__  settings.py  urls.py  wsgi.py
(myenv) [root@ecs-2778-dcoker-db54 myproj]#
```

```
from django.contrib import admin

from django.urls import path,include

import djangoapp.urls


urlpatterns = [

    path('admin/', admin.site.urls),

    path('djangoapp/',include(djangoapp.urls)),

]
```

5.回到项目根目录下，启动 Django 服务器

```
cd ~/myprojdir

python3 manage.py migrate

python3 manage.py runserver 192.168.0.52:8000
```

访问视图 DETAIL：http://192.168.0.52:8000/djangoapp/detail



12.查询表数据

此处在 python3 manage.py shell 的 python 编码界面中使用。

```
#获取所有记录

records = MYMODEL.OBJECTS.ALL()        // MYMODEL 是一个模型类名，对应表名为

MYMODEL，通过 objects 属性来执行各种查询，如 all()获取所有记录

# filter()根据条件过滤记录，如同 sql 中的 where 条件

filtered_records = MyModel.objects.filter(field1='value')
```

如下：

```
(myenv) [hlv@ecs-django-test myprojdir]$ python3 manage.py shell
Python 3.6.8 (default, Nov 14 2023, 16:29:52)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from djangoapp.models import Choice, Question
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
>>> q.save()
>>> q.id
1
>>>
```

```
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2024, 10, 12, 8, 40, 35, 755697, tzinfo=<UTC>)
>>> q.question_text = "What is up?"
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
>>> quit()
(myenv) [hlv@ecs-django-test myprojdir]$
```

13.原生 SQL 查询

此处在 python3 manage.py shell 的 python 编码界面中使用。语法参考：

```
# raw()执行原生 SQL 查：

# MyModel—Django 应用目录下 models.py 定义的 Model

# myapp_mymodel—数据库表名，应用名_Model 名

raw_query_result = MyModel.objects.raw('SELECT * FROM myapp_mymodel')
```

1. 在 Django 项目根目录下，启动 Django 的交互式命令行界面

```
python3 manage.py shell
```

2.导入相关模型，这里用第 10 步创建的模型 Question

```
from djangoapp.models import Choice
```

3. 执行原生 SQL 查询

```
raw_query_result = Choice.objects.raw('SELECT * FROM djangoapp_Choice')
```

4.处理查询结果

```
for result in raw_query_result:

        print(result)
```

```
(myenv) [root@ecs-2778-dcoker-db54 myprojdir]# python3 manage.py shell
Python 3.6.8 (default, Nov 14 2023, 16:29:52)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from djangoapp.models import Choice
>>> from djangoapp.models import Choice
>>> raw_query_result = Choice.objects.raw('SELECT * FROM djangoapp_Choice')
>>> for result in raw_query_result:
...         print(result)
...
Choice object (101)
Choice object (102)
>>>
```

四、 GaussDB 的适配问题总结归纳

由于本文档是用 opengauss，所以用的是 django 里的 postgresql 模块，如果是使用 GaussDB 场景，需要替换的目录和文件如下：

- django/db/backends/gaussdb
- django/db/backends/base/schema.py
- django/db/models
- django/contrib/gaussdb

1. 数据库特性差异

注释类属性

`minimum_database_version = (12,)`–>改为 `minimum_database_version = (8,)`注释类属性 `can_clone_databases = True`，不支持基于除 template0 和 template 创建数据库–>改为 `can_clone_databases = False`

在 `django/db/backends/gaussdb/creation.py` 的`_execute_create_test_db` 方法里手动调用"migrate"来解决 ut 中创建数据库不能复制的问题

```
call_command(
    "migrate",
    verbosity=max(1 - 1, 0),
    interactive=False,
    database=self.connection.alias,
    run_syncdb=True,
)
can_clone_databases = False
supports_aggregate_filter_clause = False
supports_update_conflicts = False
supported_explain_formats = {"TEXT"}
supports_json_field_contains = False
```

2. pg_class 字段差异

1) relispartition 与 parttype

在 get_table_list 这个函数使用中，pg_class 在 postgresql 和 GaussDB 中分区表的展示不同。

GaussDB 的 pg_class 中不存在 relispartition 字段，内置 sql 调整

数据库	字段	类型	描述
PostgreSQL	relispartition	bool	如果表或索引是一个分区，则为真
GaussDB	parttype	char	表或者索引是否具有分区表的性质。

			p：表示带有分区表性质 n：表示没有分区表特性 s：表示该表为二级分区表

```
-- 修改前
SELECT
    c.relname,
    CASE
        WHEN c.relispartition THEN 'p'
        WHEN c.relkind IN ('m', 'v') THEN 'v'
        ELSE 't'
    END,
    obj_description(c.oid, 'pg_class')
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('f', 'm', 'p', 'r', 'v')
    AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
    AND pg_catalog.pg_table_is_visible(c.oid)
-- 修改后
SELECT
    c.relname,
    CASE
        WHEN c.parttype <> 'n' THEN 'p'
        WHEN c.relkind IN ('m', 'v') THEN 'v'
```

```
        ELSE 't'
    END,
    obj_description(c.oid, 'pg_class')
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('f', 'm', 'p', 'r', 'v')
    AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
    AND pg_catalog.pg_table_is_visible(c.oid)
```

2) relkind 字段差异

数据库	字段	类型	描述
PostgreSQL	relkind	char	r = 普通表， i = 索引， S = 序列， t = TOAST 表， v = 视图， m = 物化视图， c = 组合类型， f = 外部表， p = 分区表， I = 分区索引
GaussDB	relkind	char	r：表示普通表, i：表示索引, I：表示分区表 GLOBAL 索引, S：表示序列, L：表示长序列, v：表示视图, c：表示复合类型, t：表示 TOAST 表, f：表示外表, m：表示物化视图

```
--修改前
SELECT
    s.relname AS sequence_name,
    a.attname AS colname
FROM
```

```
    pg_class s
    JOIN pg_depend d ON d.objid = s.oid
        AND d.classid = 'pg_class'::regclass
        AND d.refclassid = 'pg_class'::regclass
    JOIN pg_attribute a ON d.refobjid = a.attrelid
        AND d.refobjsubid = a.attnum
    JOIN pg_class tbl ON tbl.oid = d.refobjid
        AND tbl.relname = %s
        AND pg_catalog.pg_table_is_visible(tbl.oid)
WHERE
    s.relkind = 'S';
--修改后
SELECT
    s.relname AS sequence_name,
    a.attname AS colname
FROM
    pg_class s
    JOIN pg_depend d ON d.objid = s.oid
        AND d.classid = 'pg_class'::regclass
        AND d.refclassid = 'pg_class'::regclass
    JOIN pg_attribute a ON d.refobjid = a.attrelid
        AND d.refobjsubid = a.attnum
    JOIN pg_class tbl ON tbl.oid = d.refobjid
        AND tbl.relname = %s
```

```
        AND pg_catalog.pg_table_is_visible(tbl.oid)
WHERE
    s.relkind in ('S', 'L');
```

GaussDB 创建出来的名字也有差别,PG 是 `table_name_id_seq`,GaussDB 是 `table_name_id_identity`。体现在 UT 测试用例: `test_get_sequences` 里。

3. WITH ORDINALITY 语句问题

GaussDB 不支持，修改后存在返回值不一致的情况 当前出现的不一致字段：orders, type, options。

```
--修改前
SELECT
    c.conname,
    array(
        SELECT attname
        FROM unnest(c.conkey) WITH ORDINALITY cols(colid, arridx)
        JOIN pg_attribute AS ca ON cols.colid = ca.attnum
        WHERE ca.attrelid = c.conrelid
        ORDER BY cols.arridx
    ),
    c.contype,
    (SELECT fkc.relname || '.' || fka.attname
    FROM pg_attribute AS fka
    JOIN pg_class AS fkc ON fka.attrelid = fkc.oid
```

```
    WHERE fka.attrelid = c.confrelid AND fka.attnum = c.confkey[1]),
    cl.reloptions
FROM pg_constraint AS c
JOIN pg_class AS cl ON c.conrelid = cl.oid
WHERE cl.relname = %s AND pg_catalog.pg_table_is_visible(cl.oid)


--修改后
SELECT
    c.conname,
    array(
        SELECT attname
        FROM (SELECT *, rownum as arridx FROM (SELECT unnest(array[c.conkey]) as
colid)) cols
        JOIN pg_attribute AS ca ON cols.colid = ca.attnum
        WHERE ca.attrelid = c.conrelid
        ORDER BY cols.arridx
    ),
    c.contype,
    (SELECT fkc.relname || '.' || fka.attname
    FROM pg_attribute AS fka
    JOIN pg_class AS fkc ON fka.attrelid = fkc.oid
    WHERE fka.attrelid = c.confrelid AND fka.attnum = c.confkey[1]),
    cl.reloptions
```

```
FROM pg_constraint AS c

JOIN pg_class AS cl ON c.conrelid = cl.oid

WHERE cl.relname = %s AND pg_catalog.pg_table_is_visible(cl.oid)

--修改前
SELECT
    indexname,

    array_agg(attname ORDER BY arridx),

    indisunique,

    indisprimary,

    array_agg(ordering ORDER BY arridx),

    amname,

    exprdef,

    s2.attoptions
FROM (
    SELECT

        c2.relname as indexname, idx.*, attr.attname, am.amname,

        CASE

            WHEN idx.indexprs IS NOT NULL THEN

                pg_get_indexdef(idx.indexrelid)

        END AS exprdef,

        CASE am.amname

            WHEN %s THEN

                CASE (option & 1)
```

```
                    WHEN 1 THEN 'DESC' ELSE 'ASC'
                END
        END as ordering,
        c2.reloptions as attoptions
    FROM (
        SELECT *
        FROM
            pg_index i,
            unnest(i.indkey, i.indoption)
                WITH ORDINALITY koi(key, option, arridx)
    ) idx
    LEFT JOIN pg_class c ON idx.indrelid = c.oid
    LEFT JOIN pg_class c2 ON idx.indexrelid = c2.oid
    LEFT JOIN pg_am am ON c2.relam = am.oid
    LEFT JOIN
        pg_attribute attr ON attr.attrelid = c.oid AND attr.attnum = idx.key
    WHERE c.relname = %s AND pg_catalog.pg_table_is_visible(c.oid)
) s2
GROUP BY indexname, indisunique, indisprimary, amname, exprdef, attoptions;


--修改后
SELECT
    indexname,
```

```
    array_agg(attname ORDER BY arridx),
    indisunique,
    indisprimary,
    array_agg(ordering ORDER BY arridx),
    amname,
    exprdef,
    s2.attoptions
FROM (
    SELECT
        c2.relname as indexname, idx.*, attr.attname, am.amname,
        CASE
            WHEN idx.indexprs IS NOT NULL THEN
                pg_get_indexdef(idx.indexrelid)
        END AS exprdef,
        CASE am.amname
            WHEN %s THEN
                CASE (option & 1)
                    WHEN 1 THEN 'DESC' ELSE 'ASC'
                END
        END as ordering,
        c2.reloptions as attoptions
    FROM (
```

```
        SELECT *
        FROM (
            SELECT *, unnest(i.indkey) as key, unnest(i.indoption) as option, rownum
as arridx
            FROM
                pg_index i
            ) koi
    ) idx
    LEFT JOIN pg_class c ON idx.indrelid = c.oid
    LEFT JOIN pg_class c2 ON idx.indexrelid = c2.oid
    LEFT JOIN pg_am am ON c2.relam = am.oid
    LEFT JOIN
        pg_attribute attr ON attr.attrelid = c.oid AND attr.attnum = idx.key
    WHERE c.relname = %s AND pg_catalog.pg_table_is_visible(c.oid)
) s2
GROUP BY indexname, indisunique, indisprimary, amname, exprdef, attoptions;
```

4. 调用 bulk_create()方法报错

PostgreSQL 的 upsert 功能：当记录不存在时，执行插入；否则，进行更新

GaussDB 中无 ON CONFLICT 使用方式，使用 ON DUPLICATE KEY UPDATE（无法指定唯一约束）

```
# 进入 postgresql.features.DatabaseFeatures 注释一下类属性，默认为 False
# supports_update_conflicts_with_target = True
```

```
# 修改前
if on_conflict == OnConflict.IGNORE:
    return "ON CONFLICT DO NOTHING"
if on_conflict == OnConflict.UPDATE:
    return "ON CONFLICT(%s) DO UPDATE SET %s" % (
        ", ".join(map(self.quote_name, unique_fields)),
        ", ".join(
            [
                f"{field} = EXCLUDED.{field}"
                for field in map(self.quote_name, update_fields)
            ]
        ),
    )
# 修改后
if on_conflict == OnConflict.IGNORE:
    return "ON DUPLICATE KEY UPDATE NOTHING"
if on_conflict == OnConflict.UPDATE:
    return "ON DUPLICATE KEY UPDATE %s" % (
        ", ".join(
            [
                f"{field} = EXCLUDED.{field}"
                for field in map(self.quote_name, update_fields)
```

```
            ]
        ),
    )
```

5. 执行 python manage.py flush 报错

GaussDB 不支持重置自增主键（RESTART IDENTITY）

修改

```
DatabaseWrapper.data_types("AutoField":"serial",
"BigAutoField":"bigserial","SmallAutoField":"smallserial")
```

修改

```
DatabaseWrapper.data_types_suffix{
            "AutoField": "",
            "BigAutoField": "",
            "SmallAutoField": "",
     }
# 修改前
def sql_flush(self, style, tables, *, reset_sequences=False, allow_cascade=False):
    if not tables:
        return []
# Perform a single SQL 'TRUNCATE x, y, z...;' statement. It allows us
# to truncate tables referenced by a foreign key in any other table.
    sql_parts = [
```

```
        style.SQL_KEYWORD("TRUNCATE"),
        ", ".join(style.SQL_FIELD(self.quote_name(table)) for table in tables),
    ]
    if reset_sequences:
        sql_parts.append(style.SQL_KEYWORD("RESTART IDENTITY"))
    if allow_cascade:
        sql_parts.append(style.SQL_KEYWORD("CASCADE"))
    return ["%s;" % " ".join(sql_parts)]
# 修改后
def sql_flush(self, style, tables, *, reset_sequences=False, allow_cascade=False):
    if not tables:
        return []
# Perform a single SQL 'TRUNCATE x, y, z...;' statement. It allows us
# to truncate tables referenced by a foreign key in any other table.
    sql_parts = [
        style.SQL_KEYWORD("TRUNCATE"),
        ", ".join(style.SQL_FIELD(self.quote_name(table)) for table in tables),
    ]
    if allow_cascade:
        sql_parts.append(style.SQL_KEYWORD("CASCADE"))
    sql = ["%s;" % " ".join(sql_parts)]
    if reset_sequences:
```

```
        truncated_tables = {table.upper() for table in tables}
        sequences = [
            sequence
            for sequence in self.connection.introspection.sequence_list()
            if sequence["table"].upper() in truncated_tables
        ]
        sql.extend(self.sequence_reset_by_name_sql(style, sequences))
    return sql
```

6. GaussDB 的自增字段不支持 IDENTITY

GaussDB 的自增字段不支持 IDENTITY，所以创建自增字段时使用 serial 等类型

类属性调整

```
# GaussDB 不支持
# sql_alter_sequence_type = "ALTER SEQUENCE IF EXISTS %(sequence)s
AS %(type)s"
# sql_add_identity = (
#       "ALTER TABLE %(table)s ALTER COLUMN %(column)s ADD "
#       "GENERATED BY DEFAULT AS IDENTITY"
# )
# sql_drop_indentity = (
#       "ALTER TABLE %(table)s ALTER COLUMN %(column)s DROP IDENTITY IF
EXISTS"
```

```
# )
# 手动创建序列
sql_add_sequence = (
    "CREATE SEQUENCE %(sequence)s INCREMENT 1 MINVALUE 1 MAXVALUE
9223372036854775807 START 1 NOCYCLE"
)
# 指定为某一字段的默认值，使该字段具有唯一标识属性
sql_alter_column_default_sequence = "ALTER TABLE %(table)s ALTER
COLUMN %(column)s SET DEFAULT nextval('%(sequence)s')"
# 指定序列与列的归属关系
sql_associate_column_sequence = "ALTER SEQUENCE %(sequence)s OWNED
BY %(table)s.%(column)s"
# serial 等数据类型对应真实数据类型
auto_types = {
    "serial": "integer",
    "bigserial": "bigint",
    "smallserial": "smallint",
}
```

类方法`_alter_column_type_sql`调整，可对比 postgresql 代码查看

```
def _alter_column_type_sql(
    self, model, old_field, new_field, new_type, old_collation, new_collation
):
```

```
# Drop indexes on varchar/text/citext columns that are changing to a
# different type.
old_db_params = old_field.db_parameters(connection=self.connection)
old_type = old_db_params["type"]
if (old_field.db_index or old_field.unique) and (
    (old_type.startswith("varchar") and not new_type.startswith("varchar"))
    or (old_type.startswith("text") and not new_type.startswith("text"))
    or (old_type.startswith("citext") and not new_type.startswith("citext"))
):
    index_name = self._create_index_name(
        model._meta.db_table, [old_field.column], suffix="_like"
    )
    self.execute(self._delete_index_sql(model, index_name))

self.sql_alter_column_type = (
    "ALTER COLUMN %(column)s TYPE %(type)s%(collation)s"
)
# Cast when data type changed.
if using_sql := self._using_sql(new_field, old_field):
    self.sql_alter_column_type += using_sql
new_internal_type = new_field.get_internal_type()
old_internal_type = old_field.get_internal_type()
```

```
# Make ALTER TYPE with IDENTITY make sense.
table = strip_quotes(model._meta.db_table)
auto_field_types = {
    "AutoField",
    "BigAutoField",
    "SmallAutoField",
}
old_is_auto = old_internal_type in auto_field_types
  new_is_auto = new_internal_type in auto_field_types


# 如果为 serial 等类型，就替换为对应的 integer 等类型，否则不变
new_type = self.auto_types.get(new_type, new_type)
if new_is_auto and not old_is_auto:
    column = strip_quotes(new_field.column)
    # 创建序列
    sequence = f"{table}_{column}_seq"
    self.execute(
        self.sql_add_sequence
        % {
            "sequence": self.quote_name(sequence),
        }
    )
```

```
return (
    (
        self.sql_alter_column_type
        % {
            "column": self.quote_name(column),
            "type": new_type,
            "collation": "",
        },
        [],
    ),
    [
        (
            self.sql_alter_column_default_sequence
            % {
                "table": self.quote_name(table),
                "column": self.quote_name(column),
                "sequence": self.quote_name(sequence),
            },
            [],
        ),
        (
```

```
                    self.sql_associate_column_sequence
                    % {
                        "table": self.quote_name(table),
                        "column": self.quote_name(column),
                        "sequence": self.quote_name(sequence),
                    },
                    [],
                ),
            ],
        )
    elif old_is_auto and not new_is_auto:
        column = strip_quotes(new_field.column)
        fragment, _ = super()._alter_column_type_sql(
            model, old_field, new_field, new_type, old_collation, new_collation
        )
        other_actions = []
        if sequence_name := self._get_sequence_name(table, column):
            other_actions = [
                (
                    self.sql_delete_sequence
                    % {
                        "sequence": self.quote_name(sequence_name),
```

```
                },
                [],
            )
        ]
        return fragment, other_actions
    elif new_is_auto and old_is_auto and old_internal_type != new_internal_type:
        fragment, _ = super()._alter_column_type_sql(
            model, old_field, new_field, new_type, old_collation, new_collation
        )
        column = strip_quotes(new_field.column)
        db_types = {
            "AutoField": "integer",
            "BigAutoField": "bigint",
            "SmallAutoField": "smallint",
        }
        other_actions = []
        return fragment, other_actions
    else:
        return super()._alter_column_type_sql(
            model, old_field, new_field, new_type, old_collation, new_collation
        )
```

7. 数据库版本获取

可使用 SHOW product_version SQL 语句查询 GaussDB 版本

```
@cached_property
def gaussdb_version(self):
with self.temporary_connection():
    with self.connection.cursor() as cur:
        cur.execute("SHOW  product_version")
        version = cur.fetchone()[0]
    return tuple(int(i) for i in version.split('.'))
def get_database_version(self):
    """
    Return a tuple of the database's version.
    E.g. for gaussdb_version 8.102.0, return (8, 102, 0).
    """
    return self.gaussdb_version
```

因为 `ut test_get_database_version` 的原因,上面的代码做了如下修改

```
@cached_property
def gaussdb_version(self):
    with self.temporary_connection():
        with self.connection.cursor() as cur:
            cur.execute("SHOW  product_version")
            version = cur.fetchone()[0]
```

```
        return version
def get_database_version(self):
    """
    Return a tuple of the database's version.
    E.g. for gaussdb_version 8.102.0, return (8, 102, 0).
    """
    return tuple(int(i) for i in self.gaussdb_version.split('.'))
```

8. 代码目录的变动

增加 db/backends/gaussdb 目录，对应的 ut 目录: tests/backends/gaussdb

增加 django/contrib/gaussdb 目录

增加 tests/dbshell/test_gaussdb.py 文件

增加 tests/gaussdb_tests 目录

9. 名称的替换

postgres,postresql 替换为 gaussdb

PostgreSQL 替换为 GaussDB

五、 注意事项

1. 创建索引的方法不支持 gin 索引

行存表（ASTORE 存储引擎）支持的索引类型：btree（行存表缺省值）

行存表（USTORE 存储引擎）支持的索引类型：ubtree

btree 与 ubtree 是与表的存储类型 ASTORE/USTORE 强相关，在创建索引时指定索引类型与主表不对应时会自动进行转换。

在 `django/contrib/gaussdb/indexes.py` 里只保留了 BTreeIndex

2. 不支持重命名序列名

```
ALTER SEQUENCE IF EXISTS "auto_id_id_seq" AS bigint 语句不支持
```

GaussDB 使用 serial 创建自增字段（integer 类型）时会创建一个 bigint 类型的序列在

UT 测试用例: test_get_sequences 里注释了

3. EXTENSION 相关操作需要手动开启

注释 django/contrib/gaussdb/operations.py 里的 CreateExtension 类及其子类

https://support.huaweicloud.com/centralized-devg-v8-gaussdb/gaussdb-42-0554.html

```
set enable_extension = true;
```

4. 创建数据库的时候，tempalte 只支持 template0、templatem

在 django/db/backends/gaussdb/creation.py 里修改为

```
suffix += ' TEMPLATE "template0"'
```

5. GaussDB 对于数据库名字超过 63 字节会截断而不是报错，PG 是报错

<https://support.huaweicloud.com/centralized-devg-v8-gaussdb/gaussdb-42-1751.html>

6. collation "sv-x-icu" for encoding "UTF8" does not exist

```
Creating test database for alias 'default'...

Got an error creating the test database: collation "sv-x-icu" for encoding "UTF8"

does not exist
LINE 1: ... BY DEFAULT AS IDENTITY, "char_field" varchar(10) COLLATE "s...
```

django/db/backends/gaussdb/features.py，注释相关的代码

```
test_collations = {

    "deterministic": "C",

    "virtual": "sv-x-icu",

    # "non_default": "sv-x-icu",

    # "swedish_ci": "sv-x-icu",

}
```

7. 因为 GuaussDB 没有 gis,所以 gis 功能相关的代码没有体现

8. 禁用了 citext

9. Ustore 引擎不支持物化视图的创建和使用

```
ut: inspectdb.tests.InspectDBTransactionalTests. test_include_materialized_views
```

官方文档: <https://support.huaweicloud.com/centralized-devg-v8-gaussdb/gaussdb-42-0564.html#ZH-CN_TOPIC_0000001835601720>

10.Django 5.0 下使用开源的驱动,会报下面的错误

```
======================================================================
```

```
==========

ERROR: test_func_index_json_key_transform (schema.tests.SchemaTests)

----------------------------------------------------------------------

----------

Traceback (most recent call last):

  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line

103, in _execute

    return self.cursor.execute(sql)

psycopg2.errors.UndefinedObject: data type jsonb has no default operator class for

access method "ubtree"

HINT:  You must specify an operator class for the index or define a default operator

class for the data type.
```

11. Django 5.0 下使用开源的驱动，不支持 SHA1()语句

```
如 SQL 语句：

SELECT SHA1("db_functions_author"."alias") AS "sha1_alias" FROM

"db_functions_author" ORDER BY "db_functions_author"."id" ASC

ERROR: test_basic (db_functions.text.test_sha1.SHA1Tests)

----------------------------------------------------------------------

----------

Traceback (most recent call last):

  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line
```

```
105, in _execute
    return self.cursor.execute(sql, params)
psycopg2.DatabaseError: sha/sha1 is supported only in B-format database
CONTEXT:  referenced column: sha1_alias
```

12. Django 5.0 下使用开源的驱动,cannot call jsonb_object_field (jsonb -> text operator) on an array

在 model_fields 这个 ut 目录下，这个错误特别多

暂时将 feature.py 里的 supports_json_field_contains = False

```
test_usage_in_subquery (model_fields.test_jsonfield.TestQuerying) ... ERROR
(0.070) SELECT "model_fields_nullablejsonmodel"."id",
"model_fields_nullablejsonmodel"."value",
"model_fields_nullablejsonmodel"."value_custom" FROM
"model_fields_nullablejsonmodel" WHERE "model_fields_nullablejsonmodel"."id"
IN (SELECT U0."id" FROM "model_fields_nullablejsonmodel" U0 WHERE (U0."value"
-> 'c') = '14'); args=('c', <psycopg2._json.Json object at 0x70ecb4364be0>);
alias=default
----------------------------------------------------------------------
----------
======================================================================
==========
ERROR: test_usage_in_subquery (model_fields.test_jsonfield.TestQuerying)
```

```
----------------------------------------------------------------------
----------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line
105, in _execute
    return self.cursor.execute(sql, params)
psycopg2.errors.InvalidParameterValue: cannot call jsonb_object_field (jsonb ->
text operator) on an array
```

13. Django 5.0 下使用开源的驱动，cannot alter type of column "green" twice

```
======================================================================
==========
ERROR: test_alter_field_change_nullable_to_database_default_not_null
(migrations.test_operations.OperationTests)
The AlterField operation changing a null field to db_default.
----------------------------------------------------------------------
----------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line
103, in _execute
    return self.cursor.execute(sql)
```

```
psycopg2.errors.FeatureNotSupported: cannot alter type of column "green" twice

BEGIN;

CREATE TABLE "test_alflcntddnn_pony" ("id" serial NOT NULL PRIMARY KEY, "pink"

integer NOT NULL, "weight" double precision NOT NULL, "green" integer NULL,

"yellow" varchar(20) DEFAULT 'Yellow' NULL);

COMMIT;

INSERT INTO "test_alflcntddnn_pony" ("pink", "weight", "green", "yellow") VALUES

(3, 1.0, NULL, DEFAULT) RETURNING "test_alflcntddnn_pony"."id",

"test_alflcntddnn_pony"."yellow";

BEGIN;

ALTER TABLE "test_alflcntddnn_pony" ALTER COLUMN "green" SET DEFAULT 4;

UPDATE "test_alflcntddnn_pony" SET "green" = 4 WHERE "green" IS NULL; SET

CONSTRAINTS ALL IMMEDIATE;

ALTER TABLE "test_alflcntddnn_pony" ALTER COLUMN "green" SET NOT NULL;

COMMIT;

SELECT "test_alflcntddnn_pony"."id", "test_alflcntddnn_pony"."pink",

"test_alflcntddnn_pony"."weight", "test_alflcntddnn_pony"."green",

"test_alflcntddnn_pony"."yellow" FROM "test_alflcntddnn_pony" WHERE

"test_alflcntddnn_pony"."id" = 1 LIMIT 21;

INSERT INTO "test_alflcntddnn_pony" ("pink", "weight", "green", "yellow") VALUES

(3, 1.0, DEFAULT, DEFAULT) RETURNING "test_alflcntddnn_pony"."id",

"test_alflcntddnn_pony"."green", "test_alflcntddnn_pony"."yellow";
```

```
BEGIN;
ALTER TABLE "test_alflcntddnn_pony" ALTER COLUMN "green" DROP DEFAULT,
ALTER COLUMN "green" DROP NOT NULL;
```

14.Django 5.0 下使用开源的驱动,cannot call extract path from a scalar

ut：test_deep_distinct

```
======================================================================
==========
ERROR: test_deep_distinct (model_fields.test_jsonfield.TestQuerying)
----------------------------------------------------------------------
----------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line
105, in _execute
    return self.cursor.execute(sql, params)
psycopg2.errors.InvalidParameterValue: cannot call extract path from a scalar
```

15.不支持法语搜索

```
======================================================================
==========
ERROR: test_config_from_field_explicit
```

```
(gaussdb_tests.test_search.MultipleFieldsTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line 123, in _execute
    return self.cursor.execute(sql, params)
psycopg2.errors.FeatureNotSupported: Text search for French is not supported!
```

16.不支持自定义的 collation

```
======================================================================
ERROR: test_create (gaussdb_tests.test_operations.CreateCollationTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-5.0/django/db/backends/utils.py", line 103, in _execute
    return self.cursor.execute(sql)
psycopg2.errors.FeatureNotSupported: user defined collation is not yet supported.
```

17.EXCLUDE constraint is not yet supported.

18.不支持的函数

- BitXor
- JSONBAgg
- GEN_RANDOM_UUID
- phraseto_tsquery
- websearch_to_tsquery
- SIMILARITY

19.不支持 Trigram

六、 待解决问题

1. 查询的大小写问题

ut: test_ci_cs_db_collation

插入 ANDREW，查询 Andrew 没有返回结果

相同的 sql 语句，pg12 是 ok 的.

经过测试 pg11 和 oracle 也没有返回,暂时忽略.

```
CREATE TABLE "schema_author" ("id" integer NOT NULL PRIMARY KEY GENERATED
BY DEFAULT AS IDENTITY, "name" varchar(255) NOT NULL, "height" integer NULL
CHECK ("height" >= 0), "weight" integer NULL, "uuid" uuid NULL); (params None)
INSERT INTO "schema_author" ("name", "height", "weight", "uuid") VALUES
('ANDREW', NULL, NULL, NULL) RETURNING "schema_author"."id";
SELECT 1 AS "a" FROM "schema_author" WHERE "schema_author"."name" =
```

```
'Andrew' LIMIT 1;
```

2. 进行日期运算后结果不对

ut: test_negative_timedelta_update，相同的 sql 语句，pg12 是 ok 的

```
======================================================================
==========
FAIL: test_negative_timedelta_update (expressions.tests.FTimeDeltaTests)
----------------------------------------------------------------------
----------
Traceback (most recent call last):
    File "/root/python/projects/django-gaussdb-5.0/tests/expressions/tests.py", line
2122, in test_negative_timedelta_update
    self.assertEqual(e0.start, expected_start)
AssertionError: datetime.datetime(2010, 6, 15, 14, 46, 0, 747030) !=
datetime.datetime(2010, 6, 23, 9, 45, 0, 746970)
----------------------------------------------------------------------
----------
(0.037)
UPDATE "expressions_ExPeRiMeNt"
SET "start" = (((((("expressions_ExPeRiMeNt"."start" + '-1 days 86370.000000
seconds'::interval) + '-1 days 84600.000000 seconds'::interval) + '-1 days
79200.000000 seconds'::interval) + '-2 days 0.000000 seconds'::interval) + '-1
```

```
days 86399.999970 seconds'::interval)
WHERE "expressions_ExPeRiMeNt"."name" = 'e0';


args=(datetime.timedelta(days=-1, seconds=86370),
        datetime.timedelta(days=-1, seconds=84600),
        datetime.timedelta(days=-1, seconds=79200),
        datetime.timedelta(days=-2),
        datetime.timedelta(days=-1, seconds=86399, microseconds=999970),
        'e0');
ALIAS=DEFAULT (0.037)
SELECT "expressions_ExPeRiMeNt"."id",
        "expressions_ExPeRiMeNt"."name",
        "expressions_ExPeRiMeNt"."assigned",
        "expressions_ExPeRiMeNt"."completed",
        "expressions_ExPeRiMeNt"."estimated_time",
        "expressions_ExPeRiMeNt"."start",
        "expressions_ExPeRiMeNt"."end",
        "expressions_ExPeRiMeNt"."scalar"
FROM "expressions_ExPeRiMeNt"
WHERE "expressions_ExPeRiMeNt"."name" = 'e0'
LIMIT 21;
args=('e0',);
```

解决方法

- 数据库时区不同，PG 默认为 UTC，GaussDB 默认为 PRC
- INTERVAL 使用区别，如("expressions_ExPeRiMeNt"."start" + INTERVAL '−1 day 86370 seconds')，在 PG 中为(−86400 + 86370 = 30)秒，而 GaussDB 中为（−86400 − 86370）秒，需调整为("expressions_ExPeRiMeNt"."start" + INTERVAL '−1 day +86370 seconds')

修改

```
# django.db.backends.gaussdb.base.CursorWrapper
def execute(self, query, args=None):
    if (
        args is not None and isinstance(args, tuple) and len(args) > 0
    ):
        to_replace = [
            "%s" if not isinstance(item, datetime.timedelta)
            else
            "'%d days +%d.%06d seconds'::interval" % (
                item.days, item.seconds, item.microseconds)
            for item in args
        ]
        have_changes = any(
            isinstance(item, datetime.timedelta) for item in args)
```

```
        if have_changes:
            new_params = [param for repl, param in zip(to_replace, args)
                          if repl == "%s"]
            return self.cursor.execute(query % tuple(to_replace),
new_params)
    return self.cursor.execute(query, args)
```

3. sql 返回的条数不对

ut: test_filter_exists_lhs, 相同的 sql 语句，pg 是 ok 的，可以通过 ut

期待返回 2 条数据，gauss 只返回了一条数据

```
======================================================================
==========
FAIL: test_filter_exists_lhs (lookup.tests.LookupQueryingTests)
----------------------------------------------------------------------
----------
Traceback (most recent call last):
    File "/root/python/projects/django-gaussdb-5.0/tests/lookup/tests.py", line
1494, in test_filter_exists_lhs
    self.assertCountEqual(qs, [self.s2, self.s3])
AssertionError: Element counts were not equal:
First has 0, Second has 1:  <Season: 2042>
```

解决方法

指定子查询不展开的 Hint no_expand[(@queryblock)]

修改

`django/db/models/expressions.Subquery.as_sql` 里修改如下

```
def as_sql(self, compiler, connection, template=None, **extra_context):
    connection.ops.check_expression_support(self)
    template_params = {**self.extra, **extra_context}
    subquery_sql, sql_params = self.query.as_sql(compiler, connection)
    template_params["subquery"] = subquery_sql[1:-1].strip()
    no_expand = ' /*+ no_expand*/ '
    prefix = template_params["subquery"][:6]
    if prefix.lower() == 'select':
        template_params["subquery"] = template_params["subquery"].replace(
            prefix, prefix + no_expand, 1)
    template = template or template_params.get("template", self.template)
    sql = template % template_params
    return sql, sql_params
```

查询语句

```
-- 修改前
SELECT "lookup_season"."id", "lookup_season"."year", "lookup_season"."gt",
"lookup_season"."nulled_text_field",
```

```
EXISTS(SELECT 1 AS "a" FROM "lookup_season" U0 WHERE (U0."id" =
("lookup_season"."id") AND U0."year" < 2000) LIMIT 1) AS "before_20"
FROM
    "lookup_season"
WHERE
    EXISTS(SELECT 1 AS "a" FROM "lookup_season" U0 WHERE (U0."id" =
("lookup_season"."id") AND U0."year" < 2000) LIMIT 1) = ("lookup_season"."year" <
(1900));


-- 修改后
SELECT "lookup_season"."id", "lookup_season"."year", "lookup_season"."gt",
"lookup_season"."nulled_text_field",
EXISTS(SELECT /*+ no_expand*/ 1 AS "a" FROM "lookup_season" U0 WHERE (U0."id"
= ("lookup_season"."id") AND U0."year" < 2000) LIMIT 1) AS "before_20"
FROM
    "lookup_season"
WHERE
    EXISTS(SELECT /*+ no_expand*/ 1 AS "a" FROM "lookup_season" U0 WHERE
(U0."id" = ("lookup_season"."id") AND U0."year" < 2000) LIMIT 1) =
("lookup_season"."year" < (1900));
数据
INSERT INTO "public"."lookup_season" ("id", "year", "gt", "nulled_text_field")
VALUES (1, 1942, 1942, NULL);
INSERT INTO "public"."lookup_season" ("id", "year", "gt", "nulled_text_field")
```

```
VALUES (2, 1842, 1942, 'text');

INSERT INTO "public"."lookup_season" ("id", "year", "gt", "nulled_text_field")

VALUES (3, 2042, 1942, NULL);

建表语句

DROP TABLE IF EXISTS "public"."lookup_season";

CREATE TABLE "public"."lookup_season" (

    "id" int4 NOT NULL DEFAULT nextval('lookup_season_id_seq'::regclass),

    "year" int2 NOT NULL,

    "gt" int4,

    "nulled_text_field" text COLLATE "pg_catalog"."default"

)

WITH (orientation=ROW, storage_type=USTORE);
```

七、 适配 Django3.2.25 时发现遗漏的修复点

1. 移除 phrase 类型的搜索的及其 ut

```
class SearchQuery(SearchQueryCombinable, Func):

    output_field = SearchQueryField()

    SEARCH_TYPES = {

        'plain': 'plainto_tsquery',

        # 'phrase': 'phraseto_tsquery',

        'raw': 'to_tsquery',

        'websearch': 'websearch_to_tsquery',
```

```
    }


def test_phrase_search(self):
    line_qs = Line.objects.annotate(search=SearchVector('dialogue'))
    searched = line_qs.filter(search=SearchQuery('burned body his away',
search_type='phrase'))
    self.assertSequenceEqual(searched, [])
    searched = line_qs.filter(search=SearchQuery('his body burned away',
search_type='phrase'))
    self.assertSequenceEqual(searched, [self.verse1])
```

2. 移除 TrigramSimilar, Unaccent 相关的 lookup 及其 ut

```
from .lookups import (
    SearchLookup,
    TrigramSimilar,
    TrigramStrictWordSimilar,
    TrigramWordSimilar,
    Unaccent,
)


# 更改为如下,并移除相关代码
```

```
from .lookups import (
    SearchLookup,
)
```

3. 文件夹重命名

```
django/contrib/gaussdb/templates/postgres 改为
```

```
django/contrib/gaussdb/templates/gaussdb
```

4. 移除 BrinIndex 及相关 ut

```
class BrinIndex(GaussdbIndex):
```

5. 修复 ut: test_array_agg_filter

```
    def test_array_agg_filter(self):
        values = AggregateTestModel.objects.aggregate(
            arrayagg=ArrayAgg("integer_field", filter=Q(integer_field__gt=0)),
        )
        self.assertEqual(values, {"arrayagg": [1, 2]})
```

改为

```
    def test_array_agg_filter(self):
        values = AggregateTestModel.objects.aggregate(
            arrayagg=ArrayAgg("integer_field", filter=Q(integer_field__gt=0)),
        )
        self.assertEqual(values, {'arrayagg': [None, 1, 2, None]})
```

6. 修复 ut: test_add_with_options

```
self.assertIndexExists(table_name, ["pink"], index_type="btree")
```

改为

```
self.assertIndexExists(table_name, ["pink"], index_type="ubtree")
```

7. CURRENT_TIMESTAMP 在 pg model 下为 sql 执行时间,非事务开始时间,未通过 ut,增加注释以提醒

```
def test_transaction_now(self):
    """
    gaussdb will not pass this ut: https://support.huaweicloud.com/centralized-devg-v8-gaussdb/gaussdb-42-1702.html
    guassdb must in A model can pass this ut, we're in pg model.
    The test case puts everything under a transaction, so two models
    updated with a short gap should have the same time.
    """
    m1 = NowTestModel.objects.create()
    m2 = NowTestModel.objects.create()

    NowTestModel.objects.filter(id=m1.id).update(when=TransactionNow())
    sleep(0.1)
    NowTestModel.objects.filter(id=m2.id).update(when=TransactionNow())
```

```
    m1.refresh_from_db()
    m2.refresh_from_db()

    self.assertIsInstance(m1.when, datetime)
    self.assertEqual(m1.when, m2.when)
```

8. 修复 ut: test_name_auto_generation

```
    def test_name_auto_generation(self):
        index = self.index_class(fields=["field"])
        index.set_name_with_model(CharFieldModel)
        self.assertRegex(
            index.name, r"gaussdb_te_field_[0-9a-f]{6}_%s" %
self.index_class.suffix
        )
```

改为

```
    def test_name_auto_generation(self):
        index = self.index_class(fields=["field"])
        index.set_name_with_model(CharFieldModel)
        # in set_name_with_model will use table_name[:11],
        # so 'gaussdb_tests_charfieldmodel'[:11] is gaussdb_tes
        self.assertRegex(
            index.name, r"gaussdb_tes_field_[0-9a-f]{6}_%s" %
```

```
self.index_class.suffix

        )
```

9. 修复 uttest_suffix

```
    def test_suffix(self):

        self.assertEqual(BTreeIndex.suffix, "btree")
```

改为

```
    def test_suffix(self):

        self.assertEqual(BTreeIndex.suffix, "ubtree")
```

10.修复 ut: test_btree_parameters

```
self.assertEqual(constraints[index_name]["options"], ["fillfactor=80"])
```

改为

```
self.assertEqual(constraints[index_name]['options'],

                         ['fillfactor=80', 'storage_type=USTORE'])
```

11. 因为 citext 的移除,移除 ut: test_citext_values 在 django/db/models/fields/json.py 里增加 GaussDBOperatorLookup

在如下的类里增加 `gaussdb_operator` 字段

- HasKey
- HasKeys
- HasAnyKeys

12.将 btree 的 index 后缀改为 ubtree 的后缀,并在 DatabaseSchemaEditor 类里增加_constraint_names 方法

```
def _constraint_names(self, model, column_names=None, unique=None,
                      primary_key=None, index=None, foreign_key=None,
                      check=None, type_ =None, exclude=None):
    """Return all constraint names matching the columns and conditions."""
    if type_ is not None:
        type_ = 'ubtree'
        return super()._constraint_names(model, column_names, unique,
primary_key, index, foreign_key, check, type_,
                                         exclude)
```

13.BaseDatabaseSchemaEditor 的_constraint_names 方法里修改

```
if column_names is None or column_names == infodict["columns"]:
```

改为

```
if ( column_names is None or sorted(column_names) ==
sorted(infodict["columns"]) ):
```

14.SHA 相关的修改，增加了 GaussDBSHA2Mixin

涉及 SHA224, SHA256, SHA384, SHA512

尽管 sql 语法正确，开源的驱动还是会报错: psycopg2.DatabaseError: sha2 is supported only in B-format database

```
class GaussDBSHA2Mixin:
    def as_gaussdb(self, compiler, connection, **extra_content):
        return super().as_sql(
            compiler,
            connection,
            template='SHA2(%%(expressions)s, %s)' % self.function[3:],
            **extra_content,
        )

class SHA256(MySQLSHA2Mixin, OracleHashMixin, PostgreSQLSHAMixin,
GaussDBSHA2Mixin, Transform):
    function = 'SHA256'
    lookup_name = 'sha256'
```

15.在 in 操作符里的子查询加上 no_expand

```
======================================================================
==========
FAIL: test_distinct_on_in_ordered_subquery
```

```
(distinct_on_fields.tests.DistinctOnTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/root/python/projects/django-gaussdb-3.2.25/tests/distinct_on_fields/tests.py", line 133, in test_distinct_on_in_ordered_subquery
    self.assertSequenceEqual(qs, [self.p1_o2, self.p2_o1, self.p3_o1])
AssertionError: Sequences differ: <QuerySet [<Staff: p1>, <Staff: p2>, <Staff: p3>]> != [<Staff: p1>, <Staff: p2>, <Staff: p3>]
```

sql 语句

```
SELECT
    "distinct_on_fields_staff"."id",
    "distinct_on_fields_staff"."name",
    "distinct_on_fields_staff"."organisation"
FROM
    "distinct_on_fields_staff"
WHERE
    "distinct_on_fields_staff"."id" IN ( SELECT /*+ no_expand*/ DISTINCT ON ( "distinct_on_fields_staff"."name" ) "distinct_on_fields_staff"."id" FROM "distinct_on_fields_staff" ORDER BY "distinct_on_fields_staff"."name" ASC, "distinct_on_fields_staff"."id" ASC )
ORDER BY
    "distinct_on_fields_staff"."name" ASC;
```

修改 django/db/models/lookups.py

```
def as_sql(self, compiler, connection):
    max_in_list_size = connection.ops.max_in_list_size()
    if self.rhs_is_direct_value() and max_in_list_size and len(self.rhs) >
max_in_list_size:
        return self.split_parameter_list_as_sql(compiler, connection)
    res = super().as_sql(compiler, connection)
    if res is not None:
        list_res = list(res)
        if connection.vendor == 'gaussdb' and 'IN (SELECT' in res[0]:
            list_res[0] = list_res[0].replace('IN (SELECT', 'IN (SELECT /*+
no_expand*/', 1)
        return tuple(list_res)
    return res
```

16.子查询的 no_expand 里添加 vendor 的判断

```
if connection.vendor == 'gaussdb':
```

17.django/contrib/gaussdb/locale 里把.po 文件里的 PostgreSQL 改为 GaussDB,并重新生成为.mo 文件

八、 测试验证

1. 由于 django 组件，带有 web 开发功能模块，所以可以用浏览器访问 Django 项目的

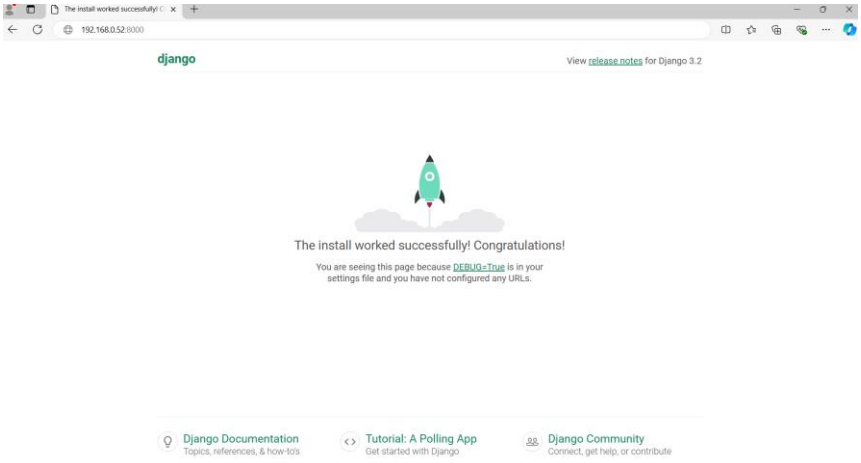
开发服务器网址。即：**GaussDB** 服务器域名或 **IP** 地址，后面跟上:8000

注意：由于我的测试是在华为云服务器 ECS 上，所以用 web 浏览器测试，需要一台 windows 的服务器，当跳板机。来访问 192.168.0.52 云服务器上的 Django 服务。

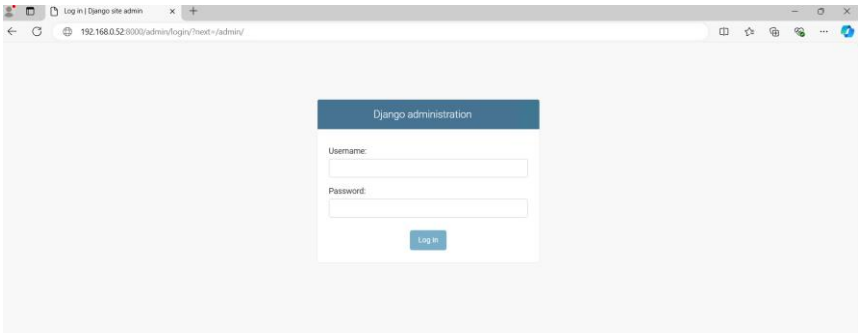
即：在浏览器中输入该连接

http://192.168.0.52:8000/　出现这样的页面说明，连接数据库成功。

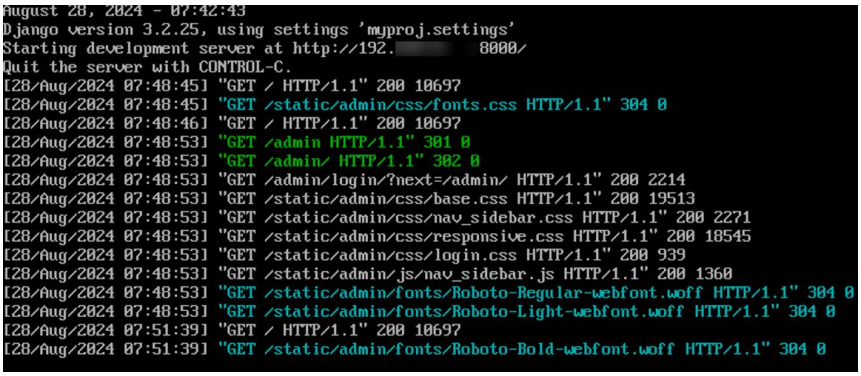
如图：



如果在地址 **URL** 末尾附加/**admin**，使用 **python3 manage.py createsuperuser** 创建的管理员操作。



如果浏览器连接成功，服务器端会显示一些日志，如下：



九、 参数资料

官方文档--Django app 说明

官方文档--自定义模型字段

官方文档--编写并运行测试