Yukon

Release 1.0.1

Aug 23, 2023

Contents


Yukon

CHAPTER 1

产品简介

Yukon（禹贡[1]），基于openGauss数据库[2]扩展地理空间数据的存储和管理能力，提供专业的GIS（Geographic Information System）功能，赋能传统关系型数据库。

Yukon 支持二三维一体化的空间数据存储能力:

1. Yukon for openGauss，适用于开源数据库openGauss和商用GaussDB数据库;
2. Yukon for PostgreSQL，适用于PostgreSQL数据库。

许可说明参见 LICENSE.TXT。

此文档在线地址: Yukon Doc

[1] 《禹贡》是《尚书》中的一篇，是中国古代文献中最古老和最有系统性地理观念的著作。
[2] openGauss 官网

CHAPTER 2

模块组织结构

目前，Yukon 基于 openGauss/PostgreSQL 扩展的模块包括：

1. postgis：与数据库适配的 PostGIS 矢量模块；
2. postgis_raster：与数据库适配的 PostGIS 栅格模块；
3. postgis_sfcgal：与数据库适配的 PostGIS 三维算法相关模块；
4. yukon_geomodel：Yukon自有的三维模型数据模块；
5. yukon_geogridcoder：Yukon自有的空间网格编码模块。

模块之间的依赖关系如图：

Fig. 1: 模块依赖图

对于与PostGIS适配的三个模块（postgis、postgis_raster、postgis_sfcgal），本文档提供入门级教程， 更详细的文档可参考 PostGIS 文档； 此外，当前版本尚未适配的能力清单见 *A.3.2* 已知问题 。

CHAPTER 3

安装与卸载

目前 Yukon 1.0.1 版本支持开源数据库openGauss和PostgreSQL，以及商用数据库GaussDB。 在安装 Yukon 之前必须先安装数据库。 开源数据库安装指导参见： openGauss 或 PostgreSQL 。 商用数据库GaussDB请联系高斯数据库服务提供商。

Note: 1、openGauss 支持的操作系统有： centos_x86_64、openeuler_aarch64、openeuler_x86_64，请优先在这些操作系统进行安装。

2、Yukon 1.0.1 兼容GaussDB的内核版本有：GaussDB_503.1.0_SPC1200、GaussDB_503.1.0_SPC1300。

数据库安装成功后，开始 Yukon 产品的安装。

3.1 获取安装包

下载地址:

链接：https://pan.baidu.com/s/1PkcGEpYjqT2sTdKm6cd-NA?pwd=c0bx 提取码：c0bx

安装包

- Yukon-1.0.1-openGauss3.1.0-CentOS_x64.tar.gz
- Yukon-1.0.1-openGauss3.1.0-openeuler_aarch64.tar.gz
- Yukon-1.0.1-postgres13-CentOS_x64.tar.gz
- Yukon_Pro-1.0.1-GaussDB_503.1.0_SPC1200-Kylin_aarch64.tar.gz
- Yukon_Pro-1.0.1-GaussDB_503.1.0_SPC1200-Kylin_x86.tar.gz

1. 从 Yukon 开源仓库下载对应平台的安装包
    1. 登录 Yukon For openGauss 源码托管网站下载适配 openGauss3.1.0的安装包。
    2. 登录 Yukon For PostgreSQL 源码托管网站下载适配 PostgreSQL13 的安装包

3. 登录账号，点击 Yukon-1.0.1-xxx-xxx.tar.gz 下载。

2. 检查安装包
   解压安装包，检查安装目录及文件是否齐全。在安装包所在目录执行以下命令：

```
mkdir yukon
tar -xf Yukon-1.0.1-xxx-xxx.tar.gz -C yukon
ls -l yukon
```

执行 ls 命令，显示类似如下信息：

```
ls -l
drwxr-xr-x   8 root root     138 Dec 29 21:52 .
dr-xr-x---. 20 root root    4096 Dec 30 14:33 ..
drwxr-xr-x   2 root root      75 Dec 29 21:52 bin
drwxr-xr-x   2 root root     278 Dec 29 21:52 data
drwxr-xr-x   2 root root    8192 Dec 29 21:52 extension
-rw-r--r--   1 root root 7649776 Dec 29 21:52 Yukon 1.0.1 说明文档.pdf
-rw-r--r--   1 root root    4367 Dec 29 21:52 install.sh
drwxr-xr-x   2 root root     220 Dec 29 21:52 lib
drwxr-xr-x   2 root root    4096 Dec 29 21:52 license
lrwxrwxrwx   1 root root      13 Dec 29 21:52 yk_tool -> ./bin/yk_tool
drwxr-xr-x   2 root root    4096 Dec 29 21:52 yukon_dep
```

3.2 软硬件环境要求

3.2.1 硬件环境要求

硬件环境建议参考 openGauss 的硬件环境要求。

3.2.2 软件环境要求

软件类型	配置描述
Linux 操作系统	ARM: - openEuler 20.03LTS - 麒麟 v10 X86: - openEuler 20.03LTS - Centos 7.6
数据库	- openGuass3.1.0 - GuassDB - PostgreSQL 13

3.3 软件依赖要求

为了尽可能的与操作系统的默认软件保持兼容，我们在软件的安装包中自带和操作系统软件源中版本一致的依赖软件，列表如下:

软件名称	**Centos(7.6)版本**	**openEuler(20.03)版本**
libtiff	4.0.3	4.1.0
libcurl	7.29.0	7.71.1
gmp	6.0.0	6.2.0
mpfr	3.1.1	4.1.0
libuuid	2.23.2	2.35.2
json-c	0.11	0.15
libxml2	2.9.1	2.9.10

3.4 GaussDB相关GUC参数要求

GUC参数详细说明和设置方法请参考GaussDB官方文档。

1、检查GUC兼容性配置参数behavior_compat_options，关闭部分GUC兼容性配置项，如果未关闭可能会出现插件功能不可用或执行结果异常的情况。

具体参数如下： allow_procedure_compile_check proc_implicit_for_loop_variable plpgsql_dependency proc_outparam_override

2、检查GUC参数是否符合以下取值要求

enable_default_ustore_table=off 说明：目前不支持ustore存储模式，默认值为on，需要关闭设置为off。

a_format_version=” 说明：必须设为’‘,默认值即为空，若值为”10c”会导致postgis_raster模块创建失败。

recovery_redo_workers=1 说明：极致RTO特性中每个ParseRedoRecord线程对应的PageRedoWorker数量，默认值即为1，不支持并行回放，若有修改请恢复默认设置。（503.2.0版本之后已修复）

3.5 安装包安装

3.5.1 单节点安装

安装时请使用数据库所属用户，这里假设 openGauss/GaussDB 数据库所属用户为 omm, PostgreSQL 所属用户为 postgres

1. 使用 omm/postgres 用户登录到 Yukon 包要安装的主机，解压 Yukon 压缩包到任意目录（假定目录为 /opt/software/yukon）

```
tar -xf Yukon-1.0.1-xxx-xxx.tar.gz -C /opt/software/yukon
```

2. 进入解压后的目录

```
cd /opt/software/yukon
```

3.检查数据库环境变量

```
--检查pg_config命令是否正常运行以及输出结果是否正确
pg_config

--检查PGDATA环境变量，输出结果是否为空以及是否正确
echo $PGDATA
--检查LD_LIBRARY_PATH环境变量，输出结果是否为空以及是否包含数据库lib目录
echo $LD_LIBRARY_PATH
```

4.配置数据库环境变量[可选]

如果检查数据库环境变量存在问题，请按照以下步骤重新配置PATH、LD_LIBRARY_PATH、PGDATA环境变量，并且配置后需要重新检查。

(1)PostgreSQL配置参考

```
--设置PGHOME环境变量，指定数据库安装目录，请根据实际数据库安装参数进行设置
export PGHOME=/opt/pg13
--设置PATH环境变量
export PATH=$PATH:$PGHOME/bin
--设置PGDATA环境变量，指定数据库的数据目录位置，请根据实际数据库安装参数进行设置
export PGDATA=$PGHOME/data
--设置LD_LIBRARY_PATH环境变量
echo export LD_LIBRARY_PATH='$LD_LIBRARY_PATH':$PGHOME/lib >>/home/pg13/.
↪bashrc
source  /home/pg13/.bashrc
```

(2)openGauss/GaussDB配置参考 openGauss/GaussDB数据库安装后会生成用户环境变量文件，PATH/LD_LIBRARY_PATH一般会自动加载,不需要手动; 如果遇到需要手动加载的情况，可直接source用户环境变量文件。

```
--手动source环境变量文件
source /home/omm/env_single
--设置PGDATA环境变量，需要根据数据库的数据目录datanode进行设置，可查看数据库配置文
件cluster_config.xml
 export PGDATA=$GAUSSHOME/dn1
```

5. 执行安装脚本安装 Yukon

```
sh install.sh -i
```

上述命令中使用 -i 参数表示安装，卸载时可使用 -r 参数。

6. 安装完成后会显示如下界面

```
\ \    / /      | |
 \ \_/ /_    _ | | __ ____    _ __
  \    /| | | || |/ // _ \ |  _ \
   | | | |_| ||   <| (_) || | | |
   |_|  \____||_|\_\\___/ |_| |_|
```

Note: openGauss 加载环境变量，使用脚本一键安装的单机版环境变量一般为 source /home/user/env_single，手动安装集群式安装环境变量一般为 source /home/user/.bashrc，完成后重新安装。

3.5.2 多节点安装

每个节点分别执行单节点安装，具体步骤参考“单节点安装”。

3.6 编译安装

1. 获取 Yukon 源码

安装 git 工具，在 Yukon 的 Gitee 仓库获取源码，以 Yukon for openGauss 为例：

```
git clone https://gitee.com/opengauss/Yukon.git
```

2. 准备编译环境

   安装下列软件依赖包

   - gcc/g++
   - gdal-devel
   - geos-devel
   - json-c-devel
   - proj-devel
   - libxml2-devel
   - sfcgal

   以上安装包可以通过源码安装，也可以直接使用 yum 工具直接安装。如果还缺少其他依赖，可自行安装。

   三方库的编译过程可参考 编译 。

3. 开始编译

   准备好编译环境以后，就可以开始进行编译安装：

```
# 如果您想使用 SFCGAL 作为 PostGIS 的处理后端，那么在准备安装环境时，需要预先安装 SFCGAL

进入Yukon目录后运行配置文件：
cd Yukon
./configure --with-pgconfig=/pg_config --without-topology  --without-address-
↪standardizer \
            --without-interrupt-tests CFLAGS='-O2 -fpermissive -DPGXC  -pthread '␣
↪CC=g++
make
make install
```

Note: 安装时可能会提示 openGauss 目录下缺少部分头文件，此时需要从 openGauss源码库 中的 src/include 目录 下拷贝对应文件到提示的目录下，出于方便也可以直接全部覆盖。

如果在安装 gdal、 geos、 json-c、 libxml2、porj、 sfcgal时自定义了安装路径，那么运行配置文件时需要添加以下的配置选项：

```
--with-gdalconfig=/path/gdal-config --with-geosconfig=/path/geos-config \
--with-xml2config=/path/xml2-config --with-jsondir=/path/json-c \
--with-projdir=/path/proj4 --with-sfcgal=/path/sfcgal-config
```

同时，需要将这些依赖的动态库的路径添加到 /etc/ld.so.conf 文件中，然后使用 `ldconfig` 命令加载路径,或者将路径添加到 *LD_LIARARY_PATH* 环境变量中。

4. 开始使用

   以上安装完成后，可以进入数据库操作。这里我们以 openGauss 为例：

```
--- 切换到 omm (数据库安装用户)
su omm
--- 连接到 postgres 数据库
gsql -d postgres
```

```
--- 创建 postgis 扩展
CREATE EXTENSION postgis;
--- 创建 postgis_raster 扩展
CREATE EXTENSION postgis_raster;
--- 创建 yukon_geomodel 扩展
CREATE EXTENSION yukon_geomodel;
--- 创建 yukon_geogridcoder 扩展
CREATE EXTENSION yukon_geogridcoder;
-- 如果在编译时支持了 SFCGAL 则可以创建 SFCGAL 扩展
CREATE EXTENSION postgis_sfcgal;
```

3.7 容器安装

目前 1.0.1 版本提供的 Docker 镜像如下：

- supermap/yukon:1.0.1-opengauss3.1.0-amd64
- supermap/yukon:1.0.1-opengauss3.1.0-arm64
- supermap/yukon:1.0.1-postgresql13-amd64

3.7.1 启动Yukon

```
--Yukon for openGauss
docker run --name Yukon --privileged=true -d -e GS_PASSWORD=Bigdata@123 supermap/
↪yukon:1.0.1-opengauss3.1.0-amd64
--Yukon for PostgreSQL
docker run --name Yukon --privileged=true -d -e POSTGRES_PASSWORD=Bigdata@123␣
↪supermap/yukon:1.0.1-postgresql13-amd64
```

3.7.2 环境变量

openGauss镜像支持以下变量的设定：

- `GS_PASSWORD`： **【必须设置】**，该参数设置了 openGauss 数据库的超级用户 omm 以及外部连接用户 gaussdb 的密码。openGauss 安装时默认会创建 omm 超级用户，测试用户 gaussdb 是在 entrypoint.sh 中自定义创建的用户，openGauss 设置的密码要符合复杂度要求：

  - 最少包含8个字符。
  - 不能和用户名、当前密码（ALTER）、或当前密码反序相同。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字，非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。
- GS_NODENAME：【可选设置】，该参数设置了 openGauss 数据库的节点名称，默认为 gaussdb，可以在 entrypoint.sh 中进行查看
- GS_USERNAME：【可选设置】，该参数设置了 openGauss 数据库的外部连接用户名，默认为 gaussdb，可以在 entrypoint.sh 中进行查看
- GS_PORT:【可选设置】，该参数设置了 openGauss 数据库的连接端口，默认为 5432，可以在 entrypoint.sh 中进行查看。

PostgreSQL 镜像支持以下变量的设定：

- POSTGRES_PASSWORD：**【必须设置】**，设置 PostgreSQL 密码

3.7.3 体验 Yukon

稍等片刻，等 Yukon 启动后，可以使用如下命令体验 Yukon。以 Yukon for openGauss 为例：

```
sudo docker exec -it Yukon  /bin/bash
su omm
gsql -d postgres
```

连接成功后，可以看到类似如下的字符：

```
[omm@367dd3bb5d10 /]$ gsql -d postgres
    gsql ((openGauss 3.1.0 build 590b0f8e) compiled at 2021-09-30 14:29:04 commit 0␣
↪last mr  )
    Non-SSL connection (SSL connection is recommended when requiring high-security)
    Type "help" for help.

openGauss=#
```

postgres 数据库默认已经创建了 *postgis* , *postgis_raster* , *postgis_sfcgal* , *yukon_geomodel*, *yukon_geogridcoder* 扩展，可以直接使用。

```
-- 查询 yukon_geomodel 的版本及其编译时间
select yukon_version();
-- 输出 1.0.1 Compiled at:2022-12-24 08:49:42 Commit ID:e1cdae4
```

3.7.4 外部连接数据库

openGauss 的默认监听端口为 5432，如果想要从容器外部访问数据库，则需要在 docker run 的时候指定 -p 进行端口映射：

```
docker run --name Yukon --privileged=true -d -e GS_PASSWORD=Bigdata@123 -p 5432:5432␣
↪supermap/yukon:1.0.1-opengauss3.1.0-amd64
```

现在就可以使用 openGauss 的数据库管理工具 DataStudio，或者使用开源的 DBeaver 连接数据库（第一次连接时间可能较长，需要等待数据库安装初始化完成）。

3.7.5 持久化存储数据

容器一旦被删除，容器内的所有数据和配置也均会丢失，而从镜像重新运行一个容器的话，则所有数据又都是呈现在初始化状态，因此对于数据库容器来说，为了防止因为容器的消亡或者损坏导致的数据丢失，需要进行持久化存储数据的操作。通过在 `docker run` 的时候指定 `-v` 参数来实现，或者可以指定已经建立好的 Volume 。使用以下命令将 openGauss 的所有数据文件存储在宿主机的 /Yukon/opengauss 下。

```
mkdir -p /Yukon/opengauss
docker run --name Yukon --privileged=true -d -e GS_PASSWORD=Bigdata@123 \
    -v /Yukon/opengauss:/var/lib/opengauss  -p 5432:5432 \
    supermap/yukon:1.0.1-opengauss3.1.0-amd64
```

3.8 主备节点安装

如果是在 openGauss 集群下安装 Yukon，需要在主机间执行命令，传送文件等操作。在主机之间建立互信的基础上，只需执行安装包里的脚本文件 `sh install_cluster.sh -f hostfile -U username` 即可（openGauss安装集群时，会在主机间建立互信，如果此时没有互信，可以再重新手动建立互信）。

- `hostfile`：存放有openGauss中所有主机IP的配置文件。

例： /opt/software/openGauss> vim hostfile

- `username`：openGauss集群所使用的安装用户，默认为 `omm` 用户。

3.9 数据库参数修改

数据库参数的修改均位于数据目录下的 `postgresql.conf` 和 `pg_hba.conf` 两个文件中，下面为假定的配置文件所在位置：

openGauss/GaussDB

```
/opt/openGauss/cluster/dn1/postgresql.conf
/opt/openGauss/cluster/dn1/pg_hba.conf
```

PostgreSQL13

```
/opt/postgres13/data/postgresql.conf
/opt/postgres13/data/pg_hba.conf
```

1. 修改 openGauss/GaussDB 的加密方式

   对于 openGauss/GaussDB 来说，需要将其加密方式改为 MD5，否则将会导致数据库工具软件无法连接。

   ```
   # openGauss/GaussDB: /opt/openGauss/cluster/dn1/postgresql.conf
   将 password_encryption_type 修改为 0 ，并取消注释
   password_encryption_type = 0
   ```

2. 修改监听地址

   一般情况下，我们并不会只在本地连接到数据库，也会在其他主机连接数据库，因此，我们需要修改一下监听的网卡 IP 地址，使其他主机也能够连接到数据库。这里我们修改为 `*` 表示监听所有网卡地址。

```
# openGauss/GaussDB: /opt/openGauss/cluster/dn1/postgresql.conf
# PostgreSQL: /opt/postgres13/data/postgresql.conf
listen_addresses = '*'          # what IP address(es) to listen on;
```

3. 修改可接受的远程 IP 地址

   一般情况下，我们会通过 IP 将数据库可接受的连接接限制在某个范围。因此我们可以在 `pg_hba.conf` 加入下面的一行表示可接受位于 `192.168.13.1/24` 网段内的所有主机。如果想接受所有请求，可以将子网掩码设置为 0.

```
# openGauss/GaussDB: /opt/openGauss/cluster/dn1/pg_hba.conf
# PostgreSQL: /opt/postgres13/data/pg_hba.conf
host      all                             all             192.168.13.1/24
↪          md5
```

4. openGauss/GaussDB 数据库提供了gs_guc工具设置配置文件（“postgresql.conf”、“pg_hba.conf”）中的参数，详见 openGauss

---

Note: gs_guc 工具不支持包含’.’符号的参数设置。例如：postgis.backend 无法通过 gs_guc 进行配置。可以通过 set 命令进行修改。

数据库参数修改完成后重新启动数据库，使参数生效。

3.10 验证

在本节我们会使用一款开源的数据库工具软件 DBeaver(22.3.0) 来连接数据库。

1. 创建用户

   - openGauss/GaussDB

     openGauss/GaussDB 在外部连接时不能使用 omm 用户，因此这里我们新建一个用户用于测试，后续您可以将其删除。

     1. 连接到数据库

```
gsql -d postgres -p 5432
```

     2. 创建用户

        创建一个用户名为 test 密码为 Bigdata@123 的新用户。

```
CREATE USER test with PASSWORD 'Bigdata@123' SYSADMIN;
```

        输出如下则说明创建成功：

```
NOTICE:  The encrypted password contains MD5 ciphertext, which is not
↪secure.
CREATE ROLE
```

        如果没有输出 MD5 的提示，请检查是否将 openGauss/GaussDB 的加密方式修改为 MD5。

   - PostgreSQL

     1. 连接到数据库

```
psql -d postgres -p 5432
```

2. 创建用户

   创建一个用户名为 test 密码为 Bigdata@123 的新用户。

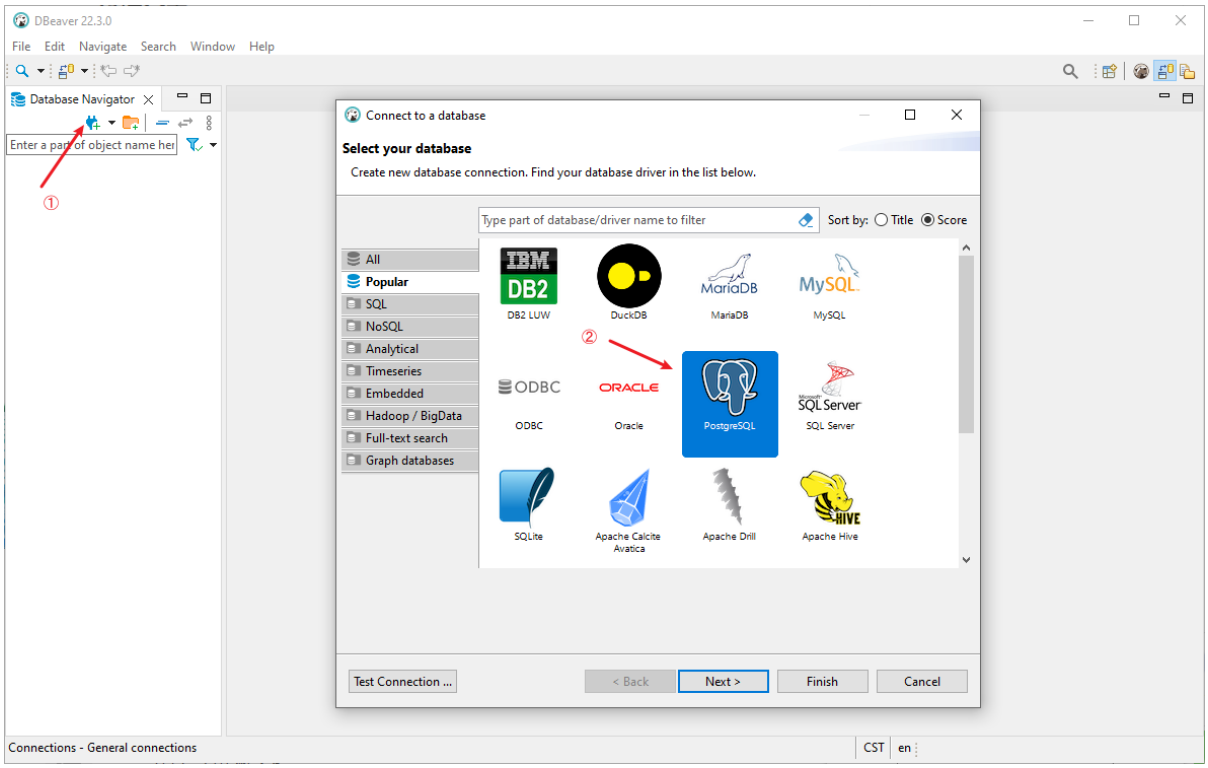
```
CREATE USER test WITH PASSWORD 'Bigdata@123' Superuser;
```

   输出如下则说明创建成功:

```
CREATE ROLE
```

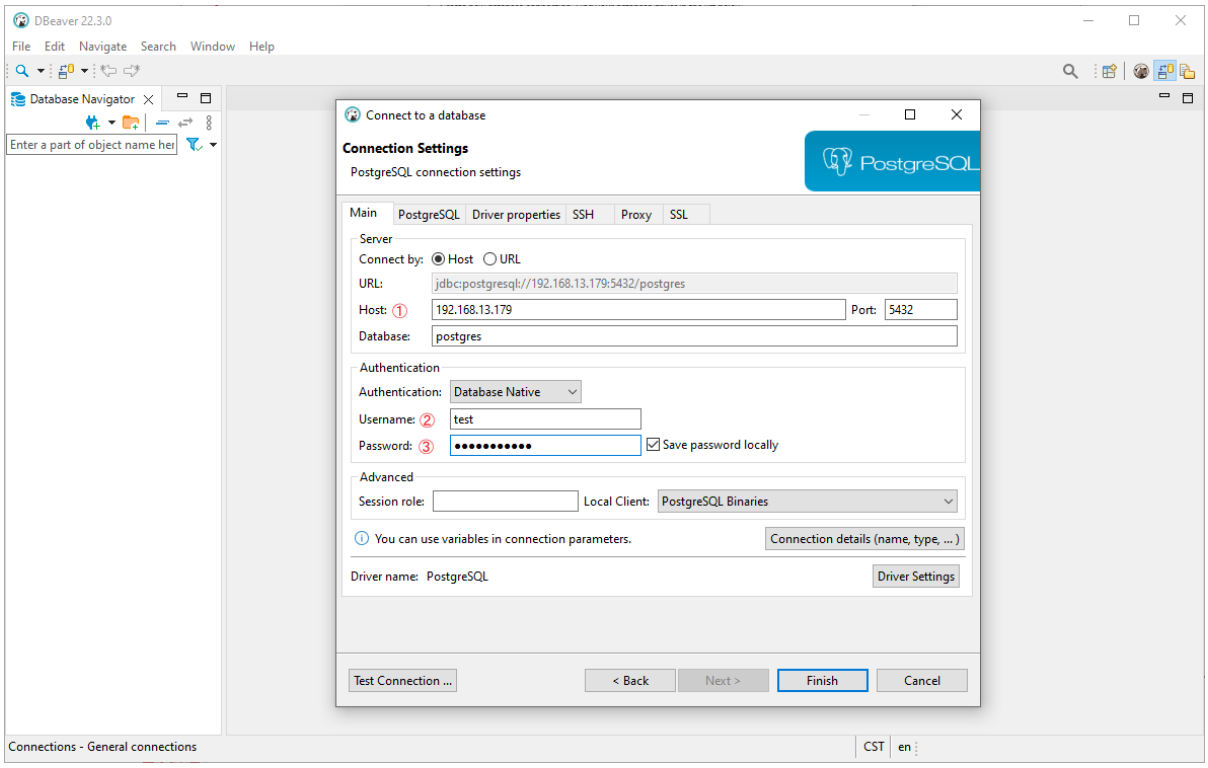
2. 新建一个 数据库连接

   这里我们统一使用 PostgreSQL JDBC 来连接数据库



1. 输入数据库连接信息

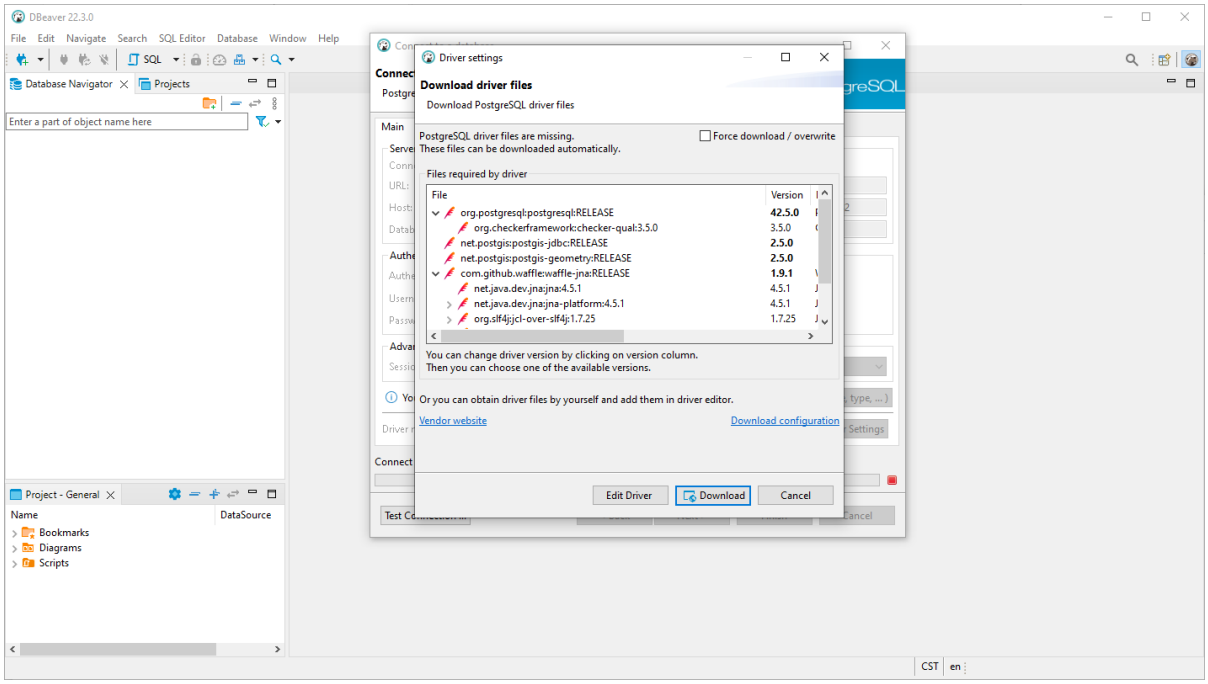
   填写我们第一步创建好的用户信息



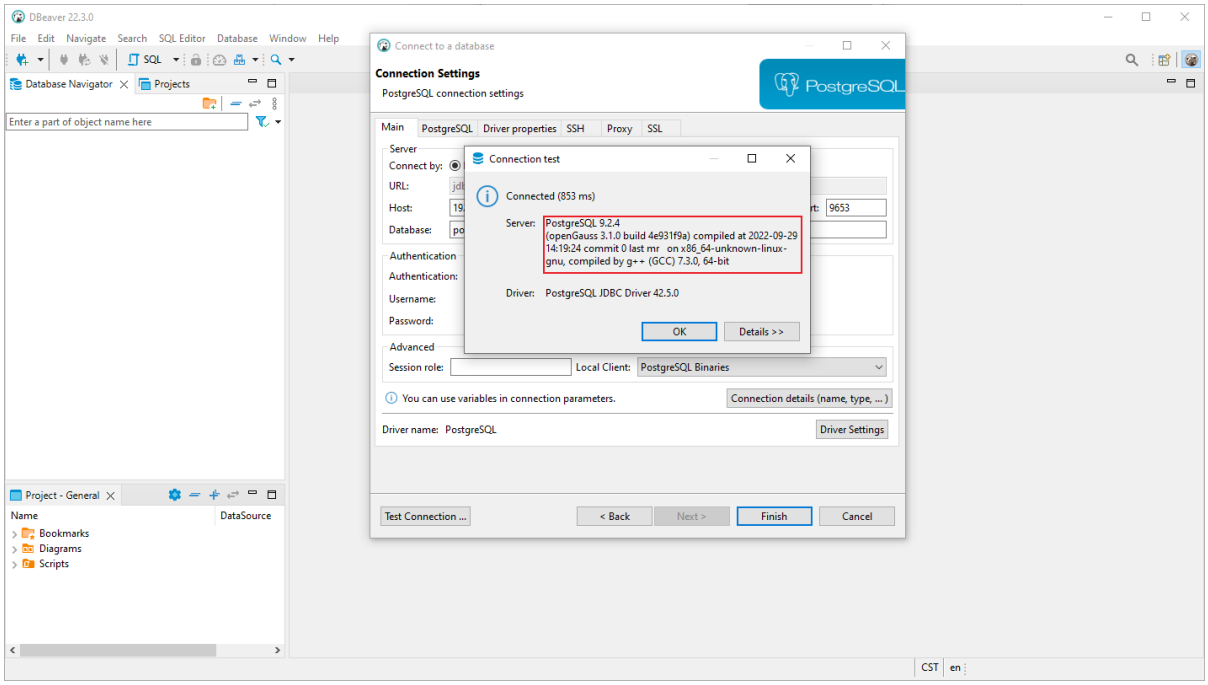
在 Host 位置写入数据库主机的 IP 地址，在 Username 写入我们刚刚创建的用户名 `test`，在 Password 位置写入我们的密码 `Bigdata@123`。

2. 测试连接

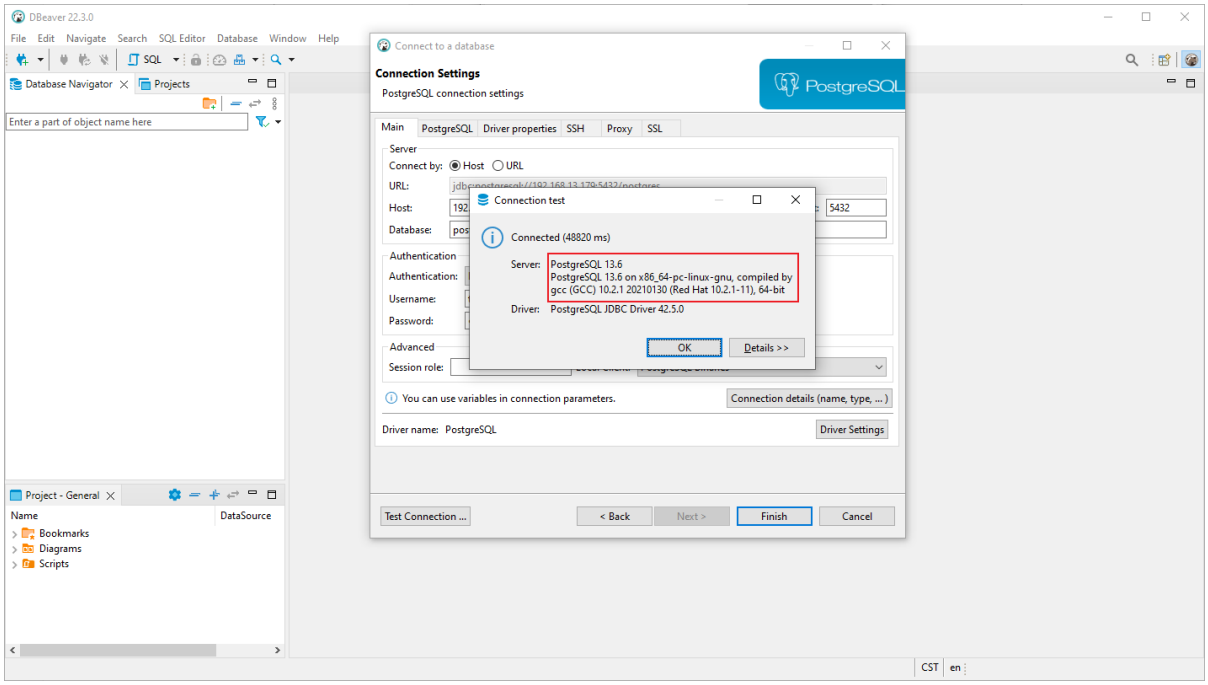
点击左下角的 Test Connection 测试是否可以连接到数据库。如果连接成功会显示数据库的信息。第一次连接时需要下载 JDBC 驱动，点击下载即可。



openGauss/GaussDB 连接成功时，显示如下：



PostgreSQL 连接成功时，显示如下：



3. 显示矢量数据

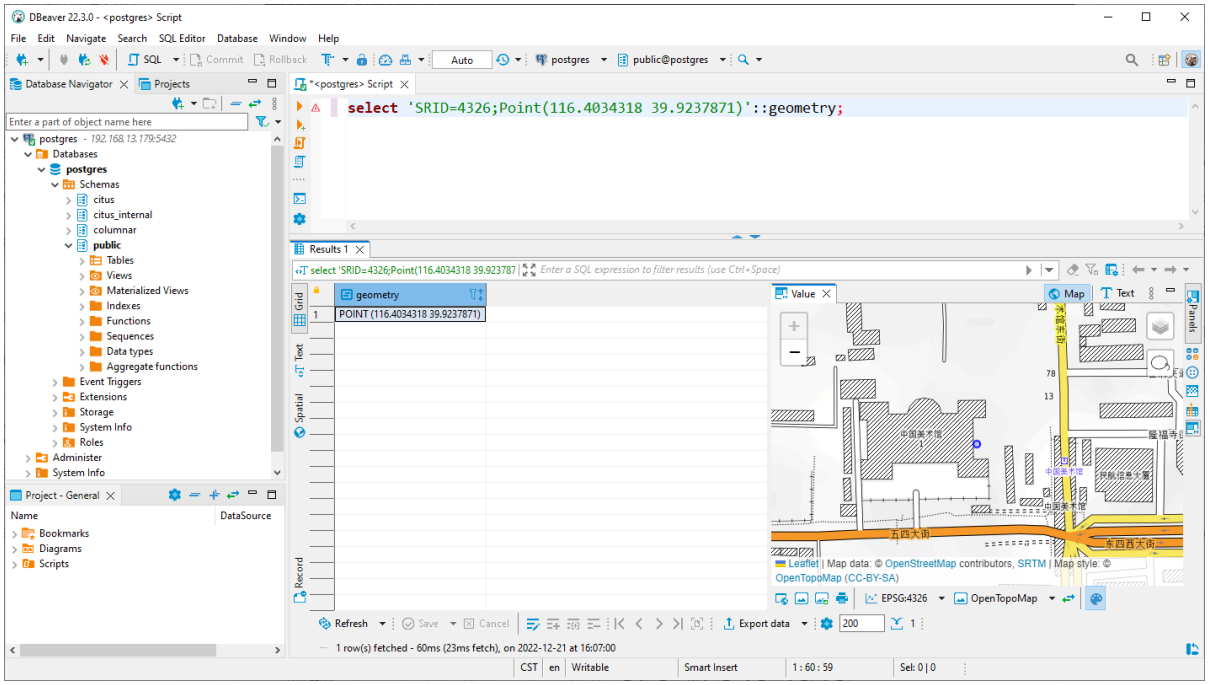
   DBeaver 自带了显示 GIS 数据的能力，我们可以使用他来查看 GIS 数据。

   首先创建 postgis 扩展

```
CREATE EXTENSION POSTGIS;
```

   查询矢量数据

```
select 'SRID=4326;Point(116.4034318 39.9237871)'::geometry;
```



3.11 升级

Yukon提供扩展模块的升级功能，支持从1.0升级到当前1.0.1版本。 以Yukon for openGauss为例，操作过程:

安装新版本Yukon 1.0.1，登录到需要升级的数据库，对不同的扩展模块执行SQL:

```
ALTER EXTENSION name UPDATE [ TO new_version ]
```

例子:

要将yukon_geogridcoder更新到版本 1.0.1，请执行以下操作:

```
#升级到最新版本
alter extension yukon_geogridcoder update;
```

或

```
#升级到指定版本
alter extension yukon_geogridcoder update to '1.1';
```

Note: 建议重启数据库后再进行升级操作,以及openGauss/GaussDB需要在postgresql.conf中添加兼容参数support_extended_features=on，才能正常升级，参考 openGauss文档 。

3.12 卸载

如果使用安装包进行安装，执行 `sh install.sh -r` 命令即可卸载，也可以使用yk_tool -r 进行卸载，详

情见yk_tool工具说明 *yk_tool* 。

如果使用源码进行编译安装，在源码根目录下，执行命令 `make uninstall` 进行卸载。

如果使用 Docker 进行安装，直接删除 Docker 容器和镜像即可。

您也可以手动检查以下三个地方查看是否有卸载残留，以openGauss为例：

- `$GAUSSHOME/lib/postgresql/` 目录下与 postgis 和 yukon 相关的文件
- `$GAUSSHOME/share/postgresql/extension/` 目录下与 postgis 和 yukon 相关的文件
- `$GAUSSHOME/lib` 目录下是否有yukon相关的三方库文件

CHAPTER 4

入门教程

4.1 准备工作

在使用空间数据库之前，我们需要了解用户及其权限、数据库的字符集和表空间等数据库基础内容。如果您已具备相关基础，可以略过本节。

4.1.1 创建用户

在安装 openGauss/PostgreSQL 数据库时，会自动创建一个初始用户，初始用户具有系统的最高权限，能够执行所有的操作。这里我们通过命令行登录到数据库，并创建一个新的系统管理员用户。

```
# 登录到数据库
--openGauss
gsql -d postgres
--PostgreSQL
psql -d postgres

# 创建新用户 yukontest
--openGauss
CREATE USER yukontest WITH SYSADMIN password "Bigdata@123";
--PostgreSQL
CREATE USER yukontest WITH SUPERUSER password 'Bigdata@123' ;
```

如果新创建的用户不带有 SYSADMIN / SUPERUSER 属性，很多操作都会没有权限。关于用户和权限的相关问题可以参考 openGauss 文档 、PostgreSQL ;

4.1.2 创建表空间

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

- 如果初始化数据库所在的分区或者卷空间已满，又不能逻辑上扩展更多空间，可以在不同的分区上创建和使用表空间

- 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能
  - 一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种固态设备
  - 一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上
- 管理员通过表空间可以设置占用的磁盘空间。用以在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间

这里我们使用刚才创建的 yukontest 用户来创建一个表空间

```
# 登录到数据库(openGauss为例)
gsql -d postgres -U yukontest -W Bigdata@123

# 创建表空间 yukonspace
CREATE TABLESPACE yukonspace LOCATION '/home/omm/data';
```

注意：目录 `/home/omm/data` 必须已经存在且具有可访问权限。

4.1.3 创建数据库

创建一个新的数据库。缺省情况下新的数据库将通过复制标准系统数据库 `template0` 来创建。且仅支持使用 `template0` 来创建

Note:

1. 只有拥有 CREATEDB 权限的用户才可以创建新数据库，系统管理员默认拥有此权限。
2. 不能在事务块中执行创建数据库语句。
3. 在创建数据库过程中，出现类似 `Permission denied` 的错误提示，可能是由于文件系统上数据目录的权限不足。出现类似 `No space left on device` 的错误提示，可能是由于磁盘满引起的。

如果您在数据库中存储中文字符，推荐选择 GBK 编码，当然您也可以选择 UTF8 编码。

这里我们创建一个 yukontutorial 数据库来进行后面的学习。

```
# 使用 yukontest 用户连接到 postgres 数据库(openGauss为例)
gsql -d postgres -U yukontest -W Bigdata@123

# 创建数据库 yukontutorial ，并指定数据库编码为 'UTF8' 表空间为上面创建好的 yukonspace
CREATE DATABASE yukontutorial ENCODING='UTF8' TABLESPACE=yukonspace;
```

有关更多创建数据库的操作可以参考 openGauss 的文档

4.1.4 创建扩展

安装一个扩展,Yukon目前共提供 postgis、postgis_raster、postgis_sfcgal、yukon_geomodel、yukon_geogridcoder五个扩展，postgis为基础扩展，被其他扩展依赖，因此安装其他扩展之前请先安装基础扩展;

语法

PostgreSQL

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
    [ VERSION version ]
    [ CASCADE ]
```

openGauss/GaussDB

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ FROM old_version ]
```

参数说明

IF NOT EXISTS

如果系统已经存在一个同名的扩展，不会报错。这种情况下会给出一个提示。请注意该参数不保证系统存在的扩展和现在脚本创建的扩展相同。

extension_name

将被安装扩展的名字。

schema_name

扩展的实例被安装在该模式下，扩展的内容可以被重新安装。指定的模式必须已经存在，如果没有指定，扩展的控制文件也不指定一个模式，这样将使用默认模式。

Note: 注意： 扩展不认为它在任何模式里面：扩展在一个数据库范围内的名字是不受限制的，但是一个扩展的实例是属于一个模式的。

---

version

安装扩展的版本，可以写为一个标识符或者字符串.默认的版本在扩展的控制文件中指定。

CASCADE

自动安装此扩展所依赖但尚未安装的任何扩展。它们的依赖项同样以递归方式自动安装。该子句（如果给定）适用于以这种方式安装的所有扩展。语句的其他选项不适用于自动安装的扩展;特别是，始终选择其默认版本

old_version

当你想升级安装“old style” 模块中没有的内容时,你必须指定FROM old_version。这个选项使CREATE EXTENSION 运行一个安装脚本将新的内容安装到扩展中，而不是创建一个新的实体.注意SCHEMA指定了包括这些已存在实体的模式。

示例

```
--在当前数据库安装postgis扩展
CREATE EXTENSION postgis;

--在当前数据库安装postgis扩展到指定模式下
CREATE EXTENSION postgis with schema schema_name;
```

Note: 建议创建扩展时使用 [WITH] [SCHEMA schema_name] 参数进行扩展，解决权限有关问题。

4.2 使用矢量数据

4.2.1 创建 postgis 扩展

在命令行使用 yukontest 用户连接到 yukontutorial 数据库：

```
# 连接到数据库(openGauss为例)
gsql -d yukontutorial -U yukontest -W Bigdata@123
```

```
--创建 postgis 扩展
CREATE EXTENSION postgis;
```

显示如下信息，则表示创建成功：

CREATE EXTENSION

4.2.2 写入数据

新建表格 global_points，并写入点数据：

```
-- 创建带geometry列的表
CREATE TABLE global_points (id SERIAL PRIMARY KEY,name VARCHAR(64),location␣
↪geometry(POINT,3857));

-- 写入10个点对象
INSERT INTO global_points (name, location) VALUES ('Town1', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12057531.405152922 4096788.3009192282)') );
INSERT INTO global_points (name, location) VALUES ('Town2', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12057879.323089447 4096794.06898925)') );
INSERT INTO global_points (name, location) VALUES ('Town3', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12057241.150712628 4096835.404659481)') );
INSERT INTO global_points (name, location) VALUES ('Town4', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12057396.37461059 4096885.372392862)') );
INSERT INTO global_points (name, location) VALUES ('Town5', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12056940.710538926 4096886.9123589676)') );
INSERT INTO global_points (name, location) VALUES ('Town6', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12058128.24460281 4096938.23117457)') );
INSERT INTO global_points (name, location) VALUES ('Town7', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12058378.134595742 4097081.4360874156)') );
INSERT INTO global_points (name, location) VALUES ('Town8', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12058636.607321415 4097235.1257181372)') );
INSERT INTO global_points (name, location) VALUES ('Town9', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12058917.488660585 4097395.7746241256)') );
INSERT INTO global_points (name, location) VALUES ('Town10', ST_GeomFromEWKT(
↪'SRID=3857;POINT (12059180.102471316 4097548.7255362184)') );
```

4.2.3 空间查询

查询距离指定点500米范围内的对象：

```
SELECT * FROM global_points WHERE ST_DWithin(location, ST_GeomFromEWKT('SRID=3857;
↪POINT (12058378 4097081)'), 500);
```

输出结果：

```
 id | name  |                        location
----+-------+-----------------------------------------------------
  6 | Town6 | 0101000020110F000046C9D307C2FF6641D920971DD5414F41
  7 | Town7 | 0101000020110F0000BB9B4E44E1FF664162B6D1B71C424F41
  8 | Town8 | 0101000020110F0000522D6F93010067412C88179069424F41
(3 rows)
```

4.3 栅格数据

4.3.1 创建 postgis_raster 扩展

在命令行使用 yukontest 用户连接到 yukontutorial 数据库:

```
# 连接到数据库(openGauss为例)
gsql -d yukontutorial -U yukontest -W Bigdata@123
```

```
--创建 postgis_raster 扩展
CREATE EXTENSION postgis_raster;
```

显示如下信息，则表示创建成功:

CREATE EXTENSION

创建成功后，退出 gsql 控制台。

4.3.2 导入数据

用 raster2pgsql 工具导入范例数据 hillshade.tif 数据(openGauss为例):

```
raster2pgsql -s 0 /SampleData/hillshade.tif -t 256x256 | gsql -d yukontutorial
```

导入成功后，再次登录数据库，刷新元数据信息:

```
# 连接到数据库
gsql -d yukontutorial -U yukontest -W Bigdata@123
# 刷新元数据信息
select AddRasterConstraints('hillshade', 'rast');
# 查看信息
select * from raster_columns where r_table_name='hillshade';
```

4.3.3 使用栅格数据

提取指定像素的值:

```
select ST_Value(rast, 1, 10, 10, true) from hillshade ;
```

CHAPTER 5

接口参考

5.1 三维模型数据模块

yukon_geomodel 模块提供对空间三维模型数据的存储和管理。使用前数据库需创建 postgis 和 yukon_geomodel 扩展。相关参考见：三维模型数据组织及存储格式、矢量面拉伸构建三维模型对象。

5.1.1 数据类型

GEOMODEL

三维模型几何对象类，其类型包括MESH、SURFACE、ANYTYPE。支持客户端以二进制形式读写。

MODEL_ELEM

三维模型元素类，其类型包括EntitySkeleton（骨架）、EntityMaterial3D（材质）、EntityTexture（纹理）。支持客户端以二进制形式读写。

GEOMODEL和MODEL_ELEM数据不能直接使用DDL命令操作，需通过数据库函数（如AddGeomodelColumn()、DropGeomodelColumn()）来完成数据的新增、删除等操作。

5.1.2 管理函数

AddGeoModelColumn

向指定的表格添加 geomodel 列，并自动创建子表。

语法

```
text AddGeoModelColumn(schema_name varchar, table_name varchar, column_name varchar,␣
↪new_srid_in integer, _type varchar DEFAULT 'ANYTYPE');

参数:
    schema_name - 模式名称
    table_name - 表名
    column_name - 列名
    new_srid_in - 坐标系
    _type - geomodel类型: ANYTYPE、MESH、SURFACE
返回:
    成功返回success,失败返回fail。
```

示例

```
SELECT AddGeoModelColumn('public','geomodel_test','geog',4326);

------------------
success
```

DropGeomodelColumn

删除指定表格的 geomodel 列，并自动删除子表。

语法

```
text DropGeomodelColumn(schema_name varchar, table_name varchar);

参数:
    schema_name - 模式名称
    table_name - 表名
返回:
    成功返回success,失败返回fail。
```

示例

```
SELECT DropGeomodelColumn('public', 'geomodel_test');

------------------
success
```

DropGeomodelTable

删除带 geomodel 列的表及其子表。

语法

```
text DropGeomodelTable(varchar schema_name, varchar table_name);

参数:
    schema_name - 模式名称
    table_name - 表名
返回:
    成功返回success,失败返回fail。
```

示例

```
SELECT DropGeomodelTable('public', 'geomodel_test');

-----------------
success
```

UpdateGeomodelSRID

更新 geomodel 列的 SRID，包括对象和元数据。

语法

```
text UpdateGeomodelSRID(varchar schema_name, varchar table_name, varchar column_name,␣
↪integer srid);

参数:
    schema_name - 模式名称
    table_name - 表名
    column_name - 列名
    srid - 指定的空间参考标识符
返回:
    成功返回success,失败返回fail。
```

示例

```
SELECT UpdateGeomodelSRID('public', 'geomodel_test','geog',4326);

-----------------
success
```

5.1.3 构造函数

ST_MakeSkeletonFromTIN

从 TIN 对象构建模型的骨架。

语法

```
model_elem ST_MakeSkeletonFromTIN(geometry geom,text skeleton_name,text material_
↪name);

参数:
    geom - geometry对象，TIN 类型
    skeleton_name - 骨架的名字
    material_name - 骨架上挂接的材质名
返回:
    返回模型对象的骨架。
```

示例

无

ST_MakeDefaultMaterial

构造一个默认的材质对象。

语法

```
model_elem ST_MakeDefaultMaterial(text material_name);

参数:
    material_name - 骨架上挂接的材质名
返回:
    返回模型对象的材质。
```

示例

无

ST_MakeGeomodel

从骨架对象构造 geomodel 对象。

语法

```
geomodel ST_MakeGeomodel(model_elem objskeleton);

参数:
    objskeleton - 骨架对象
返回:
    返回 geomodel 对象。
```

示例

无

ST_MakeHashID

模型对象的名称转为 Hash 值，作为子表的 ID 唯一标识。

语法

```
int8 ST_MakeHashID(text elemname);

参数:
    elemname - 模型对象的名称
返回:
    返回 Hash 值。
```

示例

```
SELECT  ST_MakeHashID('material1');

---------------------
6152814707455701482
```

5.1.4 空间索引

对 geomodel 列创建 GIST 索引。

语法

```
GIST(modelcol);

参数:
    modelcol - geomodel 列名
```

示例

```
CREATE INDEX building_tree ON building USING GIST (modelcol);
```

5.1.5 操作函数

ST_GeoModelType

获取 geomodel 列的类型，值域：MESH、SURFACE、ANYTYPE。

语法

```
text ST_GeoModelType(geomodel geog);

参数:
    geog - geomodel对象
返回:
    返回geomodel类型，值域：MESH、SURFACE、ANYTYPE。
```

示例

```
select  ST_GeoModelType(
↪'0100000018000000031313 95F3030303030303030383732443646453130 5F47656F3B0000005B505F5A3031303032333372E6A
↪'::geomodel);

----------------
MESH
```

ST_Boundary

获取 geomodel 对象的包络盒。

语法

```
box3d ST_Boundary(geomodel geog);

参数:
    geog - geomodel对象
返回:
    返回 geomodel 对象的包络盒几何。
```

示例

```
SELECT
ST_BOUNDARY(
↪'0000000DECE3E242ECE3E2425299B1415299B141ACA5C24AC0A5C24A20030000019E359B197B5C5C40F43AB644F832364(
↪'::geomodel);
```

(continues on next page)

(continued from previous page)

```
------------
BOX3D(113.445159912109 22.1998634338379 6378198,113.445159912109 22.1998634338379␣
↪6378208)
```

ST_3DExtent

获取 geomodel 对象的包络盒。

语法

```
box3d ST_3DExtent(geomodel geog);

参数:
    geog - geomodel对象
返回:
    返回 geomodel 对象的包络盒几何。
```

示例

```
SELECT
ST_3DExtent(
↪'0000000DECE3E242ECE3E2425299B1415299B141ACA5C24AC0A5C24A20030000019E359B197B5C5C40F43AB644F832364(
↪'::geomodel);

------------
BOX3D(113.445159912109 22.1998634338379 6378198,113.445159912109 22.1998634338379␣
↪6378208)
```

ST_Extent

获取 geomodel 对象的最小外接矩形。

语法

```
box2d ST_Extent(geomodel geog);

参数:
    geog - geomodel对象
返回:
    返回 geomodel 对象的最小外接矩形。
```

示例

```
SELECT
ST_Extent(
↪'0000000DECE3E242ECE3E2425299B1415299B141ACA5C24AC0A5C24A20030000019E359B197B5C5C40F43AB644F832364(
↪'::geomodel);

------------
BOX(113.445159912109 22.1998634338379,113.445159912109 22.1998634338379)
```

ST_SRID

获得 geomodel 对象的空间参考标识码epsg code。

语法

```
integer ST_SRID(geomodel geog);

参数:
    geog - geomodel对象
返回:
    返回 geomodel 对象的空间参考标识码epsg code。
```

示例

```
SELECT ST_SRID('0010E60DC0E3E242CB'::geomodel);

-------------
4326
```

ST_SetSRID

设置 geomodel 对象的空间参考标识码epsg code。

语法

```
geomodel ST_SetSRID(geom geomodel, srid integer);

参数:
    geog - geomodel对象
    srid - epsg code，空间参考标识码
返回:
    返回已设置指定空间参考系的 geomodel 对象。
```

示例

```
SELECT ST_SetSRID('0000000DC0E3E242CB'::geomodel, 4326);

-------------------
0010E60DC0E3E242CB
```

5.1.6 操作符

&&

判断 geomodel 对象和 geometry 对象是否相交。

语法

```
boolean Operator &&;

返回:
    相交返回 true，不相交返回 false。
```

示例

```
select '0000000DC0E3E242CB'::geomodel && st_makepoint(1, 1);
```

(continues on next page)

(continued from previous page)

```
------------
false
```

5.2 空间网格编码

yukon_geogridcoder 模块提供空间网格编码的相关操作，目前支持 GeoSOT 编码。使用前数据库需创建 yukon_geogridcoder 扩展。 相关参考见： *Yukon*支持*GeoSOT*编码的基本原理及特性、 *Yukon*中*GeoSOT* 编码的基本能力、 基于*GeoSOT*编码的多图层穿越查询。

5.2.1 数据类型

GeoSOTGrid

空间网格编码数据类型。 支持为 GeoSOTGrid 列创建 B树索引，支持为 GeoSOTGrid 数组创建 GIN 索引（Generalized Inverted Index）。

5.2.2 网格构造

ST_GeoSOTGrid

获取 geometry 对象的网格编码，默认获取对象的最小外包网格编码，也可设置指定层级，获取对象指定层级的空间网格编码。

语法

```
geosotgrid[] ST_GeoSOTGrid (geom geometry, int level);
geosotgrid[] ST_GeoSOTGrid (geom geometry);

参数:
   geometry - 几何对象
   level - 网格编码层级
返回:
   空间对象的空间网格编码。
```

示例

```
SELECT ST_GeoSOTGrid(st_geomfromtext('POINT(116.315  39.91027777777778)', 4490), 15);

---------------------------
{074EACB000000000000F0000}
```

ST_GeoSOTGridAgg

获取 geometry 对象在指定层级范围内的网格编码。

语法

```
geosotgrid[] ST_GeoSOTGridAgg(geom geometry, int level_max, int level_min);

参数:
   geometry - 几何对象
   level_max - 最大编码层级
   level_min - 最小编码层级
返回:
   空间对象的空间网格编码。
```

示例

```
SELECT ST_GeoSOTGridAgg('srid=4490;polygon((0.1 0.1, 0.2 0.2, 0.3 0.1, 0.1 0.1))
↪'::geometry, 11, 9);

----------------------------
{00000000000000000000B0000,00000400000000000000B0000}
```

ST_GeoSOTGridZ

返回 z 方向上指定层级的网格编号，用海拔基准面开始向上的第 N 层网格来表示。

语法

```
int ST_GeoSOTGridZ(float8 altitude, int level);

参数:
   altitude - 海拔高度
   level - 网格编码层级
返回:
   z 方向上的网格编号。
```

示例

```
SELECT ST_GeoSOTGridZ(9300, 15);

----------
5
```

ST_Aggregate

通过精细层网格编码聚合粗糙层网格编码。

语法

```
geosotgrid ST_Aggregate(geosotgrid grid, int level);
geosotgrid[] ST_Aggregate(geosotgrid[] gridarray,int level);

参数:
   grid - 网格编码
   gridarray - 网格编码数组
   level - 网格编码层级
返回:
   空间对象的空间网格编码。
```

示例

```
SELECT ST_Aggregate('074EA000000000000A0A0000'::geosotgrid, 9);

---------------------------
074E80000000000009090000
```

5.2.3 网格访问

ST_GetLevel

获取网格对象的编码层级。

语法

```
int ST_GetLevel(grid geosotgrid);

参数:
   geosotgrid - 网格对象
返回:
   网格编码层级。
```

示例

```
SELECT ST_GetLevel('000004000000000000B0000');

---------------------------
11
```

ST_GetLevelExtremum

获取网格编码数组的编码层级范围。

语法

```
int[] ST_GetLevelextremum(geosotgrid[] gridarray);

参数:
   gridarray - 网格编码数组
返回:
   网格编码层级范围,{最小层级level_min，最大层级level_max}
```

示例

```
SELECT ST_GetLevelextremum(array['074526EEF400000013130000','07452C4701C0000013150000
↪'::geosotgrid]);

---------------------------
{19,21}
```

ST_HasZ

网格编码是否含Z方向。

语法

```
bool ST_HasZ(geosotgrid grid);

参数:
   grid - 网格对象
返回:
   网格编码含z方向返回true，不含z方向返回false。
```

示例

```
SELECT ST_HasZ(unnest(ST_GeoSOTGrid(st_geomfromtext('POINT(116.315 39.91027777777778)
↪', 4490), 15)));

---------------------------
false
```

ST_AltitudeFromGeoSOTGridZ

返回 Z 方向指定层级下第 N 个网格对应的海拔高度，单位米。

语法

```
float ST_AltitudeFromGeoSOTGridZ(int4 z_num,int level);

参数:
   z_num - Z方向上的第几个网格
   level - 网格编码层级
返回:
   海拔高度。
```

示例

```
SELECT ST_AltitudeFromGeoSOTGridZ(5, 15);

---------------------------
9203.23364591971
```

5.2.4 转换函数

ST_AsText

将网格编码转换为标准规范的文本字符串。

语法

```
text ST_AsText(geosotgrid grid);

参数:
   grid - 网格对象
返回:
   网格编码的文本字符串
```

示例

```
SELECT ST_AsText('074EACB000000000000F0000'::geosotgrid);

---------------------------
G001310322-230230
```

ST_GeoSOTGridFromText

将网格编码文本字符串转为 geosotgrid 类型的网格编码。

语法

```
geosotgrid ST_GeoSOTGridFromText(cstring geosotgrid2d);
geosotgrid ST_GeoSOTGridFromText(cstring geosotgrid2d, cstring geosotgrid_z);

参数:
   geosotgrid2d - 二维网格编码字符串
   geosotgrid_z - 三维网格编码z方向字符串
返回:
   geosotgrid 类型的网格编码。
```

示例

```
SELECT ST_GeoSOTGridFromText('G001310322-230230', '101');

---------------------------
00B21A4984C0000000000104000F0001
```

ST_GeomFromGeoSOTGrid

返回网格对象的几何信息，二维网格返回POLYGON，三维网格返回POLYHEDRALSURFACE Z

语法

```
geometry ST_GeomFromGeoSOTGrid(geosotgrid grid);
geometry[] ST_GeomFromGeoSOTGrid(geosotgrid[] gridarray);

参数:
   grid - 网格编码
   gridarray - 网格编码数组
返回:
   geosotgrid 类型的网格编码
```

示例

```
SELECT ST_ASTEXT(ST_GeomFromGeoSOTGrid('074EACB000000000000F0000'::geosotgrid));

---------------------------
POLYGON ((116.3 39.9, 116.3 39.916666666666664, 116.31666666666666 39.916666666666664,␣
↪116.31666666666666 39.9, 116.3 39.9))
```

5.2.5 操作符

&&

网格编码相交计算，此操作符不仅可对同一层级的网格编码计算是否相交，对多个不同层级的网格编码同样可以计算相交关系。

语法

```
boolean &&(geosotgrid[] gridsA,geosotgrid[] gridsB);

参数:
  gridsA - 网格编码数组A
  gridsB - 网格编码数组B
返回:
  存在相交则返回 true，不存在相交则返回 false。
```

示例

```
SELECT ST_GeoSOTGrid(ST_MAKEENVELOPE(1, 1, 2, 2, 4490), 15) && ST_GeoSOTGrid(ST_
↪MAKEENVELOPE(1.5, 1.5, 2.5, 2.5, 4490), 15);

--------------
true
```

@>

网格编码包含关系计算。

语法

```
boolean @>(geosotgrid[] gridsA,geosotgrid[] gridsB);

参数:
  gridsA - 网格编码数组A
  gridsB - 网格编码数组B
返回:
  若包含则返回 true，不包含相交则返回 false。
```

示例

```
SELECT ST_GeoSOTGrid(ST_MAKEENVELOPE(1, 1, 2, 2, 4490), 15) @> ST_GeoSOTGrid(ST_
↪MAKEENVELOPE(1.2, 1.2, 1.8, 1.8, 4490), 15);

--------------
true
```

<@

网格编码被包含关系计算。

语法

```
boolean <@(geosotgrid[] gridsA,geosotgrid[] gridsB);

参数:
  gridsA - 网格编码数组A
```

(continues on next page)

(continued from previous page)

```
  gridsB - 网格编码数组B
返回:
  若被包含则返回 true，不被包含则返回 false。
```

示例

```
SELECT ST_GeoSOTGrid(ST_MAKEENVELOPE(1.5, 1.5, 2, 2, 4490), 15) <@ ST_GeoSOTGrid(ST_
↪MAKEENVELOPE(1, 1, 2, 2, 4490), 15);

--------------
true
```

5.3 版本函数

5.3.1 Yukon_Version()

获取 Yukon 的版本号、编译时间等信息。

语法

```
text Yukon_version();

返回:
   Yukon的版本号、编译时间等信息的文本。
```

示例

```
SELECT Yukon_version();

------------------------
1.0.1 Compiled at:2022-12-26 08:49:42 Commit ID:e1cdae4
```

5.4 注意事项

5.4.1 移除扩展

Yukon支持从数据库移除扩展模块。

语法

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

示例

移除postgis扩展：

DROP EXTENSION postgis;

– 如果有对象依赖该扩展，需要加CASCADE关键字进行级联删除

DROP EXTENSION postgis CASCADE;

注意事项

该能力由数据库内核提供，移除扩展将从当前数据库中删除指定扩展模块的相关配置，使用CASCADE同时将自动删除与该模块相关的数据，但不会级联删除其他扩展，建议提前做好相关数据备份。 参考 DROP EXTENSION 语法

CHAPTER 6

数据库备份与恢复

这里我们使用 openGauss 内置的备份与恢复工具： `gs_dump` 和 `gs_restore` 。

PostgreSQL 可以使用 `pg_dump` 和 `pg_restore` 工具来进行备份和恢复。

6.1 备份

`gs_dump` 用法：

```
gs_dump [OPTION]... [DBNAME]
```

以下是一些常用的选项：

- `-f`: 输出的文件名或者目录名
- `-F`: 输出的文件格式 `c`: 自定义格式 `d`:目录格式 `t`: tar 格式 `p`:简单文本模式（默认格式）
- `-Z`: 压缩文件的压缩等级，范围为 0-9
- `-t`: 只备份数据库中的某一张表
- `-T`: 备份除这个表之外的其余表

在 yukontutorial 数据库中创建一个 backtest 的表，然后进行备份：

```
-- 创建 backtest 表
CREATE TABLE backtest(geom geometry);
-- 插入数据
INSERT INTO backtest(geom) values('Point(1 2)');
-- 查看插入的数据
SELECT ST_ASTEXT(geom) from backtest;
```

创建好数据就可以来备份数据了，退出数据库然后在命令行执行：

```
# 这里只备份 yukontutorial 数据库中的 backtest 表，输出格式为自定义格式，输出文件名为 testdump.
↪bak
gs_dump  yukontutorial -t backtest -Fc -f testdump.bak
```

显示如下信息，则表示备份成功：

```
gs_dump[port='5432'][yukontutorial][2021-10-19 04:05:43]: The total objects number is␣
↪379.
gs_dump[port='5432'][yukontutorial][2021-10-19 04:05:43]: [100.00%] 379 objects have␣
↪been dumped.
gs_dump[port='5432'][yukontutorial][2021-10-19 04:05:43]: dump database yukontutorial␣
↪successfully
gs_dump[port='5432'][yukontutorial][2021-10-19 04:05:43]: total time: 517  ms
```

6.2 恢复

gs_restore 用法：

```
gs_restore [OPTION]... FILE
```

以下是一些常用的选项：

- -d: 数据库
- -j: 多线程恢复

使用上面备份的数据表进行还原：

```
-- 先创建一个新的数据库
CREATE DATABASE restoretest;

-- 切换到这个数据库 restoretest
-- 创建 postgis 扩展
CREATE EXTENSION postgis;
```

断开连接进行还原

```
gs_restore -d restoretest testdump.bak
```

显示如下信息，则表示还原成功：

```
start restore operation ...
table backtest complete data imported !
Finish reading 4 SQL statements!
end restore operation ...
restore operation successful
total time: 427  ms
```

Warning: 因为备份的数据库中含有 geometry 类型的数据，因此在还原的时候要先创建 postgis 扩展，才能恢复数据库，否则将会报错。

CHAPTER 7

知识库

7.1 三维模型数据组织及存储格式

7.1.1 对象组织结构

三维模型对象存储为GeoModel类型，由带局部坐标系的模型对象（ModelNode）及其放置的位置、姿态等信息组成，对象组织结构见下图：

ModelNode由精细层和LOD层（用PagedLOD表示，可选）数据组成，精细层和LOD层的基本组成单元均为PagedPatch；

每个PagedPatch包含多个Geode；Geode是一个数据包，由实体对象（ModelElement）组成，通过Geode上的矩阵，可以把相同的实体放置在不同的位置，实现模型数据的实例化存储。

ModelElement的子类包括骨架（ModelSkeleton）、材质（ModelMaterial）和纹理（ModelTexture）。

7.1.2 存储策略

在存储策略上，GeoModel存储在主表中，Geode仅存储实体对象的名字；实体对象单独存储在数据集子表中，基于实体对象名字的64位HashCode编码作为对象的ID。

7.1.3 源码路径

yukon_geomodel模块对应Yukon仓库 的/geomodel 目录；

geomodel对象相关代码实现在 /geomodel/libUGC。

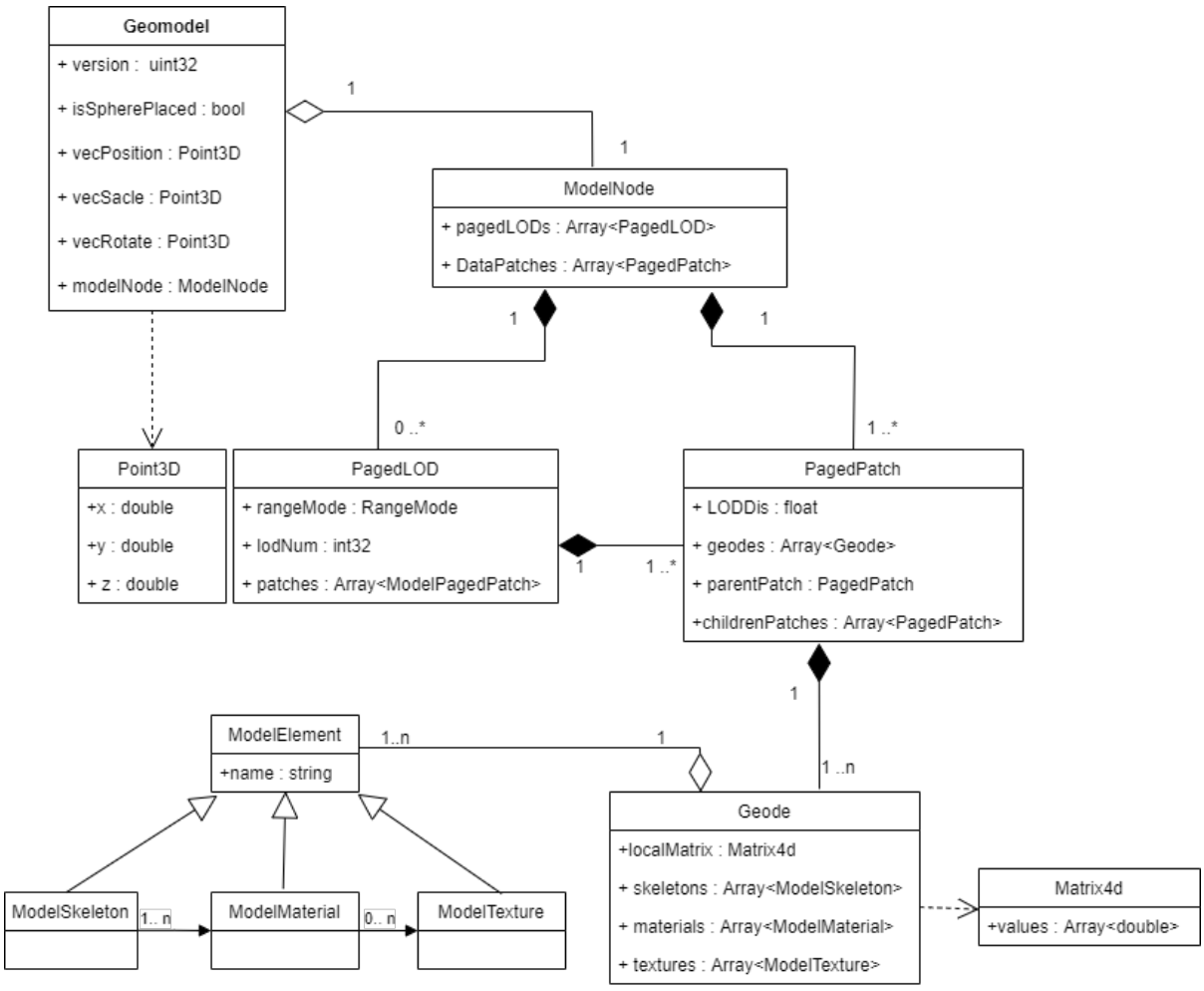


Fig. 1: 图 geomodel对象组织结构

7.2 Yukon支持GeoSOT编码的基本原理及特性

Yukon的网格编码模块，提供了geosotgrid数据类型，描述GeoSOT的网格对象。每个网格对象可以是二维的，也可以是三维的；知道自己的层级；可以转换成标准编码文本，易读易懂并与其它系统互联互通；也可以转换为geometry对象，放在二维地图或三维场景中直观地表达自己的空间位置。 Yukon支持GeoSOT编码的全部32层级，精度可达厘米级，支持Z方向； 目前支持的坐标值区间：

X[0,180]，Y[0, 88]，Z[-6302.106722602182,28680.1711252437]（X、Y方向单位为度，Z方向单位为千米）。

参见 *Yukon中GeoSOT 编码的基本能力* 。

7.2.1 高效检索

Yukon中基于网格的空间过滤查询，是将geometry对象从地理空间转换到网格空间，落在相同网格的对象即为可能有交点的对象，将作为检索结果输出。

Yukon提供对geosotgrid的查询加速：

1）对geosotgrid列创建B树索引；

2）对geosotgrid数组列创建gin索引；

7.2.2 与空间索引的区别与联系

空间索引通常针对单个图层，是单图层实现高效空间过滤的重要手段； GeoSOT编码则是切换到网格空间，从网格的视角管理空间数据，是空间数据仓库管理的重要手段：把GIS中分层或分专题管理的数据，按网格管理。

使用示例 基于*GeoSOT*编码的多图层穿越查询 。

7.2.3 对坐标系的支持

GeoSOT编码对应的地理框架为China2000（EPSG：4490），空间对象编码前需要进行坐标系转换；编码存储后，仍然可以基于geosotgrid进行空间过滤。 即非China2000坐标系的空间数据，也可以使用geosotgrid编码。

7.3 POSTGRES_FDW 使用说明

7.3.1 安装

openGauss 数据库默认情况下已经提供了 postgres_fdw 扩展，postgreSQL 数据库默认不提供扩展，可以通过位于源码的 contrib/postgres_fdw 目录进行编译安装。Yukon 的安装包中也直接提供了这个扩展。

7.3.2 使用

1. 创建 postgres_fdw 扩展

```
CREATE EXTENSION postgres_fdw;
```

2. 创建外部服务器

```
CREATE SERVER foreign_server
        FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host '192.83.123.89', port '5432', dbname 'foreign_db');
```

3. 创建用户映射

```
CREATE USER MAPPING FOR local_user
        SERVER foreign_server
        OPTIONS (user 'foreign_user', password 'password');
```

4. 创建外部表

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text
)
        SERVER foreign_server
        OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

CREATE FOREIGN TABLE 中声明的列数据类型和其他性质必须要匹配实际的远程表。列名也必须匹配，不过也可以为个别列附上column_name选项以表示它们在远程服务器上对应哪个列。在很多情况中，要手工构造外部表定义，使用 IMPORT FOREIGN SCHEMA 会更好。

7.4 OGR_FDW 使用说明

Note: 此插件目前只支持 Yukon for Postgres 版本,安装包中包含有此插件。

7.4.1 安装

Centos 7 安装:

1. 添加 PostgreSQL 软件源

```
yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/
↪pgdg-redhat-repo-latest.noarch.rpm
```

2. 安装 ogr_fdw

```
yum install -y ogr_fdw_13.x86_64
```

源码安装

源码下载

下载好源码后，然后 ./configure ,make,make install 即可。

需要注意的是：在编译时有可能需要导出 pg_config ， gdal_config 所在路径.

Note: 如果想要 ogr_fdw 插件支持 Oracle 数据库，则需要手动配置编译环境:

安装下列软件包:

oracle-instantclient-basic

oracle-instantclient-sqlplus

oracle-instantclient-devel

下载 SDK 软件包 ， 并将其解压到 `/usr/lib/oracle/21/client64` 目录下。此时该目录下有如下内容:

```
bin lib sdk SDK_LICENSE SDK_README
```

导出 `export ORACLE_HOME=/usr/lib/oracle/21/client64` 环境变量

然后再次 configure gdal 库就可以看到 OCI 支持。

```
OCI support:  yes
```

7.4.2 使用

使用 `create extension ogr_fdw` 创建扩展

Shapefile 文件

- 创建 server

```
CREATE SERVER shpdriver
    FOREIGN DATA WRAPPER ogr_fdw
    OPTIONS (
        datasource '/home/pg13/data/shp',
        format 'ESRI Shapefile'
        );
```

其中 datasource 为 Shapefile 文件所在路径。format 为格式。

- 创建外部表

```
create foreign table shptable(
    geom geometry
    )
    server shpdriver
    options (layer 'bjroad');
```

其中 server 指定为我们上边创建的 server 名字，option 中添加了一个必须的 layer 选项，指定我们要连接到的图层。

- 查询数据

```
select count(*) from shptable;
select * from shptable;
```

FileGDB 文件

- 创建 server

```
CREATE SERVER filegdbserver
    FOREIGN DATA WRAPPER ogr_fdw
    OPTIONS (
        datasource '/home/pg13/data/bjroad.gdb.zip',
        format 'OpenFileGDB' );
```

这里我们在 options 中指定数据格式为 OpenFileGDB

- 创建外部表

```
create foreign table filegdbtable(
    geom geometry(MULTILINESTRING, 4490)
    )
    server filegdbserver
    options (layer 'bjroad_1');
```

指定要链接的图层为 bjroad_1

- 查询数据

```
select st_astext(geom)
from filegdbtable;
```

Oracle Spatial 数据库

- 创建 server

```
CREATE SERVER ocidriver
    FOREIGN DATA WRAPPER ogr_fdw
    OPTIONS (
        datasource 'OCI:supermap/
↪supermap@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.13.
↪179)(PORT=1521))(CONNECT_DATA=(SID=helowin)))',
        format 'OCI');
```

这里 datasource 中，我们指定的 Oracle Spatial 数据库地址，用户名，密码及 SID。格式为 OCI.

- 创建外部表

```
create foreign table ocitable(

    geom geometry

    )

    server ocidriver

    options (

        layer 'BJROAD');
```

这里我们使用刚才导入的 bjroad.shp 数据，指定要链接的图层为 BJROAD.

- 查询数据

```
select count(*) from ocitable;
```

导入所有图层

如果想导入某个 schema 下的所有图层，可以使用 import schema 方法。如果想导入所有 schema 请使用 ogr_all:

```
create schema shpschema;

import foreign schema ogr_all
    from server shpdriver into shpschema;


create schema filegdbschema;

import foreign schema bjroad
    from server filegdbserver into filegdbschema;

create schema ocishcema;

import foreign schema ogr_all
    from server ocidriver into ocishcema;
```

如果在表名或者列名中含有中文字符，请使用如下选项来禁用过滤:

```
IMPORT FOREIGN SCHEMA ogr_all

    FROM SERVER fgdbtest

    INTO fgdbpreserve

    OPTIONS (

        launder_table_names 'false',

        launder_column_names 'false'

        );
```

7.5 ORACLE_FDW 使用说明

Note: 此插件目前只支持 Yukon for Postgres 版本,安装包中包含有此插件。

7.5.1 安装

编译安装

1. 下载源码

   可以从 oracle_fdw 的 github 下载最新的源码

2. 导出环境变量

   oracle_fdw 安装时需要 `pg_config` 和 `oracle` 库的相关支持。

```
export PATH=$PATH:../pg_confg_path
export ORACLE_HOME=/usr/lib/oracle/21/client64
```

3. 编译、安装

   开始编译前，可以使用 pg_config 查看其路径是否正确。使用如下命令开始编译并安装。

```
make
make install
```

7.5.2 使用说明

快速使用

1. 创建 oracle_fdw 扩展

```
create extension oracle_fdw;
```

2. 创建外部服务器

```
CREATE SERVER oracleserver
    FOREIGN DATA WRAPPER oracle_fdw
   OPTIONS (
       dbserver '192.168.13.179:1521/orcl',
       isolation_level 'read_only',
       nchar 'off');
```

3. 添加使用权限

```
-- 添加使用权限
GRANT USAGE ON FOREIGN SERVER oracleserver TO supermap;
```

4. 创建用户映射

```
-- 创建用户映射
CREATE USER MAPPING FOR postgres SERVER oracleserver
         OPTIONS (user 'oracle', password '123456');
```

5. 导入模式/创建外部表

```
-- 创建模式
create schema oracle_schema;
-- 从外部 server 导入
IMPORT FOREIGN SCHEMA "SCHEMA" FROM SERVER oracleserver INTO oracle_schema␣
↪OPTIONS (case 'lower');
```

6. 查询数据

```
select * from oracle_schema.smdtv_2 s where smid < 100;
```

选项说明

SERVER 选项

- dbserver (必填)：oracle 数据库的连接信息

- isolation_level (选填)：oracle 事务隔离级别，默认为`serializable`， 可选项：`serializable`, `read_committed`,`read_only`
- nchar (选填，布尔型)：oracle 字符转换，默认为 `off`

USER MAPPING 选项

- user(必填)：oracle 用户名
- password(必填)：oracle 密码

外部表选项

- table(必填)：oracle 表名

导入模式选项

- case:控制在导入外部表时，表名和字段名是否大小写
  - `keep`: 保持原有大小写形式，一般来说都是大写
  - `lower`: 将所有的表名和字段名都变成小写
  - `smart`:只将表名或字段名全部为大写的变为小写，这也是默认选项
- readonly: 将所有导入的表设置为只读

更多说明看参考官方文档说明

7.6 Citus分布式扩展使用说明

7.6.1 简介

Citus 是一个 PostgreSQL 扩展，可将 PostgreSQL 转换为分布式数据库，因此您可以在任何规模上实现高性能。

Citus使用分片和复制在多台机器上横向扩展PostgreSQL。它的查询引擎在这些服务器上执行SQL进行并行化查询，以便在大型数据集上实现实时的响应。

Citus集群由一个中心的协调节点(CN)和若干个工作节点(Worker)构成。

- coordinate：协调节点，存储元数据，不存实际数据。该节点直接对用户开放，等于一个客户端。
- worker：工作节点，不存储元数据，存储实际数据，执行协调节点发来的查询请求。

7.6.2 安装部署

- Yukon安装包里带有Citus所需的相关文件，安装完Yukon后，您只需执行下面的节点配置步骤即可。
- 或者从 Citus源码库 里下载源码，解压文件，进入citus目录，导出 `pgconfig` 文件目录， 如：`export PG_CONFIG=/opt/pg13/bin/pg_config`，运行 `configure` 配置文件， `./configure` ，如果出错提示缺少部分库文件，可以直接安装对应的库，如：`yum install -y libzstd-devel lz4-devel`，或者使用 `without` 选项，屏蔽该需求

编译安装

```
make -j && make install
```

单节点：

请将以下内容添加到：postgresql.conf

```
shared_preload_libraries = 'citus'
```

重启PostgreSQL后，登录数据库创建扩展

```
create extension citus;
```

多节点：

如果要设置多节点集群，使用 Citus 扩展配置其他 PostgreSQL 节点，并添加它们以形成 Citus 集群：

编辑 postgresql.conf 配置文件，设置 listen_addresses = '*'，监听地址改为全部

请将以下内容添加到 pg_hba.conf 配置文件中的ipv4配置下

```
host all all 0.0.0.0/0 trust
```

重启PostgreSQL使配置生效

登录数据库，在协调节点上添加其他工作节点的ip地址， 数据库端口号

```
select * from master_add_node('192.168.xxx.xxx', 5432);
```

查询添加成功的节点，如果添加成功，则会显示已添加的节点信息

```
select * from master_get_active_worker_nodes();

node_name       node_port
------------------------+
192.168.xxx.xxx 5432
```

Note: 集群需要在每个节点上同样部署Postgresql数据库 + Citus扩展，所用数据库名、默认用户名保持一致，数据库都需要创建Citus扩展以及其他所需要的扩展，确保各节点之间可通信，选择一台机器作为协调节点，其他机器作为工作节点 （已经存在的表，想创建分布式表，首先在另外的机器上创建同名数据库，创建扩展）

7.6.3 使用说明

create_distributed_table()

定义分布式表并创建其分片，如果未指定分发方法，则该函数默认为“哈希”分布

参数名称 描述	
table_name	需要分发的表的名称
distribution_column	要将表分布到的列
distribution_type（可选）	表的分布方法，允许的值为追加或哈希，默认为“哈希”

alter_distributed_table()

更改分布式表的分布列、分片计数或共置属性

参数名称	描述
table_name	将要更改的分布式表的名称
distribution_column（可选）	新分发列的名称
shard_count（可选）	新分片计数
colocate_with（可选）	当前分布式表将与之共置的表。可能的值为，用于启动新的共置组，或用于共置的另一个表的名称
cascade_to_colocated（可选）	当此参数设置为“true”时，更改也将应用于以前与该表共置的所有表，并且将保留共置。如果它是“false”，则此表的当前共置将被破坏

master_add_node()

添加工作节点

参数名称	描述
ip	IP地址
port	端口号

undistribute_table()

撤消create_distributed_table或create_reference_table的操作

参数名称	描述
table_name	要取消分发的分布式表或引用表的名称
cascade_via_foreign_keys（可选）	当此参数设置为“true”时，undistribute_table还会通过外键取消分布与table_name相关的所有表。请谨慎使用此参数，因为它可能会影响许多表

7.6.4 扩容

对于新添加的节点，Citus不会自动移动数据到新节点上，因此我们可以手动平移数据到新的节点上，达到扩容减压的目的，目前有以下两个方法可以实现

- 可以手动复制某工作节点的表到新节点，然后修改 pg_dist_placement 表的表名和grids值，删除原工作节点上该表，此时总表数不变，完成扩容减压。

创建一张分布式表，以哈希值进行分发

```
create table t1(id serial primary key, geom geometry);

set citus.shard_count = 6;

select create_distributed_table('t1', 'id', 'hash');

with a as (select id ,(random()*170)::float x, (random()*80)::float y from generate_
↪series(1, 600) t(id))
insert into t1(geom) select st_makeenvelope(x, y, x+0.001, y+0.001, 4490) from a;
```

首先，查看每个节点对应的groupid值

```
select * from pg_dist_node;

nodeid|groupid|nodename
↪|nodeport|noderack|hasmetadata|isactive|noderole|nodecluster|metadatasynced|shouldhaveshards|
------+-------+--------------+---------+--------+-----------+--------+---------+--------
↪---+--------------+-----------------+
     2|      2|192.168.12.205|   13000|default |false      |true    |primary |default␣
↪    |false          |true             |
     1|      1|192.168.12.122|   13000|default |false      |true    |primary |default␣
↪    |false          |true             |
```

查看每一张表的shardid值

```
select * from citus_shards;

table_name|shardid|shard_name|citus_table_type|colocation_id|nodename
↪|nodeport|shard_size|
----------+-------+----------+----------------+-------------+--------------+--------+-
↪---------+
t1        | 102161|t1_102161 |distributed     |            8|192.168.12.122|   13000|␣
↪    24576|
t1        | 102162|t1_102162 |distributed     |            8|192.168.12.205|   13000|␣
↪    16384|
t1        | 102163|t1_102163 |distributed     |            8|192.168.12.122|   13000|␣
↪    24576|
t1        | 102164|t1_102164 |distributed     |            8|192.168.12.205|   13000|␣
↪    16384|
t1        | 102165|t1_102165 |distributed     |            8|192.168.12.122|   13000|␣
↪    16384|
t1        | 102166|t1_102166 |distributed     |            8|192.168.12.205|   13000|␣
↪    16384|
```

查看每张分片表分布在哪一个节点上

```
select * from pg_dist_placement where shardid in (select shardid from pg_dist_shard␣
↪where logicalrelid='tb1'::regclass);

placementid|shardid|shardstate|shardlength|groupid|
-----------+-------+----------+-----------+-------+
        154| 102161|         1|          0|      1|
        155| 102162|         1|          0|      2|
        156| 102163|         1|          0|      1|
        157| 102164|         1|          0|      2|
        158| 102165|         1|          0|      1|
        159| 102166|         1|          0|      2|
```

在协调节点上加入新的worker

```
select * from master_add_node('192.168.12.201', 5432);
```

检查pg_dist_node元数据表，新的worker节点的groupid为5

```
nodeid|groupid|nodename
↪|nodeport|noderack|hasmetadata|isactive|noderole|nodecluster|metadatasynced|shouldhaveshards|
------+-------+--------------+---------+--------+-----------+--------+---------+--------
↪---+--------------+-----------------+
     2|      2|192.168.12.205|   13000|default |false      |true    |primary |default␣
↪    |false          |true             |
```

(continues on next page)

(continued from previous page)

```
     1|      1|192.168.12.122|   13000|default |false      |true     |primary |default␣
↪    |false          |true             |
     5|      5|192.168.12.201|    5432|default |false      |true     |primary |default␣
↪    |false          |true             |
```

复制分片，现在1、2号节点上各3片，为了保持均衡，我们可以移动部分分片到5号节点上

下面移动1号节点上的分片t1_102161到5号节点上：

在1号节点上创建PUBLICATION（此步操作是在1号工作节点上执行，不是在协调节点上）

```
create PUBLICATION pub_shard for table tb1_102046;
```

在3号节点上创建分片表和SUBSCRIPTION（此步操作是在3号工作节点上执行，不是在协调节点上，表名、表结构和要复制的表保持一致）

```
create table t1_102161(id int primary key, geom geometry);

CREATE SUBSCRIPTION sub_shard
    CONNECTION 'host=192.168.12.122, port=13000, dbname=testcitus'
    PUBLICATION pub_shard;
```

在协调节点上锁表，防止数据被修改

```
begin;
lock table tb1 IN EXCLUSIVE MODE;
```

等待3号节点上的表数据和1号节点上的表数据完全同步，解锁表，修改元数据表

```
end;
update pg_dist_placement set groupid=5 where shardid=102161 and groupid=1;
```

在1号节点上删除分片表和PUBLICATION

```
DROP PUBLICATION pub_shard;

drop table tb1_102046;
```

在5号节点上删除SUBSCRIPTION

```
DROP SUBSCRIPTION sub_shard;
```

扩容完成

- alter_distributed_table函数，可以自动将已有的分片表再次重新平均分配到每个节点，如2个节点共2张表，添加新节点后，修改分片表数为3，则分布情况为3个节点共3张表，也可以达到扩容减压的目的。

7.6.5 使用案例

首先根据集群环境设置分表数，分表时会平均的分配到每个工作节点上，如count=32，工作节点有两个，则会在每台工作节点上创建16张表，在我们的测试中，使用GIN索引查询网格编码时，每个工作节点的表数不超过cpu逻辑核数的一半时效果最佳。

案例环境为 3台 8核x86 机器，组合为 1cn + 2worker 的集群。

```
set citus.shard_count = 8;
```

创建表

```
create table polygon_grid21(id serial primary key, geom geometry, grids geosotgrid[]);
```

按id列上的哈希值对源表进行分发

```
select create_distributed_table('polygon_grid21', 'id');
```

往表 polygon_grid21 里插入数据

```
with a as (select (random()*170)::float x, (random()*80)::float y from generate_
↪series(1, 1000000))
insert into polygon_grid21(geom, grids) select st_makeenvelope(x, y, x+0.001, y+0.001,
↪ 4490), ST_GeoSOTGrid(st_makeenvelope(x, y, x+0.001, y+0.001, 4490), 21) from a;
```

查看分表情况、哈希分布范围

```
select * from citus_shards;

table_name    |shardid|shard_name           |citus_table_type|colocation_id|nodename ␣
↪    |nodeport|shard_size|
--------------+-------+---------------------+----------------+-------------+----------
↪----+--------+----------+
polygon_grid21| 102126|polygon_grid21_102126|distributed     |            1|192.168.
↪12.122|   13000|  66879488|
polygon_grid21| 102127|polygon_grid21_102127|distributed     |            1|192.168.
↪12.205|   13000|  66551808|
polygon_grid21| 102128|polygon_grid21_102128|distributed     |            1|192.168.
↪12.122|   13000|  66568192|
polygon_grid21| 102129|polygon_grid21_102129|distributed     |            1|192.168.
↪12.205|   13000|  66650112|
polygon_grid21| 102130|polygon_grid21_102130|distributed     |            1|192.168.
↪12.122|   13000|  66789376|
polygon_grid21| 102131|polygon_grid21_102131|distributed     |            1|192.168.
↪12.205|   13000|  66732032|
polygon_grid21| 102132|polygon_grid21_102132|distributed     |            1|192.168.
↪12.122|   13000|  66486272|
polygon_grid21| 102133|polygon_grid21_102133|distributed     |            1|192.168.
↪12.205|   13000|  66625536|

select * from pg_dist_shard;

table_name    |shardid|shard_name           |citus_table_type|colocation_id|nodename ␣
↪    |nodeport|shard_size|
--------------+-------+---------------------+----------------+-------------+----------
↪----+--------+----------+
polygon_grid21| 102126|polygon_grid21_102126|distributed     |            1|192.168.
↪12.122|   13000|  66879488|
polygon_grid21| 102127|polygon_grid21_102127|distributed     |            1|192.168.
↪12.205|   13000|  66551808|
polygon_grid21| 102128|polygon_grid21_102128|distributed     |            1|192.168.
↪12.122|   13000|  66568192|
polygon_grid21| 102129|polygon_grid21_102129|distributed     |            1|192.168.
↪12.205|   13000|  66650112|
polygon_grid21| 102130|polygon_grid21_102130|distributed     |            1|192.168.
↪12.122|   13000|  66789376|
```

(continues on next page)

(continued from previous page)

```
polygon_grid21| 102131|polygon_grid21_102131|distributed     |            1|192.168.
↪12.205|   13000|  66732032|
polygon_grid21| 102132|polygon_grid21_102132|distributed     |            1|192.168.
↪12.122|   13000|  66486272|
polygon_grid21| 102133|polygon_grid21_102133|distributed     |            1|192.168.
↪12.205|   13000|  66625536|
```

创建索引

```
create index gin_polygon_grid21 on polygon_grid21 using GIN(grids);
```

相交查询

```
select * from polygon_grid21 where grids && ST_GeoSOTGrid(st_makeenvelope(0, 0, 0.01,␣
↪0.01, 4490), 21);
```

删除一半数据，查询结果

```
delete from polygon_grid21 where id % 2 = 0;

select count(*) from polygon_grid21;

count |
------+
500000|
```

再次插入10w数据，查看结果

```
with a as (select (random()*170)::float x, (random()*80)::float y from generate_
↪series(1, 100000))
insert into polygon_grid21(geom, grids) select st_makeenvelope(x, y, x+0.001, y+0.001,
↪ 4490), ST_GeoSOTGrid(st_makeenvelope(x, y, x+0.001, y+0.001, 4490), 21) from a;

select count(*) from polygon_grid21;

count |
------+
600000|
```

想要获取更加详细的资料，请访问 Citus官方文档 。

7.7 数据库参数与调优说明

7.7.1 参数设置与查看

参数设置

可以通过 SET 命令进行参数设置，例如：

```
SET parameter =  value;
```

参数查看

可以通过 SHOW 命令进行参数查看，例如：

```
SHOW parameter;
```

7.7.2 参数说明

- shared_buffers

  这个参数设置数据库的共享缓冲区的大小。默认通常是128 兆字节。一个合理的 *shared_buffers* 是系统内存的 25%。

- effective_cach_size

  设置规划器对一个单一查询可用的有效磁盘缓冲区尺寸的假设。这个参数会被考虑在使用一个索引的代价估计中，更高的数值会使得 索引扫描更可能被使用，更低的数值会使得顺序扫描更可能被使用。这有助于规划器做出一个更好的代价估计。

- work_mem

  设置在写入临时磁盘文件之前查询操作(例如排序或哈希表)可使用的最大内存容量。如果指定值时没有单位，则以千字节为单位。默认值是4兆字节。 注意对于一个复杂查询，可能会并行运行好几个排序或者哈希操作；每个操作都会被允许使用这个参数指定的内存量，然后才会开始写数据到临时文件。 同样，几个正在运行的会话可能并发进行这样的操作。因此被使用的总内存可能是 work_mem 值的好几倍，在选择这个值时一定要记住这一点。

- maintenance_work_mem

  设置在维护性操作（例如VACUUM、CREATE INDEX 和 ALTER TABLE ADD FOREIGN KEY）中使用的最大的内存量。 如果指定值时没有单位，则以千字节为单位， 其默认值是 64 兆字节（64MB）。因为在一个数据库会话中，一个时刻只有一个这样的操作可以被执行，并且一个数据库安装通常不会有太多这样的操作并发执行， 把这个数值设置得比work_mem大很多是安全的。 更大的设置可以改进清理和恢复数据库转储的性能。

CHAPTER 8

参考工具

8.1 Yukon服务端工具

8.1.1 yk_tool

提供安装、卸载和查看版本信息等功能。

语法

查询Yukon版本信息

```
yk_tool -v | --version
```

卸载Yukon

```
yk_tool -r | --remove [--pkgpath]
```

查询许可信息[仅支持GaussDB]

```
yk_tool  -l | --license
```

查询帮助信息

```
yk_tool  -? | --help
```

参数说明

-r, –remove

卸载Yukon。

–pkgpath

设置卸载的pg_config路径

-?, –help

显示帮助信息。

-v, –version

显示版本号信息。

-l, –license

显示许可信息。

8.2 SuperMap GIS

SuperMap 11i 组件和桌面通过 SDX+ for Yukon 引擎访问禹贡空间数据库，支持的数据类型包括：二三维点/线/面和三维模型数据集。 暂不支持栅格数据集。

相关数据和服务可以通过 SuperMap iServer 进行管理和发布。

SuperMap iManager 11 支持定制 Yukon 站点。编排文件：

Yukon for openGauss 。

Yukon for PostgreSQL 。

8.3 QGIS

QGIS 是一个自由软件的桌面 GIS 软件，提供空间数据的显示、编辑和分析功能。可以通过 PostGIS 插件连接 YukonDB，支持的空间数据类型包括：

- 二/三维点、线、面
- 栅格数据

三维模型数据无法识别。

CHAPTER 9

范例集

9.1 geometry 对象定义及构造

9.1.1 geometry 类型

定义geometry列，可以不指定子类型，则表示可以存储任意子类型。

```
-- 指定srid 为 4326
create table testgeoms ( id int primary key, geom geometry);
```

geometry列，也可以指定子类型，支持的类型名见下一小节，示例如下：

```
-- 指定 geom 列存储 linestring
create table testlinestring (id serial, geom geometry(linestring, 4326) );
```

指定子类型时，可带单引号、双引号或不带引号，也不区分大小写。

9.1.2 geometry 子类型

geometry 有15种子类型，包括：

Table 1: geometry 类型

子类型	描述	构成
POINT	点	/
LINESTRING	线，折线	由点串构成
POLYGON	面	由 n 个部分组成，每个部分是首尾相连的线串
POLYHEDRALSURFACE	多面体表面	由 n 个部分组成，每个部分都是一个 Polygon
MULTIPOINT	多点	由 n 个 Point 子对象组成
MULTILINESTRING	多线	由 n 个 LineString 子对象组成
MULTIPOLYGON	多面	由 n 个 Polygon 子对象组成
CIRCULARSTRING	线，由圆弧连接而成	用点串描述。三个点确定一段圆弧，前一个圆弧的最后一个点与后一个圆弧的第一个点共用；特别地，如果圆弧的第一个点与第三个点重合，则第一，二个点作为直径，以此来表达圆形
COMPOUNDCURVE	复合线	由 n 个部分组成，每个部分可以是LineString 或 CircularString，且前一部分的最后一个点与后续部分的第一个点重合，保证复合线对象的连续性
MULTICURVE	多(曲)线	由 n 个子对象组成，每个子对象可以是CircularString 或 LineString 或 CompoundCurve
CURVEPOLYGON	复合面	由 n 个部分组成，每个部分是首尾相连的 CircularString 或 LineString 或 CompoundCurve。与 Polygon 类似，都表达一个闭合的区域，区别在于是否有CircularString对象参与构造
MULTISURFACE	多面	由 n 个子对象组成，每个子对象可以是Polygon 或 CurvePolygon 类型。与 MultiPolygon类似，都表达多面对象，区别在于是否有CircularString对象参与构造
GEOMETRYCOLLECTION	复合对象	由任意子类型构成
TRIANGLE	三角形	由首尾相连的4个点构成
TIN	不规则三角网	由 n 个 Triangle 组成

以上每种类型，可以指定是否带Z或M值。例如，点可以指定
为：POINT、POINTZ、POINTM、POINTZM。
以上15种子类型，有一部分容易理清其关系，见下图左；当加入新的类型 **CircularString** 后，衍生出带参数化对象的线和面，进而构成PostGIS的全部15种子类型，见下图右。

9.1.3 构造示例：Point LineString Polygon

Point、LineString、Polygon、MultiPoint、MultiLineString、MultiPolygon容易理解，构造示例如下：

```
-- 指定 srid 为 4326
CREATE TABLE testgeomobj (id serial, geom geometry NOT NULL);

-- Point 对象
```

(continues on next page)

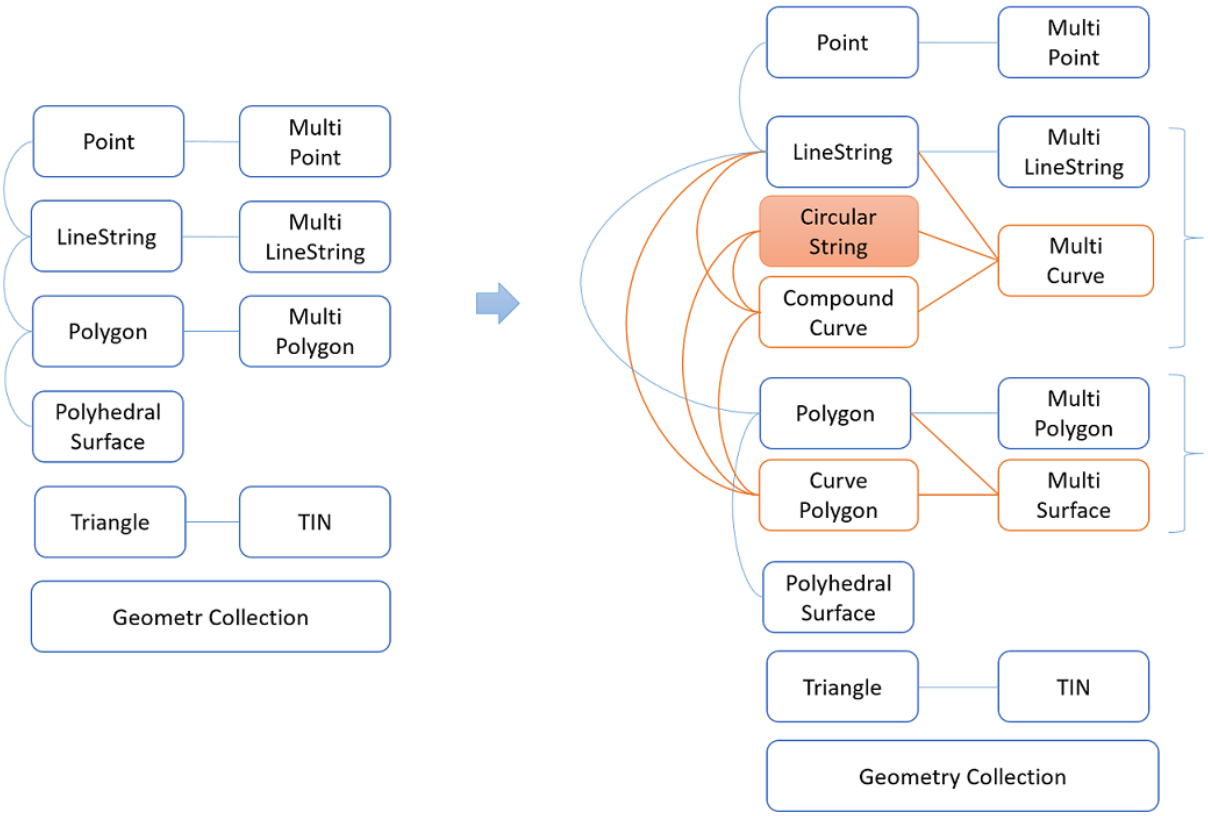


Fig. 1: PostGIS-GeometryType

(continued from previous page)

```
INSERT INTO testgeomobj (geom) VALUES ('SRID=4326;POINT(-95.363151 29.763374)');
INSERT INTO testgeomobj (geom) VALUES ('SRID=4326;POINT(-95.363151 29.763374)');
-- MultiPoint 对象
INSERT INTO testgeomobj (geom) VALUES ('SRID=4326;MULTIPOINT(-95.4 29.8,-95.4 29.8)');

-- LineString 对象
insert into testgeomobj (geom) values ('SRID=4326;LINESTRING(-71.1031880899493 42.
↪3152774590236,-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,-
↪71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248)');
-- MultiLineString 对象
insert into testgeomobj (geom) values ('SRID=4326;MultiLineString (
(-71.1031880899493 42.3152774590236,-71.1031627617667 42.3152960829043,-71.
↪102923838298 42.3149156848307,-71.1023097974109 42.3151969047397,-71.1019285062273␣
↪42.3147384934248),
(-71.1766585052917 42.3912909739571, -71.1766820268866 42.391370174323896, -71.
↪1766063012595 42.3913825660754, -71.17658265830809 42.391303365353096)
)');

-- Polygon 对象
insert into testgeomobj(geom) values ('SRID=4326;
POLYGON (
(-71.1776585052917 42.3902909739571, -71.1776820268866 42.3903701743239, -71.
↪1776063012595 42.3903825660754, -71.1775826583081 42.3903033653531,-71.
↪1776585052917 42.3902909739571),
(-71.1766585052917 42.3912909739571, -71.1766820268866 42.391370174323896, -71.
↪1766063012595 42.3913825660754, -71.17658265830809 42.391303365353096, -71.
↪1766585052917 42.3912909739571)
)');
-- MultiPolygon 对象
insert into testgeomobj(geom) values ('SRID=4326; MultiPolygon (
((-71.1776585052917 42.3902909739571, -71.1776820268866 42.3903701743239, -71.
↪1776063012595 42.3903825660754, -71.1775826583081 42.3903033653531,-71.
↪1776585052917 42.3902909739571)),
((-71.1766585052917 42.3912909739571, -71.1766820268866 42.391370174323896, -71.
↪1766063012595 42.3913825660754, -71.17658265830809 42.391303365353096, -71.
↪1766585052917 42.3912909739571))
)');
```

9.1.4 构造示例：CircularString CompoundCurve

```
CREATE TABLE testgeom (id serial,geom geometry NOT NULL);
-- CircularString：由四段组成，见下图左
INSERT INTO testgeom (geom) VALUES ('CIRCULARSTRING(0 2, -1 1, 0 0, 0.5 0, 1 0, 2 1,␣
↪1 2, 0.5 2, 0 2)');
-- CompoundCurve：由圆弧和折线段组成，'LINESTRING'关键字可省略，生成的图形见下图右
INSERT INTO testgeom (geom) VALUES ('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),
↪LINESTRING(1 0, 2 0))');
```

9.1.5 构造示例：CurvePolygon

```
-- 由首尾相连的圆弧线串和折线构成
insert INTO testgeom (geom) VALUES('CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0
↪0),(1 1, 3 3, 3 1, 1 1))');
```

9.1.6 构造示例：PolyhedralSurface

PolyhedralSurface 由n个 Polygon 构成，PolyhedralSurface 对象不一定是闭合的，可以用ST_IsClosed() 方法判断。

```
-- PolyhedralSurface 对象
INSERT INTO testgeom (geom) VALUES ('POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0,
↪0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0
↪0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
↪((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )');

SELECT ST_IsClosed(geom) from testgeom;
```

9.2 shapefile 数据导入及空间查询

9.2.1 导入数据

范例数据 包含纽约市的街道、地铁站等信息，路径: nyc_shapefile

使用 shp2pgsql导入数据

导入 nyc_census_blocks 数据

```
shp2pgsql \
  -D \
  -I \
  -s 26918 \
  nyc_census_blocks.shp \
  nyc_census_blocks \
  | gsql dbname=yukontutorial
```

其中:

- -D: 使用 dump format 方式，这个要比默认的 insert format 快很多。
- -I: 加载数据完成后，为 geometry 类型的列创建一个索引
- -s: 数据的 srid 值。
- nyc_census_blocks.shp: 要加载的数据集
- nyc_census_blocks: 加载后在数据库中创建的表名
- gsql dbname=yukontutorial : 将数据导入到 yukontutorial 数据库中

当你看到有如下输出时，则说明数据导入成功

```
Field popn_total is an FTDouble with width 11 and precision 0
Field popn_white is an FTDouble with width 11 and precision 0
Field popn_black is an FTDouble with width 11 and precision 0
Field popn_nativ is an FTDouble with width 11 and precision 0
Field popn_asian is an FTDouble with width 11 and precision 0
Field popn_other is an FTDouble with width 11 and precision 0
Shapefile type: Polygon
Postgis type: MULTIPOLYGON[2]
SET
SET
BEGIN
NOTICE:  CREATE TABLE will create implicit sequence "nyc_census_blocks_gid_seq" for␣
↪serial column "nyc_census_blocks.gid"
CREATE TABLE
NOTICE:  ALTER TABLE / ADD PRIMARY KEY will create implicit index "nyc_census_blocks_
↪pkey" for table "nyc_census_blocks"
ALTER TABLE
                         addgeometrycolumn
-------------------------------------------------------------------
 public.nyc_census_blocks.geom SRID:26918 TYPE:MULTIPOLYGON DIMS:2
(1 row)

CREATE INDEX
COMMIT
ANALYZE
```

然后重新连接到 yukontutorial 数据库，输入 \d 查看新创建的表：

```
yukontutorial=# \d
                                  List of relations
 Schema |            Name             |   Type   | Owner |              Storage
--------+-----------------------------+----------+-------+-------------------------------
↪----
 public | geography_columns           | view     | omm   |
 public | geometry_columns            | view     | omm   |
 public | geomodel_columns            | view     | omm   |
 public | nyc_census_blocks           | table    | omm   | {orientation=row,
↪compression=no}
 public | nyc_census_blocks_gid_seq   | sequence | omm   |
 public | raster_columns              | view     | omm   |
 public | raster_overviews            | view     | omm   |
 public | spatial_ref_sys             | table    | omm   | {orientation=row,
↪compression=no}
(8 rows)
```

同理，依次导入剩余的数据集：

- nyc_neighborhoods.shp
- nyc_streets.shp
- nyc_subway_stations.shp

你可能会好奇 .shp 文件到底是什么？

通常来说 shp 文件一般指的是 .shp, .shx,.dbf 还有一些其他类型文件且前缀名称一致的文件集合。shp 即 shape file ,通常单独的一个 shp 文件时没有什么用的，必须和其他类型的文件结合使用。

必要文件：

- `.shp`：存储地理要素的几何信息
- `.shx`：存储要素几何图形的索引信息
- `.dbf`: 存储地理要素的属性信息（非几何信息）

可选文件：

- `.prj`：存储空间参考信息，即地理坐标系统信息和投影坐标系统信息。使用 well-known 文本格式进行描述。

那 SRID 又是什么?

大多数导入过程都是不言自明的，但即使是经验丰富的 GIS 专业人员也可能被 SRID 难倒。

`SRID 表示 Spatial Reference IDentifier（空间参考标识符）`。它定义了我们数据的地理坐标系统和投影的所有参数。

SRID 很方便，因为它将有关地图投影的所有信息（可能非常复杂）打包（更具体的说应该是映射）到一个数字中。

你可以在以下链接中查找我们在上面使用的投影的定义：SRID

9.2.2 数据操作

我们想知道纽约市街道的总长度是多少应该怎么做?

```
select SUM(ST_LENGTH(GEOM)) from nyc_streets;
```

你可以看到如下输出：

```
yukontutorial=# select  SUM(ST_LENGTH(GEOM)) from nyc_streets ;
       sum
---------------
 10418904.7172
(1 row)
```

这样我们就可以知道纽约市街道总长度为 10418904.7172m

`ST_Length(geometry geom)`：如果 geom 的类型是 `LineString` 或者 `MultiLineString`,那么就返回这个 geometry 的长度。

`sum`：聚合函数，可以计算 `nyc_streets` 表中每一行 geometry 对象长度的总和。

我们想知道纽约市最西边的地铁站是哪一个?

```
select st_x(geom),name from nyc_subway_stations order by st_x(geom) limit 1;
```

输出结果：

```
yukontutorial=# select st_x(geom),name from nyc_subway_stations order by st_x(geom)␣
↪limit 1;
       st_x        |     name
-------------------+-------------
 563292.117258056 | Tottenville
(1 row)
```

`ST_X(geometry a_point);`: 如果 geometry 是一个点，则返回它的 x 坐标，否则返回 NULL。

现在我们知道位于纽约市最西边的地铁站是 Tottenville。

我们想知道距离 Broad St 地铁站 10m 范围内的街道有那些?

我们可以先查询一下 `Broad St` 地铁站的具体位置:

```
select st_astext(geom) from nyc_subway_stations where name='Broad St';
```

输出结果:

```
yukontutorial=# select st_astext(geom) from nyc_subway_stations where name='Broad St';
               st_astext
-----------------------------------------
 POINT(583571.905921312 4506714.34119218)
(1 row)
```

现在我们知道了 `Borad St` 地铁站的具体地点为 `POINT(583571.905921312 4506714.34119218)`，接下来可以查询距离地铁站 10m 范围内的街道了。

```
SELECT name
FROM nyc_streets
WHERE ST_DWithin(
        geom,
        ST_GeomFromText('POINT(583571.905921312 4506714.34119218)',26918),
        10
      );
```

输出结果:

```
   name
-----------
 Wall St
 Broad St
 Nassau St
(3 rows)
```

现在我们可以知道距离 `Broad St` 地铁站 10m 内的街道有这 3 个。

`ST_AsText(geometry g1)`: 返回 WKT 形式的 geometry 数据。

`ST_GeomFromText(text WKT, integer srid);`: 返回由 WKT 形式表示的 geometry 数据类型。同时指定它的 SRID。

`ST_DWithin(geometry g1, geometry g2, double precision distance_of_srid)`: 如果 g1 和 g2 相距 distance_of_srid 之内，则返回真，否则返回假。

我们想知道纽约市人口密度最大的社区是哪一个?

```
SELECT
  n.name,
  Sum(c.popn_total) / (ST_Area(n.geom) / 1000000.0) AS popn_per_sqkm
FROM nyc_census_blocks AS c
JOIN nyc_neighborhoods AS n
ON ST_Intersects(c.geom, n.geom)
```

(continues on next page)

(continued from previous page)

```
GROUP BY n.name, n.geom
ORDER BY 2 DESC limit 1;
```

输出结果：

```
       name         |  popn_per_sqkm
--------------------+------------------
 North Sutton Area | 68435.1328377268
(1 row)
```

从结果中我们可以看到 North Sutton Area 的人口密度是最大的。

`ST_Intersects( geometry geomA , geometry geomB )`: 返回 geomA 和 geomB 是否相交

9.3 矢量面拉伸构建三维模型对象

本节用到的模块有：postgis、postgis_sfcgal、yukon_geomodel。 范例数据 ： sampledata/region.shp，作为建筑物的矢量底面。

9.3.1 导入底面数据

使用 shp2pgsql 工具导入 region.shp 数据，设置导入结果为 dt1, geometry 列名 smgeometry。

9.3.2 执行 SQL 脚本

构建函数：遍历矢量数据表，对每个面对象进行拉伸、三角化后写入指定的模型数据表。函数实现如下：

```
-- geomodel_table 待写入的模型数据表；polygon_table 二维面数据表； polygon_table 中待拉伸的面
对象列名
CREATE OR REPLACE FUNCTION ExtrudePolygon2GeoModel(geomodel_table varchar, polygon_
↪table varchar, polygon_field varchar)
RETURNS boolean
AS $$
DECLARE
sql text;
cnt int;
begin
    sql = 'select count(*) from '||polygon_table||';';
    execute sql into cnt;

    -- 构造默认材质，并写入子表
    sql = 'insert into '|| geomodel_table ||'_elem ' || 'values (ST_MakeHashID(
↪'||quote_literal('material')||'),   ST_MakeDefaultMaterial('||quote_literal(
↪'material')||') );';
    raise notice '%',sql;
    execute sql;

    for i in 1..cnt
    loop
    -- 构造默认材质，矢量面拉伸 --> 转三角面 --> 转骨架对象，并写入子表
    sql = 'insert into '|| geomodel_table||'_elem '||'values (ST_MakeHashID('||quote_
↪literal('skeleton_'||i-1)||'),
```

(continues on next page)

(continued from previous page)

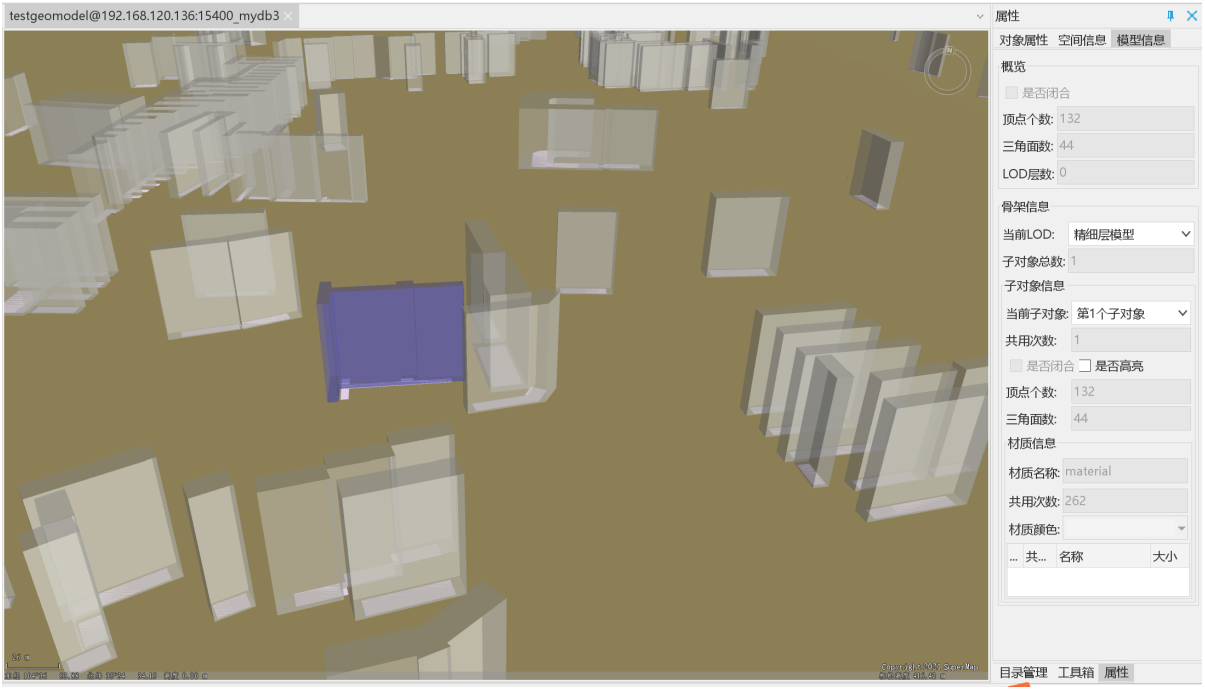
```
    (select ST_MakeSkeletonFromTIN(ST_Tesselate(ST_Extrude('|| polygon_field ||',0, 0,
↪ 100)), '||quote_literal('skeleton_'||i-1)||', '||quote_literal('material')||')
    from '|| polygon_table ||' where smid = '||i||'));';
    execute sql;
    -- raise notice '%',sql;
    -- 构造模型对象，写入主表
    sql = 'insert into '|| geomodel_table ||' values('||i||', (select  ST_
↪MakeGeomodel( ( select elemcol from '|| geomodel_table ||'_elem ' ||' where id= ST_
↪MakeHashID('||quote_literal('skeleton_'||i-1)||') ) ) ) );';
    execute sql;
    -- raise notice '%',sql;
    end loop;
    RETURN true;
END; $$ LANGUAGE 'plpgsql';
```

构造模型数据表，执行 ExtrudePolygon2GeoModel 函数

```
-- 创建待写入的模型表，指定id字段为主键
create table testgeomodel(id int primary key);
-- 添加geomodel列，指定 SRID=3857
select AddgeoModelcolumn('public', 'testgeomodel', 'geomodelcol', 3857);
--执行拉伸构建函数
select  ExtrudePolygon2GeoModel('testgeomodel', 'dt1', 'smgeometry');
```

9.3.3 查看模型数据

使用SuperMap桌面，打开Yukon数据源，可以在三维场景中加载testgeomodel数据：



9.4 矢量对象自相交检查

postgis 提供的 ST_IsValid 函数即可检测对象是否自相交。

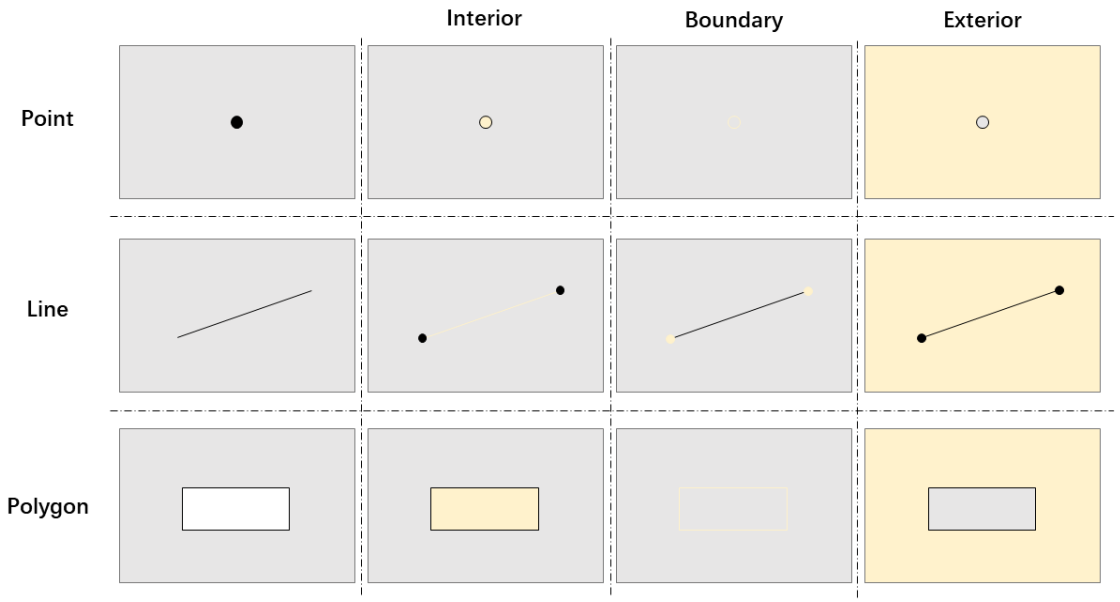
```
SELECT ST_IsValid(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
ST_IsValid(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly;
-- results
NOTICE: Self-intersection at or near point 0 0
good_line|bad_poly|
---------+---------+
true     |false    |
```

相关函数还有 ST_IsValidReason，给出非法的原因；ST_IsValidDetail 给出非法原因的同时，定位到具体的非法空间位置。

9.5 维度扩展九交模型

9.5.1 什么是九交模型

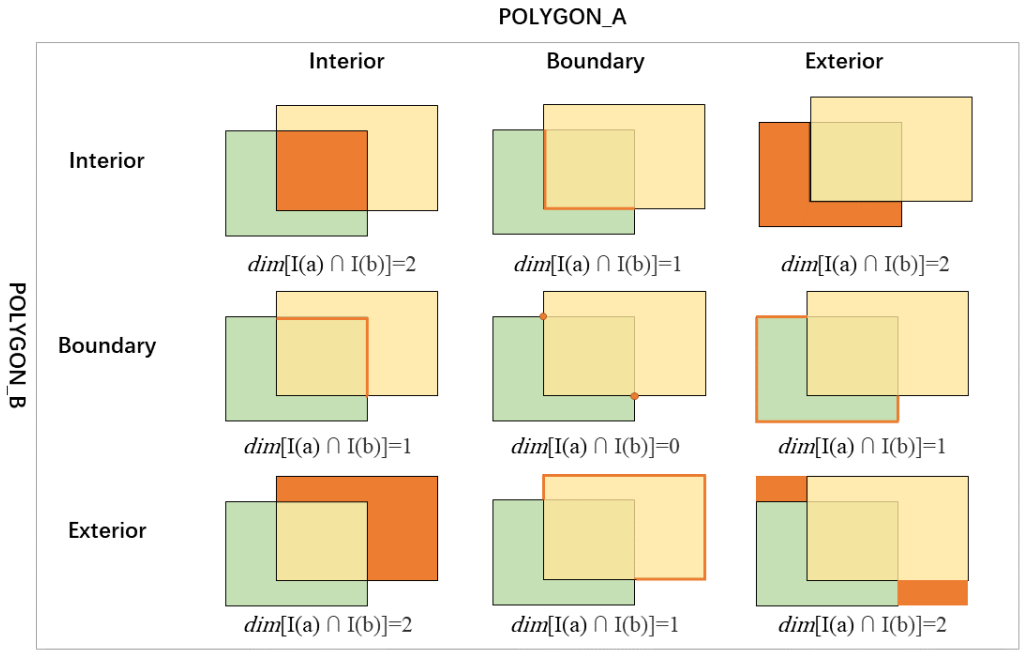
Spatial Join，即空间连接，其依据是空间关系。空间关系看起来直观形象，但是存在空间问题固有的复杂性和不确定性等问题，为了定性的去描述空间关系，我们通过一系列模型进行数字化和量化的表达与运算。其中使用最广泛的是九交模型。 九交模型把空间几何要素划分为内、外、边界三个部分，通过对几何要素这三个部分的关系判断，来定义空间要素的空间关系。



- 点要素，内部是点，边界是空集，外部是平面上除点以外的所有其他部分。
- 线要素，内部、边界和外部不容易划分，内部是以端点为界限的线的那一部分，边界是线性要素的端点，外部是平面中除内部和边界外的所有其他部分。
- 面要素，内部是以环为边界的里面的那一部分；边界是环本身；外部是边界外的一切。

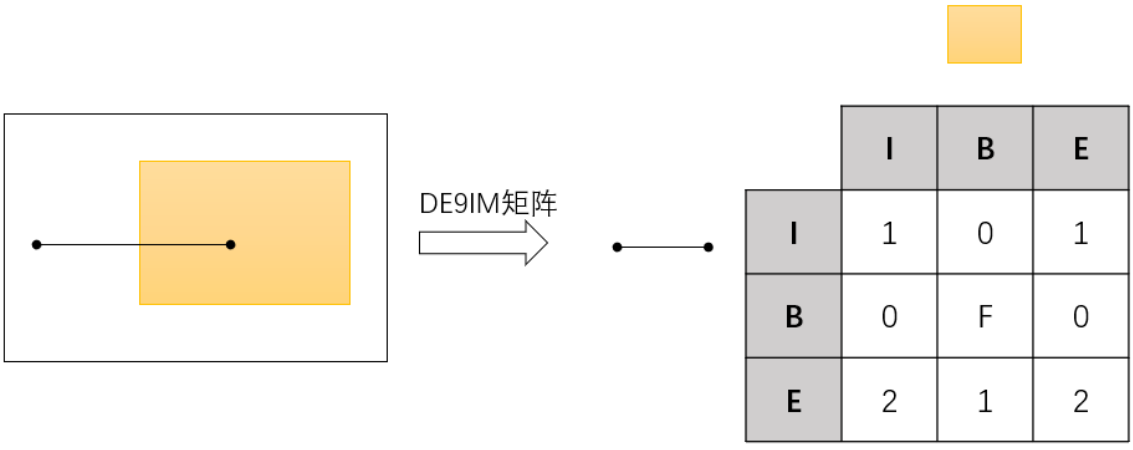
9.5.2 什么是维度扩展九交模型

维度扩展九交模型（Dimensionally Extended 9-Intersection Model，DE9IM）是对九交模型的应用延展，可以表达更加多样化的空间拓扑关系。因此，使用以上对对象内部、外部和边界的定义，我们可以用一组要素的内部/边界/外部九个可能性交集维度表示任何一对空间要素之间的关系，并利用数学矩阵来表达。关于它们的交集的DE9IM矩阵如下：



进一步地，对于上例中的多边形，内部的交集是二维区域，因此矩阵的对应部分用”2”填充。边界仅在零维点处相交，因此对应矩阵部分用”0”填充。

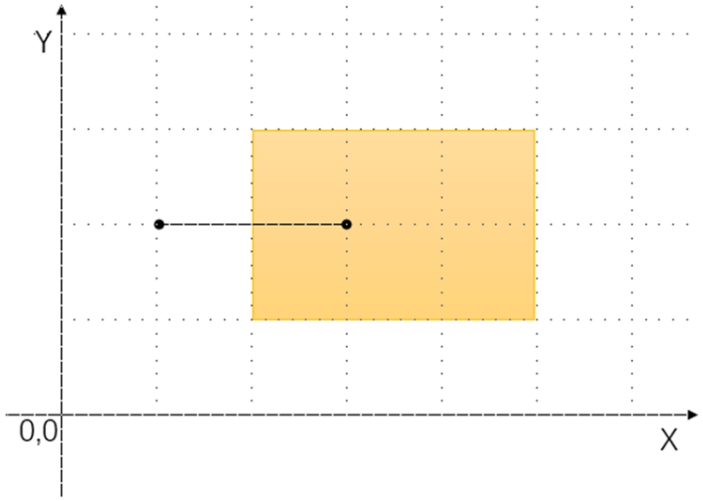
当两个几何图形的这三个部分（内部/边界/外部）之间没有交集时，将用”F”填充矩阵中对应的部分。请注意，以上两个要素的边界实际上根本不相交（线的端点与多边形的内部相交，而不是与多边形的边界相交，反之亦然），因此B/B单元用”F”填充。



我们通过（ST_Relate）函数让计算机自动填写DE9IM矩阵。DE9IM矩阵的强大之处在于使用它们作为匹配参数来查找彼此之间具有特定关系的几何图形。

9.5.3 查找具有空间关系的集合图形

前面的示例可以使用简单的矩形和直线进行简化，其空间关系与上面的多边形和线串的空间关系相同：



使用SQL生成DE9IM信息：

```
SELECT ST_Relate(
         'LINESTRING(1 2, 3 2)',
         'POLYGON((2 1, 5 1, 5 3, 2 3))'
);
```

输出结果如下：

```
101
0F0
212
```

9.5.4 示例实践

假设我们需要在街区（Blocks）和道路（Roads）的两组要素数据中，寻找符合要求的道路，该处需符合一下两个条件：

1.该道路必须位于街区内部

2.且该路口必须临街区边界的位置。

接下来我们利用DE9IM在数据库中找到所有符合这一规则的路口：

首先，在数据库中加入Blocks和Roads，按序插入数据：

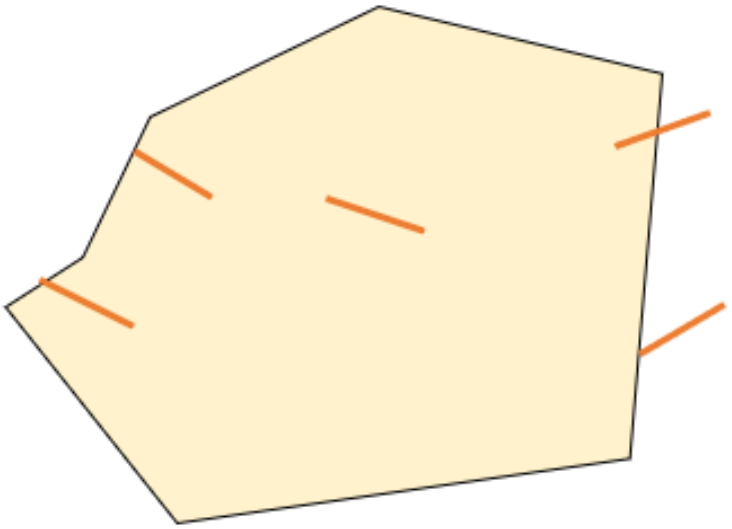
```
CREATE TABLE Blocks ( id serial primary key, geom geometry );
CREATE TABLE Roads ( id serial primary key, good boolean, geom geometry );

INSERT INTO Blocks ( geom )
```

(continues on next page)

(continued from previous page)

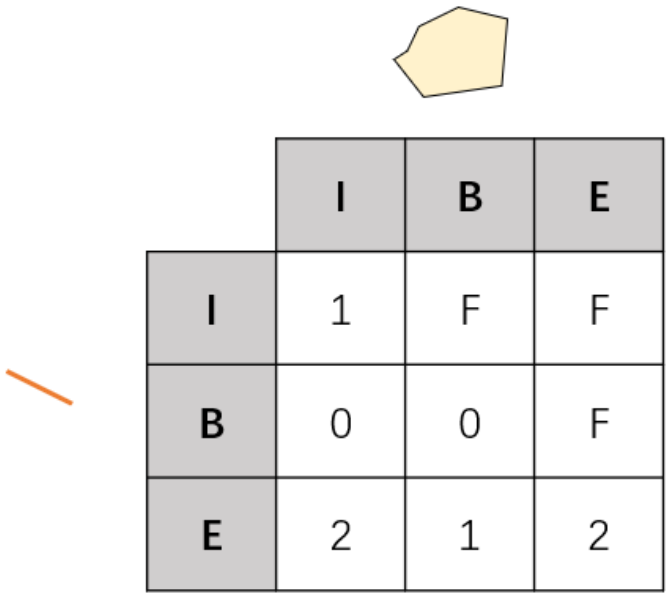
```
  VALUES ( 'POLYGON ((100 200, 140 230, 180 310, 320 380, 470 340, 460 120, 200 70,
↪100 200))');

INSERT INTO Roads ( geom, good )
  VALUES
        ('LINESTRING (170 290, 205 272)',true),
        ('LINESTRING (120 215, 176 197)',true),
        ('LINESTRING (280 270, 340 250)',false),
        ('LINESTRING (450 195, 510 310)',false),
        ('LINESTRING (460 180, 520 200)',false);
```

两组要素数据如下图所示：



- 根据上述需求，符合要求的道路具有以下特点：
- 道路内部与街区内部有一个线性（一维）相交
- 道路边界与街区内部有一个点（0维）相交
- 道路边界与街区边界也有一个点（0维）相交
- 道路内部与街区外部没有相交（F）

所以关于街区（Blocks）和道路（Roads）的DE9IM矩阵类似于下图所示：



	I	B	E
I	1	F	F
B	0	0	F
E	2	1	2

```
id | good | geom
----+------+----------------------------------------------------------------------------------
↪----------
  1 | t    |␣
↪0102000000020000000000000000406540000000000020724000000000000A06940000000000000007140
  2 | t    |␣
↪0102000000020000000000000000005E400000000000E06A4000000000000066400000000000A06840
(2 rows)
```

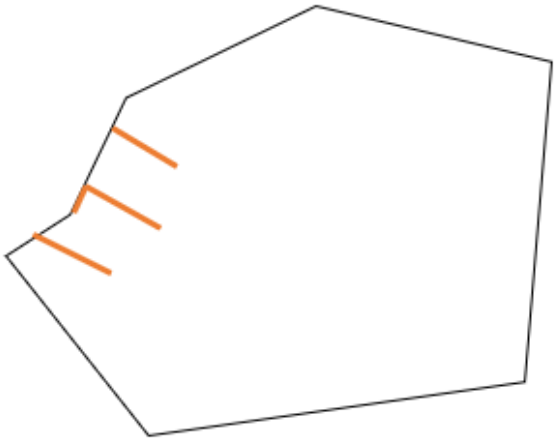
(*注意，ST_Relate的三参数版本（重载函数）的使用，如果前两个几何图形参数的关系与第三个DE9IM模型参数匹配，返回ture，如果不匹配，则返回false)

另外，对于更松散的匹配搜索，第三个参数允许DE9IM数据模型字符串使用通配符：

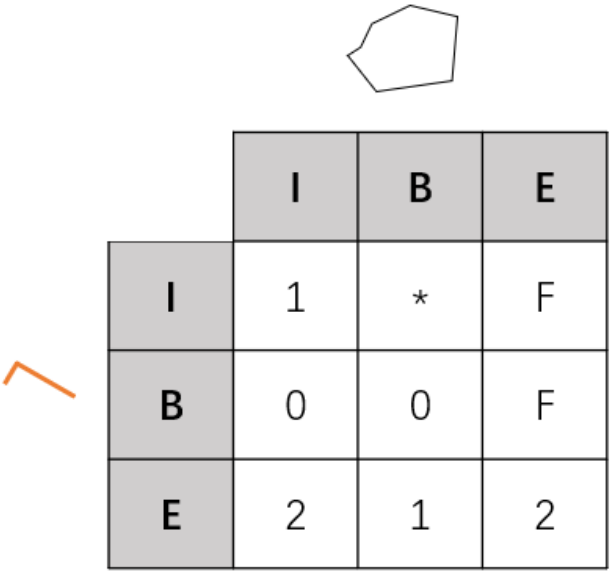
- “*”表示此单元格中的任何值都可以接受
- “T”表示任何非假值（0、1或2）都可以接受

例如，我们在示例图形中添加一个与街区边界具有二维相交的道路：

```
INSERT INTO Roads ( geom, good )
  VALUES ('LINESTRING (140 230, 150 250, 210 230)',true);
```



将新增的加到在ST_Relate函数中符合要求，则需要修改ST_Relate函数中的第三个参数，因为道路内部和街区边界相交是1或F的情况皆可，因此，我们使用”*“通配符覆盖所有情况。



	I	B	E
I	1	*	F
B	0	0	F
E	2	1	2

SQL语句如下所示：

```
SELECT Roads.*
FROM Roads JOIN Blocks ON ST_Intersects(Roads.geom, Blocks.geom)
WHERE ST_Relate(Roads.geom, Blocks.geom, '1*F00F212');
```

输出以下结果：

```
id | good |                                                                  geom

  1 | t    |␣
↪0102000000020000000000000000406540000000000020724000000000000A0694000000000000007140
```

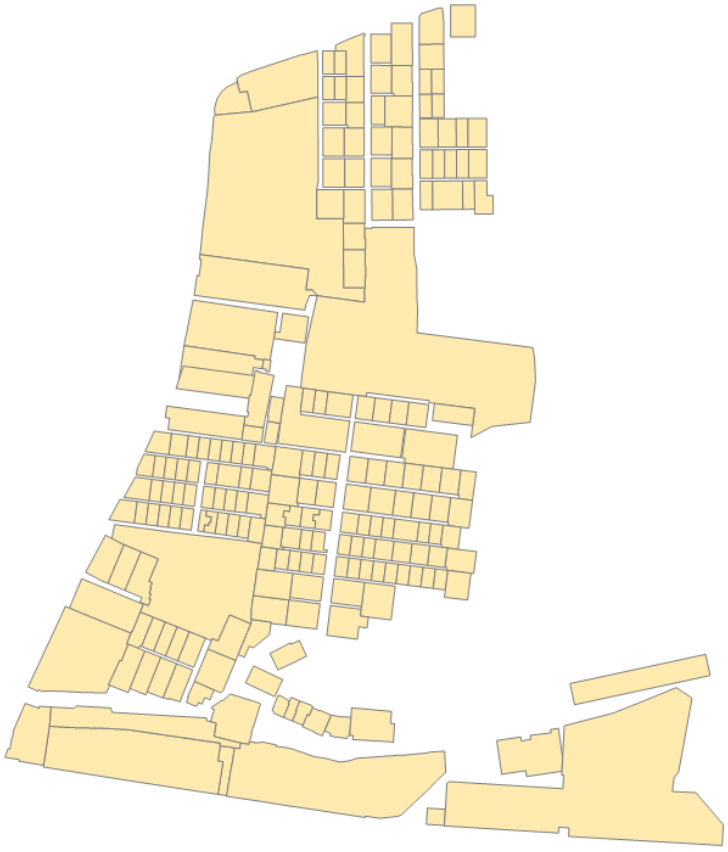
(continues on next page)

(continued from previous page)

```
  2 | t     |␣
↪0102000000020000000000000000005E400000000000E06A4000000000000066400000000000A06840
  6 | t     |␣
↪0102000000030000000000000000080614000000000000C06C4000000000000C062400000000000406F4000000000000406A40
(3 rows)
```

9.5.5 应用场景：地籍图斑数据质量校验

利用DE9IM模型对地籍数据库图斑进行拓扑关系分析，可大大降低数据检查的工作量，基于拓扑规则的拓扑关系验证方法合理有效。 首先，利用SuperMap iDesktopX桌面工具将地籍数据集（DCK_ZD.udb)导入到Yukon数据库中，如图所示



```
SELECT a.SmID, b.SmID
FROM dck_zd_1 a, dck_zd_1 b
WHERE ST_Intersects(a.smgeometry, b.smgeometry)
  AND ST_Relate(a.smgeometry, b.smgeometry, '2********')
  AND a.SmID != b.SmID
LIMIT 10;
```

(*注：2********为判定空间叠加的要求矩阵；且由于数据地块数据要素较多，因此输出限定10组)

输出以下结果：

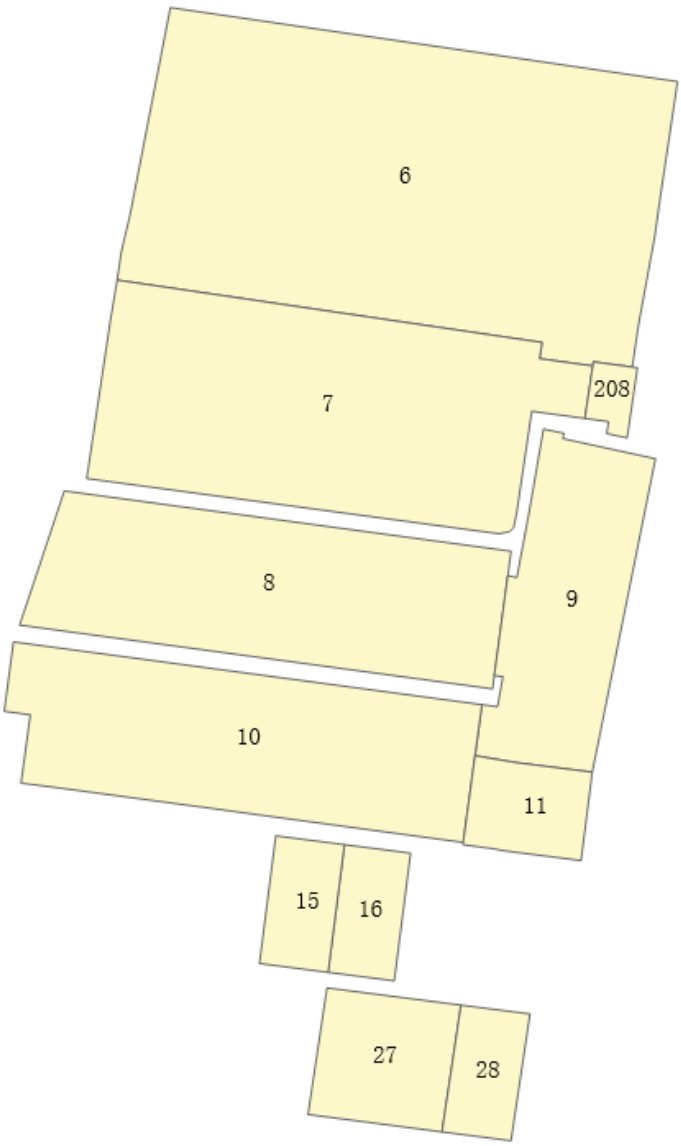
```
smid | smid
------+------
```

(continues on next page)

(continued from previous page)

```
   6 |  208
   7 |    8
   8 |    7
   8 |    9
   9 |    8
  10 |   11
  11 |   10
  15 |   16
  16 |   15
  27 |   28
(10 rows)
```

如图所示，实现对地籍数据进行校验检查，快速筛选具有空间叠加关系的地块：



9.6 Yukon中GeoSOT 编码的基本能力

示例数据：demo-geosot.udbx ，包含二维点、线、面、三维模型（精模和BIM）；

SQL客户端：DBeaver或其他postgre客户端工具

可视化工具：DBeaver、QGIS或SuperMap iDesktopX；本文为展示三维场景效果，使用SuperMap iDesktopX。

9.6.1 二维线、面数据编码

对二维点、线、面数据的geometry列计算geosotgrid，并转换为geometry对象；

示例数据：building_point_demo、builing_line_demo、building_region_demo。

```
---1. 对二维点对象编码
---1.1 创建表存储二维网格结果：building_point_grid22
create table building_point_grid22
    (smid serial4 not null ,
    smgeometry geometry(polygon, 4490),
    asText varchar(765),
    grid GeoSOTGrid,idGeo int4) ;
---1.2 为 building_point_demo 表的smgeometry列构造22层网格，并将格网转换成geometry对象，写入
with a as (select smid,ST_GeoSOTGrid(smgeometry,22) as grids from building_point_demo␣
↪ )
    insert into building_point_grid22(smgeometry,asText,grid,idGeo)
    select ST_GeomFromGeoSOTGrid(unnest(grids)),ST_AsText(unnest(grids)),
↪unnest(grids),smid from a;

---2. 对二维线对象编码
---2.1 创建表存储二维网格结果：building_line2d_grid22
create table building_line2d_grid22
    (smid serial4 not null ,
    smgeometry geometry(polygon, 4490),
    asText varchar(765),
    grid GeoSOTGrid,idGeo int4) ;
---2.2 为 builing_line_demo 表的smgeometry列构造22层网格，并将格网转换成geometry对象，写入
with a as (select smid,ST_GeoSOTGrid(smgeometry,22) as grids from builing_line_demo  )
    insert into building_line2d_grid22(smgeometry,asText,grid,idGeo)
    select ST_GeomFromGeoSOTGrid(unnest(grids)),ST_AsText(unnest(grids)),
↪unnest(grids),smid from a;

---3. 对二维面对象编码
---3.1 创建表存储二维网格结果：building_region2d_grid22
create table building_region2d_grid22
    (smid serial4 not null,
     smgeometry geometry(polygon, 4490),
     asText varchar(765),
     grid GeoSOTGrid,idGeo int4);
---3.2 为 building_region_demo 表的 smgeometry 列构造22层网格，并将格网转换成geometry对象，写
入
with a as (select smid,ST_GeoSOTGrid(smgeometry,22) as grids from building_region_
↪demo )
    insert into building_region2d_grid22(smgeometry,asText,grid,idGeo)
    select ST_GeomFromGeoSOTGrid(unnest(grids)),ST_AsText(unnest(grids)),
↪unnest(grids),smid from a;
```

新生成的building_point_grid22、building_line2d_grid22和building_region2d_grid22表格可视化效果如下：

9.6.2 三维模型对象编码

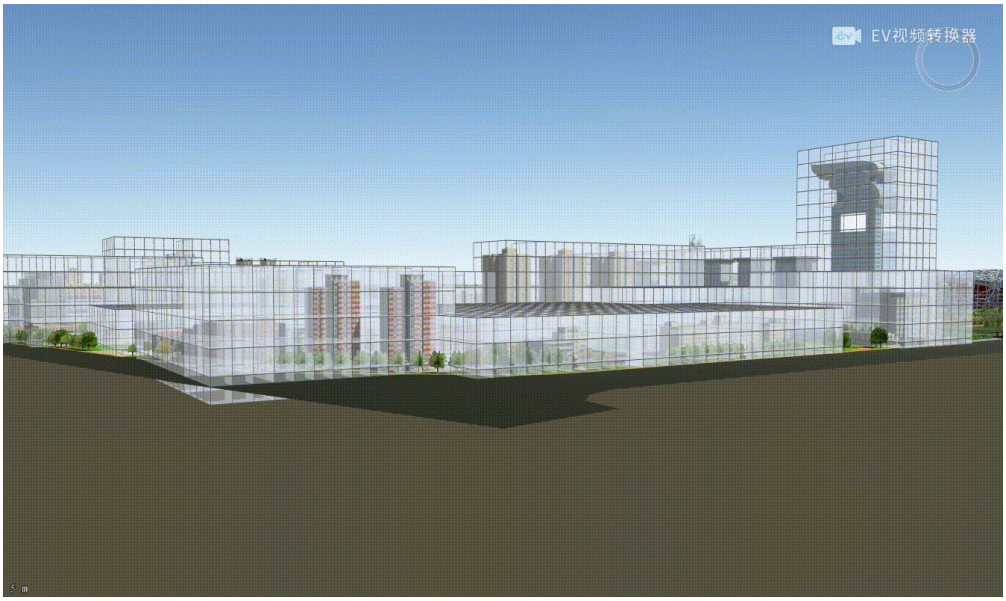
精模数据编码

取三维模型对象的包围盒，计算三维编码，并转换为geometry对象；

示例数据：building_1。

```
---4. 对三维模型对象编码
---4.1 创建表存储三维网格：building_1_grid22
create table building_1_grid22 (smid serial4 not null , smgeometry␣
↪geometry(multipolygonz, 4490),asText varchar(765),grid GeoSOTGrid,idGeo int4);
---4.2 为 building_1 表的 smgeometry 列构造22层网格，并将格网转换成geometry对象，写入
--- 注：st_boundary(geomodel)方法得到的Box3D，z方向起算点在地心，GeoSOT的z方向起算点在地表，所以
需要进行平移处理
with a as (select smid,ST_GeoSOTGrid(st_translate(st_setsrid(st_boundary(smgeometry),
↪4490), 0, 0, -6378137),22) as grids from building_1 )
    insert into building_1_grid22(smgeometry,asText,grid,idGeo)
    select ST_GeomFromText(regexp_replace(st_astext( ST_
↪GeomFromGeoSOTGrid(unnest(grids)) ),'POLYHEDRALSURFACE','MULTIPOLYGON')),ST_
↪AsText(unnest(grids)),unnest(grids),smid from a;
```

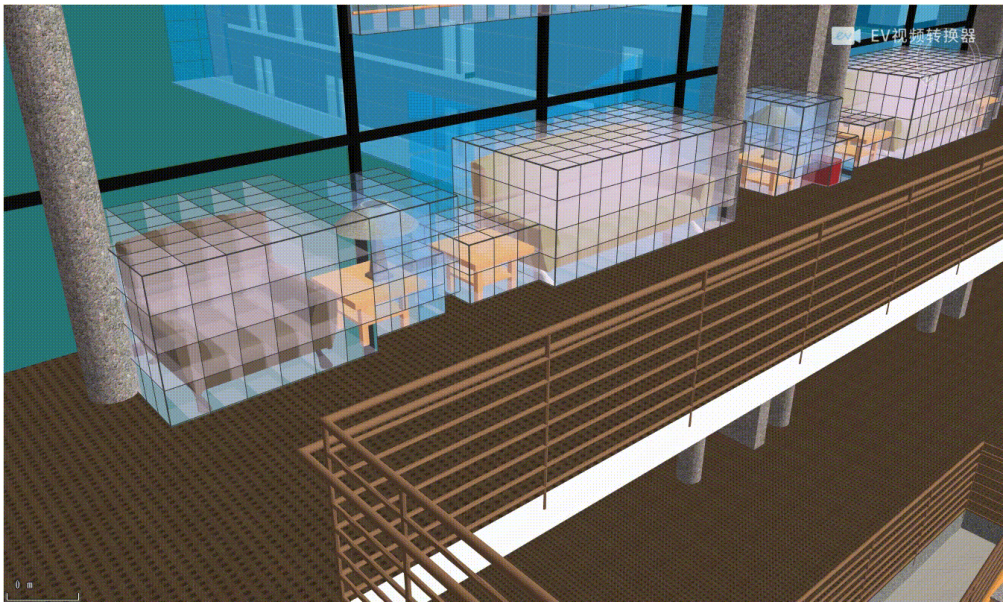
新生成的building_1_grid22表格与原始三维模型叠加后可视化效果如下：



BIM数据编码

步骤同上。示例数据：

效果图如下：



9.6.3 网格层级聚合

从已有的15层网格生成12层网格。 示例数据：bj_1。

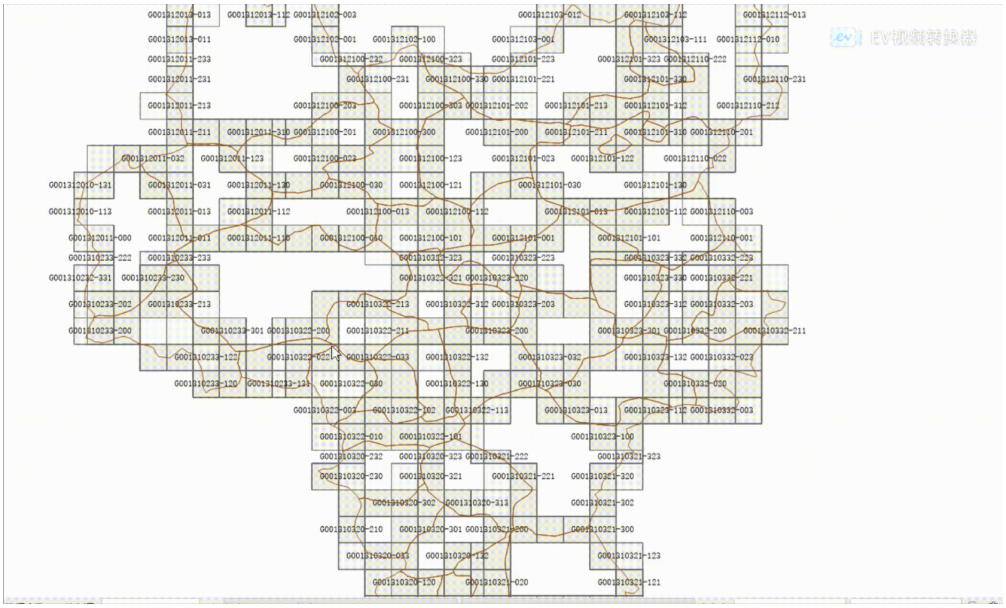
```
-- 5. 生成粗糙层网格
-- 5.1 创建表存储15层级网格数据：building_line_grid15
create table building_line_grid15 (smid serial4 not null, smgeometry geometry(polygon,
↪ 4490),asText varchar(765),grid GeoSOTGrid,idGeo int4);
```

(continues on next page)

(continued from previous page)

```
-- 5.1 为 bj_1表的smgeometry列构造15层网格，并将格网转换成geometry对象，写入
with a as (select smid,ST_GeoSOTGrid(smgeometry,15) as grids from bj_l  )
        insert into building_line_grid15(smgeometry,asText,grid,idGeo)
        select ST_GeomFromGeoSOTGrid(unnest(grids)),ST_AsText(unnest(grids)),
↪unnest(grids),smid from a;
-- 5.2 创建表存储12层级网格数据：building_line_grid12
create table building_line_grid12 (smid serial4 not null, smgeometry geometry(polygon,
↪ 4490),asText varchar(765),grid GeoSOTGrid,idGeo int4);
-- 5.3 从步骤5.1的15层网格building_line_grid15生成12层网格
with a as (select ST_Aggregate(grid,12) grid , idGeo from building_line_grid15 )
    insert into building_line_grid12 (smgeometry,asText,grid,idGeo) (select ST_
↪GeomFromGeoSOTGrid(grid),
    ST_AsText(grid), grid,idGeo from a );
```

geosot编码的第12层为不规则网格，本例中新生成的12层网格可视化效果如下：

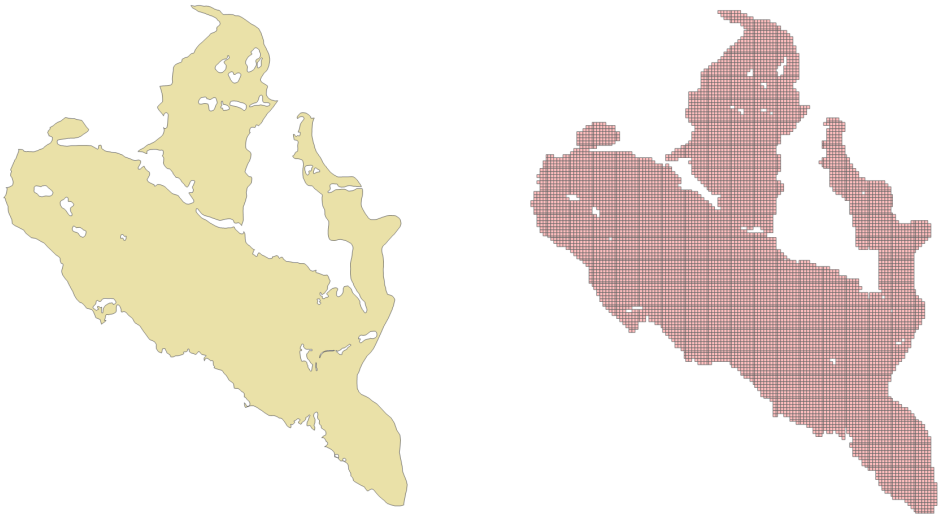


9.6.4 带洞的面对象编码

对带洞的面对象编码处理过程与普通geometry一致；示例数据：hole。

```
-- 对 hole 里的对象进行编码，并转成geometry存在tb_18表中
CREATE SEQUENCE public.tb_18_id_seq INCREMENT BY 1  MINVALUE 1 MAXVALUE 2147483647␣
↪START 1 CACHE 1 NO CYCLE;
CREATE TABLE public.tb_18 (id int4 NOT NULL DEFAULT nextval('tb_18_id_seq'::regclass)␣
↪primary key, g geometry);

insert into tb_18( g)
select ST_GeomFromGeoSOTGrid(unnest(ST_GeoSOTGrid(smgeometry, 18) ) ) from hole h ;
```

网格化前后效果如下：

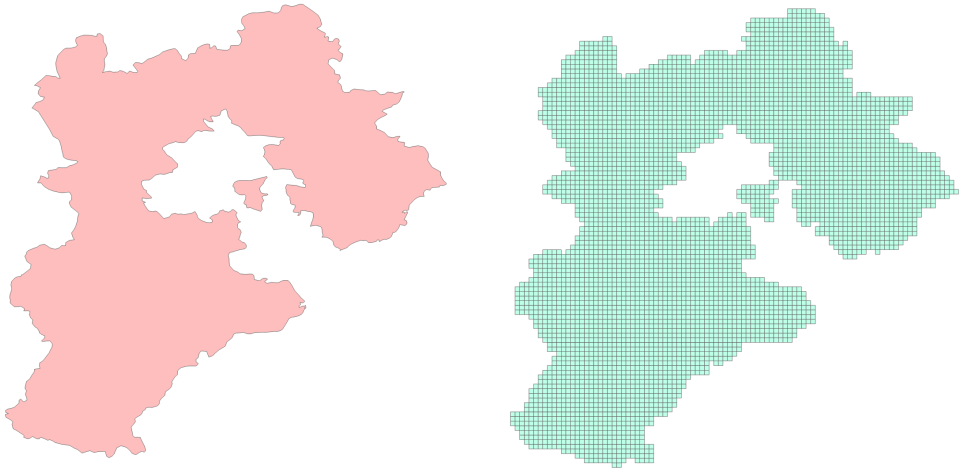


9.6.5 带飞地的面对象编码

对带飞地的面对象编码处理过程与普通geometry一致；示例数据：hebei。

```
-- 对示例数据里的对象进行编码，并转成geometry存在tb_13表中
CREATE SEQUENCE public.tb_13_id_seq INCREMENT BY 1  MINVALUE 1 MAXVALUE 2147483647␣
↪START 1 CACHE 1 NO CYCLE;
CREATE TABLE public.tb_13 (id int4 NOT NULL DEFAULT nextval('tb_13_id_seq'::regclass)␣
↪primary key, g geometry);

insert into tb_13( g)
select ST_GeomFromGeoSOTGrid(unnest(ST_GeoSOTGrid(smgeometry, 13) ) ) from hebei ;
```

网格化前后效果如下：

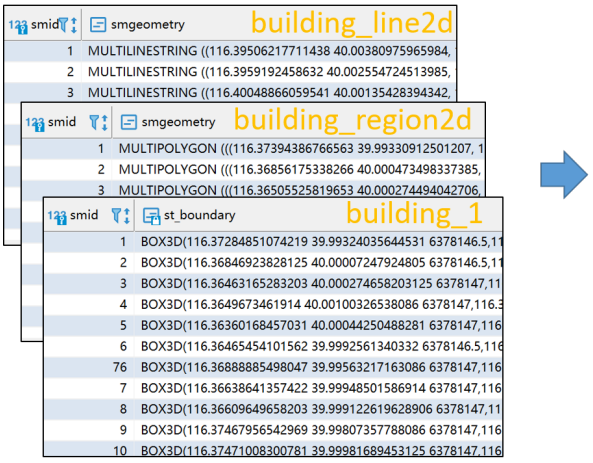


9.7 基于GeoSOT编码的多图层穿越查询

在海量空间数据管理系统中，经常用到多图层查询的应用，比如查看指定范围内有哪些图层，各图层在此区域大约有多少对象并获取属性相关的统计数据。 传统的实现方式，是对所有图层进行范围过滤，尽管有各图层有空间索引进行加速，随着图层数量的增加，仍无法满足实时或近实时查询效果。 面向网格的空间数据管理方式，能够很好地解决这个问题。本文详细说明其使用过程。

9.7.1 基本原理

该使用场景的基本原理：对原始数据的各图层对象进行编码，保留对象ID和网格，汇总到一张网格编码表，示例如下：



原始数据

tablename	smid	grids_level_22
building_line2d	1	{0764045538E0000000160000,0764045538F0000000160000,
building_line2d	2	{076404551C0000000160000,076404551C1000000016000
building_line2d	3	{076404554D6000000160000,076404554D7000000016000
building_line2d	4	{0764045518500000001600000,0764045519000000001600
building_line2d	5	{076404554DD0000000160000,0764045558800000001600
building_line2d	6	{0764045561200000001600000,0764045561300000001600
building_line2d	7	{076405002BC0000000160000,076405002BD0000000160000
building_region2d	1	{074EAE7A47A0000000160000,074EAE7A47B0000000160000
building_region2d	2	{0764045005B0000000160000,0764045005E0000000160000
building_region2d	3	{0764044545200000001600000,0764044544900000001600
building_region2d	4	{0764044547A0000000160000,0764044547B0000000160000
building_region2d	5	{0764044541F0000000160000,0764044544A0000000160000
building_region2d	6	{074EAE6FE5C0000000160000,074EAE6FE5D0000000160000
building_region2d	7	{074EAE6FF3400000001600000,074EAE6FF3500000001600000
building_1	1	{00B21A49A3D232B5A0824000001600001,00B21A49A3D232
building_1	2	{00B2880083401291208240000016000 1,00B2880083401291
building_1	3	{00B2880083093281E0824000001600 1,00B2880083093283
building_1	4	{00B28800830932A7E0824000001600 1,00B28800830932B5
building_1	5	{00B2880083093205E0824000001600 1,00B2880083093207

网格数据表

查询过程： 1. 计算对象压盖的网格，层级必须与被查询列的网格层级一致； 2. 计算出的网格数组与网格编码表的网格数组列进行求交，有相同网格的行被命中； 3. 命中的行再与原始数据做ID关联。

9.7.2 脚本及过程说明

```
-- 1. 创建网格编码表格 tb_geogrid
------1.1  创建自增序列对象
CREATE SEQUENCE public.tb_geogrid_id_seq INCREMENT BY 1  MINVALUE 1 MAXVALUE␣
↪2147483647 START 1 CACHE 1 NO CYCLE;
------1.2  创建表格 tb_geogrid，编码值将存储在 grids_level_15 列
CREATE TABLE public.tb_geogrid (id int4 NOT NULL DEFAULT nextval('tb_geogrid_id_seq
↪'::regclass), tablename text NULL,
    smid int8 NULL, grids_level_15 _geosotgrid NULL);

-- 2. 各图层对象编码计算并数据写入tb_geogrid表的grids_level_15列，15层编码；图层名以 xxx 示意
------ 注意： st_geosotgrid(smgeometry, 15) 中，smgeometry 是否带z值，决定了生成的网格是否
带z值；同一列中的网格必须具有相同层级和z方向标记
insert into tb_geogrid(tablename, smid, grids_level_15 ) (select 'xxx', smid ,st_
↪geosotgrid(smgeometry, 15) from xxx);

-- 3. 为 geogrid表 的 grids_level_15 列创建gin索引
CREATE INDEX idx_tb_geogrid_grids ON public.geogrid USING gin (grids_level_15);
```

(continues on next page)

(continued from previous page)

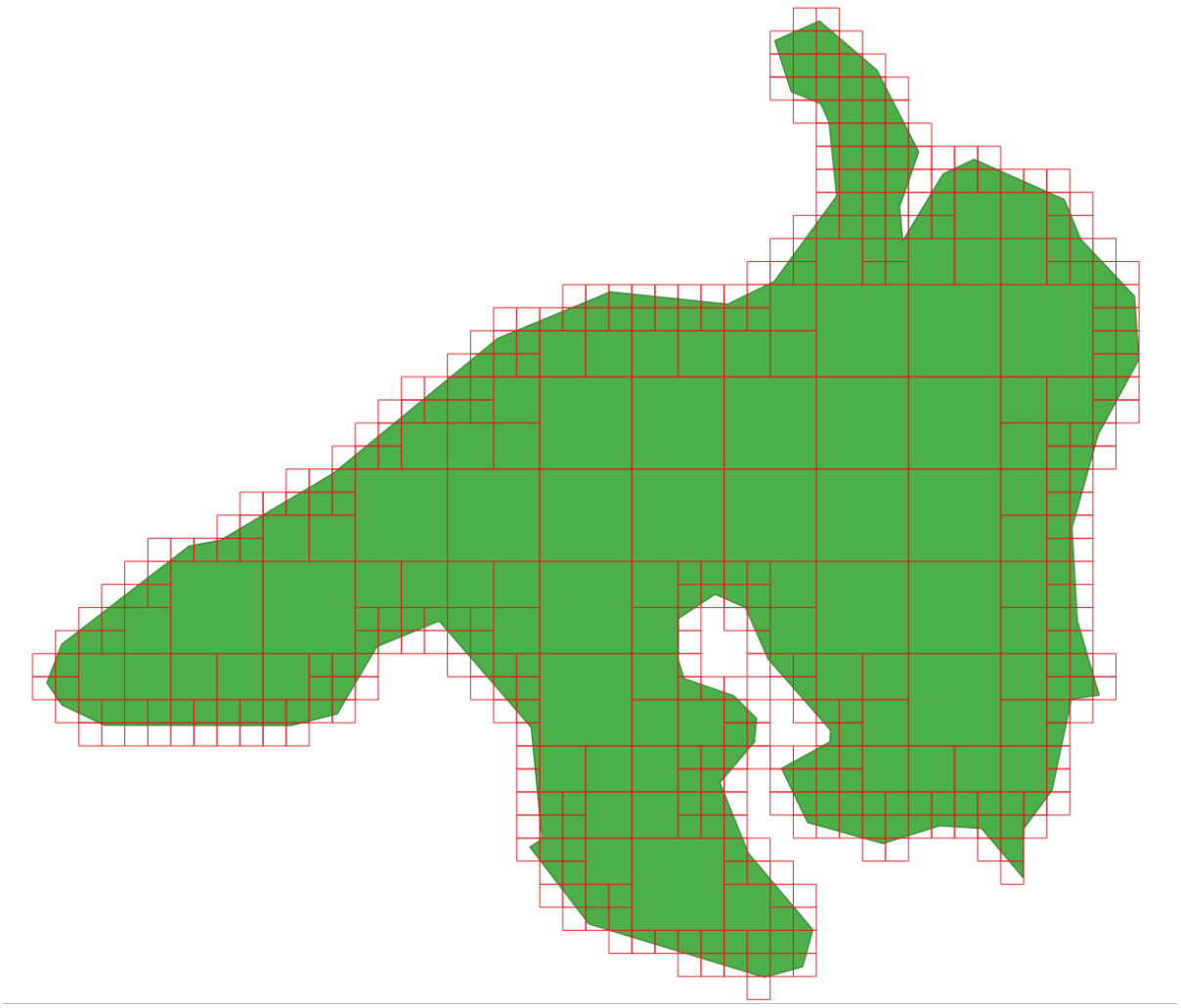
```
-- 4. 查询过滤
select tablename,count(*) from tb_geogrid  where grids_level_15 && st_geosotgrid(ST_
↪MakeEnvelope(116.4, 40, 116.45, 40.05, 4490), 22) group by tablename;
```

本例提供的基于网格编码的多图层穿越查询方法，不影响原始数据，提供高效检索性能的同时，也适用于对只读的存量数据管理。

9.8 基于混合层级空间网格编码的查询

*Yukon*支持*GeoSOT*编码的基本原理及特性 中详述了空间网格编码的基本原理，空间网格编码通常为GeoSOTGrid 类型的数组。 Yukon提供的接口 ST_GeoSOTGrid( )，可通过指定的网格层级参数计算空间网格编码，但这种方式构造出的网格均为同一网格层级。当对象的空间范围较大时，仍使用同一层级的网格，构造出的空间网格编码数组长度将超过设定值域范围，无法存储。另一方面，对于这些范围大的对象，采用太小尺度的网格进行管理和查询等，也是没有必要的，特别是在要素查询中，获得同等查询精度的前提下，小尺度的网格将带来更多的性能损耗。

因此，Yukon提供接口 ST_GeoSOTGridAgg( )，通过设置一个范围的网格层级（level_min,level_max），计算混合层级的空间网格编码。每个对象对应的空间网格编码数组将包含（level_min,level_max）范围下多个层级的空间网格编码，并提供对此空间网格编码数组创建 GIN 索引，进行查询。



以下将对基于此混合层级空间网格编码，进行数据查询的过程进行说明。

示例数据: region_test.udbx

9.8.1 主要步骤及结果说明：

1. 创建数据库中空间对象的空间网格编码。数据集名region_test，对应的网格编码表region_grid22_15。

```
CREATE TABLE region_grid22_15 (id serial4 not null,tablename text NULL,smid int8 NULL,
↪       grids22_15 geosotgrid[] NULL);

INSERT INTO region_grid22_15 (tablename, smid, grids22_15) (select 'region_test',␣
↪smid ,st_geosotgridagg(smgeometry, 22, 15) from region_test);
```

2. 对空间网格编码数组创建 GIN 索引。

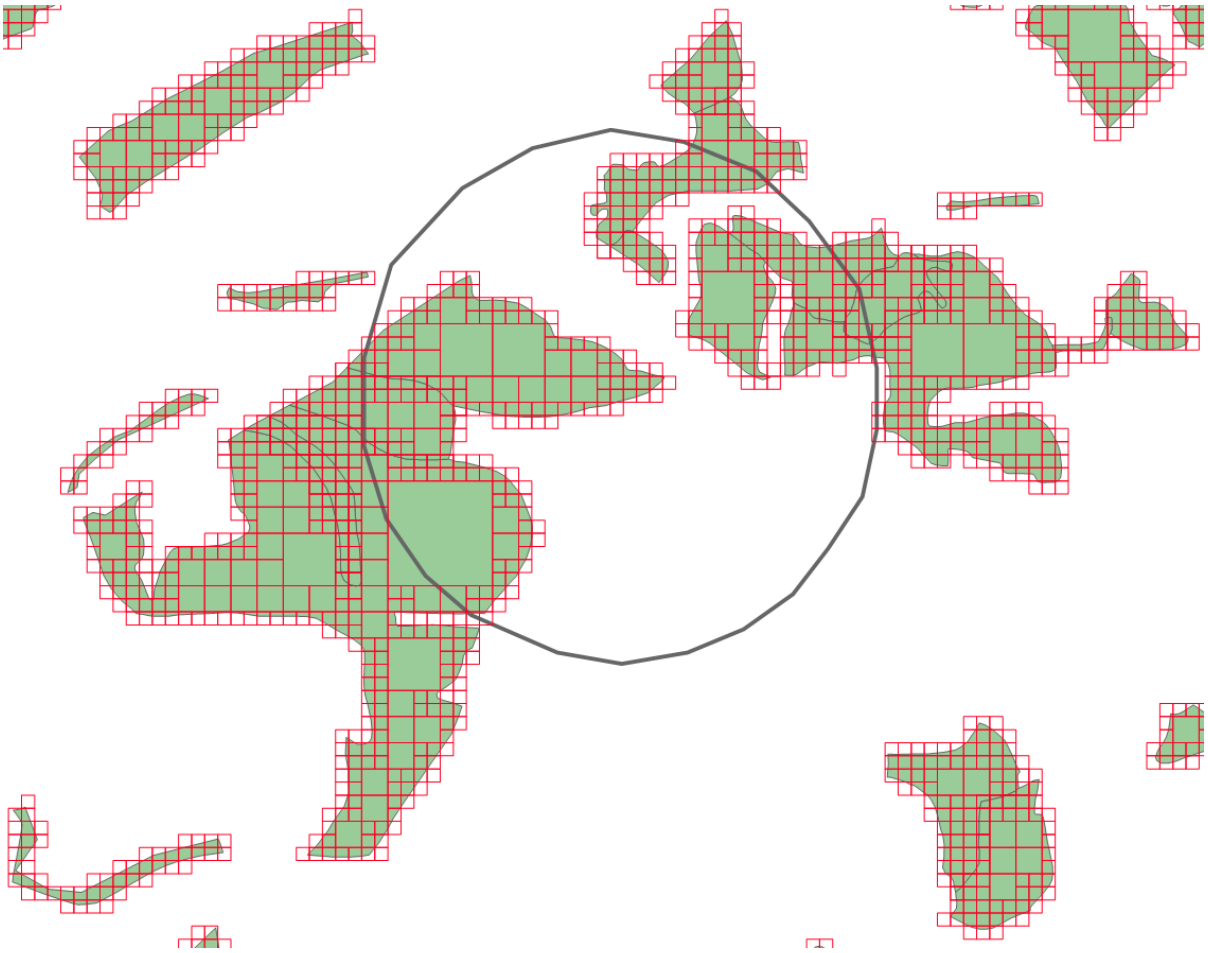
```
CREATE INDEX idx_region_grid22_15 ON region_grid22_15 USING gin (grids22_15);
```

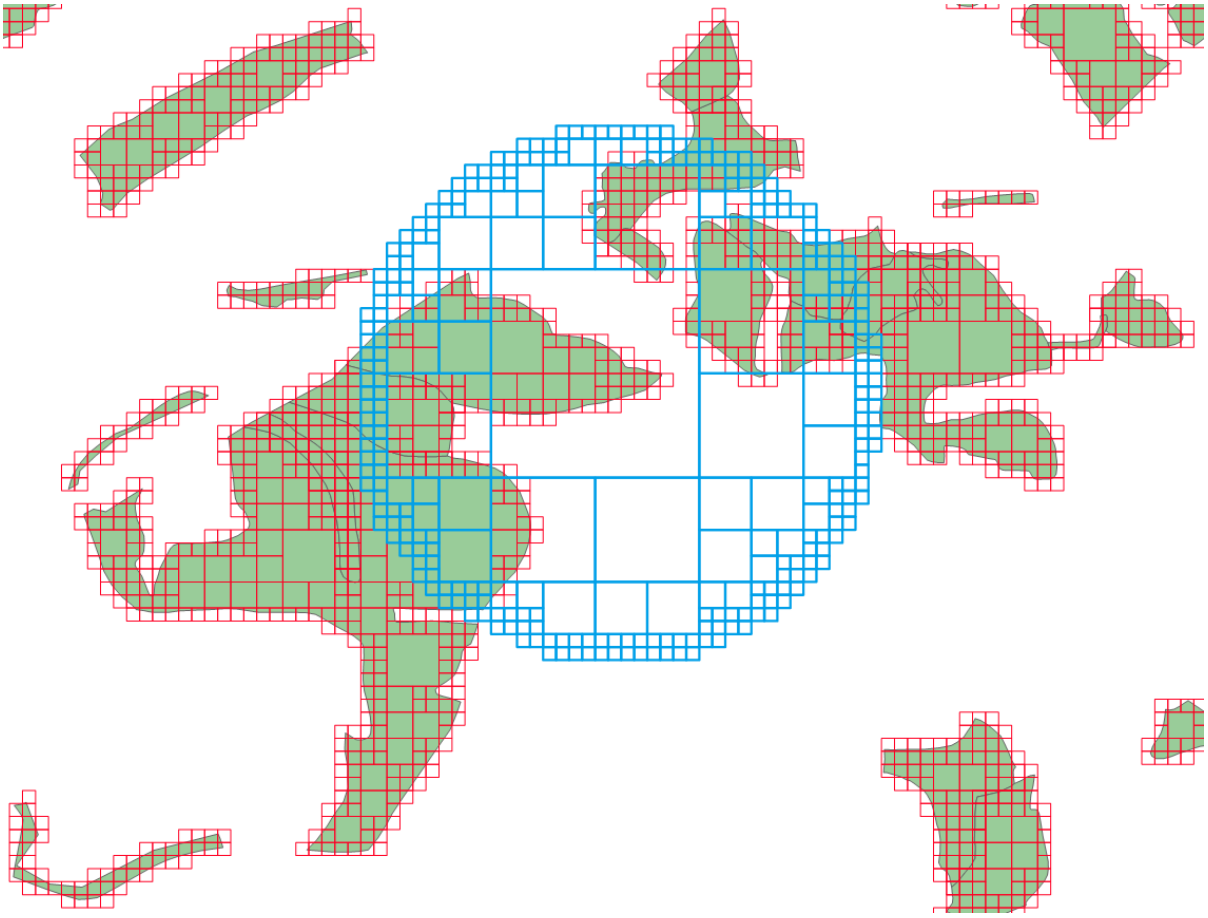
3. 创建查询范围的空间网格编码。下图圆框为查询范围，查询范围的数据集名roi，对应的网格编码表roi_grid22_15。

```
CREATE TABLE roi_grid22_15 (id serial4 not null,tablename text NULL,smid int8 NULL,␣
↪grids22_15 geosotgrid[] NULL);

INSERT INTO roi_grid22_15 (tablename, smid, grids22_15) (select 'roi', smid ,st_
↪geosotgridagg(smgeometry, 22, 15) from roi);
```

(continues on next page)

(continued from previous page)





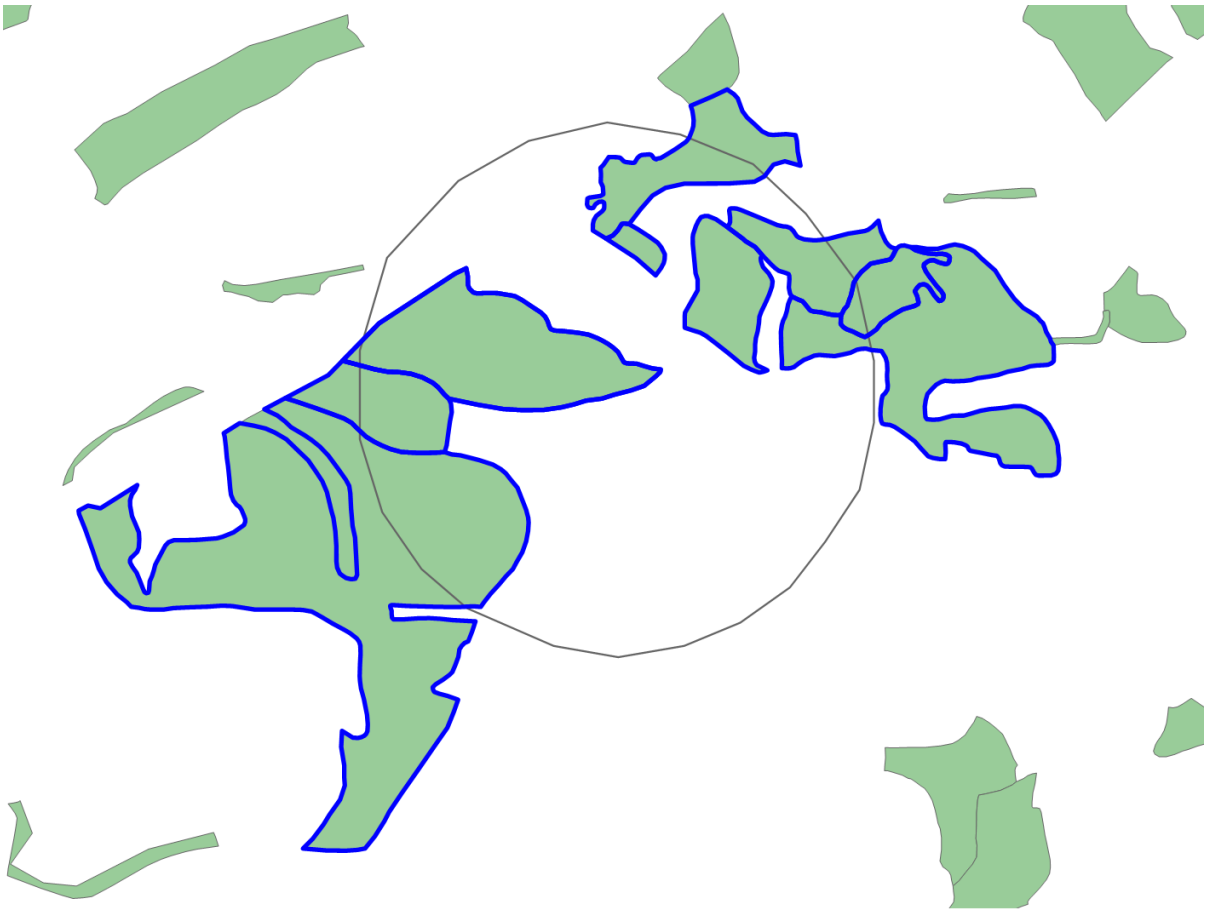
4. 执行网格相交查询，采用操作符&&，对两个网格编码表求交集。

```
select region_grid22_15.smid from region_grid22_15, roi_grid22_15 where roi_grid22_15.
↪grids22_15 && region_grid22_15.grids22_15;
```

特殊说明： 查询框的网格编码层级level_min必须小于等于空间数据网格编码的level_min，才能将查询框覆盖的空间对象完整查出。

5. 根据4中的查询结果，关联到对应的空间对象，得到最终查询结果。

```
with region_roi as (select region_grid22_15.smid from region_grid22_15, roi_grid␣
↪where aa_1000_grid.grids22_15 && roi_grid.grids22_15)

select region_test.* from region_test, region_roi where region_test.smid = region_roi.
↪smid;
```



9.8.2 空间网格编码层级如何设置?

这里设置的是查询范围的空间网格编码层级。

空间网格编码层级的设置，需要考虑查询精度和性能两个因素。层级越高，意味着网格单元对应的空间范围越小，查询的精度会相对提高，但性能会相应降低。这里建议以下几种场景下使用混合层级的空间网格编码进行查询：

- 查询范围面积较大，采用同一层级的空间网格编码无法存储。
- 查询范围面积较大，虽然可以采用同一层级编码，但性能较差，在不相对损失精度的前提下，采用混合层级的空间网格编码。

当选择了混合层级的空间网格编码后，层级范围的选择需注意：

- 查询框的level_min ⩽ 空间对象的level_min，才能将查询框覆盖的空间对象完整查询。
- 查询框的level_max ⩾ 空间对象的level_max，查询结果可以更加准确。

最后，在选择单层级空间网格编码还是混合层级空间网格编码时，需考虑的情况是比较复杂的。因此还需要综合考虑实际情况，选择最合适的方式。

CHAPTER 10

FAQ

- 为什么我无法卸载 openGauss的Yukon插件?

  由于目前 openGauss 的白名单机制，自定义插件的卸载，除PostGIS 等名单内的插件，其它均不支持卸载。请参考 详情参考 插件卸载

- 为什么一直连接不到数据库?

  1. 检查防火墙状态，防火墙开启会导致连接数据库失败，请确保已关闭防火墙。
  2. 连接出现“NullPointerException”的错误，由于 openGauss 默认使用 sha256 加密，你需要修改为md5加密，尝试解决方法：检查postgresql.conf的password_encryption_type=0并且已启用，如果被注释，请启用参数，数据库重启后重新创建用户名和密码。你也可以参考 连接问题 。
  3. 出现“拒绝从外部以非加密保护的密码进行连接”的错误，检查 pg_hba.conf，IPv4 local connections中新增 *host all all 0.0.0.0/0 md5* 。

- 为什么在创建扩展时提示没有权限访问动态库?

  检查当前启动数据库的操作系统用户是否有相关动态库文件（参看具体报错信息）的读取权限。

- openGauss 中的 simpleinstall 和 文档中的安装有那些不同?

  simpleinstall 只支持在一台机器上，安装单机、或者一主一备。 只有内核，没有 om 工具

  文档中的复杂安装是 OM 安装，可以安装 一主 N 备，包括 CM。 支持升级、增删节点等功能。

CHAPTER 11

联系我们

如果您有任何问题，可以到我们的源码托管网站提交 ISSUE，我们会为您尽快解答。

Yukon for openGauss；

Yukon for PostgreSQL。

CHAPTER 12

附录

12.1 发行说明

12.1.1 A.0 详细版本历史

Yukon版本	post-gis版本	post-gis_sfcgal版本	post-gis_raster版本	yukon_geomodel版本	yukon_geogridcoder版本
1.0.1(for PostgreSQL)	3.2.2	3.2.2	3.2.2	1.0.1	1.0.1
1.0.1(for openGauss)	3.2.1	3.2.1	3.2.1	1.0.1	1.0.1
1.0.1(for GaussDB)	3.2.1	3.2.1	3.2.1	1.0.1	1.0.1
1.0(for PostgreSQL)	3.1.2	3.1.2	3.1.2	1.0	1.0
1.0(for openGauss)	3.2.0	3.2.0	3.2.0	1.0	1.0

12.1.2 A.1 Release Yukon 1.0.1

发布日期：2022/12/30

A.1.1 新特性

- 支持计算空间对象的最小外包空间网格。涉及接口：ST_GeoSOTGrid();
- 支持一次创建指定网格层级范围内的多层级空间网格。涉及接口：ST_GeoSOTGridAgg();
- 支持获取所有空间网格的层级范围。涉及接口：ST_GetLevelExtremum();

- geosotgrid[] 数据类型扩展操作符 <@、@>，支持计算网格数组间的包含和被包含关系；
- 支持通过 GIN 索引对多层级网格下的空间对象进行查询。
- 新增 yk_tool 工具，可进行安装与卸载、查看当前版本信息。

A.1.2 缺陷修复

Yukon for openGauss:

- 修复通过 GIST 索引查询时的崩溃问题；
- 修复多线程下对 Geometry 解析错误的问题；
- 修复空字符无法解析的问题。涉及接口：ST_AsX3D()、ST_AsKML()、ST_AsGML( )、ST_GeomFromGML( )、ST_GeomFromKML( )、ST_AsLatLonText( )、UpdateGeometrySRID()、DropGeometryColumn(
- 修复 UpdateGeomodelSRID() 接口更新空间坐标参考系无效的问题；

Yukon for PostgreSQL:

- 修复 UpdateGeomodelSRID( ) 接口更新空间坐标参考系无效的问题。

A.1.3 已知问题

同 1.0.0 Beta 版本，参见 *A.3.2* 已知问题， Yukon for GaussDB和Yukon for openGauss的已知问题相同。

Yukon for GaussDB 推荐优先使用 postgis,postgis_raster, postgis_sfcgal 三个模块，可试用 yukon_geomodel 和 yukon_geogridcoder 模块。

12.1.3 A.2 Release Yukon 1.0.0

发布日期：2022/06/30

A.2.1 功能列表

优化边缘网格划分。

A.2.2 已知问题

同 1.0.0 Beta 版本。

12.1.4 A.3 Release Yukon 1.0.0 Beta

发布日期：2022/05/20

A.3.1 新增功能列表

- 适配 PostGIS 3.2 的三个模块：postgis、postgis_raster、postgis_sfcgal
- 新增 yukon_geogridcoder 模块

A.3.2 已知问题

由于 openGauss 和 PostgreSQL 的差异，PostGIS的矢量和栅格功能还有以下测试未通过:

1. postgis模块

- st_clusterintersecting 不支持自定义的聚合函数
- st_makeline 不支持自定义的聚合函数
- st_asflatgeobuf 不支持自定义的聚合函数
- st_collect 不支持自定义的聚合函数
- st_polygonize 不支持自定义的聚合函数
- st_clusterwithin 不支持自定义的聚合函数
- st_asgeojson(record feature, text geomcolumnname, integer maxdecimaldigits=9, boolean pretty_bool=false) 不支持（缺少 IsValidJsonNumber）
- st_concavehull 不支持（调用不支持的聚合函数）
- st_asgeobuf 不支持自定义的聚合函数
- st_fromflatgeobuftotable 不支持（通常需要结合 ST_AsFlatGeobuf 一起使用，故不支持）
- st_fromflatgeobuf 不支持（通常需要结合 ST_AsFlatGeobuf 一起使用，故不支持）
- st_asmvtgeom 不支持自定义的聚合函数
- st_asmvt 不支持自定义的聚合函数
- st_clusterdbscan 不支持自定义窗口函数
- st_clusterkmeans 不支持自定义窗口函数
- addauth 不支持长事务
- checkauth 不支持长事务
- disablelongtransactions 不支持长事务
- enablelongtransactions 不支持长事务
- lockrow 不支持长事务
- unlockrows 不支持长事务
- 不支持中断操作
- 不支持 BRIN 索引

2. postgis_raster 模块

- ST_Nearestvalue 不支持
- ST_Dumpvalue 不支持
- ST_Intersection 不支持
- ST_Clip 不支持

12.1.5 A.4 Release Yukon 1.0.0 Alpha

发布日期：2021/11/30

A.4.1 功能列表

- 适配 PostGIS 2.4 的三个模块：postgis、postgis_raster、postgis_sfcgal
- 新增 yukon_geomodel 模块，开放 geomodel 对象的存储结构

A.4.2 已知问题

由于 openGauss 和 PostgreSQL 的差异，PostGIS的矢量和栅格功能还有以下测试未通过:

1. postgis模块

- loader/Latin1：varchar 默认为字节数而不是字符数
- loader/Latin1-implicit： varchar 默认为字节数而不是字符数
- cluster： 目前不支持 `window` 函数
- long_xact： 和 WEB 相关，暂未测试
- typmod： copy 不支持 exception
- 同时目前不支持中断操作。

2. postgis_raster模块

- rt_tile： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_summarystats： 缺少 `st_summarystatsagg` 聚合函数，待修复
- rt_histogram： 缺少聚合函数 `st_summarystatsagg`，待修复
- rt_quantile： 缺少聚合函数 `st_summarystatsagg`，待修复
- rt_createoverview： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_union： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_elevation_functions： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_iscoveragetile： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_mapalgebra： 聚合函数不支持 `internal` 参数，导致 `st_union` 无法使用，待修复
- rt_mapalgebrafct： 函数已废弃，推荐使用 `ST_MapAlgebra`

12.2 编译

环境：

操作系统： Centos7:1806

编译器： GCC 7.3

CMake: 3.19

约定：

1. 我们所有编译的软件都会安装到 `/home/3rd_inst`
2. 系统已安装编译需要的必备工具
    - autoconf
    - automake

- libtool

12.2.1 依赖库编译

名称	Centos(7.6)版本	openEuler(20.03)版本	操作	备注
libtiff-devel	4.0.3	4.1.0	yum install libtiff-devel	PROJ 依赖
libcurl-devel	7.29.0	7.71.1	yum install libcurl-devel	PROJ 依赖
gmp-devel	6.0.0	6.2.0	yum install gmp-devel	CGAL 依赖
mpfr-devel	3.1.1	4.1.0	yum install mpfr-devel	CGAL 依赖
boost-devel	1.53.0	1.73.0	yum install boost-devel	CGAL/SFCGAL 依赖
libuuid-devel	2.23.2	2.35.2	yum install libuuid-devel	
JSON-C	0.11	0.15	yum install json-c-devel	
LIBXML2	2.9.1	2.9.10	yum install libxml2-devel	
SQLite3	3.39	3.39	编译	PROJ 依赖
PROJ	8.1.1	8.1.1	编译	
GEOS	3.10.4	3.10.4	编译	
GDAL	3.3.2	3.3.2	编译	
CGAL	4.13.2	4.13.2	编译	
SFCGAL	1.3.8	1.3.8	编译	
PROTOBUF-CPP	3.16.0	–	编译	PROTOBUF-C 依赖
PROTOBUF-C	1.4.0	–	编译	

SQLite3

1. 下载源码

```
wget https://www.sqlite.org/2022/sqlite-autoconf-3390400.tar.gz
```

2. 解压软件包

```
tar -xf sqlite-autoconf-3390400.tar.gz
```

3. 修改源代码

   在后续编译 GDAL 时，会使用到 SQLite3 的一些特性，所以这里我们需要修改一下源代码，进入 sqlite3 目录，在 `sqlite3.c` 文件的第 25 行添加 `#define SQLITE_ENABLE_COLUMN_METADATA 1`,或者直接使用下面的命令修改

```
sed -i '25 i #define SQLITE_ENABLE_COLUMN_METADATA 1' sqlite3.c
```

4. 检查环境，生成编译脚本

```
./configure --prefix=/home/3rd_inst/sqlite3
```

5. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/sqlite3 看到编译好的 SQLite3.

PROJ

1. 下载源码

```
wget http://download.osgeo.org/proj/proj-8.1.1.tar.gz
```

2. 解压软件包

```
tar -xf proj-8.1.1.tar.gz
```

3. 导出环境变量

   Porj 编译需要使用 SQLite3 3.11 以上版本，因此这里我们使用我们刚刚编译的 SQLite3，首先将其 pkg-config 文件路径导出：

```
export PKG_CONFIG_PATH=/home/3rd_inst/sqlite3/lib/pkgconfig:$PKG_CONFIG_PATH
```

   同时使用我们编译好的 sqlite3 程序：

```
export PATH=/home/3rd_inst/sqlite3/bin/:$PATH
```

4. 检查环境，生成编译脚本

```
./configure --prefix=/home/3rd_inst/proj
```

5. 编译，安装

```
make -j4 && make install
```

   编译完成后就可以在 /home/3rd_inst/proj 看到编译好的 Proj.

GEOS

1. 下载源码

```
wget https://download.osgeo.org/geos/geos-3.10.4.tar.bz2
```

2. 解压软件包

```
tar -xf geos-3.10.4.tar.bz2
```

3. 检查环境，生成编译脚本

```
cmake3 -DCMAKE_INSTALL_PREFIX=/home/3rd_inst/geos .
```

4. 编译，安装

```
make -j4 && make install
```

   编译完成后就可以在 /home/3rd_inst/geos 看到编译好的 GEOS.

GDAL

1. 下载源码

```
wget http://upload.osgeo.org/gdal/3.3.2/gdal-3.3.2.tar.gz
```

2. 解压软件包

```
tar -xf gdal-3.3.2.tar.gz
```

3. 导出环境变量

```
export LIBRARY_PATH=/home/3rd_inst/proj/lib/:$LIBRARY_PATH
export CPATH=/home/3rd_inst/proj/include:$CPATH
```

4. 检查环境，生成编译脚本

```
./configure --prefix=/home/3rd_inst/gdal
```

5. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/gdal 看到编译好的 GDAL.

CGAL

1. 下载源码

```
wget https://github.com/CGAL/cgal/archive/refs/tags/releases/CGAL-4.13.2.tar.gz
```

2. 解压软件包

```
tar -xf CGAL-4.13.2.tar.gz
```

3. 检查环境，生成编译脚本

```
cmake3 -DCMAKE_INSTALL_PREFIX=/home/3rd_inst/cgal -DCMAKE_BUILD_TYPE=Release
```

4. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/cgal 看到编译好的 CGAL.

SFCGAL

1. 下载源码

```
wget https://github.com/Oslandia/SFCGAL/archive/refs/tags/v1.3.8.tar.gz
```

2. 解压软件包

```
tar -xf v1.3.8.tar.gz
```

3. 检查环境，生成编译脚本

```
cmake3 -DCMAKE_INSTALL_PREFIX=/home/3rd_inst/sfcgal -DCMAKE_PREFIX_PATH=/home/3rd_
↪inst/cgal/lib64/cmake/CGAL
```

4. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/sfcgal 看到编译好的 SFCGAL.

JSON-C

1. 下载源码

```
wget https://github.com/json-c/json-c/archive/refs/tags/json-c-0.15-20200726.tar.
↪gz
```

2. 解压软件包

```
tar -xf json-c-0.15-20200726.tar.gz
```

3. 检查环境，生成编译脚本

```
cmake3 -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/home/3rd_inst/json-c
```

4. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/json-c 看到编译好的 JSON-C.

LIBXML2

1. 下载源码

```
wget https://github.com/GNOME/libxml2/archive/refs/tags/v2.9.12.tar.gz
```

2. 解压软件包

```
tar -xf v2.9.12.tar.gz
```

3. 检查环境，生成编译脚本

```
sh autogen.sh --prefix=/home/3rd_inst/libxml2
```

4. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/libxml2 看到编译好的 LIBXML2.

PROTOBUF-CPP

1. 下载源码

```
wget https://github.com/protocolbuffers/protobuf/releases/download/v3.16.0/
↪protobuf-cpp-3.16.0.tar.gz
```

2. 解压软件包

```
tar -xf protobuf-cpp-3.16.0.tar.gz
```

3. 检查环境，生成编译脚本

```
./configure --prefix=/home/3rd_inst/protobuf
```

4. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/protobuf 看到编译好的 PROTOBUF-CPP.

PROTOBUF-C

1. 下载源码

```
wget https://github.com/protobuf-c/protobuf-c/releases/download/v1.4.0/protobuf-c-
↪1.4.0.tar.gz
```

2. 解压软件包

```
tar -xf protobuf-c-1.4.0.tar.gz
```

3. 导出环境变量

这里我们指明刚刚安装的 Protobuf-cpp 的 pkg-config 文件路径

```
export PKG_CONFIG_PATH=/home/3rd_inst/protobuf/lib/pkgconfig:$PKG_CONFIG_PATH
```

4. 检查环境，生成编译脚本

```
./configure --prefix=/home/3rd_inst/protobuf-c
```

5. 编译，安装

```
make -j4 && make install
```

编译完成后就可以在 /home/3rd_inst/protobuf-c 看到编译好的 PROTOBUF-C.

到此为止，所有的三方依赖库已经编译完成，这里我们新建一个 enable 文件,用于导出我们在后续会用到的环境变量

```
#!/bin/bash

PREFIX=/home/3rd_inst

export LIBRARY_PATH=${PREFIX}/sqlite3/lib:${PREFIX}/proj/lib:${PREFIX}/gdal/lib:$
↪{PREFIX}/protobuf/lib:${PREFIX}/protobuf-c/lib:${PREFIX}/libxml2/lib:${PREFIX}/geos/
↪lib64:${PREFIX}/cgal/lib64:${PREFIX}/sfcgal/lib64:${PREFIX}/json-c/lib64:$LIBRARY_
↪PATH
```

(continues on next page)

(continued from previous page)

```
export LD_LIBRARY_PATH=${PREFIX}/sqlite3/lib:${PREFIX}/proj/lib:${PREFIX}/gdal/lib:$
↪{PREFIX}/protobuf/lib:${PREFIX}/protobuf-c/lib:${PREFIX}/libxml2/lib:${PREFIX}/geos/
↪lib64:${PREFIX}/cgal/lib64:${PREFIX}/sfcgal/lib64:${PREFIX}/json-c/lib64:$LD_
↪LIBRARY_PATH

export PKG_CONFIG_PATH=${PREFIX}/sqlite3/lib/pkgconfig:${PREFIX}/proj/lib/pkgconfig:$
↪{PREFIX}/geos/lib64/pkgconfig:${PREFIX}/gdal/lib/pkgconfig:${PREFIX}/sfcgal/lib64/
↪pkgconfig:${PREFIX}/protobuf/lib/pkgconfig:${PREFIX}/protobuf-c/lib/pkgconfig:$
↪{PREFIX}/json-c/lib64/pkgconfig:${PREFIX}/libxml2/lib/pkgconfig:$PKG_CONFIG_PATH

export PATH=${PREFIX}/sqlite3/bin:${PREFIX}/proj/bin:${PREFIX}/geos/bin:${PREFIX}/
↪gdal/bin:${PREFIX}/sfcgal/bin:${PREFIX}/protobuf/bin:${PREFIX}/protobuf-c/bin:$
↪{PREFIX}/libxml2/bin:$PATH
```

Note: 还有一个 ugc 的文件夹是 yukon_geomodel 的依赖文件，你可以从安装包中找到这个文件。然后将路径加入到“LD_LIBRARY_PATH“ 中。

12.2.2 Yukon for openGauss 编译

注意：在编译之前需要将 openGauss 缺少的头文件从其源码包中拷贝到数据库对应的 Include 目录下。

1. 下载源码

```
git clone https://gitee.com/opengauss/yukon.git
```

2. 检查环境，生成编译脚本

```
./configure --without-topology  --without-address-standardizer --without-
↪interrupt-tests CFLAGS='-O2 -fpermissive -DPGXC  -pthread ' CC=g++
```

3. 编译安装

```
make -j4 && make install
```

到此，就可以在 openGauss 中使用 Yukon 了。

12.2.3 Yukon for PostgreSQL 编译

1. 下载源码

```
git clone https://gitee.com/isupermap/yukon4pgsql.git
```

2. 生成 configure

```
sh autogen.sh
```

3. 检查环境，生成编译脚本

```
./configure --without-topology
```

4. 编译，安装

```
make -j4 && make install
```

到此，就可以在 PostgreSQL 中使用 Yukon 了。