openGauss的AI特性实践

实验指导手册

目录

[1 实验环境介绍 3](#_Toc7544)

[1.1 实验介绍 3](#_Toc20023)

[1.1.1 关于本实验 3](#_Toc13046)

[1.1.2 读者知识背景 3](#_Toc50)

[1.1.3 实验设备介绍 3](#_Toc17607)

[2 关卡一：openGauss数据安装及基本操作 4](#_Toc22046)

[2.1 实验介绍 4](#_Toc6403)

[2.1.1 关于本实验 4](#_Toc28422)

[2.1.2 实验目的 4](#_Toc18071)

[2.2 实验任务及步骤 4](#_Toc20995)

[2.2.1 安装前准备 4](#_Toc23698)

[2.2.2 数据库安装 6](#_Toc19677)

[2.2.3 关卡验证 7](#_Toc18630)

[3 关卡二：openGauss数据导入及基本操作 8](#_Toc14493)

[3.1 实验介绍 8](#_Toc8336)

[3.1.1 关于本实验 8](#_Toc23488)

[3.1.2 实验目的 8](#_Toc17604)

[3.2 实验任务及步骤 8](#_Toc7916)

[3.2.1 数据初始化 8](#_Toc25102)

[3.2.2 列存表的创建与使用 12](#_Toc12366)

[3.2.3 关卡验证 16](#_Toc6471)

[4 关卡三：openGauss的AI4DB特性应用 17](#_Toc21275)

[4.1 实验介绍 17](#_Toc14227)

[4.1.1 关于本实验 17](#_Toc17340)

[4.1.2 实验目的 17](#_Toc25664)

[4.2 实验任务及步骤 17](#_Toc7851)

[4.2.1 Index-advisor：索引推荐 17](#_Toc6839)

[4.2.2 关卡验证 22](#_Toc26394)

# 实验环境介绍

## 实验介绍

### 关于本实验

openGauss是关系型数据库，采用客户端/服务器，单进程多线程架构，支持单机和一主多备部署方式，备机可读，支持双机高可用和读扩展。

本实验主要内容为openEuler弹性云服务上安装部署openGauss数据库，并进行简单的数据库相关操作；完成数据库相关操作后，进行列存表的使用与物化视图的使用；后续进行openGauss的AI4DB的特性实验，提升数据库性能工作。

### 读者知识背景

本课程为创新实践课，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

具有基本的数据库知识背景，同时熟悉华为云界面，了解基本Linux知识。

### 实验设备介绍

* 组网说明

本实验环境为华为云环境，需要购买openEuler弹性云服务。

* 设备介绍

为了满足openGauss安装部署实验需要，建议每套实验环境采用以下配置：

软件类型和配置描述如下：

实验设备配套关系

|  |  |
| --- | --- |
| **设备名称** | **设备型号** |
| Linux操作系统 | openEuler20.3LTS |
| 工具 | Putty WinSCP TPCH Madlib TensorFlow |
| Python | Python 3.7.X  Python 2.7.X |

# 关卡一：openGauss数据安装及基本操作

## 实验介绍

### 关于本实验

本实验完成openGauss的安装及基本操作。

### 实验目的

掌握openGauss快速安装。

## 实验任务及步骤

### 安装前准备

用root用户名，使用之前设置的密码登录ECS。



创建openGauss数据库的安装用户omm及其属组dbgrp。

groupadd -g 1000 dbgrp

useradd -g dbgrp -u 1000 -d /home/omm omm

修改omm用户密码。

passwd omm

输入修改的omm用户密码，建议设置成复杂密码。

创建openGauss安装路径。

mkdir -p /opt/software/openGauss/

使用yum安装依赖包。

yum install -y libaio-devel ncurses-devel pam-devel libffi-devel libtool libtool-devel libtool-ltdl openssl-devel bison golang flex dkms-2.6.1-5.oe1.noarch python3-devel patch

替换python版本，将python3的链接设置为python的。

cd /usr/bin

mv python python.bak

ln -s python3 /usr/bin/python

检查python的版本。

python -V

返回值为：Python 3.7.4

使用vi打开logind.conf文件。

vi /etc/systemd/logind.conf

**按下i字母键修改RemoveIPC的值为no并去掉前面的#。**

RemoveIPC=no

**先按esc键，然后输入:wq退出并保存。**

:wq

重新加载配置参数。

systemctl daemon-reload

systemctl restart systemd-logind

检查RemoveIPC是否关闭。

systemctl show systemd-logind | grep RemoveIPC

返回值为：RemoveIPC=no

修改路径的用户属组及权限。

chown omm:dbgrp -R /opt/software

chmod 755 -R /opt/software

### 数据库安装

切换到omm用户。

su - omm

进入openGauss文件夹。

cd /opt/software/openGauss/

下载openGauss安装包。

wget https://opengauss.obs.cn-south-1.myhuaweicloud.com/2.0.1/arm/openGauss-2.0.1-openEuler-64bit.tar.bz2

解压openGauss压缩包。

tar -jxf openGauss-2.0.1-openEuler-64bit.tar.bz2 -C /opt/software/openGauss

进入安装脚本目录。

cd /opt/software/openGauss/simpleInstall

执行install.sh脚本安装openGauss。

sh install.sh -w Gauss\_234

在脚本执行过程中输入**yes**，按回车继续。

数据库登录。

exit

su - omm

gsql -d postgres -p 5432 -r

查询数据库版本。

select version();

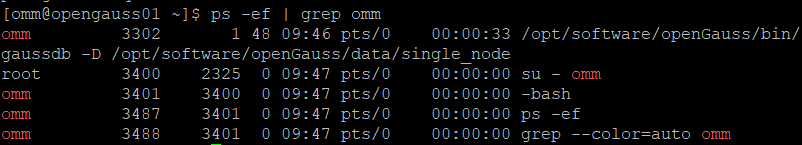
退出数据库。

\q

### 关卡验证

对数据库进程进行截图验证，需包含数据库服务器的主机名，**截图保存为1.png**

ps -ef | grep omm



回到root用户下。

exit

# 关卡二：openGauss数据导入及基本操作

## 实验介绍

### 关于本实验

本实验关卡，在安装完成的openGauss数据库上，进行TPCH和target数据初始化操作；完成openGauss中的列存表的创建、修改与查询。

### 实验目的

掌握openGauss数据库的基本维护方法；

掌握openGauss数据库列存表的创建与使用；

## 实验任务及步骤

### 数据初始化

使用root登录，进入/opt/software目录下。

cd /opt/software

下载TPC-H测试包。

git clone <https://github.com/gregrahn/tpch-kit.git>

将tpch-hit目录的属组修改为omm用户。

chown omm:dbgrp -R /opt/software/tpch-kit/

切换至用户omm，并进入测试包的目录。

su - omm

cd /opt/software/tpch-kit/dbgen/

生成MakeFile文件。

make -f Makefile

连接openGauss数据库。

gsql -d postgres -p 5432

创建测试数据库tpch并退出。

CREATE DATABASE tpch;

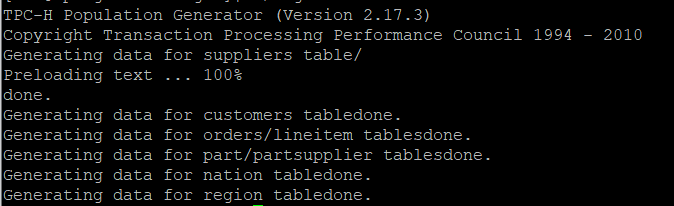
\q

执行创建对象脚本。

gsql tpch -f dss.ddl

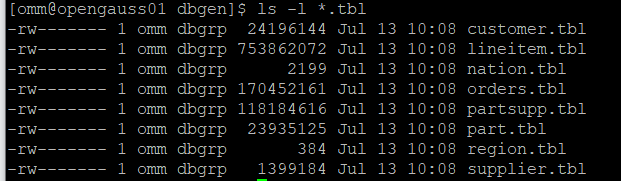
产生测试数据。

./dbgen -vf -s 1

当返回如下内容，表示数据生产完成。

使用ls命令查看回显生成的数据。

ls -l \*.tbl



编辑加载脚本LoadData.sh。

vi LoadData.sh

将以下内容添加进LoadData.sh脚本中。

for i in `ls \*.tbl`; do

table=${i/.tbl/}

echo "Loading $table..."

sed 's/|$//' $i > /tmp/$i

gsql tpch -q -c "TRUNCATE $table"

gsql tpch -c "\\copy $table FROM '/tmp/$i' CSV DELIMITER '|'"

done

使用:wq退出并保存。

:wq

执行加载脚本LoadData.sh，将数据加载进数据库中。

sh LoadData.sh

返回结果：

Loading customer...

Loading lineitem...

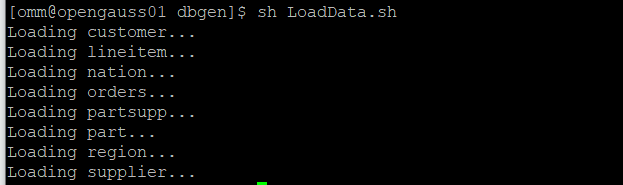
Loading nation...

Loading orders...

Loading partsupp...

Loading part...

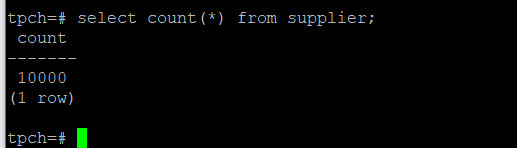
Loading region...

Loading supplier...

**登录数据库验证，将查询结果截图，截图保存为2.png**

gsql -d tpch -p 5432

select count(\*) from supplier;

此处应有返回记录，数据量为10000。

退出数据库。

\q

将相关查询拷贝进查询目录中，并进入queries/目录中。

cp dists.dss queries/

cp qgen queries/

cd queries/

生成查询语句。

vi genda.sh

将以下内容复制到genda.sh中。

for i in {1..22}; do

./qgen -d $i>$i\_new.sql

./qgen -d $i\_new | sed 's/limit -1//' | sed 's/day (3)/day/' > queries.sql

done

使用:wq退出并保存。

:wq

执行genda.sh脚本，生成查询语句。

sh genda.sh

验证生成的查询语句。

ls -l queries.sql

上传target.sql脚本至/opt/software目录下。

cd /opt/software

wget https://practicecourse.obs.myhuaweicloud.com/target.sql

切换至root用户，将脚本的属组修改为omm。

exit

chown omm:dbgrp /opt/software/target.sql

切换至omm用户将target.sql中的内容导入到tpch数据库中。

su - omm

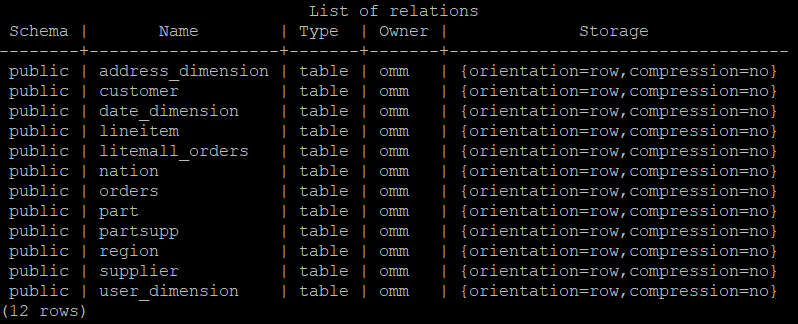
gsql -d tpch -p 5432 -r -f /opt/software/target.sql > /opt/software/target.log

**导入过程可能会持续几分钟，请耐心等待。**

登录数据库进行验证。

gsql -d tpch -p 5432

\dt

此处加上TPCH测试的表，总共有12张表。退出数据库。

\q

### 列存表的创建与使用

登录tpch 数据库。

gsql -d tpch -p 5432

创建与litemall\_orders表表结构相同的列存表。

CREATE TABLE "public"."litemall\_orders\_col" (

"order\_id" INTEGER NOT NULL DEFAULT NULL,

"order\_sn" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"order\_status" SMALLINT DEFAULT NULL,

"user\_key" INTEGER DEFAULT NULL,

"address\_key" INTEGER DEFAULT NULL,

"goods\_price" numeric(10,2) DEFAULT NULL,

"freight\_price" numeric(10,2) DEFAULT NULL,

"coupon\_price" numeric(10,2) DEFAULT NULL,

"integral\_price" numeric(10,2) DEFAULT NULL,

"groupon\_price" numeric(10,2) DEFAULT NULL,

"order\_price" numeric(10,2) DEFAULT NULL,

"actual\_price" numeric(10,2) DEFAULT NULL,

"pay\_id" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"pay\_date" INTEGER DEFAULT NULL,

"pay\_time" INTEGER DEFAULT NULL,

"ship\_sn" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"ship\_channel" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"ship\_date" INTEGER DEFAULT NULL,

"ship\_time" INTEGER DEFAULT NULL,

"confirm\_date" INTEGER DEFAULT NULL,

"confirm\_time" INTEGER DEFAULT NULL,

"message" varchar(512) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"comments" SMALLINT DEFAULT NULL,

"aftersale\_status" SMALLINT DEFAULT NULL,

"refund\_amount" numeric(10,2) DEFAULT NULL,

"refund\_type" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"refund\_content" varchar(127) COLLATE "pg\_catalog"."default" DEFAULT NULL,

"refund\_date" INTEGER DEFAULT NULL,

"refund\_time" INTEGER DEFAULT NULL,

"end\_date" INTEGER DEFAULT NULL,

"end\_time" INTEGER DEFAULT NULL,

"add\_date" INTEGER DEFAULT NULL,

"add\_time" INTEGER DEFAULT NULL,

"update\_date" INTEGER DEFAULT NULL,

"update\_time" INTEGER DEFAULT NULL,

"deleted" TINYINT DEFAULT NULL

)

with (orientation=column );

将litemall\_orders表中的数据插入litemall\_orders\_col中。

insert into litemall\_orders\_col select \* from litemall\_orders;

使用\timing开启SQL执行时间的返回。

\timing

使用SQL语句查询，2020年上半年litemall\_orders行存表中的order\_price的总和。

select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';

获得结果为：

sum

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

310586483.00

(1 row)

Time: 265.077 ms

使用SQL语句查询，2020年上半年litemall\_orders\_col列存表中的order\_price的总和。

select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

获得结果为：

sum

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

310586483.00

(1 row)

Time: 27.130 ms

使用SQL语句查询，2020年上半年litemall\_orders行存表中的order\_price的平均值。

select avg(order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';

获得结果为：

avg

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

3105.8648300000000000

(1 row)

Time: 369.490 ms

使用SQL语句查询，2020年上半年litemall\_orders\_col列存表中的order\_price的平均值。

select avg(order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

获得结果为：

avg

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

3105.8648300000000000

(1 row)

Time: 16.704 ms

使用SQL语句查询，litemall\_orders行存表中order\_id为6的order\_price的值。

select order\_price from litemall\_orders where order\_id=6;

获得结果为：

order\_price

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

2469.00

(1 row)

Time: 4.434 ms

使用SQL语句查询，litemall\_orders\_col列存表中order\_id为6的order\_price的值。

select order\_price from litemall\_orders\_col where order\_id=6;

获得结果为：

order\_price

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

2469.00

(1 row)

Time: 5.663 ms

使用SQL语句，将litemall\_orders行存表中order\_id为6的order\_price修改为2468。

update litemall\_orders set order\_price=2468 where order\_id=6;

获得结果为：

UPDATE 1

Time: 3.902 ms

使用SQL语句，将litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468。

update litemall\_orders\_col set order\_price=2468 where order\_id=6;

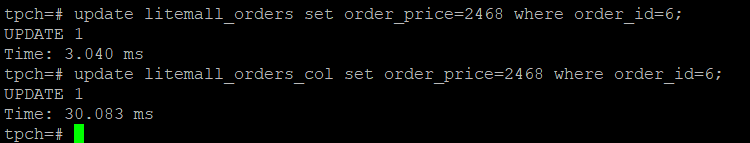
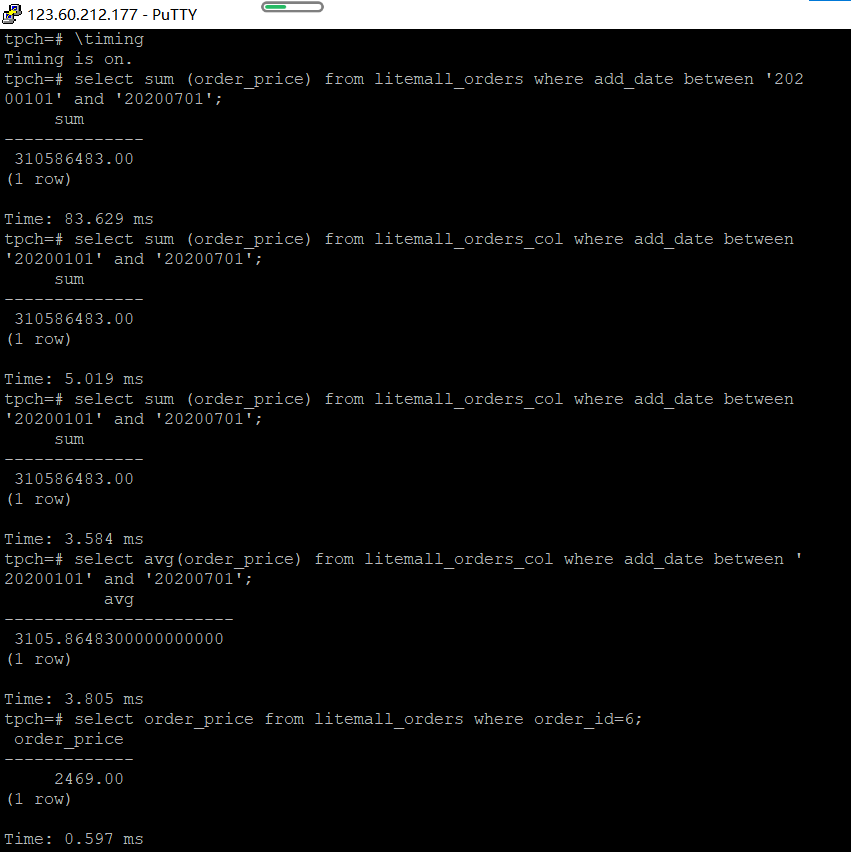
获得结果为：

UPDATE 1

Time: 36.075 ms

### 关卡验证

对3.2.2列存表的创建与使用中**步骤5~步骤12的结果**截图验证，**截图保存为2.png**



退出数据库。

\q

# 关卡三：openGauss的AI4DB特性应用

## 实验介绍

### 关于本实验

本实验关卡，在安装完成的openGauss数据库上，使用AI4DB的功能，对TPCH测试中的SQL语句进行优化。

### 实验目的

掌握openGauss数据库的AI4DB的功能；

掌握openGauss数据库的SQL优化能力；

掌握openGauss数据库的索引创建与使用方法。

## 实验任务及步骤

### Index-advisor：索引推荐

登录数据库。

gsql -d tpch -p 5432

使用SQL查询2020年3月订单收入，并进行排序。

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;

使用explain，对该SQL加以分析。

EXPLAIN

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

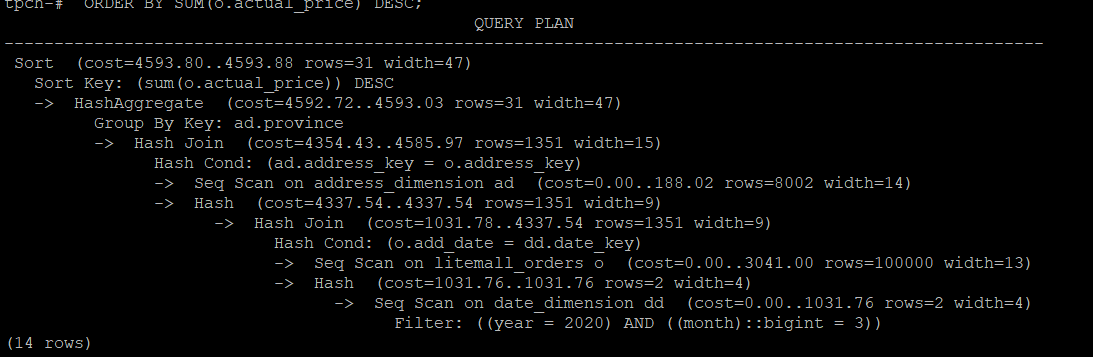
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;

获得执行计划结果为：

使用索引推荐功能，对查询语句进行推荐。

select \* from gs\_index\_advise('

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC');

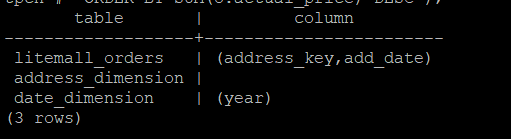
获得索引推荐结果如下：

table | column

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

litemall\_orders | (address\_key,add\_date)

address\_dimension |

date\_dimension | (year)

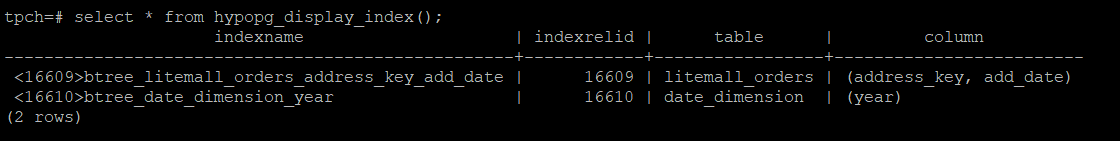
在litemall\_orders和date\_dimension表上创建虚拟索引列。

select \* from hypopg\_create\_index('create index on litemall\_orders(address\_key,add\_date)');

select \* from hypopg\_create\_index('create index on date\_dimension(year)');

查看创建的虚拟索引列。

select \* from hypopg\_display\_index();

获得创建虚拟列的结果：

indexname | indexrelid | table | column

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

<16715>btree\_litemall\_orders\_address\_key\_add\_date | 16715 | litemall\_orders | (address\_key, add\_date)

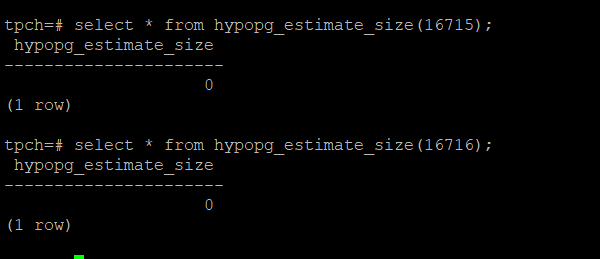
<16716>btree\_date\_dimension\_year | 16716 | date\_dimension | (year)

获取索引虚拟列大小结果（单位为：字节）。

其中**16715**和**16716**为上一步中查询到的indexrelid，此处查询需要进行替换成自己系统中查询到的值。

select \* from hypopg\_estimate\_size(**16715**);

select \* from hypopg\_estimate\_size(**16716**);

返回结果为：

hypopg\_estimate\_size

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

4177920

(1 row)

hypopg\_estimate\_size

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

696320

(1 row)

开启GUC参数enable\_hypo\_index，

该参数控制数据库的优化器进行EXPLAIN时是否考虑创建的虚拟索引。通过对特定的查询语句执行explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

set enable\_hypo\_index = on;

再次使用explain，对该SQL加以分析。

EXPLAIN

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM litemall\_orders o,

address\_dimension ad,

date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

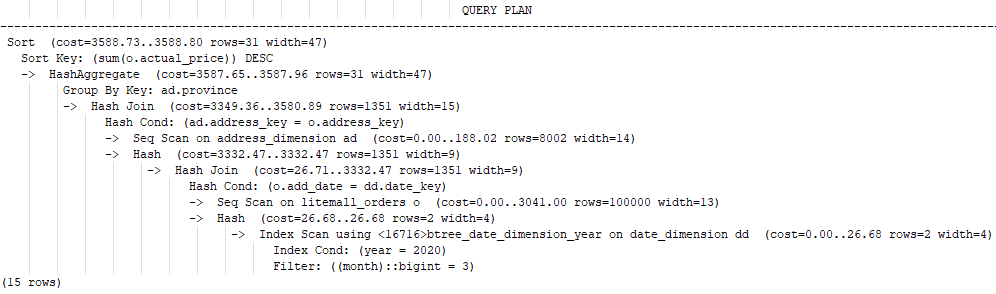
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;

获得执行计划：



删除某一个索引虚拟列。

其中**16715**和**16716**为上一步中查询到的indexrelid，此处查询需要进行替换成自己系统中查询到的值。

select \* from hypopg\_drop\_index(**16715**);

返回结果：

hypopg\_drop\_index

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

f

(1 row)

删除所有索引虚拟列，将执行结果截图。

select \* from hypopg\_reset\_index();

返回结果：

hypopg\_reset\_index

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

(1 row)

查看索引虚拟列，将执行结果截图。

select \* from hypopg\_display\_index();

返回结果：

indexname | indexrelid | table | column

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

(0 rows)

针对执行的queries.sql进行优化，为customer、order、part表增加主键索引，学员也可以针对queries.sql中的查询内容进行进一步优化。

ALTER TABLE "public"."customer" ADD CONSTRAINT "pk\_customer" PRIMARY KEY ("c\_custkey");

ALTER TABLE "public"."orders" ADD CONSTRAINT "pk\_orders" PRIMARY KEY ("o\_orderkey");

ALTER TABLE "public"."part" ADD CONSTRAINT "pk\_part" PRIMARY KEY ("p\_partkey");

在order表的o\_orderdate列，以及lineitem表的l\_shipdate列上创建索引，可以通过分析queries.sql查询，进行更深入的优化。

CREATE INDEX idx\_o\_orderdate ON orders(o\_orderdate);

CREATE INDEX idx\_l\_shipdate ON lineitem(l\_shipdate);

退出数据库。

\q

### 关卡验证

完成索引推荐功能的优化，使用索引推荐以及虚拟索引列的创建，对创建的索引虚拟列前后的执行时间进行对比截图；完成对queries查询的优化，对比优化前后的区别。

对4.2.1 Index-advisor：索引推荐中**步骤10~步骤15的结果**截图验证，**截图保存为3.png**

