openGauss 安全体系创新实践课

华为技术有限公司

关卡一、openGauss 数据安装及基本操作

openGauss 数据安装及基本操作，作业提交任务如下：

任务一：数据库状态验证

1. 查询数据库状态成功截图

```
[omm@opengauss01 openGauss-server]$ gs_ctl status
[2023-04-17 14:02:07.397][29720][][gs_ctl]: gs_ctl status,datadir is /opt/software/openGauss/
dest/data
gs_ctl: server is running (PID: 29592)
/opt/software/openGauss/dest/bin/gaussdb "-D" "/opt/software/openGauss/dest/data"
[omm@opengauss01 openGauss-server]$
```

任务二：数据库服务进程验证

1. 查看数据库服务进程截图（包含数据库服务器的主机名）

```
[omm@opengauss01 openGauss-server]$ ps -ef|grep omm
root        8481    5456  0 13:51 pts/0    00:00:00 su - omm
omm         8482    8481  0 13:51 pts/0    00:00:00 -bash
omm        29592       1  1 13:59 ?        00:00:03 /opt/software/openGauss/dest/bin/gaussdb
-D /opt/software/openGauss/dest/data
omm        29747    8482  0 14:03 pts/0    00:00:00 ps -ef
omm        29748    8482  0 14:03 pts/0    00:00:00 grep --color=auto omm
[omm@opengauss01 openGauss-server]$
```

任务三：实践思考题

思考题 1：为什么需要通过源码编译，安装数据库？

答：

1. 满足不同的运行平台，Linux 发型版本众多，但是每个版本采用的软件或者内核版本都不一样，而我们的二进制包所依赖的环境不一定能够正常运行，所以大部分软件直接提供源码。

2. 方便定制，满足不同的需求，很多时候我们所需要的软件都是可以定制的，需要什么就安装什么，大多数二进制代码都是一键装全，所以自由度并不高。

3. 方便运维、开发人员维护，我们的源码是可以打包二进制的，但是对于这个软件的打包都会有一份代价不小的额外工作，包括维护，所以如果是源码的话，软件产商会直接维护，但是如果是二进制的话，一般都是 Linux 发行商提供

关卡二、openGauss 数据导入及行存列存

任务一：数据初始化验证

1. 查询 supplier 表的行数，并将结果进行图：

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

```
tpch=# select count(*) from supplier;
 count
-------
 10000
(1 row)

tpch=# select count(*) from supplier;
```

任务二：行存表与列存表执行效率对比

1. 2020 年上半年 litemall_orders 行存表与 litemall_orders_col 列存表中的 order_price 的总和查询，并对比执行效率截图

```
select sum (order_price) from litemall_orders where add_date between '20200101' and '20200701';
select sum (order_price) from litemall_orders_col where add_date between '20200101' and '20200701';
```

```
tpch=# select sum (order_price) from litemall_orders where add_date between '20200101' and '2
0200701';
     sum
--------------
 310586483.00
(1 row)

Time: 244.198 ms
tpch=#
```

```
tpch=# select sum (order_price) from litemall_orders_col where add_date between '20200101' an
d '20200701';
     sum
--------------
 310586483.00
(1 row)

Time: 23.488 ms
tpch=#
```

2. 2020 年上半年 litemall_orders 行存表与 litemall_orders_col 列存表中的 order_price 的平均值查询，并对比执行效率截图

```
select avg (order_price) from litemall_orders where add_date between '20200101' and '20200701';
```

```
tpch=# select avg(order_price) from litemall_orders where add_date between '20200101' and '20
200701';
          avg
-----------------------
 3105.8648300000000000
(1 row)

Time: 326.075 ms
tpch=#
```

```
select avg (order_price) from litemall_orders_col where add_date between '20200101' and '20200701';
```

```
tpch=# select avg(order_price) from litemall_orders_col where add_date between '20200101' and
 '20200701';
         avg
-----------------------
 3105.8648300000000000
(1 row)

Time: 15.180 ms
tpch=#
```

3. 查询 litemall_orders 行存表与 litemall_orders_col 列存表中 order_id 为 6 的 order_price 的值，并对比执行效率截图。

```
select order_price from litemall_orders where order_id=6;
```

```
tpch=# select order_price from litemall_orders where order_id=6;
 order_price
-------------
     2469.00
(1 row)

Time: 1.488 ms
tpch=#
```

```
select order_price from litemall_orders_col where order_id=6;
```

```
tpch=# select order_price from litemall_orders_col where order_id=6;
 order_price
-------------
     2469.00
(1 row)

Time: 4.865 ms
tpch=#
```

4. 将 litemall_orders 行存表与 litemall_orders_col 列存表中 order_id 为 6 的 order_price 修改为 2468，并对比执行效率截图。

```
update litemall_orders set order_price=2468 where order_id=6;
```

```
tpch=# update litemall_orders set order_price=2468 where order_id=6;
UPDATE 1
Time: 4.363 ms
tpch=#
```

```
update litemall_orders_col set order_price=2468 where order_id=6;
```

```
tpch=# update litemall_orders_col set order_price=2468 where order_id=6;
UPDATE 1
Time: 34.214 ms
tpch=#
```

任务三：实践思考题

思考题 1：

行存表与列存表在执行相同的 SQL 语句时，为何执行的时间不同？

答：

1. 行存表的写入是一次完成。如果这种写入建立在操作系统的文件系统上，可以保证写入过程的成功或者失败，数据的完整性因此可以确定。

2. 列存表由于需要把一行记录拆分成单列保存，写入次数明显比行存表多（意味着磁头调度次数多，而磁头调度是需要时间的，一般在 1ms~10ms），再加上磁头需要在盘片上移动和定位花费的时间，实际时间消耗会更大。所以，行存表在写入上占有很大的优势。

3. 还有数据修改,这实际也是一次写入过程。不同的是，数据修改是对磁盘上的记录做删除标记。行存表是在指定位置写入一次，列存表是将磁盘定位到多个列上分别写入，这个过程仍是行存表的列数倍。所以，数据修改也是以行存表占优。

思考题 2：

在执行哪些类型 SQL 时，行存表效率更高？在执行哪些类型 SQL 时，列存表效率更高？

答：在执行数据写入语句，比如 UPDATE、INSERT 类型 SQL 时，行存表效率更高。在执行数据读取语句，比如 SELECT、LIMIT 类型 SQL 时，列存表效率更高。

关卡三：openGauss 物化视图应用

任务一：物化视图的使用

1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

```
SELECT * FROM test_view;
```

```
tpch=# SELECT * FROM test_view;
 username | gender | totalspend
----------+--------+------------
 杨兰娟   |      2 |  119391.00
 柳高梅   |      2 |  116155.00
 韦小全   |      1 |  114072.00
 贲艳梅   |      2 |  112565.00
 强兰丽   |      2 |  111925.00
 滑小刚   |      1 |  110089.00
 席长梅   |      2 |  108247.00
 翁晓婷   |      2 |  107988.00
 娄高伟   |      1 |  107323.00
 苏长刚   |      0 |  104640.00
 喻高伟   |      1 |  102536.00
 袁晓轩   |      1 |  101835.00
 伏成峰   |      1 |  101725.00
 毕晓刚   |      1 |  101057.00
 金高芳   |      2 |  100322.00
(15 rows)

tpch=#
```

2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

```
SELECT * FROM v_order;
```

```
tpch=# SELECT * FROM v_order;
 count
-------
    15
(1 row)

tpch=#
```

3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

```
SELECT * FROM v_order;
```

```
tpch=# SELECT * FROM v_order;
 count
-------
    14
(1 row)

tpch=#
```

4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

```
SELECT * FROM vi_order;
```

```
tpch=# SELECT * FROM vi_order;
 username | totalspend
----------+------------
 柳高梅   |  116155.00
 韦小全   |  114072.00
 贲艳梅   |  112565.00
 强兰丽   |  111925.00
 滑小刚   |  110089.00
 席长梅   |  108247.00
 翁晓婷   |  107988.00
 娄高伟   |  107323.00
 苏长刚   |  104640.00
 喻高伟   |  102536.00
 袁晓轩   |  101835.00
 伏成峰   |  101725.00
 毕晓刚   |  101057.00
 金高芳   |  100322.00
(14 rows)

tpch=#
```

5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

```
SELECT * FROM vi_order;
```

```
tpch=# SELECT * FROM vi_order;
 username | totalspend
----------+------------
 柳高梅   |  116155.00
 韦小全   |  114072.00
 贲艳梅   |  112565.00
 强兰丽   |  111925.00
 滑小刚   |  110089.00
 席长梅   |  108247.00
 翁晓婷   |  107988.00
 娄高伟   |  107323.00
 苏长刚   |  104640.00
 喻高伟   |  102536.00
 袁晓轩   |  101835.00
 伏成峰   |  101725.00
 毕晓刚   |  101057.00
 金高芳   |  100322.00
 杨兰娟   |  119391.00
(15 rows)

tpch=#
```

6. 对表进行操作后，刷新全量物化视图，查询物化视图结果，将执行结果截图。

```
SELECT * FROM vi_order;
```

```
tpch=# SELECT * FROM vi_order;
 username | totalspend
----------+------------
 柳高梅   |  116155.00
 韦小全   |  114072.00
 贲艳梅   |  112565.00
 强兰丽   |  111925.00
 滑小刚   |  110089.00
 席长梅   |  108247.00
 翁晓婷   |  107988.00
 娄高伟   |  107323.00
 苏长刚   |  104640.00
 喻高伟   |  102536.00
 袁晓轩   |  101835.00
 伏成峰   |  101725.00
 毕晓刚   |  101057.00
 金高芳   |  100322.00
 杨兰娟   |  119391.00
 马景涛   |  139391.00
(16 rows)

tpch=#
```

任务二：实践思考题

思考题 1：全量物化视图与增量物化视图有哪些差别?

答:

1. 全量物化视图：仅支持对已创建的物化视图进行全量更新，而不支持进行增量更新。创建全量物化视图语法和 CREATE TABLE AS 语法类似。

2. 增量物化视图：可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据刷新。与全量创建物化视图的不同在于目前增量物化视图所支持场景较小。

思考题 2：物化视图适用那些使用场景?

答：物化视图的应用场景有两种：1. 用于查询优化 2. 用于高级复制

关卡四：openGauss 密态数据库特性应用

任务一：物化视图的使用

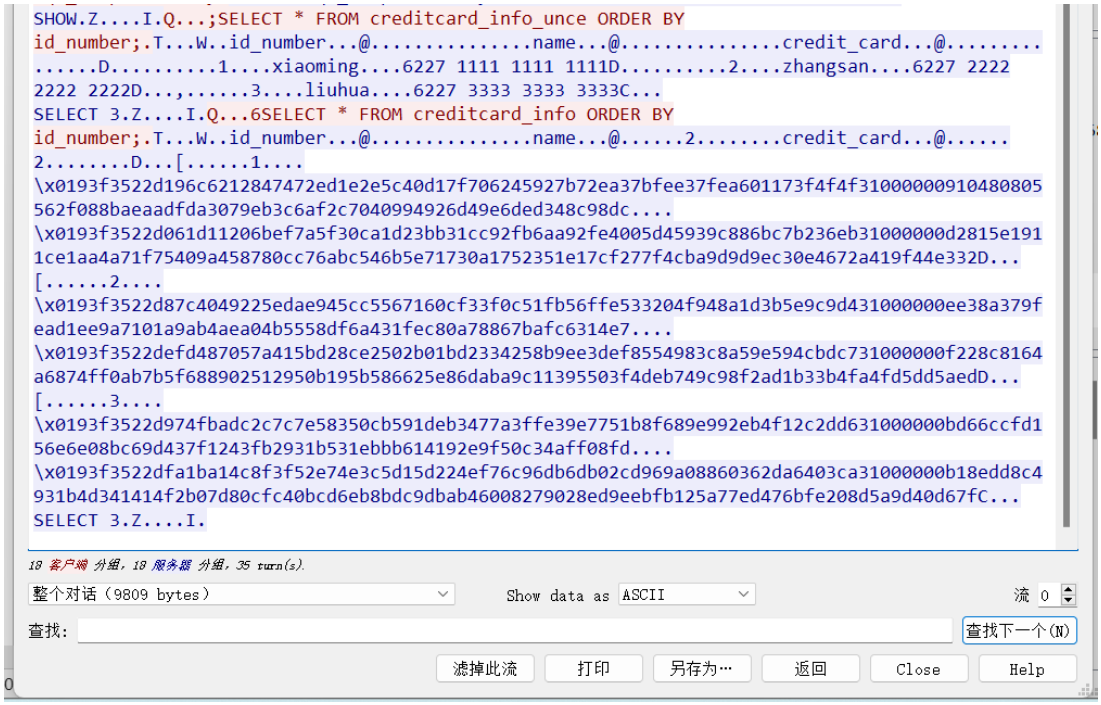
1. 通过 tcpdump 抓取数据流，此 putty 窗口暂时保持不动，将执行结果截图：

```
[root@opengauss01 ~]# tcpdump -i any tcp port 5432 and host 127.0.0.1 -w /opt/encryption.cap
dropped privs to tcpdump
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

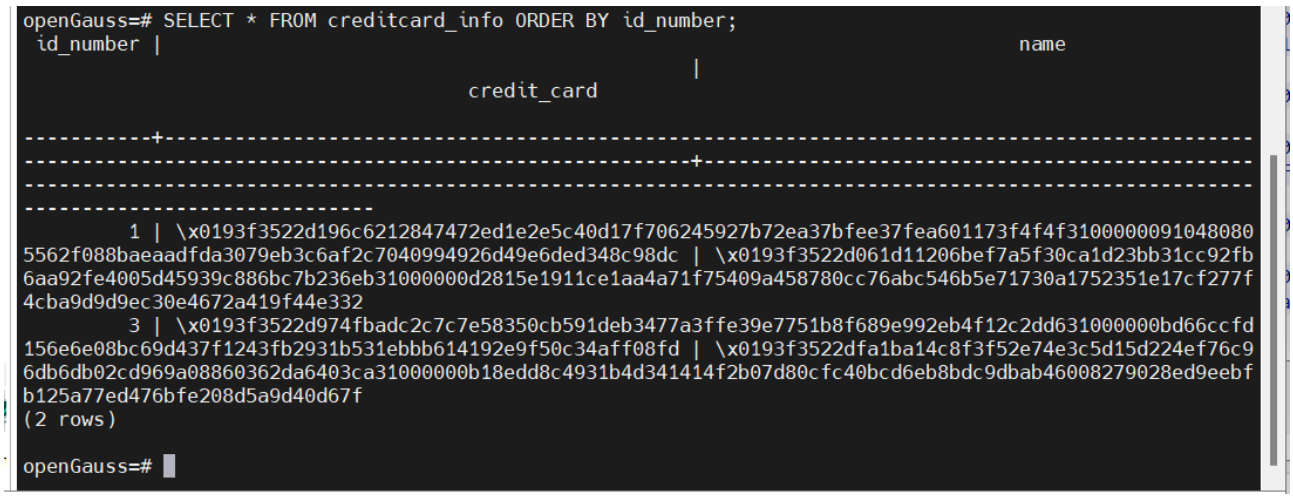
2. 将加密表和非加密表查询结果截图：

```
openGauss=# SELECT * FROM creditcard_info_unce ORDER BY id_number;
 id_number |   name   |     credit_card
-----------+----------+---------------------
         1 | xiaoming | 6227 1111 1111 1111
         2 | zhangsan | 6227 2222 2222 2222
         3 | liuhua   | 6227 3333 3333 3333
(3 rows)

openGauss=# SELECT * FROM creditcard_info ORDER BY id_number;
 id_number |   name   |     credit_card
-----------+----------+---------------------
         1 | xiaoming | 6227 1111 1111 1111
         2 | zhangsan | 6227 2222 2222 2222
         3 | liuhua   | 6227 3333 3333 3333
(3 rows)

openGauss=#
```

3. 用 wireshark 解析加密表和非加密表的差异时，非加密表 name 列和 credit_card 列是明文，加密表 name 列和 credit_card 列均是密文，将执行结果截图：



4.查询加密表，查询到的结果为密文，将执行结果截图：



任务二：实践思考题

思考题 1：

数据实际存储在物理磁盘上的时候是明文还是密文？数据的加解密的动作是在客户端完成的还是服务端完成的？

答：数据实际存储在物理磁盘上的时候是密文。数据的加解密的动作都是在客户端完成的。