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@ecs-6375 openGauss-server]$ gs_ctl status
[2022-05-15 11:02:29.069][223960][][gs_ctl]: gs_ctl status,datadir is /opt/software/openGauss/data
gs_ctl: server is running (PID: 223811)
/opt/software/openGauss/bin/gaussdb "-D" "/opt/software/openGauss/data"
```

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@ecs-6375 openGauss-server]$ ps -ef | grep omm
root        4368    3689  0 10:40 pts/1    00:00:00 su - omm
omm         4369    4368  0 10:40 pts/1    00:00:00 -bash
omm       223811       1  1 10:59 pts/1    00:00:01 /opt/software/openGauss/bin/gaussdb -D /opt/software/openGauss/data
omm       223932    4369  0 11:01 pts/1    00:00:00 ps -ef
omm       223933    4369  0 11:01 pts/1    00:00:00 grep --color=auto omm
```

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？

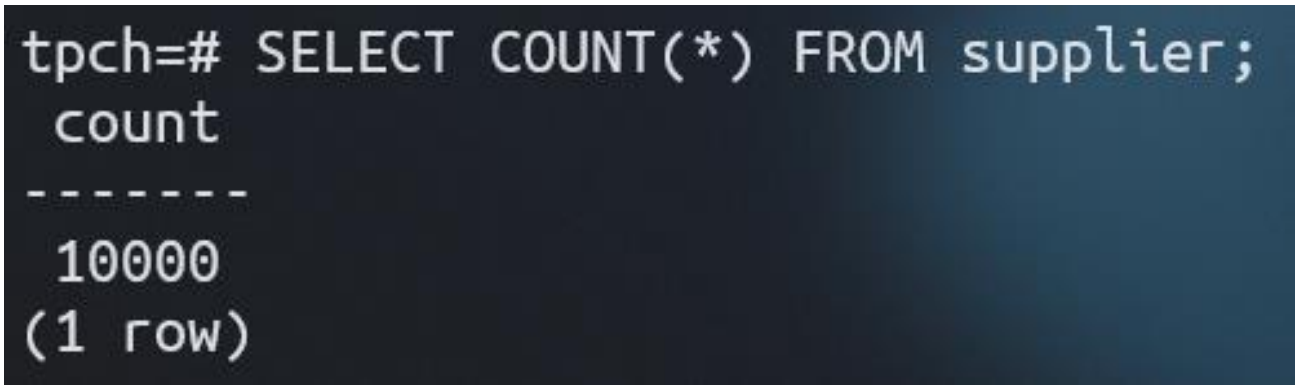
1. 通过 git 拉取到本地的代码可以随时通过 `git pull` 与远程仓库同步，保持版本最新，并能够第一时间获取到 bug fix；而如果使用 rpm 或 deb 等软件包，则一般需要等系统维护者进行充分的测试才能够发布，通常会落后若干个小版本。
2. 相比起其它方式，使用开源源码编译有更小的风险遇到恶意代码。

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，并将结果进行截图：

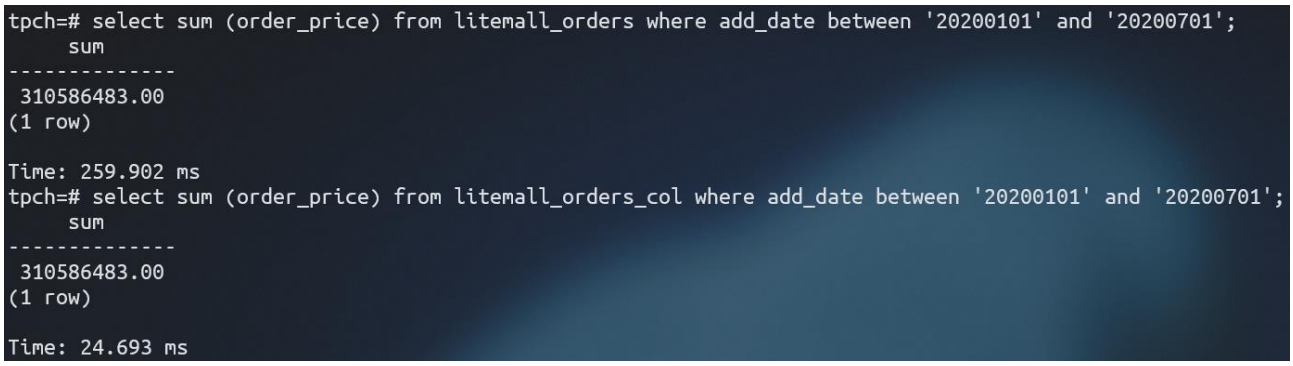
```
select count(*) from supplier;
```



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';
```



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

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

Time: 369.746 ms
tpch=# select avg(order_price) from litemall_orders_col where add_date between '20200101' and '20200701';
          avg
----------------------
 3105.8648300000000000
(1 row)

Time: 16.857 ms
```

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

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

Time: 2.361 ms
tpch=# select order_price from litemall_orders_col where order_id=6;
 order_price
-------------
     2469.00
(1 row)

Time: 6.698 ms
```

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

```
tpch=# update litemall_orders set order_price=2468 where order_id=6;
UPDATE 1
Time: 4.540 ms
tpch=# update litemall_orders_col set order_price=2468 where order_id=6;
UPDATE 1
Time: 74.163 ms
```

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?

行存表在磁盘上以行为单位存储数据，列存表则以列为单位，索引数据、压缩数据的方式都不同。比如列存，由于每列数据类型相同，所以容易压缩数据。

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？

行存表效率高：

1. SQL 操作涉及一张表中的大多数列时，如 `SELECT * FROM table`。
2. 插入新记录，或更新记录时。
3. 条件查询指定唯一条件时。

列存表效率高：

1. 涉及对于某一列的运算时，如 `AVG()` 或 `SUM()` 函数。
2. 数据极度庞大复杂时。

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)

Time: 3.127 ms
```

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：

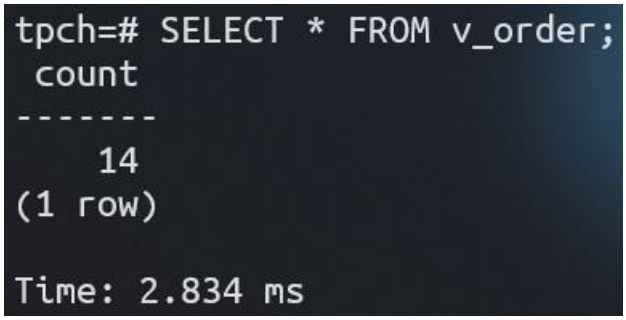
```
SELECT * FROM v_order;
```

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

Time: 3.079 ms
```

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;
```



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;
```

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;
```

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?

1. 前者不支持增量更新，而后者可以通过 `REFRESH INCREMENTAL MATERIALIZED VIEW`进行增量更新。
2. 前者的适用范围较广，而后者所支持场景较少，目前仅支持基表扫描语句或者 UNION ALL 语句。

思考题 2：物化视图适用哪些使用场景?

物化视图与普通视图最大的区别在于，前者能够存储数据而后者只是虚拟的映射。所以，物化视图适合用于在本地维护一份只读的远程数据库副本。比如建立一个快照，或者是数据的定期统计分析等。

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：

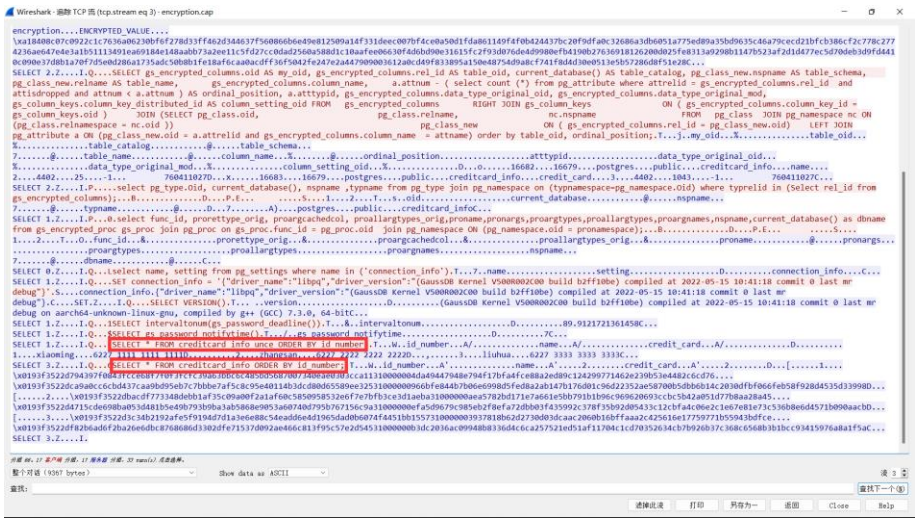
```
[root@ecs-6375 ~]# tcpdump -i any tcp port 5432 and host 127.0.0.1 -w /opt/encryption.cap
dropped privs to tcpdump
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

2. 将加密表和非加密表查询结果截图：

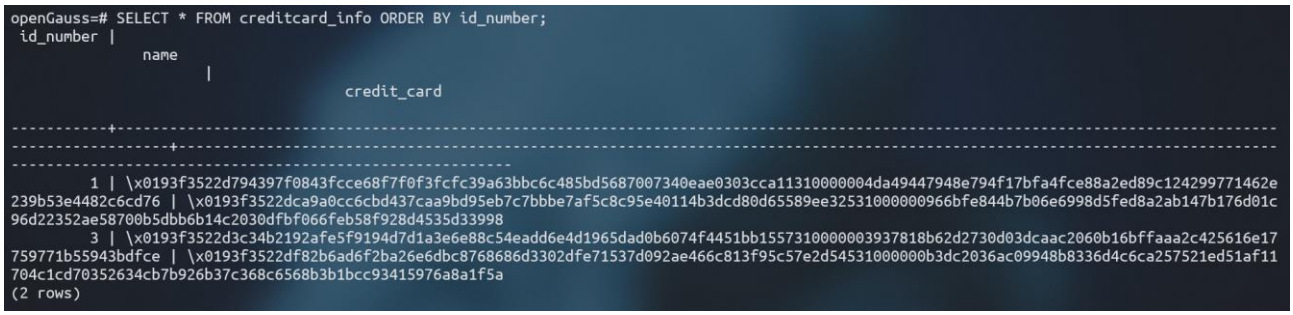
```
openGauss=# SELECT * FROM creditcard_info_unce ORDER BY id_number;
 id_number |   name   |     credit_card
-----------+----------+---------------------
         1 | xiaoming | 6227 1111 1111 1111
         2 | zhangsan | 6227 2222 2222 2222
         3 | liuhua   | 6227 3333 3333 3333
(3 rows)

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)
```

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？

存储在物理磁盘上的数据是密文，因为服务器理论上不知道如何解密这些数据。

加解密的动作均在客户端完成。因为：如果加密动作在服务端完成，那么客户端发送的查询或更新语句就是明文；如果解密动作在服务端完成，那么服务端返回的查询结果也是明文。只有客户端统一完成了加解密动作，才能保证双方交流的信息在传输过程中始终保持加密状态。