openGauss AI 特性创新实践课

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-7b85 ~]$ gs_ctl status
[2021-12-24 14:11:10.860][2631][][gs_ctl]: gs_ctl status,datadir is /opt/software/openGauss/data
gs_ctl: server is running (PID: 2564)
/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-7b85 ~]$ ps -ef|grep omm
root        2496    2358  0 14:09 pts/0    00:00:00 su - omm
omm         2497    2496  0 14:09 pts/0    00:00:00 -bash
omm         2564       1  1 14:10 pts/0    00:00:01 /opt/software/openGauss/bin/gaussdb -D /opt/softwa
re/openGauss/data
omm         2652    2497  0 14:12 pts/0    00:00:00 ps -ef
omm         2653    2497  0 14:12 pts/0    00:00:00 grep --color=auto omm
```

实验思考题：为什么需要通过源码编译，安装数据库？

x86 和鲲鹏使用的指令是不一样的，在鲲鹏上使用的是精简指令集，而在 x86 上使用的是复杂指令集。主要有如下三点差异：

- 架构差异：x86 和鲲鹏处理器（arm） (aarch64) 属于不同的架构。
- 指令集差异：x86 复杂指令集，鲲鹏处理器 – 精简指令集。
- 向量寄存器差异：x86 和鲲鹏处理器使用向量寄存器不同，向量指令集也存在差异。

一个在 x86 上运行的程序根本不可能毫无阻碍地就可以在鲲鹏上使用，必须经过编译，同时也证明了迁移的必要性；所以我们需要进行迁移的准备，在经过迁移准备和迁移分析之后进行编译迁移，所以我们需要在鲲鹏处理器上下载源码，然后将源码进行编译，在进行安装数据库。

关卡二、openGauss 数据导入及基本操作

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;;
```

```
[omm@ecs-7b85 dbgen]$ gsql -d tpch -p 5432 -r
gsql ((GaussDB Kernel V500R002C00 build b2ff10be) compiled at 2021-12-23 20:39:58 commit 0 last mr  de
bug)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

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

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';
```

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

Time: 257.255 ms
```

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

```
tpch=# select sum (order_price) from litemall_orders_col where add_date between '20200101' and '202007
01';
     sum
-------------
 310586483.00
(1 row)

Time: 24.697 ms
```

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 '20200701';
         avg
----------------------
 3105.8648300000000000
(1 row)

Time: 365.819 ms
```

```
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 '2020070
1';
         avg
----------------------
 3105.8648300000000000
(1 row)

Time: 16.309 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;
```

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

Time: 2.358 ms
```

```
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: 6.450 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;
```

```
tpch=# update litemall_orders set order_price=2468 where order_id=6;
UPDATE 1
Time: 12.223 ms
```

```
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: 72.996 ms
```

任务三：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: 3.059 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: 3.023 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;
```

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

Time: 2.881 ms
```

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)
```

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=# 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)
```

实践思考题 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？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？

1、首先，对于行存表与列存表的情况，数据读取时，行存储通常会将一行数据完全读出，而如果出现只需要其中几列数据的情况，就会存在冗余列；而列存储每次读取的数据是集合的一段或者全部，没有冗余性的问题；两种存储的数据分布。由于列存储的每一列数据类型是同质的，不存在二义性问题。比如说某列数据类型为整型(int)，那么它的数据集合一定是整型数据。这种情况使数据解析变得十分容易。相比之下，行存储则要复杂得多，因为在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，这个操作很消耗 CPU，增加了解析的时间。所以，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。

2、行存储执行一下类型的 SQL 语句效率高：适合随机的增删改查操作；需要在行中选取所有属性的查询操作；需要频繁插入或更新的操作，其操作与索引和行的大小更为相关。

3、列存储执行一下类型的 SQL 语句效率高：我们不需要将一行数据的全部属性全部读取，只需要其中几列数据属性的情况；而且列存表适合数据仓库和分布式的应用。

实践思考题 2：全量物化视图与增量物化视图有哪些差别?

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。创建全量物化视图语法和 CREATE TABLE AS 语法一致，不支持对全量物化视图指定 NodeGroup 创建。

增量物化视图顾名思义就是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进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者 UNION ALL 语句。

关卡三、openGauss 的 AI4DB 特性应用

任务一：使用 X-Tuner 进行参数优化

1. 执行 TPCH 脚本，获得测试时间，将执行结果截图：

```
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log
```

```
[omm@ecs-7b85 ~]$ tail -10 /opt/software/tpch-kit/dbgen/queries/queries01.log
 13        |      888 | 6737713.99
 17        |      861 | 6460573.72
 18        |      964 | 7236687.40
 23        |      892 | 6701457.95
 29        |      948 | 7158866.63
 30        |      909 | 6808436.13
 31        |      922 | 6806670.18
(7 rows)

total time: 1199005  ms
```

2. 使用 root 用户，执行 X-Tuner 进行参数建议优化，将执行结果截图

```
gs_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm
```

```
********************************** Recommended Knob Settings ***********************************
+--------------------------+-----------+----------+----------+---------+
|           name           | recommend |   min    |   max    | restart |
+--------------------------+-----------+----------+----------+---------+
| default_statistics_target |    1000   |   100    |   1000   |  False  |
|    effective_cache_size   |  21602940 |  187192  | 21602940 |  False  |
|  effective_io_concurrency |    200    |   150    |   250    |  False  |
|      enable_mergejoin     |    off    |    0     |    1     |  False  |
|      enable_nestloop      |    off    |    0     |    1     |  False  |
|      max_connections      |    370    |    50    |   741    |   True  |
| max_prepared_transactions |    370    |    50    |   741    |   True  |
|     max_process_memory    |  28803920 | 22403048 | 28803920 |   True  |
|      random_page_cost     |    1.0    |    1.0   |    2.0   |  False  |
|       shared_buffers      |   187192  |  187196  |  215272  |   True  |
|        wal_buffers        |    5849   |   2048   |   5849   |   True  |
+--------------------------+-----------+----------+----------+---------+
```

3.重启完成后，获取参数值：

```
cd /opt/software/openGauss/data
cat postgresql.conf|grep -E 'shared_buffers|max_connections|effective_cache_size|effective_io_concurrency|wal_buffers|random_page_cost|default_statistics_target'
```

```
[omm@ecs-7b85 data]$ cat postgresql.conf|grep -E 'shared_buffers|max_connections|effective_cache_size|
effective_io_concurrency|wal_buffers|random_page_cost|default_statistics_target'
max_connections = 370                   # (change requires restart)
# Note:  Increasing max_connections costs ~400 bytes of shared memory per
shared_buffers = 186864                 # min 128kB
bulk_write_ring_size = 2GB              # for bulkload, max shared_buffers
#standby_shared_buffers_fraction = 0.3 #control shared buffers use in standby, 0.1-1.0
effective_io_concurrency = 200          # 1-1000; 0 disables prefetching
wal_buffers = 5839                      # min 32kB
random_page_cost = 1                    # same scale as above
effective_cache_size = 21602940
default_statistics_target = 1000        # range 1-10000
# max_locks_per_transaction * (max_connections + max_prepared_transactions)
[omm@ecs-7b85 data]$
```

任务二：使用 Index-advisor 对 select 查询语句进行优化，并通过对比执行计划，得到优化前后的不同。

1. 使用 explain，对查询 2020 年 3 月订单表收入并进行排序的 SQL 加以分析，将结果截图。

```
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;
```

```
tpch=#  EXPLAIN
tpch-# SELECT ad.province AS province, SUM(o.actual_price) AS GMV
tpch-#   FROM litemall_orders o,
tpch-#        address_dimension ad,
tpch-#        date_dimension dd
tpch-#  WHERE o.address_key = ad.address_key
tpch-#    AND o.add_date = dd.date_key
tpch-#    AND dd.year = 2020
tpch-#    AND dd.month = 3
tpch-#  GROUP BY ad.province
tpch-#  ORDER BY SUM(o.actual_price) DESC;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
--
 Sort  (cost=4593.80..4593.88 rows=31 width=47)
   Sort Key: (sum(o.actual_price)) DESC
   ->  HashAggregate  (cost=4592.72..4593.03 rows=31 width=47)
         Group By Key: ad.province
         ->  Hash Join  (cost=4354.43..4585.97 rows=1351 width=15)
               Hash Cond: (ad.address_key = o.address_key)
               ->  Seq Scan on address_dimension ad  (cost=0.00..188.02 rows=8002 width=14)
               ->  Hash  (cost=4337.54..4337.54 rows=1351 width=9)
                     ->  Hash Join  (cost=1031.78..4337.54 rows=1351 width=9)
                           Hash Cond: (o.add_date = dd.date_key)
                           ->  Seq Scan on litemall_orders o  (cost=0.00..3041.00 rows=100000 width=13
)
                           ->  Hash  (cost=1031.76..1031.76 rows=2 width=4)
                                 ->  Seq Scan on date_dimension dd  (cost=0.00..1031.76 rows=2 width=4
)
                                       Filter: ((year = 2020) AND ((month)::bigint = 3))
(14 rows)

tpch=#
```

2. 使用索引推荐功能，对查询语句进行推荐，将执行结果截图。

```
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');
```

```
tpch=#  select * from gs_index_advise('
tpch'# SELECT ad.province AS province, SUM(o.actual_price) AS GMV
tpch'#   FROM litemall_orders o,
tpch'#        address_dimension ad,
tpch'#        date_dimension dd
tpch'#  WHERE o.address_key = ad.address_key
tpch'#    AND o.add_date = dd.date_key
tpch'#    AND dd.year = 2020
tpch'#    AND dd.month = 3
tpch'#  GROUP BY ad.province
tpch'#  ORDER BY SUM(o.actual_price) DESC');
 schema |       table       |          column
--------+-------------------+------------------------
 public | litemall_orders   | (address_key,add_date)
 public | address_dimension |
 public | date_dimension    | (year)
(3 rows)
```

3. 查看创建的虚拟索引列，将执行结果截图。

```
select * from hypopg_display_index();
```

```
tpch=# select * from hypopg_display_index();
                  indexname                   | indexrelid |      table      |         column
----------------------------------------------+------------+-----------------+------------------------
 <24576>btree_litemall_orders_address_key_add_date |      24576 | litemall_orders | (address_key, add_date)
 <24577>btree_date_dimension_year             |      24577 | date_dimension  | (year)
(2 rows)
```

4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

```
select * from hypopg_estimate_size(16715);
select * from hypopg_estimate_size(16716);
```

```
tpch=# select * from hypopg_estimate_size(16715);
 hypopg_estimate_size
----------------------
                    0
(1 row)

tpch=#  select * from hypopg_estimate_size(16716);
 hypopg_estimate_size
----------------------
                    0
(1 row)
```

5.再次使用 explain，对该 SQL 加以分析，将执行结果截图。

```
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;
```

```
tpch=# tpch=# 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;
                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=3579.58..3579.65 rows=31 width=47)
   Sort Key: (sum(o.actual_price)) DESC
   ->  HashAggregate  (cost=3578.50..3578.81 rows=31 width=47)
         Group By Key: ad.province
         ->  Hash Join  (cost=3340.21..3571.74 rows=1351 width=15)
               Hash Cond: (ad.address_key = o.address_key)
               ->  Seq Scan on address_dimension ad  (cost=0.00..188.02 rows=8002 width=14)
               ->  Hash  (cost=3323.32..3323.32 rows=1351 width=9)
                     ->  Hash Join  (cost=17.56..3323.32 rows=1351 width=9)
                           Hash Cond: (o.add_date = dd.date_key)
                           ->  Seq Scan on litemall_orders o  (cost=0.00..3041.00 rows=100000 width=13)
                           ->  Hash  (cost=17.53..17.53 rows=2 width=4)
                                 ->  Index Scan using <24577>btree_date_dimension_year on date_dimension dd  (cost=0.00..17.53 rows=2 width=4)
                                       Index Cond: (year = 2020)
                                       Filter: ((month)::bigint = 3)
(15 rows)
```

6. 删除某一个索引虚拟列，将执行结果截图。

```
select * from hypopg_drop_index(16715);
```

```
tpch=# select * from hypopg_drop_index(16715);
 hypopg_drop_index
-------------------
 f
(1 row)
```

7. 删除所有索引虚拟列，将执行结果截图。

```
select * from hypopg_reset_index();
```

```
tpch=# select * from hypopg_reset_index();
 hypopg_reset_index
--------------------

(1 row)
```

8. 查看索引虚拟列，将执行结果截图。

```
select * from hypopg_display_index();
```

```
tpch=# select * from hypopg_display_index();
 indexname | indexrelid | table | column
-----------+------------+-------+--------
(0 rows)
```

任务三：通过创建索引，对 queries.sql 中的 SQL 语句进行优化，并对比优化前后 queries.sql 执行的时间。

1. 重新执行 queries.sql 查询，将执行结果截图：

```
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log
```

```
[omm@ecs-7b85 data]$  tail -10 /opt/software/tpch-kit/dbgen/queries/queries02.log
 13        |     888 | 6737713.99
 17        |     861 | 6460573.72
 18        |     964 | 7236687.40
 23        |     892 | 6701457.95
 29        |     948 | 7158866.63
 30        |     909 | 6808436.13
 31        |     922 | 6806670.18
(7 rows)

total time: 332174  ms
```

挑战一：进一步优化 queries.sql 中的查询语句，使得前后执行时间出现倍数级的提升。

1. 重新执行 queries.sql 查询，将执行结果截图：

```
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries03.log
```

```
[omm@ecs-7b85 data]$  tail -10 /opt/software/tpch-kit/dbgen/queries/queries02.log
 13         |      888 | 6737713.99
 17         |      861 | 6460573.72
 18         |      964 | 7236687.40
 23         |      892 | 6701457.95
 29         |      948 | 7158866.63
 30         |      909 | 6808436.13
 31         |      922 | 6806670.18
(7 rows)

total time: 331933  ms
```

实践思考题 1：根据 X-Tuner 给出的参数优化，在哪些参数上进行了优化，为何要对这些参数进行优化?

在 order 表的 o_orderdate 列，以及 lineitem 表的 l_shipdate 列上创建了索引，因为这两个参数是经常需要根据范围进行搜索和排序的参数。

实践思考题 2：索引的使用，对于执行 SQL 有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化?

1、 索引使用的好处：通过唯一性索引（unique）可确保数据的唯一性；加快数据的检索速度；加快表之间的连接；减少分组和排序时间；使用优化隐藏器提高系统性能。在平时的 SQL 使用中，我们经常在 WHERE 子句中的列上面创建索引，这样加快条件的判断速度。

2、 对于数据库的优化，我们一般有八大经典的方式进行数据库的优化：选取最适用的字段属性；使用连接（JOIN）来代替子查询(Sub-Queries)；使用联合(UNION)来代替手动创建的临时表；必要的时候使用事务，利用事务的四大特性，我们的语句块中的每条语句都操作成功要么失败，保持数据的一致性和完整性，事物以 BEGIN 关键字开始，COMMIT 关键字结束。在这之间的一条 SQL 操作失败，那么，ROLLBACK 命令就可以把数据库恢复到 BEGIN 开始之前的状态；锁定表，尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有成千上万的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。有些情况下我们可以通过锁定表的方法来获得更好的性能；使用外键，锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键；当然除此之外，我们优化 SQL 语句，编写合适的 SQL 语句应该是最常用的数据库优化方法。

关卡四、openGauss 的 DB4AI 特性应用

任务一：在 gs_model_warehouse 系统表中查看训练后的模型信息，将执行结果截图：

```
postgres=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
```

```
openGauss=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
       modelname        | modelowner |        createtime        | processedtuples | discardedtuples |
 pre_process_time | exec_time | iterations | outputtype |     modeltype      |
                                query                                                                | m
odeldata |                weight                 |                                                 hyperparam
etersnames                                       |              hyperparametersvalues
 |          hyperparametersoids          |  coefnames   |  coefvalues  | coefoids |         trainingsc
oresname          |          trainingscoresvalue          | modeldescribe
-----------------------+------------+-----------------------+----------------+-----------------+
-----------------+-----------+-----------+-----------+------------------+----------------------
-------------------------------------------------------------------------------------------------+--
---------+-----------------------------------------+-----------------------------------------------
-------------------------------------------------+-----------------------------------------------
-+-----------------------------------------+--------------+-------------+----------+-------------------
-----------------+-----------------------------------------+--------------
 house_binary_classifier |         10 | 2021-12-24 15:59:57.4793 |              15 |               0 |
               0 |   .002683 |        100 |         16 | svm_classification | CREATE MODEL house_bina
ry_classifier USING svm_classification FEATURES tax, bath, size TARGET price < 100000 FROM houses; |
        | {-.0623524,.000946067,.037792,.0225448} | {batch_size,decay,lambda,learning_rate,max_iterat
ions,max_seconds,optimizer,tolerance,seed,verbose} | {1000,.95,.01,.8,100,0,gd,.0005,1640332797,false}
 | {23,701,701,701,23,23,1043,701,23,16} | {categories} | {false,true} |          | {accuracy,f1,preci
sion,recall,loss} | {.666667,.666667,.625,.714286,7.10655} |
(1 row)
```

任务二：观察新模型的信息，将执行结果截图。

```
postgres=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
```

```
openGauss=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
        modelname        | modelowner |         createtime         | processedtuples | discardedtuples
 | pre_process_time | exec_time | iterations | outputtype |     modeltype      |
                                                  query
                                     | modeldata |                  weight                   |
                                    hyperparametersnames                                     |
          hyperparametersvalues              |          hyperparametersoids          |  coefnames
|  coefvalues  | coefoids |         trainingscoresname          |            trainingscoresvalue
      | modeldescribe
-------------------------+------------+----------------------------+----------------+----------------
-+------------------+-----------+------------+------------+--------------------+--------------------
-----------------------------------------------------------------------------------------------------
---------------------------------------+-----------+-------------------------------------------------+---
-----------------------------------------------------------------------------------------------------+---
-------------------------------------------------+-----------------------------------------+-------------
+--------------+----------+-------------------------------------+-----------------------------------
------+--------------
 house_binary_classifier |         10 | 2021-12-24 16:00:49.835598 |              15 |               0
 |               0 |    .01167 |        100 |         16 | svm_classification | CREATE MODEL house_bi
nary_classifier USING svm_classification FEATURES tax, bath, size TARGET price < 100000 FROM houses WI
TH batch_size=1, learning_rate=0.001; |           | {-.00224325,4.50771e-05,.00226251,.000456642} | {b
atch_size,decay,lambda,learning_rate,max_iterations,max_seconds,optimizer,tolerance,seed,verbose} | {1
,.95,.01,.001,100,0,gd,.0005,1640332849,false} | {23,701,701,701,23,23,1043,701,23,16} | {categories}
| {false,true} |          | {accuracy,f1,precision,recall,loss} | {.733333,.714286,.714286,.714286,9.7
0613} |
(1 row)
```

任务三：利用训练好的逻辑回归模型预测数据，并与 SVM 算法进行比较，将执行结果截图。

```
postgres=# SELECT tax, bath, size, price, price < 100000 AS price_actual, PREDICT BY
house_binary_classifier (FEATURES tax, bath, size) AS price_svm_pred, PREDICT BY
house_logistic_classifier (FEATURES tax, bath, size) AS price_logistic_pred FROM houses;
```

```
openGauss=# SELECT tax, bath, size, price, price < 100000 AS price_actual, PREDICT BY house_binary_cla
ssifier (FEATURES tax, bath, size) AS price_svm_pred, PREDICT BY house_logistic_classifier (FEATURES t
ax, bath, size) AS price_logistic_pred FROM houses;
 tax  | bath | size | price  | price_actual | price_svm_pred | price_logistic_pred
------+------+------+--------+--------------+----------------+---------------------
  590 |    1 |  770 |  50000 | t            | t              | t
 1050 |    2 | 1410 |  85000 | t            | t              | t
   20 |    1 | 1060 |  22500 | t            | t              | t
  870 |    2 | 1300 |  90000 | t            | t              | t
 1320 |    2 | 1500 | 133000 | f            | t              | t
 1350 |    1 |  820 |  90500 | t            | f              | f
 2790 |  2.5 | 2130 | 260000 | f            | f              | f
  680 |    1 | 1170 | 142500 | f            | t              | t
 1840 |    2 | 1500 | 160000 | f            | f              | f
 3680 |    2 | 2790 | 240000 | f            | f              | f
 1660 |    1 | 1030 |  87000 | t            | f              | f
 1620 |    2 | 1250 | 118600 | f            | f              | f
 3100 |    2 | 1760 | 140000 | f            | f              | f
 2070 |    3 | 1550 | 148000 | f            | f              | f
  650 |  1.5 | 1450 |  65000 | t            | t              | t
(15 rows)
```

实践思考题 1：分类模型与回归模型有何不同?

分类和回归的区别在于输出变量的类型；定量输出称为回归，或者说是连续变量预测；定性输出称为分类，或者说是离散变量预测。例如：预测明天的气温是多少度，这是一个回归任务；预测明天是阴、晴还是雨，就是一个分类任务。

实践思考题 2：什么是 SVM 算法?

SVM 是一个二元分类算法，线性分类和非线性分类都支持。经过演进，现在也可以支持多元分类，同时经过扩展，也能应用于回归问题；支持向量机（support vector machines，SVM）是一种二分类模型，它将实例的特征向量映射为空间中的一些点，SVM 的目的就是想要画出一条线，以 “最好地” 区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。SVM 适合中小型数据样本、非线性、高维的分类问题。

实践思考题 3：分类问题有哪些评价指标，请分别说明他们的含义?

分类问题主要有以下评价指标，分别是：准确率(Accuracy)，精确率(Precision)，召回率(Recall)，综合评价指标 F-Measure（F 值），ROC 曲线，AUC 曲线和 P-R 曲线。

1、 准确率

对于给定的测试数据集，分类器正确分类的样本数与总样本数之比

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2、 精确率

精确率是检索出相关文档数与检索出的文档总数的比率（正确分类的正例个数占分类为正例的实例个数的比例），衡量的是检索系统的查准率。

$$precision = \frac{TP}{TP + FP}$$

3、 召回率

召回率是指检索出的相关文档数和文档库中所有的相关文档数的比率（正确分类的正例个数占实际正例个数的比例），衡量的是检索系统的查全率。

$$recall = \frac{TP}{TP + FN}$$

为了能够评价不同算法优劣，在 Precision 和 Recall 的基础上提出了 F1 值的概念，来对 Precision 和 Recall 进行整体评价

$$F1 = \frac{精确率 * 召回率 * 2}{精确率 + 召回率}$$

4、 Precision 和 Recall 指标有时候会出现矛盾的情况，这样就需要综合考虑他们，最常见的方法就是 F-Measure（又称为 F-Score）。F-Score 是 Precision 和 Recall 的加权调和平均：综合评价指标

$$F = \frac{(a^2+1)P * R}{a^2(P+R)}$$

当参数 a = 1 a=1a=1 时，就是最常见的 F1.因此，F1 综合了 P 和 R 的结果，当 F1 较高时则能说明试验方法比较有效。

5、 ROC 曲线

ROC 曲线：接收者操作特征（receiver operating characteristic），ROC 曲线上每个点反映着对同一信号刺激的感受性。

6、 AUC 曲线

AUC(Area Under Curve)：ROC 曲线下的面积，介于 0.1 和 1 之间。AUC 作为数值可以直观的评价分类器的好坏，值越大越好。

AUC 的物理意义：首先 AUC 值是一个概率值。假设分类器的输出是样本属于正类的 Score（置信度），则 AUC 的物理意义为，任取一对（正、负）样本，正样本的 Score 大于负样本的 Score 的概率。

实践思考题 4：回归问题有哪些评价指标，请分别说明他们的含义?

回归问题主要有以下评价指标，分别是：平均绝对误差（Mean Absolute Error，MAE），均方误差（Mean Squared Error，MSE），R-square(决定系数)，Adjusted R-Square (校正决定系数）。

1、平均绝对误差就是指预测值与真实值之间平均相差多大 ：

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|f_i - y_i|$$

平均绝对误差能更好地反映预测值误差的实际情况.

2、均方误差。观测值与真值偏差的平方和与观测次数的比值：

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(f_i - y_i)^2$$

这也是线性回归中最常用的损失函数，线性回归过程中尽量让该损失函数最小。那么模型之间的对比也可以用它来比较。

MSE 可以评价数据的变化程度，MSE 的值越小，说明预测模型描述实验数据具有更好的精确度。

3、 决定系数

$$R^2 = 1 - \frac{\sum (Y\_actual - Y\_predict)^2}{\sum (Y\_actual - Y\_mean)^2}$$

4、 Adjusted R-Square (校正决定系数)

$$R^2\_adjusted = 1 - \frac{(1 - R^2)(n-1)}{n-p-1}$$

n 为样本数量，p 为特征数量。消除了样本数量和特征数量的影响