2024年全国大学生计算机系统能力大赛 PolarDB 数据库创新设计赛（天池杯）

决赛方案报告

学校：南开大学

队伍：数据都队

队员：王荣熙 岳志鑫 张耕嘉

填写说明：

- 正文、标题格式已经在本文中设定，请勿修改；
- 本文档应结构清晰，描述准确，不冗长拖沓；
- 提交文档时，以 PDF 格式提交；
- 本文档内容是初赛内容的组成部分，务必真实填写。如不属实，将导致奖项等级降低甚至终止参加比赛。

目　录

1. 代码及描述文档

主要修改方向如下:

polardb_build.sh：	修改相关参数
tpch_copy.sh：	设置参数
sql/：	优化 sql 语句
index.txt:	建立索引
postgresql.conf.sample.polardb_pg(src/backend/utils/misc/)：	修改参数
postgresql.conf.sample(src/backend/utils/misc/)：	修改参数

2. 性能优化设计

2.1 优化目标

提高数据查询速度、降低磁盘 I/O 开销、减少内存使用、减少表连接中间量，添加高性能索引。

2.2 优化策略

（1）通过使用单机并行,启用强制并行度加快查询速度。
（2）开启 PolarDB 预读功能，使用共享储存，减少 IO 次数。
（3）通过对 SQL 语句进行优化，使用逻辑等价、物化视图、视图或提前过滤等方式减少查询量。
（4）通过对 SQL 语句的分析添加全局索引和部分索引，增加查询速度.
（5）针对 SQL 语句设置扫描策略，例如是否顺序扫描，选择更为合适的查询计划。
（6）通过设置参数寻找更为高效的查询性能，增加查询效率。
（7）通过使用 unix socket 代替 tcp 连接，减少查询时长。

3. 性能优化实现路径

tpch_copy.sh
步骤 1:开启单机并行，对八张表设置 ePQ 的最大查询并行度：

```
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE nation SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE region SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE supplier SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE part SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE partsupp SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE customer SET (px_workers = 100);"
```

```
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE orders SET
(px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE lineitem
SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_px_dop_per_node = 8;" #加大节点并行度
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_px_optimizer_enable_hashagg = 0;" #防止内存用尽
```

步骤 2:开启 PolarDB 预读功能

guc.c

```
    //yzx 修改预分配
    #define MAX_CONFIG_VARS 100
    struct config_bool config_pool[MAX_CONFIG_VARS];
    int pool_index = 0; // 当前使用到的预分配池的索引
    void DefineCustomBoolVariable(const char *name,
                                const char *short_desc,
                                const char *long_desc,
                                bool *valueAddr,
                                bool bootValue,
                                GucContext context,
                                int flags,
                                GucBoolCheckHook check_hook,
                                GucBoolAssignHook assign_hook,
                                GucShowHook show_hook)
    {
        // 检查是否还有可用的预分配空间
        if (pool_index >= MAX_CONFIG_VARS) {
            // 如果超出预分配数量，可以选择扩展池或报错
            elog(ERROR, "Exceeded maximum number of custom bool
variables");
            return;
        }

        // 从预分配池中获取下一个可用的 config_bool
        struct config_bool *var = &config_pool[pool_index++];
        // 初始化字段
        var->gen.name = name;
        var->gen.short_desc = short_desc;
        var->gen.long_desc = long_desc;
        var->gen.context = context;
        var->gen.flags = flags;
        var->variable = valueAddr;
        var->boot_val = bootValue;
```

```
        var->reset_val = bootValue;
        var->check_hook = check_hook;
        var->assign_hook = assign_hook;
        var->show_hook = show_hook;

        // 调试输出
        if (var->assign_hook != NULL
&& !is_session_dedicated_guc(&var->gen)) {
            ELOG_PSS(DEBUG1, "conf with assign_hook is shared '%s'",
var->gen.name);
        }

        // 注册该变量
        define_custom_variable(&var->gen);
    }
```

步骤 3:对 SQL 语句进行优化

2.sql

```
WITH min_supplycost AS (
    SELECT
        ps_partkey,
        MIN(ps_supplycost) AS min_ps_supplycost
    FROM
        partsupp
        JOIN supplier ON s_suppkey = ps_suppkey
        JOIN nation ON s_nationkey = n_nationkey
        JOIN region ON n_regionkey = r_regionkey
    WHERE
        r_name = 'AMERICA'
    GROUP BY
        ps_partkey
)
SELECT
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
FROM
    part
    JOIN partsupp ON p_partkey = ps_partkey
```

```
    JOIN supplier ON s_suppkey = ps_suppkey
    JOIN nation ON s_nationkey = n_nationkey
    JOIN region ON n_regionkey = r_regionkey
    JOIN min_supplycost ON partsupp.ps_partkey =
min_supplycost.ps_partkey
                       AND partsupp.ps_supplycost =
min_supplycost.min_ps_supplycost
WHERE
    p_size = 28
    AND p_type LIKE '%COPPER'
    AND r_name = 'AMERICA'
ORDER BY
    s_acctbal DESC,
    n_name,
    s_name,
    p_partkey;
```

优化前是 **20247**ms，优化后是 **6188**ms，查询时间减少了约 69.44%。

原语句中，ps_supplycost 的最小值需要通过嵌套子查询计算，每次对 partsupp 表的扫描需要重新计算最小值，重复计算增加了查询的复杂度。我们将子查询独立提取为一个 CTE，可以一次性计算出 ps_partkey 对应的最小 ps_supplycost，避免了重复计算。并且原语句中嵌套子查询中包含多表连接，多次处理会增加临时表的创建和扫描开销。我们使用 CTE 与主查询联合，减少多次表连接，简化了查询逻辑。

3.sql

```
select
    l_orderkey,
    sum(l_extendedprice*(1-l_discount))as revenue,
    o_orderdate,
    o_shippriority
from
    customer
    join orders on c_custkey = o_custkey
    join lineitem on l_orderkey = o_orderkey
where
    c_mktsegment='BUILDING'
    and o_orderdate <date '1995-03-07'
    and l_shipdate  >date '1995-03-07'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
```

```
    o_orderdate;
```

优化前是 **104196**ms，优化后是 **27591**ms，查询时间减少了约 73.52%。

隐式链接会导致生成笛卡尔积，再通过 WHERE 条件过滤，很容易造成性能问题。显式连接明确指定了连接条件，优化器就能够更高效地构建连接计划。

4.sql

```
SELECT
    o.o_orderpriority,
    COUNT(*) AS order_count
FROM
    orders o
JOIN
    lineitem l ON o.o_orderkey = l.l_orderkey
WHERE
    o.o_orderdate >= DATE '1994-02-01'
    AND o.o_orderdate < DATE '1994-05-01'
    AND l.l_commitdate < l.l_receiptdate
GROUP BY
    o.o_orderpriority
ORDER BY
    o.o_orderpriority;
```

优化前是 **68659**ms，优化后是 **17472**ms，查询时间减少了约 74.55%。

原语句使用了 EXISTS 子查询，EXISTS 对于每条 orders 表中的记录，都会扫描 lineitem 表去逐条验证条件 l_orderkey = o_orderkey AND l_commitdate < l_receiptdate 是否成立。这种方式会导致对 lineitem 表多次扫描。而 lineitem 表又是最大的一张表，具有 1.2 亿条数据，会影响整体性能。所以我们将 EXISTS 子查询转换为 JOIN 明确连接逻辑，而我们对这个语句的主要的优化则是在建立索引上。

5.sql

```
SELECT
    n.n_name,
    SUM(l.l_extendedprice * (1 - l.l_discount)) AS revenue
FROM
    customer c
JOIN orders o ON c.c_custkey = o.o_custkey
JOIN lineitem l ON o.o_orderkey = l.l_orderkey
JOIN supplier s ON l.l_suppkey = s.s_suppkey
JOIN nation n ON s.s_nationkey = n.n_nationkey AND c.c_nationkey =
s.s_nationkey
JOIN region r ON n.n_regionkey = r.r_regionkey
WHERE
    r.r_name = 'ASIA'
```

```sql
    AND o.o_orderdate >= date '1993-01-01'
    AND o.o_orderdate < date '1994-01-01'
GROUP BY
    n.n_name
ORDER BY
    revenue DESC;
```

优化前是 **69830**ms，优化后是 **56025**ms，查询时间减少了约 19.76%。

原来的查询的方式是隐式连接，会导致笛卡尔积，会导致不必要的数据扫描和处理，对于大型表，性能会显著下降。我们使用显式 JOIN 语法，明确表之间的连接条件。我们对这个语句的主要的优化是在建立索引上。

7.sql

```sql
SELECT
    n1.n_name AS supp_nation,
    n2.n_name AS cust_nation,
    EXTRACT(YEAR FROM l_shipdate) AS l_year,
    SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM
    lineitem l
JOIN
    orders o ON o.o_orderkey = l.l_orderkey
JOIN
    customer c ON c.c_custkey = o.o_custkey
JOIN
    supplier s ON s.s_suppkey = l.l_suppkey
JOIN
    nation n1 ON s.s_nationkey = n1.n_nationkey
JOIN
    nation n2 ON c.c_nationkey = n2.n_nationkey
WHERE
    ((n1.n_name = 'PERU' AND n2.n_name = 'VIETNAM')
    OR (n1.n_name = 'VIETNAM' AND n2.n_name = 'PERU'))
    AND l_shipdate BETWEEN '1995-01-01' AND '1996-12-31'
GROUP BY
    n1.n_name,
    n2.n_name,
    EXTRACT(YEAR FROM l_shipdate)
ORDER BY
    n1.n_name,
    n2.n_name,
    l_year;
```

优化前是 68572ms，优化后是 55899ms，查询时间减少了约 18.48%。

原语句使用了一个子查询，会增加优化器解析的复杂性，可能导致中间结果存储与处理的开销。我们直接将子查询展开为主查询，将所有连接、计算和过滤的逻辑放在一个查询中完成，减少中间过程的存储和处理。并且我们用显示连接代替了隐式连接。原语句将 supp_nation、cust_nation 和 l_year 分别在子查询和外部查询中使用，这增加了分组和排序的复杂性。我们直接在主查询中分组和排序，提高了查询的效率。

8.sql

```
WITH asian_nations AS (
    SELECT n.n_nationkey
    FROM nation n
    JOIN region r ON n.n_regionkey = r.r_regionkey
    WHERE r.r_name = 'ASIA'
),
filtered_parts AS (
    SELECT p_partkey
    FROM part
    WHERE p_type = 'ECONOMY BRUSHED BRASS'
),
filtered_orders AS (
    SELECT o_orderkey, EXTRACT(YEAR FROM o_orderdate) AS o_year
    FROM orders
    WHERE o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31'
),
eligible_data AS (
    SELECT
        o.o_year,
        l.l_extendedprice,
        l.l_discount,
        n2.n_name
    FROM filtered_parts p
    JOIN lineitem l ON p.p_partkey = l.l_partkey
    JOIN supplier s ON l.l_suppkey = s.s_suppkey
    JOIN nation n2 ON s.s_nationkey = n2.n_nationkey
    JOIN partsupp ps ON p.p_partkey = ps.ps_partkey AND s.s_suppkey =
ps.ps_suppkey
    JOIN filtered_orders o ON o.o_orderkey = l.l_orderkey
    JOIN customer c ON o.o_orderkey = c.c_custkey
    JOIN asian_nations an ON c.c_nationkey = an.n_nationkey
)
SELECT
    o_year,
    SUM(CASE WHEN n_name = 'VIETNAM' THEN l_extendedprice*(1-l_discount)
ELSE 0 END)
```

```
    / SUM(l_extendedprice*(1-l_discount)) AS mkt_share
FROM eligible_data
GROUP BY o_year
ORDER BY o_year;
set enable_nestloop to default;
```

优化前是 **93189**ms，优化后是 **13298**ms，查询时间减少了约 85.73%。

原语句在主查询中对多个表进行一次性过滤和连接，这会导致大量无关数据进入连接操作，增加中间结果集的规模。我们使用 CTE 逐步裁剪数据，每一步骤只处理特定表的数据，减少中间结果规模，最后使用显示连接代替了隐式连接。

9.sql

```
SELECT
    n.n_name AS nation,
    EXTRACT(YEAR FROM o.o_orderdate) AS o_year,
    SUM(l.l_extendedprice * (1 - l.l_discount) - ps.ps_supplycost *
l.l_quantity) AS sum_profit
FROM
    nation n
JOIN supplier s ON n.n_nationkey = s.s_nationkey
JOIN lineitem l ON l.l_suppkey = s.s_suppkey
JOIN partsupp ps ON ps.ps_suppkey = s.s_suppkey AND ps.ps_partkey =
l.l_partkey
JOIN part p ON p.p_partkey = l.l_partkey
JOIN orders o ON o.o_orderkey = l.l_orderkey
WHERE
    p.p_name LIKE '%sandy%'
GROUP BY
    n.n_name,
    EXTRACT(YEAR FROM o.o_orderdate)
ORDER BY
    n.n_name,
    o_year DESC;
```

优化前是 **406243**ms，优化后是 **58211** ms，查询时间减少了约 85.67%。

我们使用显示连接代替了隐式连接。并且原语句通过嵌套子查询来查询，我们将子查询展开，直接通过显式连接完成所有计算。原语句在子查询中计算了 amount，随后在主查询中对 sum(amount)聚合。这种方式会导致冗余计算和中间数据的重复存储。而我们直接在主查询中完成计算和聚合。

10.sql

```
SELECT
    c.c_custkey,
    c.c_name,
```

```sql
    SUM(l.l_extendedprice * (1 - l.l_discount)) AS revenue,
    c.c_acctbal,
    n.n_name,
    c.c_address,
    c.c_phone,
    c.c_comment
FROM
    customer c
JOIN
    orders o ON c.c_custkey = o.o_custkey
JOIN
    lineitem l ON o.o_orderkey = l.l_orderkey
JOIN
    nation n ON c.c_nationkey = n.n_nationkey
WHERE
    o.o_orderdate >= DATE '1993-12-01'
    AND o.o_orderdate < DATE '1994-03-01'
    AND l.l_returnflag = 'R'
GROUP BY
    c.c_custkey,
    c.c_name,
    c.c_acctbal,
    c.c_phone,
    n.n_name,
    c.c_address,
    c.c_comment
ORDER BY
    revenue DESC;
```

优化前是 **406243**ms，优化后是 **58211** ms，查询时间减少了约 85.67%。

我们使用显示连接代替了隐式连接。我们对这个语句的主要的优化是在建立索引上。

12.sql

```sql
SELECT
    l.l_shipmode,
    SUM(CASE
        WHEN o.o_orderpriority IN ('1-URGENT', '2-HIGH') THEN 1
        ELSE 0
    END) AS high_line_count,
    SUM(CASE
        WHEN o.o_orderpriority NOT IN ('1-URGENT', '2-HIGH') THEN 1
        ELSE 0
    END) AS low_line_count
FROM
```

```
    orders o
JOIN
    lineitem l ON o.o_orderkey = l.l_orderkey
WHERE
    l.l_shipmode IN ('MAIL', 'SHIP')
    AND l.l_commitdate < l.l_receiptdate
    AND l.l_shipdate < l.l_commitdate
    AND l.l_receiptdate >= DATE '1996-01-01'
    AND l.l_receiptdate < DATE '1997-01-01'
GROUP BY
    l.l_shipmode
ORDER BY
    l.l_shipmode;
```

优化前是 **86642**ms，优化后是 **8069**ms，查询时间减少了约 90.69%。

我们使用显示连接代替了隐式连接。我们对这个语句的主要的优化是在建立索引上。

14.sql

```
SELECT
    100.00 * SUM(l.l_extendedprice * (1 - l.l_discount)) FILTER (WHERE
p.p_type LIKE 'PROMO%')
    / SUM(l.l_extendedprice * (1 - l.l_discount)) AS promo_revenue
FROM
    lineitem l
JOIN
    part p ON l.l_partkey = p.p_partkey
WHERE
    l.l_shipdate >= DATE '1996-07-01'
    AND l.l_shipdate < DATE '1996-08-01';
```

优化前是 **58407**ms，优化后是 **835**ms，查询时间减少了约 98.57%。

原语句使用了 CASE 表达式进行条件聚合，每次计算都需要判断 CASE 条件，即使不满足条件也需要返回 0，增加了计算复杂度。我们优化后使用 FILTER 子句，FILTER 子句只对满足条件的数据进行聚合，避免了不必要的判断和计算。并且，我们使用显示连接代替了隐式连接。

16.sql

```
SELECT
    p.p_brand,
    p.p_type,
    p.p_size,
    COUNT(DISTINCT ps.ps_suppkey) AS supplier_cnt
FROM
```

```sql
    partsupp ps
JOIN
    part p ON p.p_partkey = ps.ps_partkey
LEFT JOIN
    supplier s ON ps.ps_suppkey = s.s_suppkey AND s.s_comment LIKE
'%Customer%complaints%'
WHERE
    p.p_brand <> 'Brand#13'
    AND p.p_type NOT LIKE 'ECONOMY BRUSHED%'
    AND p.p_size IN (37, 49, 46, 26, 11, 41, 13, 21)
    AND s.s_suppkey IS NULL  -- Only include suppliers not in the
complaints list
GROUP BY
    p.p_brand,
    p.p_type,
    p.p_size
ORDER BY
    supplier_cnt DESC,
    p.p_brand,
    p.p_type,
    p.p_size;
```

优化前是 **21626**ms，优化后是 **5344**ms，查询时间减少了约 75.29%。

原语句使用 NOT IN 子查询，在数据量较大的情况下效果可能不会很好，所以我们替换为 LEFT JOIN，并且我们添加了索引，进一步加速了 LEFT JOIN 的过程，用显示连接替换了隐式连接。

18.sql

```sql
SET work_mem = '256MB';
CREATE MATERIALIZED VIEW mv_filtered_lineitem AS
SELECT
    l_orderkey,
    SUM(l_quantity) AS total_quantity
FROM
    lineitem
GROUP BY
    l_orderkey
HAVING
    SUM(l_quantity) > 315;
-- 主查询，利用物化视图加速
SELECT
    c.c_name,
    c.c_custkey,
```

```sql
    o.o_orderkey,
    o.o_orderdate,
    o.o_totalprice,
    SUM(l.l_quantity) AS total_quantity
FROM
    customer c
JOIN
    orders o ON c.c_custkey = o.o_custkey
JOIN
    mv_filtered_lineitem f ON o.o_orderkey = f.l_orderkey
JOIN
    lineitem l ON o.o_orderkey = l.l_orderkey
GROUP BY
    c.c_name,
    c.c_custkey,
    o.o_orderkey,
    o.o_orderdate,
    o.o_totalprice
ORDER BY
    o.o_totalprice DESC,
    o.o_orderdate;
SET work_mem='1024MB';
```

优化前是 **249230**ms，优化后是 **26913**ms，查询时间减少了约 89.20%。

原语句使用子查询在 lineitem 表中筛选满足条件的 l_orderkey，lineitem 表数据量大，重复分组和聚合的性能开销十分高。所以我们使用物化视图，物化视图可以将 lineitem 表的筛选结果提前计算并存储，避免了每次主查询重复计算，最后我们用显示连接替换了隐式连接。

20.sql

```sql
WITH wheat_parts AS (
    SELECT p_partkey
    FROM part
    WHERE p_name LIKE 'wheat%'
),
japan_suppliers AS (
    SELECT s.s_suppkey, s.s_name, s.s_address
    FROM supplier s
    JOIN nation n ON s.s_nationkey = n.n_nationkey
    WHERE n.n_name = 'JAPAN'
),
candidate_ps AS (
    SELECT ps.ps_partkey, ps.ps_suppkey, ps.ps_availqty
```

```
    FROM partsupp ps
    JOIN wheat_parts wp ON ps.ps_partkey = wp.p_partkey
    JOIN japan_suppliers js ON js.s_suppkey = ps.ps_suppkey
),
lineitem_agg AS (
    SELECT l.l_partkey, l.l_suppkey, 0.5 * SUM(l.l_quantity) AS
half_qty_sum
    FROM lineitem l
    JOIN candidate_ps cps ON l.l_partkey = cps.ps_partkey
                        AND l.l_suppkey = cps.ps_suppkey
    WHERE l.l_shipdate >= DATE '1997-01-01'
      AND l.l_shipdate < DATE '1998-01-01'
    GROUP BY l.l_partkey, l.l_suppkey
)
SELECT distinct js.s_name, js.s_address
FROM japan_suppliers js
JOIN candidate_ps cps ON js.s_suppkey = cps.ps_suppkey
LEFT JOIN lineitem_agg la ON la.l_partkey = cps.ps_partkey
                        AND la.l_suppkey = cps.ps_suppkey
WHERE cps.ps_availqty > la.half_qty_sum
ORDER BY js.s_name;
```

优化前是 **317881**ms，优化后是 **13772**ms，查询时间减少了约 95.67%。

原语句使用多层嵌套子查询，嵌套子查询层次较深，增加了查询解析和执行的复杂度，我们使用 CTE 将逻辑分解成多个步骤减少子查询重复计算的开销，将隐式连接替换成显示连接。

21.sql

```
SELECT
    s.s_name,
    COUNT(*) AS numwait
FROM
    supplier s
JOIN
    lineitem l1 ON s.s_suppkey = l1.l_suppkey
JOIN
    orders o ON o.o_orderkey = l1.l_orderkey
JOIN
    nation n ON s.s_nationkey = n.n_nationkey
WHERE
    o.o_orderstatus = 'F'
    AND l1.l_receiptdate > l1.l_commitdate
    AND EXISTS (
        SELECT 1
```

```sql
        FROM lineitem l2
        WHERE l2.l_orderkey = l1.l_orderkey
        AND l2.l_suppkey <> l1.l_suppkey
    )
    AND NOT EXISTS (
        SELECT 1
        FROM lineitem l3
        WHERE l3.l_orderkey = l1.l_orderkey
        AND l3.l_suppkey <> l1.l_suppkey
        AND l3.l_receiptdate > l3.l_commitdate
    )
    AND n.n_name = 'RUSSIA'
GROUP BY
    s.s_name
ORDER BY
    numwait DESC,
    s.s_name;
```

优化前是 **96998**ms，优化后是 **55804**ms，查询时间减少了约 42.47%。

子查询使用 SELECT *，处理的数据列较多，增加了不必要的开销，所以我们替换为 SELECT 1，仅验证子查询条件是否满足，而不返回具体数据，还将隐式连接替换为显示连接。

步骤 4:添加索引

index.txt

```sql
-- 1
create index i1_1 on lineitem (l_returnflag,l_linestatus) include
(l_quantity,l_extendedprice,l_discount,l_tax) where l_shipdate <= date
'1998-08-05';

-- 2
CREATE INDEX i2_1 ON partsupp (ps_partkey, ps_suppkey) INCLUDE
(ps_supplycost); --8, 9 也用
CREATE INDEX i2_2 ON part (p_partkey) INCLUDE (p_size, p_type) WHERE
p_size = 28 AND p_type LIKE '%COPPER';

-- 3
create index i3_1 on customer (c_custkey) where c_mktsegment =
'BUILDING';
create index i3_2 on orders (o_custkey) include
(o_orderdate,o_shippriority) where o_orderdate < date '1995-03-07';
create index i3_3 on lineitem (l_orderkey) include
(l_extendedprice,l_discount) where l_shipdate > date '1995-03-07';
```

```sql
-- 4
create index i4_1 on lineitem (l_orderkey) where l_commitdate <
l_receiptdate; -- 12 也会用到
create index i4_2 on orders (o_orderpriority) where o_orderdate >= date
'1994-02-01' and o_orderdate < date '1994-05-01';

-- 5
create index i5_1 on orders (o_custkey) include (o_orderkey) where
o_orderdate >= date '1993-01-01' and o_orderdate < date '1994-01-01';
CREATE INDEX i5_2 ON supplier (s_suppkey, s_nationkey);

-- 6
CREATE INDEX i6_1 ON lineitem (l_discount) INCLUDE (l_extendedprice,
l_quantity) WHERE l_shipdate >= DATE '1993-01-01' AND l_shipdate < DATE
'1994-01-01';

-- 7
CREATE INDEX i7_1 ON lineitem (l_suppkey) INCLUDE (l_extendedprice,
l_shipdate, l_quantity) WHERE l_shipdate >= DATE '1995-01-01' AND
l_shipdate < DATE '1996-12-31'; --12-20 7 没有用到，真神奇
create index i7_2 on nation (n_nationkey); -- 8,9 也用

-- 8
CREATE INDEX i8_1 ON orders (o_orderdate, o_orderkey, o_custkey) WHERE
o_orderdate BETWEEN DATE '1995-01-01' AND DATE '1996-12-31';
create index i8_2 on part (p_partkey) where p_type = 'ECONOMY BRUSHED
BRASS';
CREATE INDEX i8_4 ON customer (c_custkey, c_nationkey);

-- 9
CREATE EXTENSION pg_trgm;
CREATE INDEX i9_2 ON part USING gin(p_name gin_trgm_ops);

-- 10
create index i10_1 on lineitem (l_orderkey) include
(l_extendedprice,l_discount) where l_returnflag = 'R';
create index i10_2 on orders (o_orderkey) include (o_custkey) where
o_orderdate >= date '1993-12-01' and o_orderdate < date '1994-3-01';

-- 12
CREATE INDEX i12_1 ON lineitem (l_commitdate) INCLUDE
(l_shipmode,l_shipdate, l_receiptdate,l_orderkey) WHERE
l_receiptdate >= DATE '1996-01-01' AND l_receiptdate < DATE '1997-01-
01';
```

```
-- 14
create index i14_1 on lineitem(l_partkey) include
(l_extendedprice,l_discount) where l_shipdate >= date '1996-07-01' and
l_shipdate < date '1996-08-01';

-- 15
CREATE INDEX i15_1 ON lineitem (l_suppkey) INCLUDE (l_extendedprice,
l_discount) WHERE l_shipdate >= DATE '1996-09-01' AND l_shipdate < DATE
'1996-12-01';

-- 17
create index i17_1 on part (p_partkey) where p_brand = 'Brand#35' and
p_container = 'JUMBO PKG';
create index i17_2 on lineitem (l_partkey) include
(l_quantity,l_extendedprice);

-- 18
create index i18_1 on lineitem (l_orderkey) include (l_quantity);

-- 20
create index i20_1 on lineitem (l_partkey,l_suppkey) where
l_shipdate >= date '1997-01-01' and l_shipdate < date '1998-01-01';
create index i20_2 on part (p_partkey) where p_name like 'wheat%';

-- 21
create index i21_1 on lineitem (l_orderkey) include (l_suppkey) where
l_receiptdate <> l_commitdate;
create index i21_2 on lineitem (l_orderkey) include (l_suppkey);
create index i21_3 on orders (o_orderkey) where o_orderstatus = 'F';
```

步骤 5:设置扫描策略

1.sql 禁用顺序扫描：`set enable_seqscan = off;`

步骤 6:修改参数

1.polardb.build.sh

```
echo "polar_enable_shared_storage_mode = on
        polar_hostid = 1
        max_connections = 400
        polar_wal_pipeline_enable = true
        polar_create_table_with_full_replica_identity = off
        logging_collector = on
        log_directory = 'pg_log'
        unix_socket_directories='/tmp'
```

```
        shared_buffers = '12GB'
        synchronous_commit = off
        full_page_writes = off
        random_page_cost = 1.1
        autovacuum_naptime = 100min
        max_worker_processes = 128
        polar_use_statistical_relpages = off
        polar_enable_persisted_buffer_pool = off
        polar_nblocks_cache_mode = 'all'
        polar_enable_replica_use_smgr_cache = on
        polar_bulk_read_size = 128kB
        polar_bulk_extend_size = 4MB
        polar_index_create_bulk_extend_size = 512
        maintenance_work_mem = 1GB
        work_mem = 1024MB
        synchronous_commit = off
        fsync = off
        max_connections = 300
        max_wal_senders = 0
        hot_standby = off
        archive_mode = off
        wal_log_hints = off
        wal_level = minimal
        max_replication_slots = 0
        max_parallel_workers = 6
        max_parallel_maintenance_workers =24
        wal_buffers = '64MB'
        checkpoint_timeout = 30min
        max_wal_size = 4GB
        min_wal_size = 80MB
        checkpoint_completion_target = 0.9
        effective_cache_size = 12GB
        polar_wal_pipeline_mode = 1
        polar_enable_standby_use_smgr_cache = on" >>
$pg_bld_master_dir/postgresql.conf

# 删掉了 cflags cxxflags -g 选项
gcc_opt_level_flag="-pipe -Wall -grecord-gcc-switches -march=native -
mtune=native -fno-omit-frame-pointer -I/usr/include/et"
# 使用更大的数据块
./configure --prefix=$pg_bld_basedir --with-pgport=$pg_bld_port --with-
wal-blocksize=32 --with-blocksize=32 $common_configure_flag
$configure_flag
# 使用 SQL_ASCII 编码，关闭块校验
```

```
su_eval "$pg_bld_basedir/bin/initdb -E SQL_ASCII --locale=C -U
$pg_db_user -D $pg_bld_master_dir $tde_initdb_args"
```

2.toch_copy.sh

```
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_enable_shm_aset = on;" #开启全局共享内存
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_ss_dispatcher_count = 8;" #Dispatcher 进程的最大个数为 8
psql -h /tmp -p 5432 -U postgres -d postgres -c "select 'alter table
'||tablename||' set (parallel_workers=8);' from pg_tables where
schemaname='public';" #设置并行参数为 8
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
session_replication_role = 'replica';" #禁用外键约束
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
autovacuum = 'off';" # 关闭 autovacuum

psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
maintenance_work_mem = '4GB';" # 提升内存配置以优化索引创建
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE nation SET
(px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE region SET
(px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE supplier
SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE part SET
(px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE partsupp
SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE customer
SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE orders SET
(px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER TABLE lineitem
SET (px_workers = 100);"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_enable_px = ON;"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_px_dop_per_node = 8;"
psql -h /tmp -p 5432 -U postgres -d postgres -c "ALTER SYSTEM SET
polar_px_optimizer_enable_hashagg = 0;" #防止内存用尽
psql -h /tmp -p 5432 -U postgres -d postgres -c "update pg_class set
relpersistence ='p' where relnamespace='public'::regnamespace;" # 恢复表
的持久性设置
# 把 unlogged table 改为 logged table，生成表统计信息和 vm 文件.
```

```
psql -h /tmp -p 5432 -U postgres -d postgres -c "update pg_constraint
set convalidated=true where convalidated<>true;"
psql -h /tmp -p 5432 -U postgres -d postgres -c "vacuum analyze;"
```

3.postgresql.conf.sample

```
shared_buffers = 8GB
autovacuum = off
checkpoint_timeout = 1d
max_wal_size = 128GB
min_wal_size = 64GB
checkpoint_completion_target = 0.9
bgwriter_delay = 10ms     # 10-10000ms between rounds
bgwriter_lru_maxpages = 500   # max buffers written/round, 0 disables
bgwriter_lru_multiplier = 2.0   # 0-10.0 multiplier on buffers
scanned/round
bgwriter_flush_after = 512kB    # measured in pages, 0 disables
force_parallel_mode = on
```

4.postgresql.conf.sample.polardb_pg

```
wal_level=minimal #yzx 修改 确保 WAL 级别为 minimal，禁用 WAL 写入
max_wal_senders = 0        # 禁用 WAL 发送器
hot_standby = off           # 禁用热备份
synchronous_commit = off   # 禁用同步提交
max_wal_size=128GB
min_wal_size=64GB
bgwriter_delay=10ms
bgwriter_flush_after=512 #1MB
bgwriter_lru_maxpages=500
max_parallel_workers_per_gather =4
min_parallel_table_scan_size =0
min_parallel_index_scan_size =0
parallel_tuple_cost =0
parallel_setup_cost =0
wal_writer_flush_after=16MB
```

步骤 7:使用 unix 代替 tcp 连接

pg_hba.conf.sample

```
    # TYPE  DATABASE        USER            ADDRESS                 METHOD
    # "local" is for Unix domain socket connections only
    local   all             all                                     trust
    local   postgres        postgres                                trust
    # IPv4 local connections:
    host    all             all             127.0.0.1/32            trust
    # IPv6 local connections:
```

```
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
```

4. 性能提升效果

4.1 测试结果

- 优化前：数据查询时间为 2154.8650 秒。
- 优化后：数据查询时间为 457.7130 秒。

4.2 性能提升

- 数据查询时间减少了 78.76%。
- 详细信息如下：

SQL 语句	优化前(ms)	优化后（ms）	优化比
1.sql	105156	38878	63.02826277%
2.sql	20247	6188	69.437447525%
3.sql	104196	27591	73.52009674%
4.sql	68659	17472	74.55249858%
5.sql	69830	56025	19.76944007%
6.sql	66835	1277	98.08932446%
7.sql	68572	55899	18.48130432%
8.sql	93189	13298	85.73007544%
9.sql	406243	58211	85.67089156%
10.sql	69730	6672	90.43166499%
11.sql	9602	1878	80.44157467%
12.sql	86642	8069	90.68696475%
13.sql	11205	12046	-7.505577867%
14.sql	58407	835	98.57037684%
15.sql	131075	3726	97.15735266%
16.sql	21626	5344	75.28900398%
17.sql	74805	2676	96.42269902%
18.sql	249230	26913	89.20154075%
19.sql	17590	16282	7.436043206%
20.sql	317881	13772	95.66756113%
21.sql	96998	55804	42.46891688%
22.sql	7147	28857	-303.763817%

5. 指导老师的贡献

指导老师：温延龙
贡献：
1.对决赛的内容进行分析，判断加速 Tips 的内容是否值得去做，对当前的 polardb for postgress 的内容，以及相关的数据库的实现方法进行讲解。
2.对决赛的每个阶段的成果进行分析，检验是否符合逻辑，分析未来的优化前景。
3.对每个阶段指定目标，将总体的优化计划细分为每天的任务和查询分数的目标，每天都进行推进和成绩的更新。