Database Systems Homework #3
Exercise 5: Experiments
600.316
Spring 2016

Name: Joon Hyuck Choi, William Sun
JHED: jchoi100, wsun19

April 22, 2016

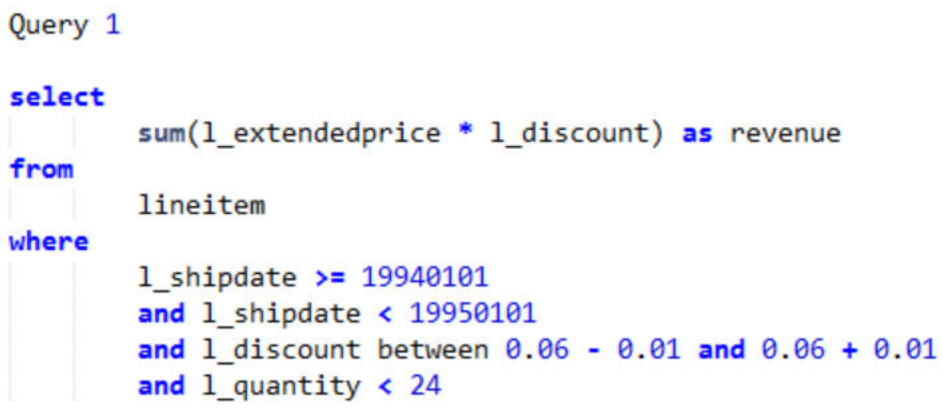
First, we give a quick outline of the material that is dealt with in this document. Our group conducted a total of ten run experiments on the five queries posted on the course website. The dataset we used is the *tpch-sf-0.001*.

For each of the five queries, we performed two runs : 1) un-optimized; 2) optimized. (As a note, the y-axes in our plots signify the running time in one-thousandth seconds.) Moreover, there was a utf-8 issue when running the *pickJoinOrder()* functions on the queries to produce actual results. Therefore, when collecting time-related data, we optimized the queries using only the *pushdownOperators()* function. However, both *pushdownOperators()* and *pickJoinOrder()* functions produced correct *.explain()* output, so we discuss those results extensively.

The document explains the different results obtained by taking the different techniques of running the queries. For each of the queries, we first give the original sql query statement, show a plot of the different running times of the two techniques, and explain the changes we made to the original query by going through the *.explain()* outputs. In the end, we give a general discussion about the optimization we applied not limited to the specific queries given. The source code written to run the queries are saved in a directory called *experiments* under the main directory of our submission.

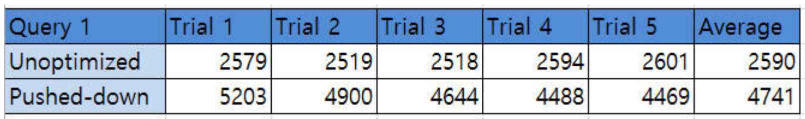
1 Query 1

The given sql query:

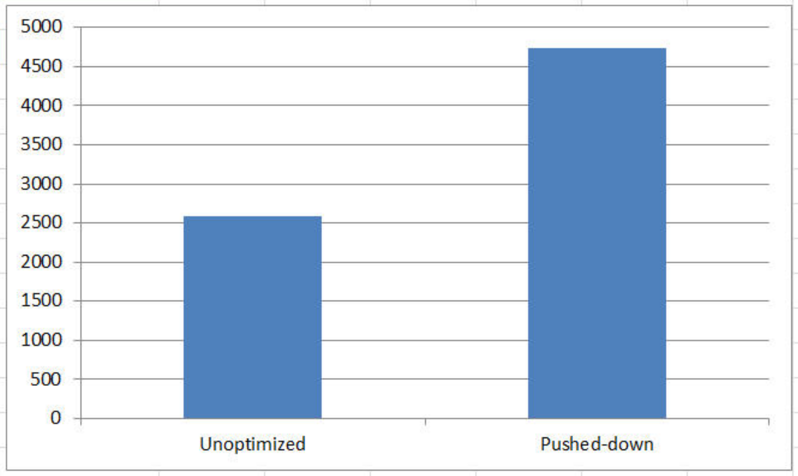


```
Query 1

select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= 19940101
        and l_shipdate < 19950101
        and l_discount between 0.06 - 0.01 and 0.06 + 0.01
        and l_quantity < 24
```

1.1 Running Times

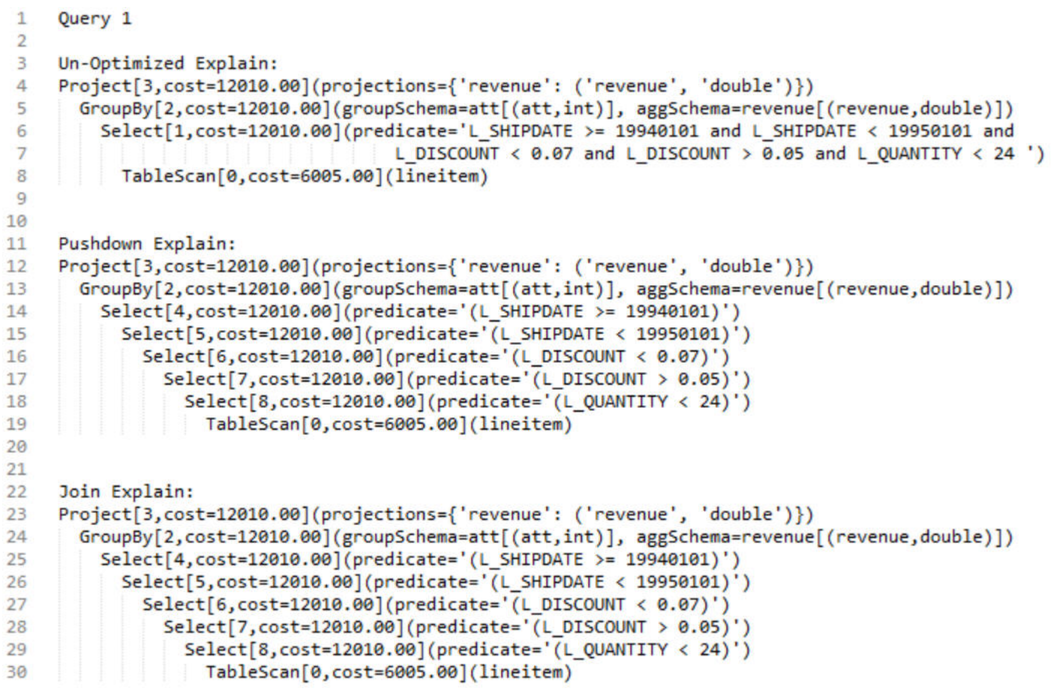


Query 1	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Unoptimized	2579	2519	2518	2594	2601	2590
Pushed-down	5203	4900	4644	4488	4469	4741



Non-optimized: 2.59 seconds
Optimized: 4.74 seconds
(Source code for each of the runs is included in our submission directory.)

1.2 Explain() Outputs



```
Query 1

Un-Optimized Explain:
Project[3,cost=12010.00](projections={'revenue': ('revenue', 'double')})
  GroupBy[2,cost=12010.00](groupSchema=att[(att,int)], aggSchema=revenue[(revenue,double)])
    Select[1,cost=12010.00](predicate='L_SHIPDATE >= 19940101 and L_SHIPDATE < 19950101 and
                                L_DISCOUNT < 0.07 and L_DISCOUNT > 0.05 and L_QUANTITY < 24 ')
      TableScan[0,cost=6005.00](lineitem)


Pushdown Explain:
Project[3,cost=12010.00](projections={'revenue': ('revenue', 'double')})
  GroupBy[2,cost=12010.00](groupSchema=att[(att,int)], aggSchema=revenue[(revenue,double)])
    Select[4,cost=12010.00](predicate='(L_SHIPDATE >= 19940101)')
      Select[5,cost=12010.00](predicate='(L_SHIPDATE < 19950101)')
        Select[6,cost=12010.00](predicate='(L_DISCOUNT < 0.07)')
          Select[7,cost=12010.00](predicate='(L_DISCOUNT > 0.05)')
            Select[8,cost=12010.00](predicate='(L_QUANTITY < 24)')
              TableScan[0,cost=6005.00](lineitem)


Join Explain:
Project[3,cost=12010.00](projections={'revenue': ('revenue', 'double')})
  GroupBy[2,cost=12010.00](groupSchema=att[(att,int)], aggSchema=revenue[(revenue,double)])
    Select[4,cost=12010.00](predicate='(L_SHIPDATE >= 19940101)')
      Select[5,cost=12010.00](predicate='(L_SHIPDATE < 19950101)')
        Select[6,cost=12010.00](predicate='(L_DISCOUNT < 0.07)')
          Select[7,cost=12010.00](predicate='(L_DISCOUNT > 0.05)')
            Select[8,cost=12010.00](predicate='(L_QUANTITY < 24)')
              TableScan[0,cost=6005.00](lineitem)
```

1.3 Discussion

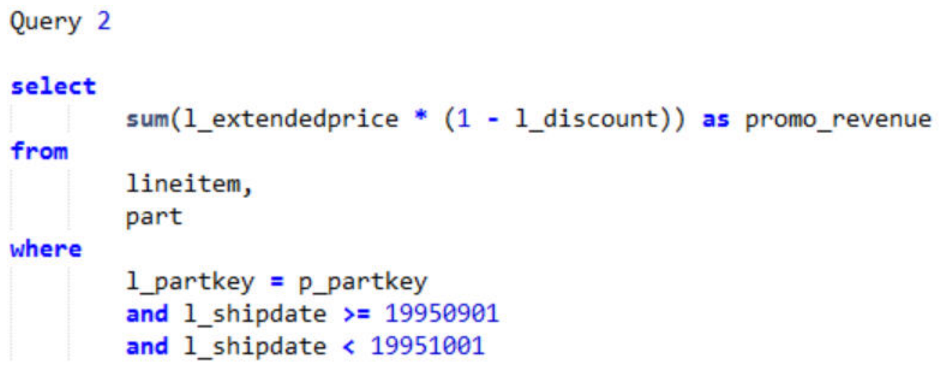
First, since this query does not involve any joins, the query plan tree before and after running the *pickjoinOrder()* function remains the same.

Morever, notice that the optimized query took about twice as long to run than the non-optimized version did. Moreover, even with our slow database implementation, the un-optimized version took only about 2 seconds to process the query. This indicates that for queries with this level of complexity, the overhead of figuring out the optimized way of processing the query made things go slower than the un-optimized run.

Such situations may happen for much more complex queries as well. Instead of using statistics and estimations, if we took the brute-force way of enumerating all possible optimization options, the overhead of optimizing would dominate the overall running time. This query was a very special case where we saw that the optimization overhead might slow things down in certain queries.

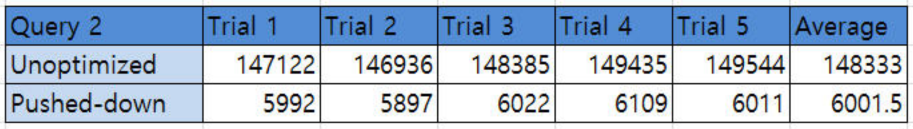
2 Query 2

The given sql query:

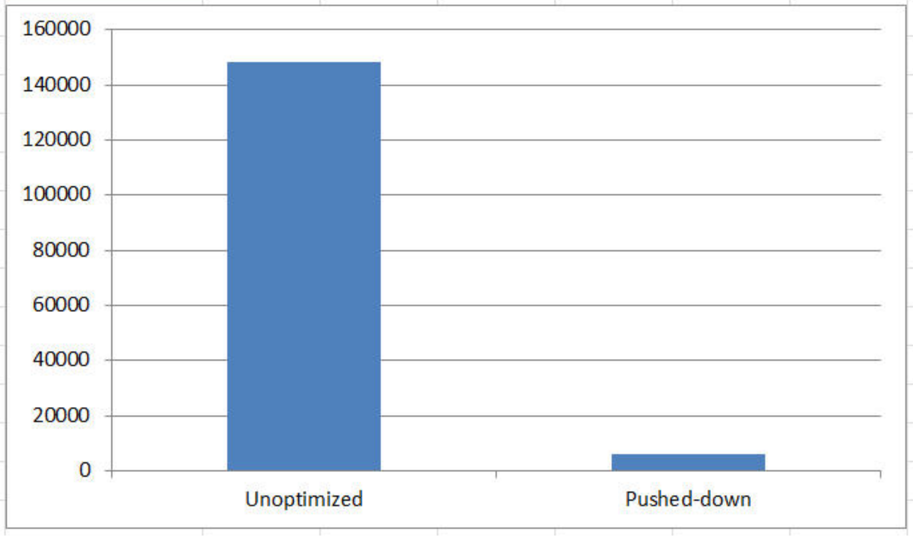


```
Query 2

select
        sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= 19950901
        and l_shipdate < 19951001
```

2.1 Running Times

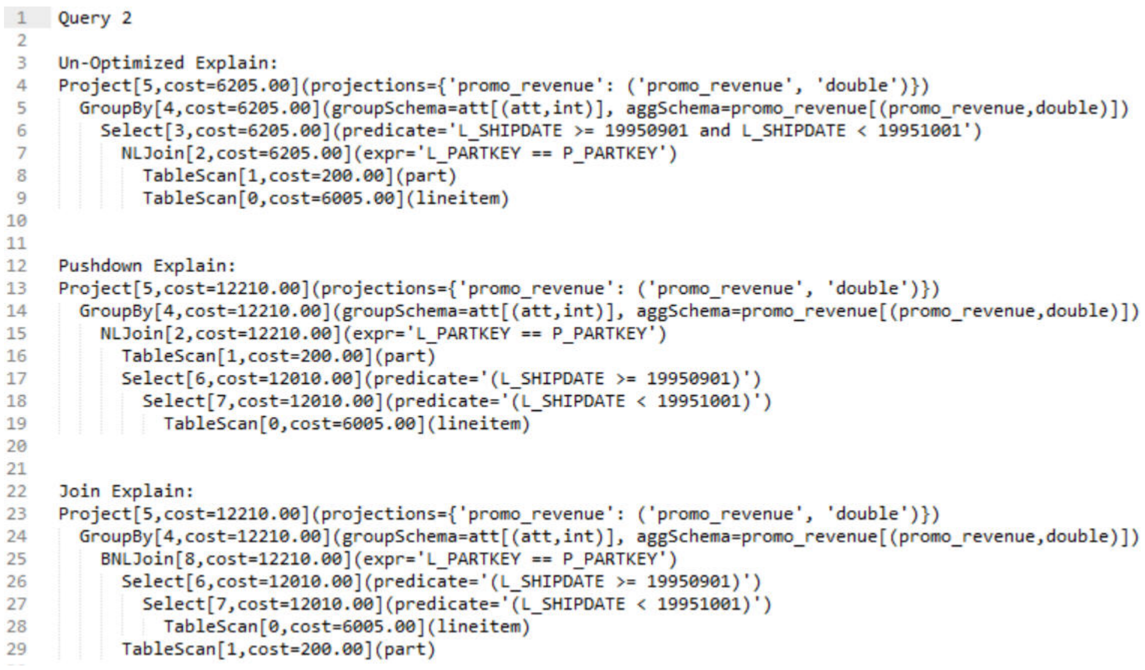


Query 2	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
Unoptimized	147122	146936	148385	149435	149544	148333
Pushed-down	5992	5897	6022	6109	6011	6001.5



Non-optimized: 148.333 seconds
Optimized: 6.001 seconds
(Source code for each of the runs is included in our submission directory.)

2.2 Explain() Outputs



```
Query 2

Un-Optimized Explain:
Project[5,cost=6205.00](projections={'promo_revenue': ('promo_revenue', 'double')})
  GroupBy[4,cost=6205.00](groupSchema=att[(att,int)], aggSchema=promo_revenue[(promo_revenue,double)])
    Select[3,cost=6205.00](predicate='L_SHIPDATE >= 19950901 and L_SHIPDATE < 19951001')
      NLJoin[2,cost=6205.00](expr='L_PARTKEY == P_PARTKEY')
        TableScan[1,cost=200.00](part)
        TableScan[0,cost=6005.00](lineitem)


Pushdown Explain:
Project[5,cost=12210.00](projections={'promo_revenue': ('promo_revenue', 'double')})
  GroupBy[4,cost=12210.00](groupSchema=att[(att,int)], aggSchema=promo_revenue[(promo_revenue,double)])
    NLJoin[2,cost=12210.00](expr='L_PARTKEY == P_PARTKEY')
      TableScan[1,cost=200.00](part)
      Select[6,cost=12010.00](predicate='(L_SHIPDATE >= 19950901)')
        Select[7,cost=12010.00](predicate='(L_SHIPDATE < 19951001)')
          TableScan[0,cost=6005.00](lineitem)


Join Explain:
Project[5,cost=12210.00](projections={'promo_revenue': ('promo_revenue', 'double')})
  GroupBy[4,cost=12210.00](groupSchema=att[(att,int)], aggSchema=promo_revenue[(promo_revenue,double)])
    BNLJoin[8,cost=12210.00](expr='L_PARTKEY == P_PARTKEY')
      Select[6,cost=12010.00](predicate='(L_SHIPDATE >= 19950901)')
        Select[7,cost=12010.00](predicate='(L_SHIPDATE < 19951001)')
          TableScan[0,cost=6005.00](lineitem)
      TableScan[1,cost=200.00](part)
```

2.3 Discussion

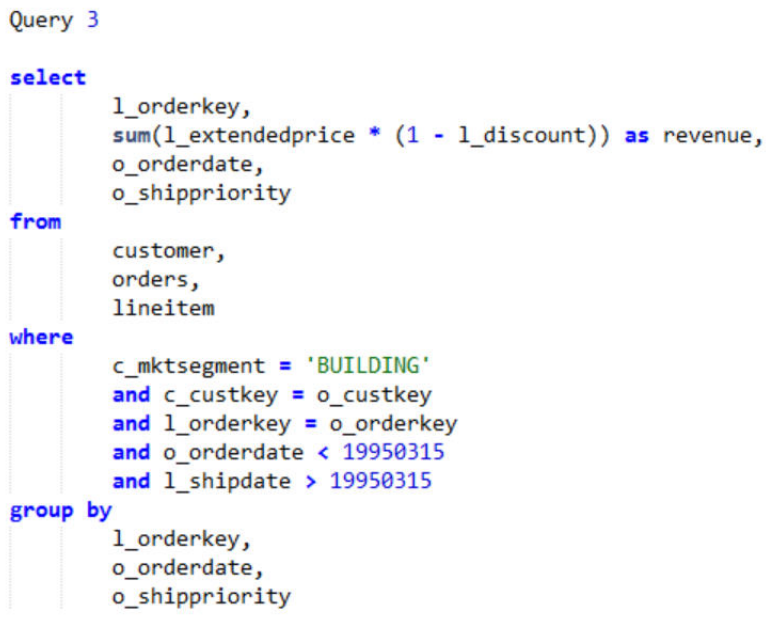
Much to our satisfaction, the *pushdownOperators* method made the query run much faster and, of couse, produce the same results. The main factor that made things faster with the optimized plan is that it pushed down all the selects as far down as it could.

For instance, in the original plan, all the *Select* predicates are above the *Join*, which necessitates joining all tuples inside the two tables to be joined. However, we know that we can filter out a lot of tuples beforehand to cut down the running time. Thus, we pushed down all the select statements as close to the base tables as possible. So we can see in the **Pushdown Explain**, the select statements reside directly above the TableScan. After filtering out unnecessary tuples, the join (which usually takes up a huge proportion of query run times) has less tuples to join.

For this query, the join optimizer changed the NLJoin to a BNLJoin because it is more cost efficient. It also changed the ordering of the join back to left deep.

3 Query 3

The given sql query:



```
Query 3

select
        l_orderkey,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate,
        o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = 'BUILDING'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < 19950315
        and l_shipdate > 19950315
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
```

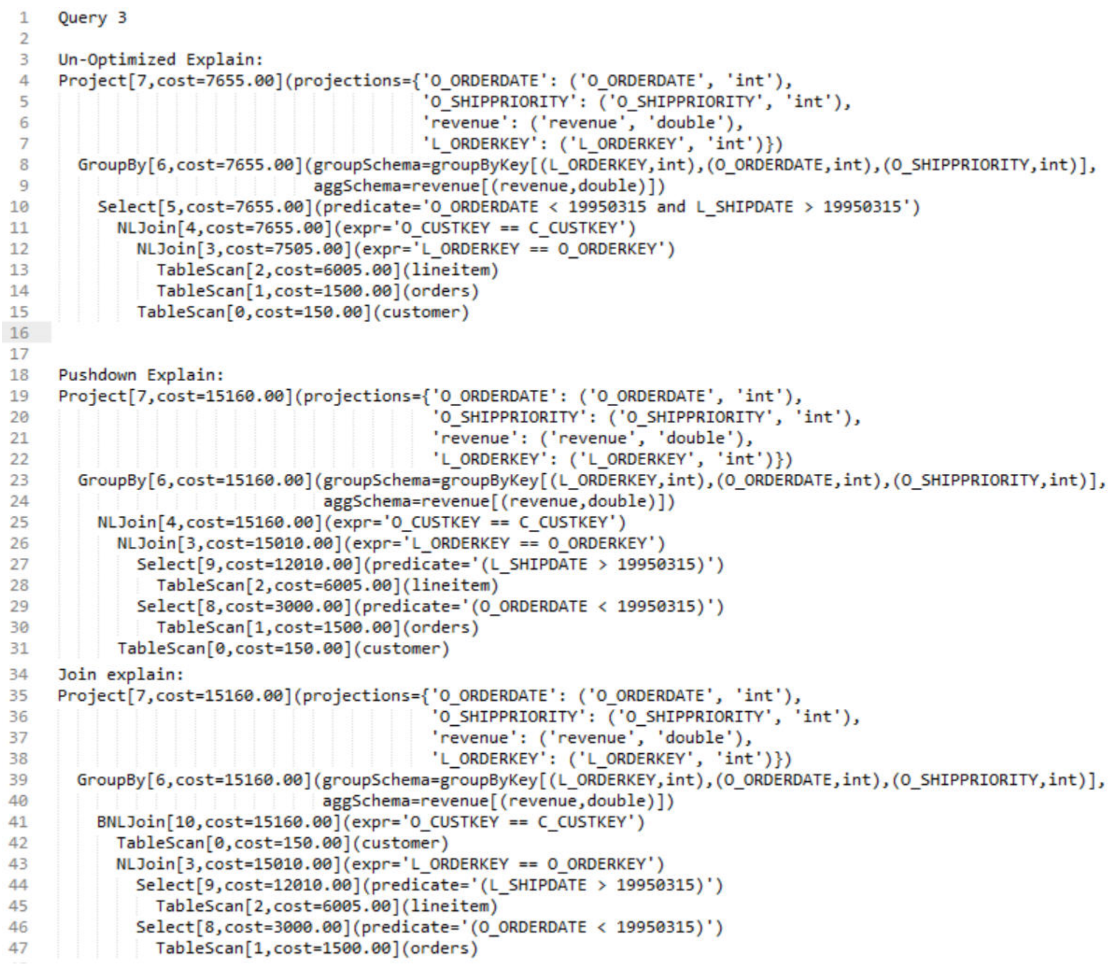
3.1 Running Times

Non-optimized:	x seconds
Optimized:	x seconds

$\rightarrow$ Unfortunately, queries 3, 4, and 5 took more than hours to run (even the optimized plans). Therefore, we had to cut them off. However, discussion on the changes in the *.explain()* outputs are given.

(Source code for each of the runs is included in our submission directory.)

3.2 Expalin() Outputs



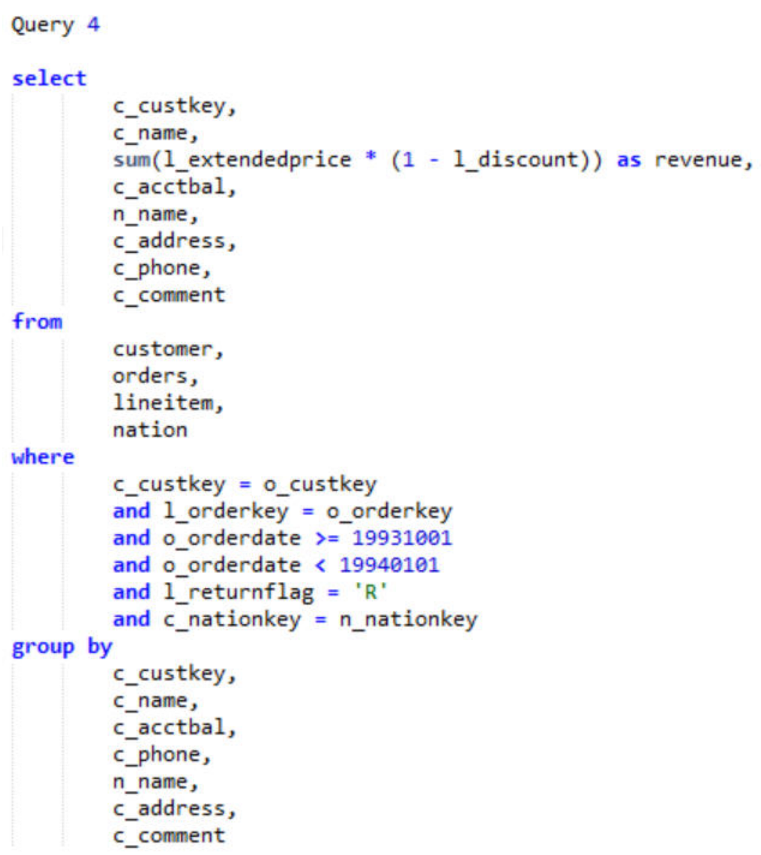
```
Query 3

Un-Optimized Explain:
Project[7,cost=7655.00](projections={'O_ORDERDATE': ('O_ORDERDATE', 'int'),
                                     'O_SHIPPRIORITY': ('O_SHIPPRIORITY', 'int'),
                                     'revenue': ('revenue', 'double'),
                                     'L_ORDERKEY': ('L_ORDERKEY', 'int')})
  GroupBy[6,cost=7655.00](groupSchema=groupByKey[(L_ORDERKEY,int),(O_ORDERDATE,int),(O_SHIPPRIORITY,int)],
                          aggSchema=revenue[(revenue,double)])
    Select[5,cost=7655.00](predicate='O_ORDERDATE < 19950315 and L_SHIPDATE > 19950315')
      NLJoin[4,cost=7655.00](expr='O_CUSTKEY == C_CUSTKEY')
        NLJoin[3,cost=7505.00](expr='L_ORDERKEY == O_ORDERKEY')
          TableScan[2,cost=6005.00](lineitem)
          TableScan[1,cost=1500.00](orders)
        TableScan[0,cost=150.00](customer)


Pushdown Explain:
Project[7,cost=15160.00](projections={'O_ORDERDATE': ('O_ORDERDATE', 'int'),
                                      'O_SHIPPRIORITY': ('O_SHIPPRIORITY', 'int'),
                                      'revenue': ('revenue', 'double'),
                                      'L_ORDERKEY': ('L_ORDERKEY', 'int')})
  GroupBy[6,cost=15160.00](groupSchema=groupByKey[(L_ORDERKEY,int),(O_ORDERDATE,int),(O_SHIPPRIORITY,int)],
                           aggSchema=revenue[(revenue,double)])
    NLJoin[4,cost=15160.00](expr='O_CUSTKEY == C_CUSTKEY')
      NLJoin[3,cost=15010.00](expr='L_ORDERKEY == O_ORDERKEY')
        Select[9,cost=12010.00](predicate='(L_SHIPDATE > 19950315)')
          TableScan[2,cost=6005.00](lineitem)
        Select[8,cost=3000.00](predicate='(O_ORDERDATE < 19950315)')
          TableScan[1,cost=1500.00](orders)
      TableScan[0,cost=150.00](customer)
Join explain:
Project[7,cost=15160.00](projections={'O_ORDERDATE': ('O_ORDERDATE', 'int'),
                                      'O_SHIPPRIORITY': ('O_SHIPPRIORITY', 'int'),
                                      'revenue': ('revenue', 'double'),
                                      'L_ORDERKEY': ('L_ORDERKEY', 'int')})
  GroupBy[6,cost=15160.00](groupSchema=groupByKey[(L_ORDERKEY,int),(O_ORDERDATE,int),(O_SHIPPRIORITY,int)],
                           aggSchema=revenue[(revenue,double)])
    BNLJoin[10,cost=15160.00](expr='O_CUSTKEY == C_CUSTKEY')
      TableScan[0,cost=150.00](customer)
      NLJoin[3,cost=15010.00](expr='L_ORDERKEY == O_ORDERKEY')
        Select[9,cost=12010.00](predicate='(L_SHIPDATE > 19950315)')
          TableScan[2,cost=6005.00](lineitem)
        Select[8,cost=3000.00](predicate='(O_ORDERDATE < 19950315)')
          TableScan[1,cost=1500.00](orders)
```

3.3 Discussion

First, the *pushdownOperators* pushed down each of the CNF-decomposed select predicates as near its base table as it can. For instance, in the un-optimized query plan, we can see that all the select filtering happens after two joins have taken place. As mentioned earlier, joins are expensive operations, and the more tuples there are in a table, the longer it takes to perform a join. So if we know in advance that certain tuples will not affect the result of the entire query, it would be cost-beneficial to filter them out from the source. This is exactly what our *pushdownOperators* is doing. (The predicate cmktsegment= ‘Building’ could not be processed and would produce error messages. We had to exclude this predicate.)

4 Query 4

The given sql query:



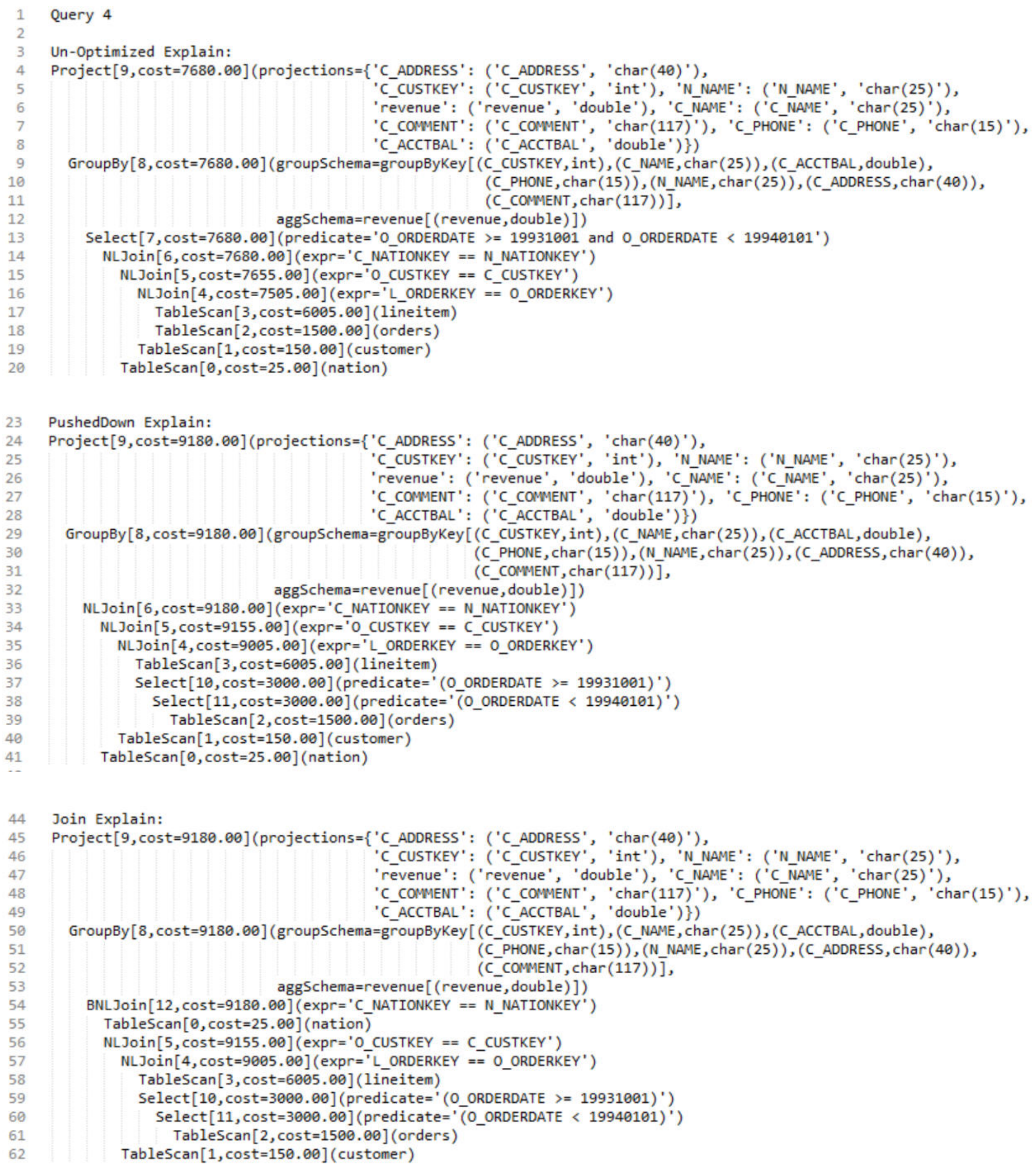
```
Query 4

select
		c_custkey,
		c_name,
		sum(l_extendedprice * (1 - l_discount)) as revenue,
		c_acctbal,
		n_name,
		c_address,
		c_phone,
		c_comment
from
		customer,
		orders,
		lineitem,
		nation
where
		c_custkey = o_custkey
		and l_orderkey = o_orderkey
		and o_orderdate >= 19931001
		and o_orderdate < 19940101
		and l_returnflag = 'R'
		and c_nationkey = n_nationkey
group by
		c_custkey,
		c_name,
		c_acctbal,
		c_phone,
		n_name,
		c_address,
		c_comment
```

4.1 Running Times

Non-optimized:	x seconds
Optimized:	x seconds

(Source code for each of the runs is included in our submission directory.)

4.2 Explain() Outputs



```
Query 4

Un-Optimized Explain:
Project[9,cost=7680.00](projections={'C_ADDRESS': ('C_ADDRESS', 'char(40)'),
                                     'C_CUSTKEY': ('C_CUSTKEY', 'int'), 'N_NAME': ('N_NAME', 'char(25)'),
                                     'revenue': ('revenue', 'double'), 'C_NAME': ('C_NAME', 'char(25)'),
                                     'C_COMMENT': ('C_COMMENT', 'char(117)'), 'C_PHONE': ('C_PHONE', 'char(15)'),
                                     'C_ACCTBAL': ('C_ACCTBAL', 'double')})
  GroupBy[8,cost=7680.00](groupSchema=groupByKey[(C_CUSTKEY,int),(C_NAME,char(25)),(C_ACCTBAL,double),
                                                  (C_PHONE,char(15)),(N_NAME,char(25)),(C_ADDRESS,char(40)),
                                                  (C_COMMENT,char(117))],
                          aggSchema=revenue[(revenue,double)])
    Select[7,cost=7680.00](predicate='O_ORDERDATE >= 19931001 and O_ORDERDATE < 19940101')
      NLJoin[6,cost=7680.00](expr='C_NATIONKEY == N_NATIONKEY')
        NLJoin[5,cost=7655.00](expr='O_CUSTKEY == C_CUSTKEY')
          NLJoin[4,cost=7505.00](expr='L_ORDERKEY == O_ORDERKEY')
            TableScan[3,cost=6005.00](lineitem)
            TableScan[2,cost=1500.00](orders)
          TableScan[1,cost=150.00](customer)
        TableScan[0,cost=25.00](nation)
```

```
PushedDown Explain:
Project[9,cost=9180.00](projections={'C_ADDRESS': ('C_ADDRESS', 'char(40)'),
                                     'C_CUSTKEY': ('C_CUSTKEY', 'int'), 'N_NAME': ('N_NAME', 'char(25)'),
                                     'revenue': ('revenue', 'double'), 'C_NAME': ('C_NAME', 'char(25)'),
                                     'C_COMMENT': ('C_COMMENT', 'char(117)'), 'C_PHONE': ('C_PHONE', 'char(15)'),
                                     'C_ACCTBAL': ('C_ACCTBAL', 'double')})
  GroupBy[8,cost=9180.00](groupSchema=groupByKey[(C_CUSTKEY,int),(C_NAME,char(25)),(C_ACCTBAL,double),
                                                  (C_PHONE,char(15)),(N_NAME,char(25)),(C_ADDRESS,char(40)),
                                                  (C_COMMENT,char(117))],
                          aggSchema=revenue[(revenue,double)])
    NLJoin[6,cost=9180.00](expr='C_NATIONKEY == N_NATIONKEY')
      NLJoin[5,cost=9155.00](expr='O_CUSTKEY == C_CUSTKEY')
        NLJoin[4,cost=9005.00](expr='L_ORDERKEY == O_ORDERKEY')
          TableScan[3,cost=6005.00](lineitem)
          Select[10,cost=3000.00](predicate='(O_ORDERDATE >= 19931001)')
            Select[11,cost=3000.00](predicate='(O_ORDERDATE < 19940101)')
              TableScan[2,cost=1500.00](orders)
        TableScan[1,cost=150.00](customer)
      TableScan[0,cost=25.00](nation)
```

```
Join Explain:
Project[9,cost=9180.00](projections={'C_ADDRESS': ('C_ADDRESS', 'char(40)'),
                                     'C_CUSTKEY': ('C_CUSTKEY', 'int'), 'N_NAME': ('N_NAME', 'char(25)'),
                                     'revenue': ('revenue', 'double'), 'C_NAME': ('C_NAME', 'char(25)'),
                                     'C_COMMENT': ('C_COMMENT', 'char(117)'), 'C_PHONE': ('C_PHONE', 'char(15)'),
                                     'C_ACCTBAL': ('C_ACCTBAL', 'double')})
  GroupBy[8,cost=9180.00](groupSchema=groupByKey[(C_CUSTKEY,int),(C_NAME,char(25)),(C_ACCTBAL,double),
                                                  (C_PHONE,char(15)),(N_NAME,char(25)),(C_ADDRESS,char(40)),
                                                  (C_COMMENT,char(117))],
                          aggSchema=revenue[(revenue,double)])
    BNLJoin[12,cost=9180.00](expr='C_NATIONKEY == N_NATIONKEY')
      TableScan[0,cost=25.00](nation)
      NLJoin[5,cost=9155.00](expr='O_CUSTKEY == C_CUSTKEY')
        NLJoin[4,cost=9005.00](expr='L_ORDERKEY == O_ORDERKEY')
          TableScan[3,cost=6005.00](lineitem)
          Select[10,cost=3000.00](predicate='(O_ORDERDATE >= 19931001)')
            Select[11,cost=3000.00](predicate='(O_ORDERDATE < 19940101)')
              TableScan[2,cost=1500.00](orders)
        TableScan[1,cost=150.00](customer)
```

4.3 Discussion

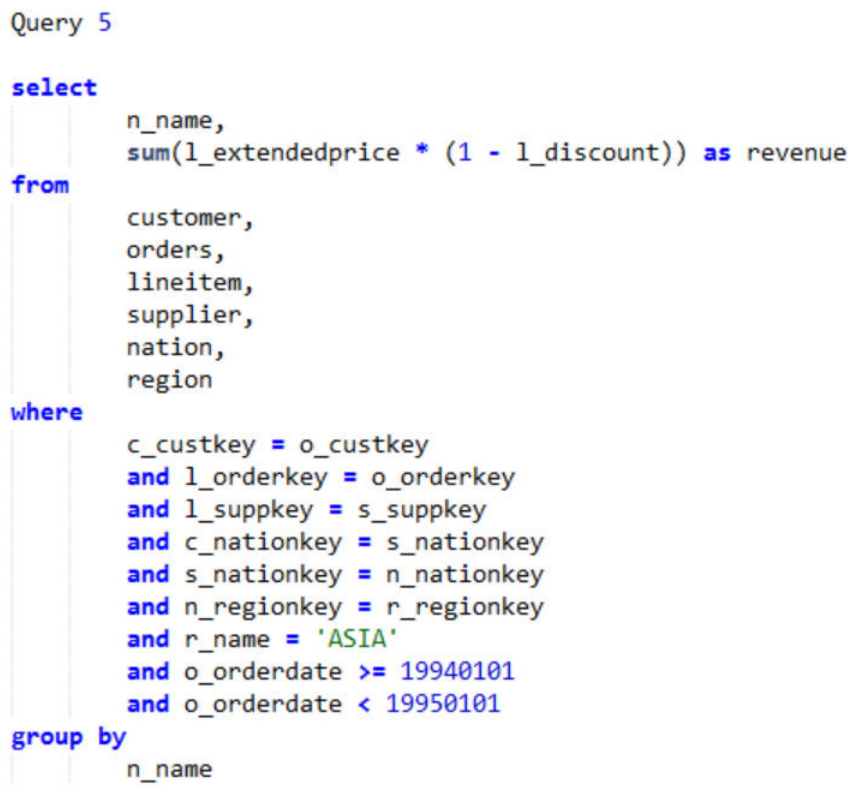
Again, similar to the previous queries, we can see that the select statements that were above the joins have been split up and placed as near the base tables as possible. Moreover, query4 involves more joins than does query2. Thus, we expect pushing down the selection operation to near the base tables before they are involved in any joins would be beneficial.

Furthermore, in this specific query, we do not have any select DNF predicates that involve fields from more than one base table. Thus, we were able to split up all the predicates into single CNF-decomposed predicates and attach each of the predicates right above its own base table. (As with query 3, the predicate lreturnflag= 'R' could not be processed and would produce error messages. We had to exclude this predicate in constructing the query.)

In this query, the join optimizer changed the first join to a BNLJoin.

5 Query 5

The given sql query:



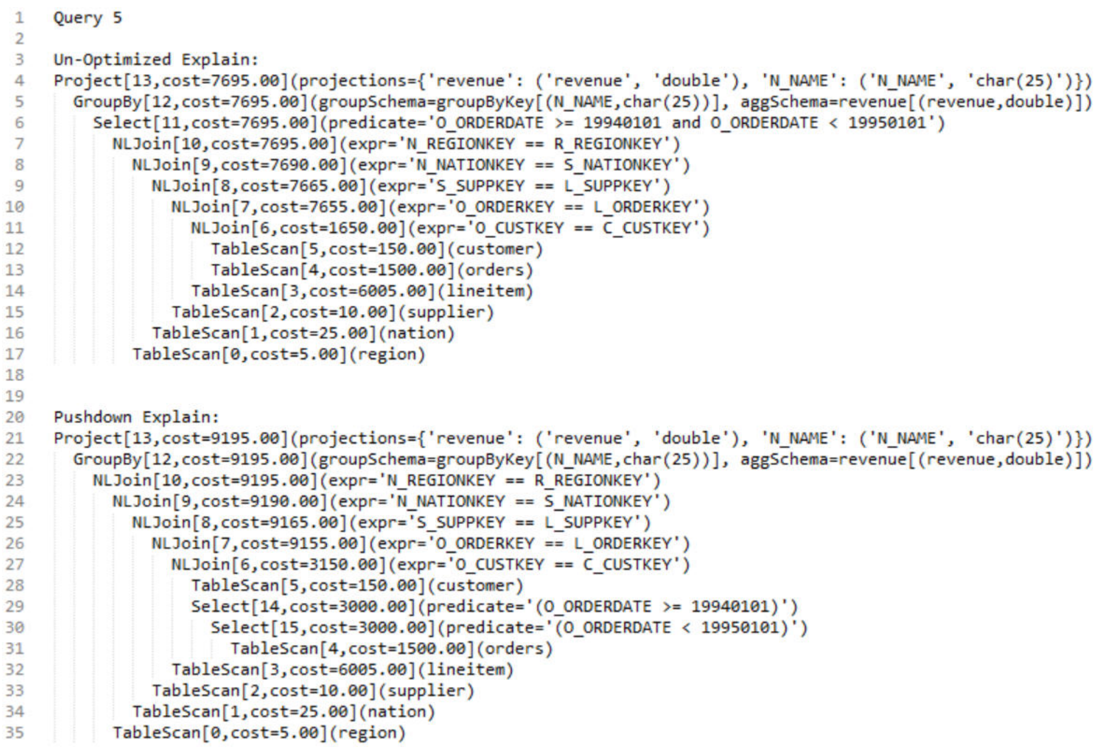
```
Query 5

select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'ASIA'
        and o_orderdate >= 19940101
        and o_orderdate < 19950101
group by
        n_name
```

5.1 Running Times

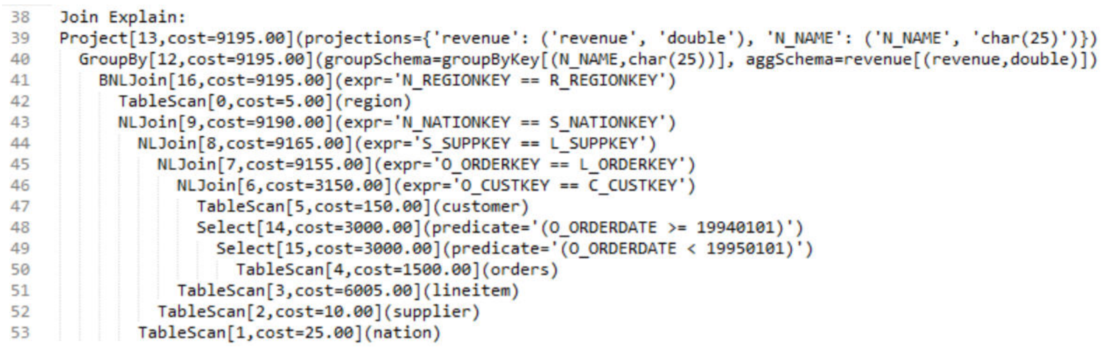
Non-optimized: x seconds
Optimized: x seconds

(Source code for each of the runs is included in our submission directory.)

5.2 Explain() Outputs



```
Query 5

Un-Optimized Explain:
Project[13,cost=7695.00](projections={'revenue': ('revenue', 'double'), 'N_NAME': ('N_NAME', 'char(25)')})
  GroupBy[12,cost=7695.00](groupSchema=groupByKey[(N_NAME,char(25))], aggSchema=revenue[(revenue,double)])
    Select[11,cost=7695.00](predicate='O_ORDERDATE >= 19940101 and O_ORDERDATE < 19950101')
      NLJoin[10,cost=7695.00](expr='N_REGIONKEY == R_REGIONKEY')
        NLJoin[9,cost=7690.00](expr='N_NATIONKEY == S_NATIONKEY')
          NLJoin[8,cost=7665.00](expr='S_SUPPKEY == L_SUPPKEY')
            NLJoin[7,cost=7655.00](expr='O_ORDERKEY == L_ORDERKEY')
              NLJoin[6,cost=1650.00](expr='O_CUSTKEY == C_CUSTKEY')
                TableScan[5,cost=150.00](customer)
                TableScan[4,cost=1500.00](orders)
              TableScan[3,cost=6005.00](lineitem)
            TableScan[2,cost=10.00](supplier)
          TableScan[1,cost=25.00](nation)
        TableScan[0,cost=5.00](region)


Pushdown Explain:
Project[13,cost=9195.00](projections={'revenue': ('revenue', 'double'), 'N_NAME': ('N_NAME', 'char(25)')})
  GroupBy[12,cost=9195.00](groupSchema=groupByKey[(N_NAME,char(25))], aggSchema=revenue[(revenue,double)])
    NLJoin[10,cost=9195.00](expr='N_REGIONKEY == R_REGIONKEY')
      NLJoin[9,cost=9190.00](expr='N_NATIONKEY == S_NATIONKEY')
        NLJoin[8,cost=9165.00](expr='S_SUPPKEY == L_SUPPKEY')
          NLJoin[7,cost=9155.00](expr='O_ORDERKEY == L_ORDERKEY')
            NLJoin[6,cost=3150.00](expr='O_CUSTKEY == C_CUSTKEY')
              TableScan[5,cost=150.00](customer)
              Select[14,cost=3000.00](predicate='(O_ORDERDATE >= 19940101)')
                Select[15,cost=3000.00](predicate='(O_ORDERDATE < 19950101)')
                  TableScan[4,cost=1500.00](orders)
            TableScan[3,cost=6005.00](lineitem)
          TableScan[2,cost=10.00](supplier)
        TableScan[1,cost=25.00](nation)
      TableScan[0,cost=5.00](region)
```



```
Join Explain:
Project[13,cost=9195.00](projections={'revenue': ('revenue', 'double'), 'N_NAME': ('N_NAME', 'char(25)')})
  GroupBy[12,cost=9195.00](groupSchema=groupByKey[(N_NAME,char(25))], aggSchema=revenue[(revenue,double)])
    BNLJoin[16,cost=9195.00](expr='N_REGIONKEY == R_REGIONKEY')
      TableScan[0,cost=5.00](region)
      NLJoin[9,cost=9190.00](expr='N_NATIONKEY == S_NATIONKEY')
        NLJoin[8,cost=9165.00](expr='S_SUPPKEY == L_SUPPKEY')
          NLJoin[7,cost=9155.00](expr='O_ORDERKEY == L_ORDERKEY')
            NLJoin[6,cost=3150.00](expr='O_CUSTKEY == C_CUSTKEY')
              TableScan[5,cost=150.00](customer)
              Select[14,cost=3000.00](predicate='(O_ORDERDATE >= 19940101)')
                Select[15,cost=3000.00](predicate='(O_ORDERDATE < 19950101)')
                  TableScan[4,cost=1500.00](orders)
            TableScan[3,cost=6005.00](lineitem)
          TableScan[2,cost=10.00](supplier)
        TableScan[1,cost=25.00](nation)
```

5.3 Discussion

Query 5 involved five joins over six tables. Thus, we expect the pushdown optimization and join order optimization to have greatest effects here. As with query 4, we can see that the original query has three select predicates (the rest of the clauses under the *where* clause being join experessions).

We can see that the CNF-decomposed predicates were attached to the base tables. (As with query 4, the predicate rname = ‘Asia’ could not be processed and would produce error messages. We had to exclude this predicate in constructing the query.) In this query, the join optimizer changed the first join to a BLNJoin.

Please see the next page for the general discussion.

6 General Discussion

Even though optimization is necessary in order to make queries run faster, we learned that for queries with small-enough sizes, the computation we go through in order to come up with the optimized way of running the query actually makes the query slower. In other words, for small enough queries, the overhead of optimization is greater than the original cost. Thus, we must first compute estimates of the query costs before we attempt to optimize the given query plan. If the cost estimate comes out to some value below our predetermined threshold, we might choose not to bother optimize the query and directly process the given query plan.

Second, we conclude that for large enough queries, pushing down the select predicates as far down as possible (as near the base tables as possible) actually makes the queries run much faster. This is because joins are often the most expensive parts of a query plan. If we know in advance that certain tuples will not be used during the entire query and thus will not affect the outcome, it would be beneficial for us to filter them out before anything else. Once the unnecessary tuples have been filtered out, we can perform less heavier joins and produce results faster than before. Depending on the selectivity of the select predicate, this way of optimization could make queries run multiple times faster than the unoptimized version of the query.

Again, this assignment was a great chance for us to confirm the theories on optimization we learned in both 600.315 and 600.316. Seeing first hand the queries run much faster and produce identical outputs was very interesting. This made us think and speculate more about what types of optimizations and tweaks commercial databases use to make their systems faster.

Thank you.