Database Systems Homework #2
Exercise 4: Experiments
600.316
Spring 2016

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

March 9, 2016

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

For each of the three queries, we performed four runs: 1) python sql query using block-nested-loops join (labeled as BNL in our plots), 2) python sql query using hash join (labeled $Hash$), 3) python block-nested-loops sql query with operator orders switched (labeled $Operator$), and 4) sqlite3 (labeled $Sqlite$). As a note, the y-axes in our plots signify the running time in seconds.

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 four techniques, and discuss the results. In the end, we give a general discussion about the four techniques listed above not limited to the three queries given. The source code written to run the queries are saved in a directory called *python_queries* under the main directory of our submission.

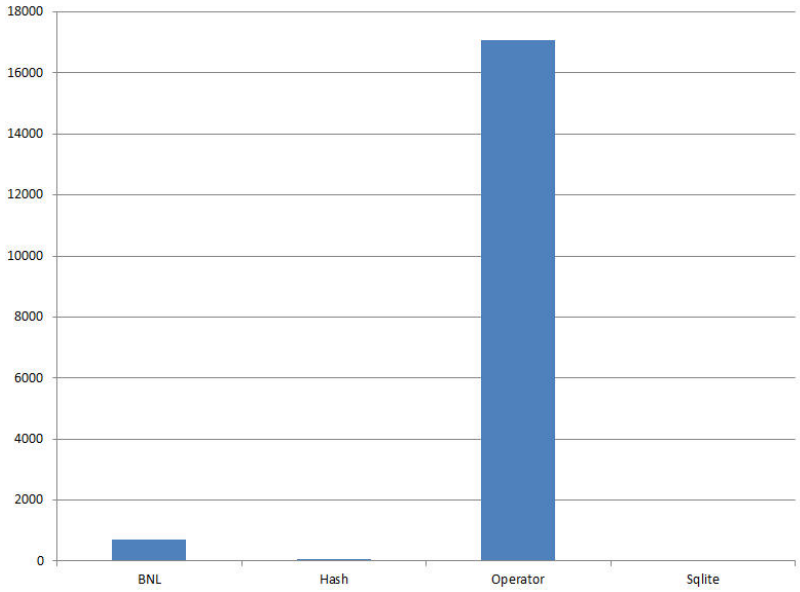
Please note that some output tuples may not be correct for some of the python implementations of the sql queries. We tried debugging the python query statements as much as we could, but the hash join techniques may not be ouputting correct results. We discuss the empirical results we obtained and their meaning nevertheless.

1 Query 1

The given sql query:

```
select p.p_name, s.s_name
from part p, supplier s, partsupp ps
where p.p_partkey = ps.ps_partkey
  and ps.ps_suppkey = s.s_suppkey
  and ps.ps_availqty = 1
union all
select p.p_name, s.s_name
from part p, supplier s, partsupp ps
where p.p_partkey = ps.ps_partkey
      and ps.ps_suppkey = s.s_suppkey
      and ps.ps_supplycost < 5;
```

1.1 Running Times



Block Nested Loops: 699.581 seconds
Hash join: 57.009 seconds
Different operator orders: 17078.928 seconds
Sqlite: 0.039 seconds
(Source code for each of the runs is included in our submission directory.)

1.2 Discussion

First, notice that the query that took the longest time to run was the $Operator$ one and the one that took shortest time to run was the $Sqlite$. The query using the hash join technique took less time than both BNL and $Operator$ which used the block-nested-loop join technique to join multiple tables.

The factor that created a great amount of difference in the running times between BNL and $Operator$ is the different ordering of select statements. In the BNL case, we placed the $.select()$ statement ahead of time to filter out certain tuples that were not going to be included in our result anyway. After filtering useless tuples from some of the tables, we then joined tables together and performed a union operation in the end.

In the $Operator$ case, we performed a selection at the last moment after having joined all the raw tables together and unioning the intermediate results. As expected, this method took much longer time than the BNL method mentioned above because this method had to join multiple tuples that had nothing to do with the end result. Without filtering out tuples that did not match the predicate clause ahead of time, we blindly joined the tables together in the first place and performed a select on the fully joined and unioned table just before returning the final result.

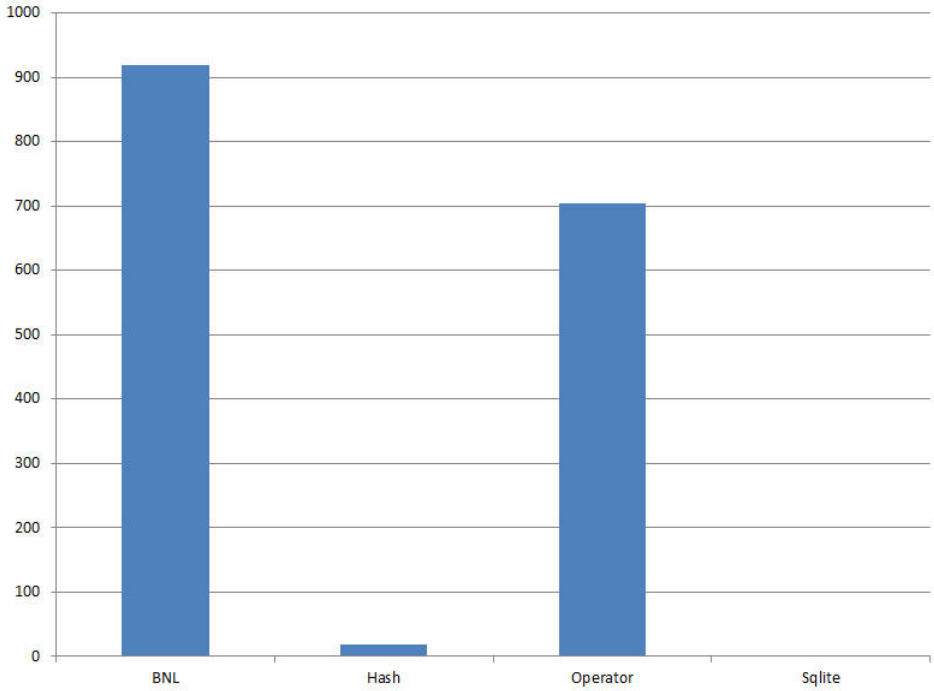
To our satisfaction, the results returned by the two runs (BNL and $Operator$) were identical, which proved that changing operator orders does not affect the correctness of the queries. Moreover, we were able to confirm that performing selections ahead of time saved a lot of time in actually running the queries.

2 Query 2

The given sql query:

```
select part.p_name, count(*) as count
from part, lineitem
where part.p_partkey = lineitem.l_partkey and lineitem.l_returnflag = 'R'
group by part.p_name;
```

2.1 Running Times



Block Nested Loops: 918.615 seconds
Hash join: 17.882 seconds
Different operator orders: 704.684 seconds
Sqlite: 0.068 seconds

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

2.2 Discussion

The above is a plot showing the running time of the four different tests we ran. Notice that the instance that the $Sqlite$ method took the shortest time to run while the BNL method took the longest time. Moreover, the query that used hash joins as the join technique took less time than the two queries *BNL* and *Operator* that used block-nested-loop joins.

The results we obtained for this query run were unexpected. The expected results were for the $Operator$ run to take longer time to produce the outputs. The reason was that the $Operator$ case pushes back its **where** clause until the end. Thus, in this case, the database engine performs extraneous joins that migt not end up in the final result. In the BNL case, we perform a **where** selection ahead of time to filter out unnecessary tuples before performing a join.

So the only difference between the BNL method and the *Operator* method was that we changed the position of the **where** clause. Contrary to our expectations, the BNL method took longer time. We were expecting this one to take shorter time because this one does a selection with the **where** clause before even joining while the *Operator* method joins all the tuples first and them filters them out.

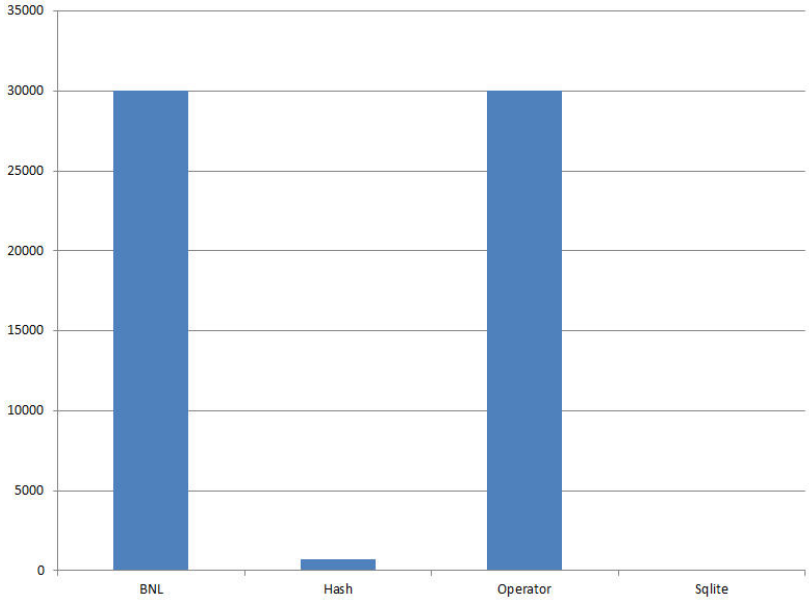
However, the results that the two queries presented were identical, which proved that the correctness is not affected by changing the order of the operators.

3 Query 3

The given sql query:

```
with temp as (
   select n.n_name as nation, p.p_name as part, sum(l.l_quantity) as num
   from customer c, nation n, orders o, lineitem l, part p
   where c.c_nationkey = n.n_nationkey
     and c.c_custkey = o.o_custkey
     and o.o_orderkey = l.l_orderkey
     and l.l_partkey = p.p_partkey
   group by n.n_name, p.p_name
)
select nation, max(num)
from temp
group by nation;
```

3.1 Running Times



Block Nested Loops: x seconds (explanation on the next page)
Hash join: 741.81 seconds
Different operator orders: x seconds (explanation on the next page)
Sqlite: 0.083 seconds
(Source code for each of the runs is included in our submission directory.)

3.2 Discussion

First of all, we had to terminate running the BNL and $Operator$ experiment runs because they took too long to run. (I started the run just before I went to bed and woke up the next morning and found the two still running.) Although this is an unusual situation that might bring up doubts about our BNL implementation, the results we obtained for BNL were correct in the previous cases. This is why the elapsed time for BNL and $Operator$ above are noted as x and not a real number.

However, we were able to conclude even in these circumstances that joins using the BNL technique takes much longer time than $Hash$ or $Sqlite$. In this case, we were not able to see which one of the two out of BNL and $Operator$ took longer time to execute. Since this query did not have a **where** clause that filtered out tuples from tables, we tested on changing the order in which we joined tables. The educated guess is that the execution plan with joins that involved greater reductions of tuples early on would run faster than the other.

Again, we were able to confirm with query 3 that the $Sqlite$ execution took the shortest time to run and the $Hash$ execution took less time than both BNL and $Operator$. On the next page, we give a general discussion about the four techniques.

4 General Discussion

We conclude that the queries run with hash joins run much faster than the ones that use block nested loop joins. Moreover, filtering out tuples before joining tables tend to give better performances. Thus, we confirm the fact stated during lecture that doing selections early on give better performance.

Evidently, the $Hash$ technique runs much faster than BNL because it allows for the engine to compare only the tuples that hashed to the same bucket. If two tuples did not hash to the same bucket (using a set of attributes to hash), they have absolutely no chances of matching up as a join. However, the BNL (or regular NL) technique naively iterates through all the tuples that exist in each relation and tries to match tuples.

Moreover, we conclude that the order in which we place operators matters in query execution. Generally, placing **where** clauses (selection in relational algebra) early on will save execution time as it will filter out tuples that don't belong in the result output ahead of time. Moreover, performing joins that will give the most dramatic cut of tuples should be performed early on. The reason is that joins are usually the most expensive operations in query execution, and if there are less tuples in the relation to join, it will take less time to execute the query. And if there are more than 1 joins in the query, it is best to perform joins that give the greatest amount of reductions early on so that the intermediate tables we use to join with other tables are as small as possible.

Finally, after having run twelve iterations of the three queries, we concluded that Sqlite has done a fantastic job in making queries run fast. Even though we acknowledge the fact that is not the fastest database engine out there, it was nevertheless much faster than our own python implementation of a simple database engine. Moreover, this experiment section was a great chance to get a first hand experience on textbook material taught in 600.315–it was cool to test different query execution plans and observe the different run times we get from them.

Thank you.