Jlanders - S3163776 – Joseph Landers – Database Systems Assignment #2 – Semester 1 2009

<u>Nested Loops Join</u>

1.)

	characters	join	guilds	NestedLoop	
	cg	nested			
		100	200	500	1000
	1	3090	1820	1140	930
	2	3100	1830	1140	930
	3	3080	1820	1150	910
	4	3080	1810	1140	900
	5	3070	1820	1130	930
	6	3080	1830	1130	910
	7	3090	1780	1150	920
	8	3110	1820	1140	910
	9	3090	1810	1170	900
	10	3080	1810	1110	910
	11	3110	1800	1140	910
	12	3060	1820	1150	920
	13	3100	1830	1140	890
	14	3270	1810	1140	920
	15	3100	1820	1140	900
	16	3140	1810	1140	910
	17	3080	1800	1160	910
	18	3090	1810	1130	900
	19	3080	1830	1160	920
	20	3120	1840	1150	910
Average		3101	1816	1142.5	912

2.)

	gc	nested			
		100	200	500	1000
	1	5330	2910	1560	1070
	2	5310	2920	1560	1060
	3	5280	2940	1540	1070
	4	5290	2980	1580	1070
	5	5350	2940	1600	1060
	6	5310	2900	1610	1060
	7	5280	2940	1580	1060
	8	5420	2950	1590	1070
	9	5290	2940	1570	1060
	10	5320	2950	1560	1060
	11	5290	2940	1540	1070
	12	5330	2930	1560	1060
	13	5330	2940	1550	1080
	14	5240	2950	1560	1060
	15	5300	2980	1570	1070
	16	5350	2960	1540	1050
	17	5330	2960	1570	1070
	18	5340	2970	1550	1080
	19	5380	2960	1560	1050
	20	5460	3030	1560	1060
Average		5326.5	2949.5	1565.5	1064.5

3.) Discuss your results. Is the data as you expected? What number of buffers would you recommend for the join process? Which order of file parameters works best?
In answering these questions, you should demonstrate understanding of the theoretical properties of the nested loop join. Restrict your answer to a maximum of half a page.

The general trend was expected – as the outer relations buffer size increases the number of disk reads should reduce as we have to iterate over the inner relation less times.

Given the results, I would recommend a buffer size between 500 and 1000 as this is where the benefit of the buffer appears to tail off.

It appears that characters join guilds works best.

Here are the expected costs for the two joins:

characters	Guilds		
Outer	Inner	Buffer	Cost
10000	4000	10	4010000
10000	4000	100	410000
10000	4000	200	210000
10000	4000	500	90000
10000	4000	1000	50000

Guilds	characters		
Outer	Inner	Buffer	Cost
4000	10000	10	4004000
4000	10000	100	404000
4000	10000	200	204000
4000	10000	500	84000
4000	10000	1000	44000

The performance of the two based on the costs should have been similar.

The guilds join characters should have a faster hashtable search because we don't have to search every record in buffer due to the guilds(many) to character(one) relationship – we can stop when we find one guild in the outer buffer. This appears to have a low cost so has not had much impact on performance.

I believe the performance is different between the two joins because the inner file record size is important to performance.

The guilds file has smaller records which results in less disk reads.
The characters file has bigger records which results in more disk reads.

Of course, we can optimise our code to block read and reduce this difference.

4.) In up to ten sentences, explain how you implemented your hash table. As part of your discussion, describe your hash function, and explain why it is suitable for hashing this type of data.

The hash table implemented was a linear hash table.

The hash table was made of pointers to records.

Once the guildID was hashed we could search the hash table at that slot.

If the pointer to the record was NULL then we had no match as there was no record (this case should never occur as we should have 100% occupancy). If the pointer was not NULL then we would check the record for a matching guildID.

For GUILDS JOIN CHARACTERS the search would terminate on a NULL record or matching. For a non-matching record the search would continue until a match was found or we had read the entire hash table.

For CHARACTERS JOIN GUILDS the search would continue until the whole hash table was processed – hence non-optimal use of the hash table as we search all records

The hash function used:

```
int hashfunction (int id, int hashtable_size)
{
   return (((438439 * id) + 34723753) % 376307) % hashtable_size;
}
```

This is a good function as:

1.) It provides a good spread of data in a semi-random distribution across the hashtable.
2.) It uses ints and a simple formula therefore will be relatively quick as compared to strings as mathematical functions can be processed in less operations on a CPU.

Jlanders - S3163776 – Joseph Landers – Database Systems Assignment #2 – Semester 1 2009

HashJoin

1.)

	characters	join	guilds	HashJoin
	100	200	500	1000
1	530	530	560	620
2	510	520	590	620
3	510	530	560	680
4	510	540	560	620
5	540	520	600	630
6	510	520	560	660
7	510	520	560	620
8	530	520	570	630
9	520	530	560	660
10	520	530	560	620
11	510	520	570	640
12	510	520	560	640
13	520	520	560	640
14	530	520	570	650
15	510	530	570	640
16	510	520	560	630
17	520	520	560	630
18	530	520	570	630
19	500	540	570	630
20	510	530	570	640
Average	517	525	567	636.5

2.)

	guilds	join	characters	HashJoin
	100	200	500	1000
1	510	520	580	640
2	520	520	550	650
3	510	530	560	690
4	520	520	570	640
5	510	530	560	650
6	520	530	560	640
7	510	510	580	660
8	510	520	560	620
9	520	550	570	630
10	520	520	570	640
11	510	540	570	630
12	510	530	560	630
13	510	510	610	630
14	520	520	570	640
15	510	510	570	680
16	510	530	590	660
17	500	560	560	650
18	530	550	560	620
19	530	530	580	640
20	520	540	570	610
Average	515	528.5	570	642.5

3. Discuss your results. Is the data as you expected? What number of buffers would you recommend for the join process? Which order of file parameters works best?
In answering these questions, you should demonstrate understanding of the theoretical properties of the hash join. Restrict your answer to a maximum of half a page.

The data wasn't what I expected. As the partitions increased I had assumed the search time would be lower as the number of search operations decreases but instead it increases. I had not factored in the extra disk cost of creating, reading and writing these partitions of data which have to sit on separate disk sectors.

In fact I did some further testing and found that performance is at a maximum around 10 partitions and decreases from there onwards. Therefore I would recommend 10 partitions for the join process.

The performance is relatively similar for either order of files in the above examples, if we analyse disk costs we can see that the order is not important as the costs are the same. Furthermore each file is read the same number of times no matter the order of parameters.

HashJoin

Outer	Inner	Cost
4000	10000	42000

HashJoin

Outer	Inner	Cost
10000	4000	42000

4. In up to ten sentences, describe your two hash functions, and explain your choice. Remember to cite any sources that you use, if appropriate.

The first hash function I used to partition the data has simple hash of:

```
id % hashtable_size;
```

For the second hash function I used the hash function explained earlier in the NestedLoop answers:

```
(((438439 * id) + 34723753) % 376307) % hashtable_size;
```

The hash functions should both provide random distributions and non-equivalent hash mappings when used together. They should also both be relatively fast as they are both simple formulas with relatively few CPU operations.

I did do some research and after testing a couple of hash functions against the basic ones given above I noted that the performance didn't noticeably improve. This is due to the relative costs of disk accesses versus memory accesses.

<u>Discussion</u>

1. In a few sentences, summarise your overall results. Which algorithm works best for joining the files?

HashJoin works best with our data set as it reduces the number of disk reads compared to Nested Loop Join.

Each data set is read twice in total and written once. Partitioning consists of a read of each data file and a write to the partitions and matching partitions consists of reading both sets of partitions once.

Cost: 3 * (M + N)

Where M = Number of records in first file (Outer File)
N = Number of records in second file (Inner File)

NestedLoop however can read the outer file once and the inner file multiple times resulting in higher disk costs.

Cost: M + (N * (M/B))

Where M = Number of records in first file (Outer File)
N = Number of records in second file (Inner File)
B = Outer Buffer Size

2. Would the algorithm that you found to work best on the supplied data files work as well for very large files? If not, which algorithm would you expect to work best and why?

For two very large files Hash Join will work just as well and will out perform Nested Loop Join.

NestedLoop

Outer	Inner	Buffer	Cost
1000000	1000000	10	###
1000000	1000000	100	10001000000
1000000	1000000	200	5001000000
1000000	1000000	500	2001000000
1000000	1000000	1000	1001000000

HashJoin

Outer	Inner	Cost
1000000	1000000	6000000

PTO

3. Would the relative performance of your algorithms be the same when joining other small data files of different relative sizes? Why? In your answer, give examples of cases where performance is likely to be the same or different. You could run additional experiments to verify your findings.

The performance of the algorithms for different data file sizes is given below:

NestedLoop

Outer	Inner	Buffer	Cost
100	1000000	10	10000100
100	1000000	100	1000100
100	1000000	200	1000100
100	1000000	500	1000100
100	1000000	1000	1000100

HashJoin

Outer	Inner	Cost
100	1000000	3000300

NestedLoop

Outer	Inner	Buffer	Cost
1000000	1000000	10	###
1000000	1000000	100	10001000000
1000000	1000000	200	5001000000
1000000	1000000	500	2001000000
1000000	1000000	1000	1001000000

HashJoin

Outer	Inner	Cost
1000000	1000000	6000000

NestedLoop

Outer	Inner	Buffer	Cost
1000000	100	10	11000000
1000000	100	100	2000000
1000000	100	200	1500000
1000000	100	500	1200000
1000000	100	1000	1100000

HashJoin

Outer	Inner	Cost
1000000	100	3000300

NestedLoop

Outer	Inner	Buffer	Cost
100	100	10	1100
100	100	100	200
100	100	200	200
100	100	500	200
100	100	1000	200

HashJoin

Outer	Inner	Cost
100	100	600

PTO

3.) Continued

For two large data files Hash Join will work better than Nested Loop Join.

For the other cases the Nested Loop Join can out perform the Hash Join given certain parameters.

If we analyse the cost formulas:

Hash Join Cost = 3 * (M + N)
Nested Loop Join Cost = M + (N * (M/B))

M = inner file record count
N = outer file record count
B = outer file buffer size

For Nested Loop Join we could reduce N and/or M to a low value or increase B to a higher value to get better costs than Hash Join.

END OF ASSIGNMENT