PDD - Assignment

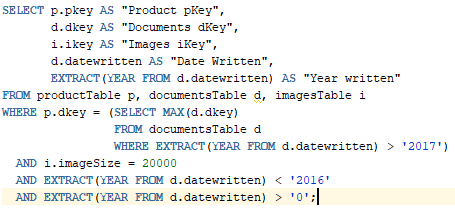
Nikkolas Diehl – 16945274  
Yash Chaudhari – 1384865

PDD assignment 2

**PART A:**

The goal of this assignment is to optimize the querying of a database. This database is made up of 3 tables, each with a million rows.

First, we need to create a complex join query with a sub-query shown below:

**Baseline Query:**

**Tables:**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Baseline Query Trace Files (2):**

We ran two baseline queries two get an average and work from there. We did this because a single query with a very short time might be an outlier.

|  |  |
| --- | --- |
| Trace One: | Trace Two: |
|  |  |

As you can see from the traces above, the times between the baseline queries varied. This is probably due to the speed of the CPU differing slightly because of other applications running at the time, or because the database is running slowly at specific or random times.

* Total count in both baselines is exactly 49,068
* Average CPU time across both queries is 3.585 seconds.
* Average elapsed time across both queries is 4.155 seconds
* Average disk access for both queries is 14,624.5
* Query count for both baselines is exactly 13,092
* Current count for both baselines is exactly 2
* Rows count for both baselines is exactly 735,965

**PART B:**

**Indexing:**

We chose to use three different types of indexing. The first was an index on the one column of each table (basically the primary keys of each table but they weren’t set up as primary keys). The second indexing we did was an index on every single column in each of the three tables. And the third and final indexing we did was on productTable on each of the main keys (pkey, dkey, ikey) and then on the last two tables, there was indexing on their main keys (dkey, and ikey).

The results are shown below:

|  |  |
| --- | --- |
| Index type one:   * Count: 49,068 * CPU time: 3.45 * Elapsed Time: 3.57 * Disk Accesses: 11,392 * Query count: 13,092 * Current: 2 * Rows: 735,965 * Count is 0% better than baseline. * CPU time is 3% better than baseline. * Elapsed time is 16% better than baseline. * Disk accesses is 28% better than baseline. * Query count is 0% better than baseline. * Current is 0% better than baseline. * Rows is 0% better than baseline |  |
| Index type two:   * Count: 51,288 * CPU time: 5.57 * Elapsed Time: 6.01 * Disk Accesses: 24,446 * Query count: 22,610 * Current: 4 * Rows: 769,206 * Count is 4% worse than baseline * CPU time is 55% worse than baseline * Elapsed time is 44% worse than baseline * Disk accesses is 67% worse than baseline * Query count is 72% worse than baseline * Current is 100% worse than baseline * Rows is 4% worse than baseline |  |
| Index type three:   * Count: 98,136 * CPU time: 6.85 * Elapsed Time: 7.06 * Disk Accesses: 18,234 * Query count: 26,184 * Current: 4 * Rows: 1,471,930 * Count is 100% worse than baseline * CPU time is 91% worse than baseline * Elapsed time is 69% worse than baseline * Disk accesses is 24% worse than baseline * Query count is 100% worse than baseline * Current is 100% worse than baseline * Rows is 100% worse than baseline |  |

The indexing results prove than indexing on multiple rows when a large select is done is highly inefficient and the only efficient way to do indexing is when indexing is done on individual rows. The individual row indexing shows an average of 6.7% better stats than the baseline queries.

**Clustering:**

We chose four different types of clustering. All the chosen types of clustering used a HASHKEY of 1999. There are a million rows though, but we could not wait the time it would have taken for a HASHKEY of 999983 which would have taken nearly 5 hours to complete.  
The first type of clustering we used was on all the main keys of each table (pkey, dkey, ikey). The second type of clustering was on every single column in each table, and the third was a cluster on the productTable using the pKey, dkey and ikey, whilst the other two tables had a cluster on only their main keys (dkey, ikey) similar to the indexing we did. The fourth and final type of clustering we did was a hybrid of the indexes and clustering. Both the indexing and the clustering where only put on the main keys for each table (pkey, dkey and ikey).

|  |  |
| --- | --- |
| Clustering type one:   * Count: 49,068 * CPU time: 3.64 * Elapsed Time: 3.69 * Disk Accesses: 15,024 * Query count: 16,037 * Current: 2 * Rows: 735,965 * Count is 0% better than baseline * CPU time is 1% worse than baseline * Elapsed time is 14% better than baseline * Disk accesses is 4% worse than baseline * Query count is 22% worse than baseline * Current is 0% better than baseline * Rows is 0% better than baseline |  |
| Clustering type two:   * Count: 49,068 * CPU time: 3.56 * Elapsed Time: 3.72 * Disk Accesses: 14,772 * Query count: 16,037 * Current: 2 * Rows: 735,965 * Count is 0% better than baseline * CPU time is 0.7% better than baseline * Elapsed time is 11% better than baseline * Disk accesses is 1% worse than baseline * Query count is 22% worse than baseline * Current is 0% better than baseline * Rows is 0% better than baseline |  |
| Clustering type three:   * Count: 49,068 * CPU time: 3.55 * Elapsed Time: 3.77 * Disk Accesses: 13,970 * Query count: 16,037 * Current: 2 * Rows: 735,965 * Count is 0% better than baseline * CPU time is 0.9% better than baseline * Elapsed time is 10% better than baseline * Disk accesses is 4.6% better than baseline * Query count is 22% worse than baseline * Current is 0% better than baseline * Rows is 0% better than baseline |  |
| Clustering type four:   * Count: 49,068 * CPU time: 3.84 * Elapsed Time: 3.95 * Disk Accesses: 15,049 * Query count: 16,037 * Current: 2 * Rows: 735,965 * Count is 0% better than baseline * CPU time is 7% worse than baseline * Elapsed time is 5% better than baseline * Disk accesses is 2.9% worse than baseline * Query count is 22% worse than baseline * Current is 0% better than baseline * Rows is 0% better than baseline |  |

As you can see from the results above, the trace files prove that using clustering on a large database is mostly inefficient unless you doing clustering on only the main keys for each table as done in the first type of clustering. The individual column clustering is, on average, 4.4% better than the baseline queries.

From all this data, we can see that on huge databases such as mine with millions of rows per column in a table, single column indexing is probably the fastest method.

**PART C:**

In order to the find the fastest query, we will use the fastest clustering/indexing method and then alter the query to get the fastest result.

The fastest method of clustering/indexing was the first form of indexing we did where all the main keys for each table where indexed. It was better than the baseline by an average of 15.6%

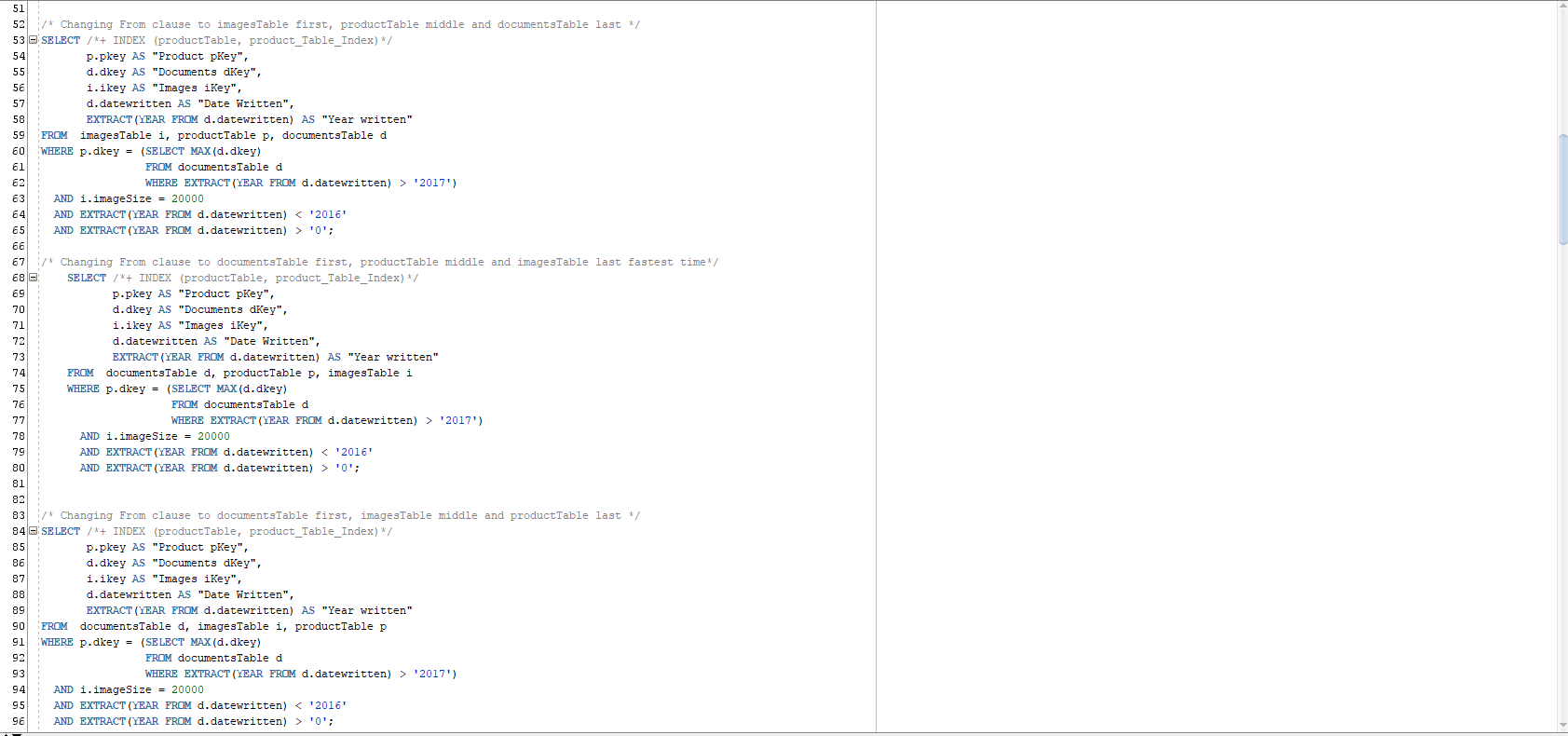
The test will be done by using an unstructured data and then selecting specific rows yet changing the order of the 3 tables within each different statements “from” clause. The fastest method was single column INDEXING we will use single column hints of INDEX to perform the test. The reason being hints such as INDEX\_COMBINE use bitmap indexing but we have unique data each row and INDEX\_JOIN requires a sufficiently small number of indexes must exist that contain all the columns required to resolve the query.

The test will be performed on each Index of the data and the “from” clause will be adjusted to try all variation to find the fastest method within each index. The fastest of the Indexes will then be compared and further utilized by parallel query optimization to reach the best possible optimization.

Below is how the test has been set up

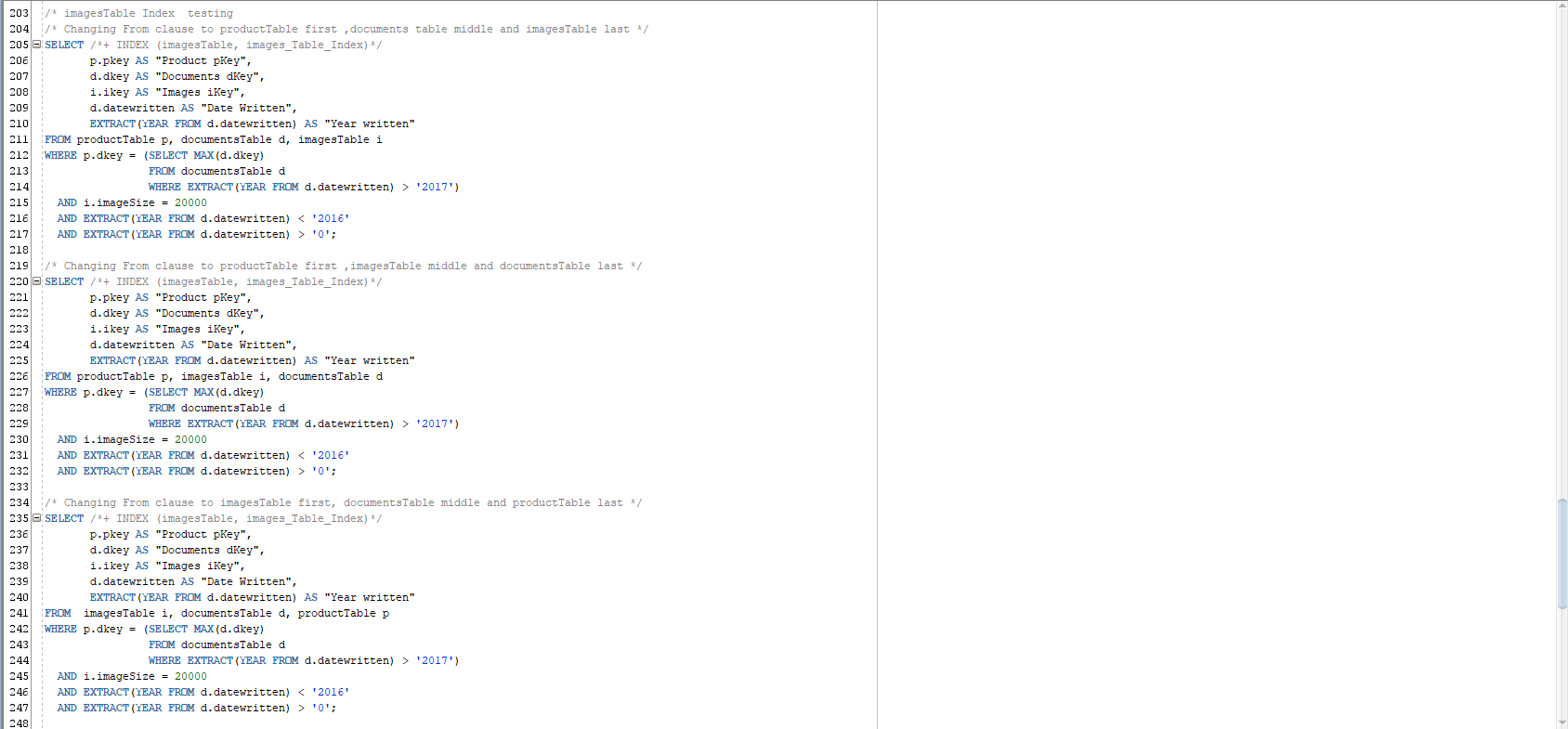
1-

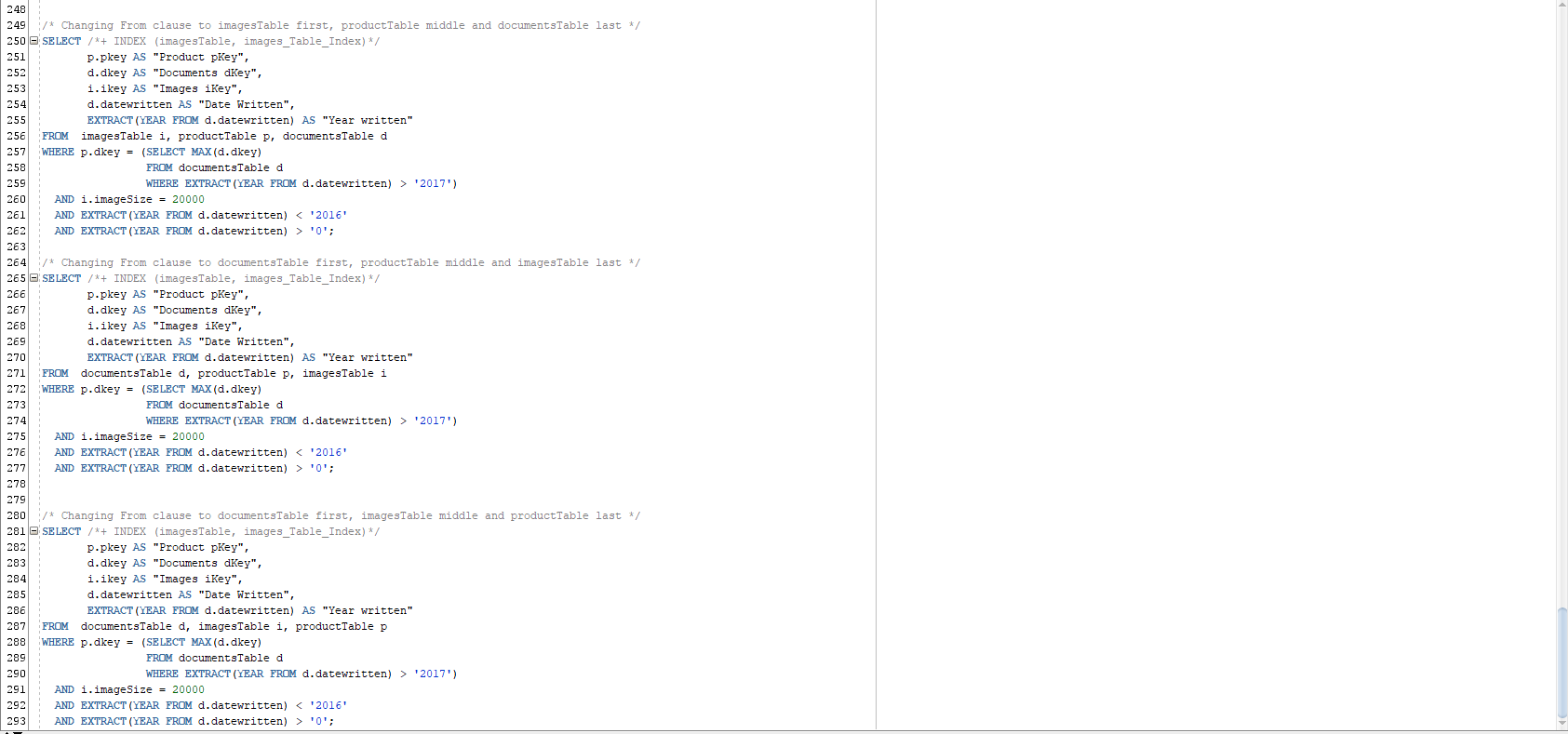


2-

3-

4-

5-

6-

Outcomes:



Now we take the fastest outcome and further utilize parallel query option. We first altered the tables by using:

****

Then we altered the fastest index which was the product\_Table\_Index:

****

We ran two different queries one with just parallelism and one with both index and parallelism below is the result to how quickly it was fetched:



We can see from this that by utilizing just indexing Hints we halved the time but then utilizing parallelism we further got better result and reduced the time by about 70%.

**PART D:**

The Oracle’s Query optimization is tasked to run SQL statements, by using the optimization query the goal was to yield most efficient usage of the hardware.

Utilizing a 3 million volume of unordered data firstly we used the Access path of indexing the data. The indexing results prove than indexing on multiple rows when a large select is done is highly inefficient and the only efficient way to do indexing is when indexing is done on individual rows. The individual row indexing shows an average of 6.7% better stats than the baseline queries. Next, we use clustering to see if it provides any benefit to the data. We find out that clustering on a large database is mostly inefficient unless you are doing clustering on only the main keys for each table. The individual column clustering is, on average, 4.4% better than the baseline queries. We took the best technique being single column Indexing then applied indexing Hints which helped reduced the time of the data retrieval by 50% compared to having an index but no Indexing Hints. Furthermore, the combination of both Parallelism and Indexing is where we see another 70% drop in retrieval of data time from a base of 3.45 seconds with Indexing to a 0.459 Seconds. This shows us the benefit of Query optimization Oracle provides especially for a large database when multiple computers are accessing data either locally or remotely shorter retrieval times are ideal. Oracle provides many different option depending such as Clusters, Indexes, Hints, Parallelism for any database you are creating a careful consideration needs to go into how to create it and which Oracle tools to use to get the best optimization out of your system.