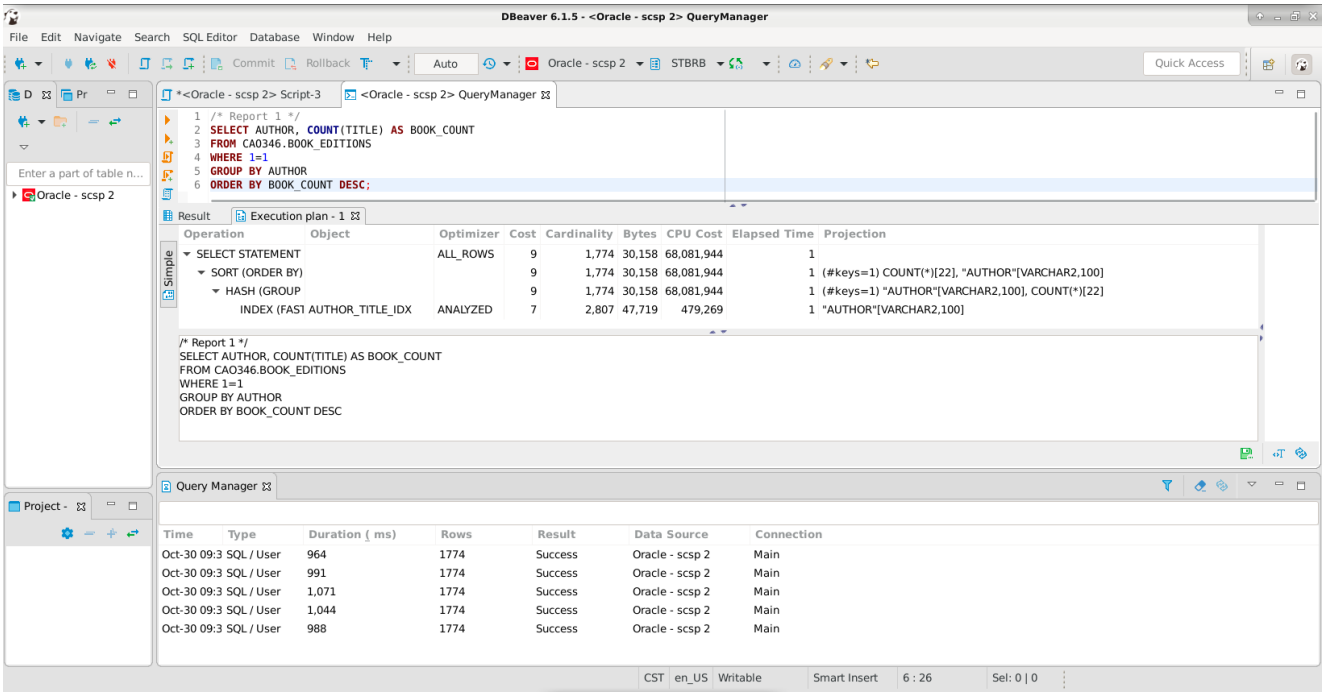
CS5300 - Programming Project 3 - Nick@Nite

Optimized Results using Indexing (Primary Keys)

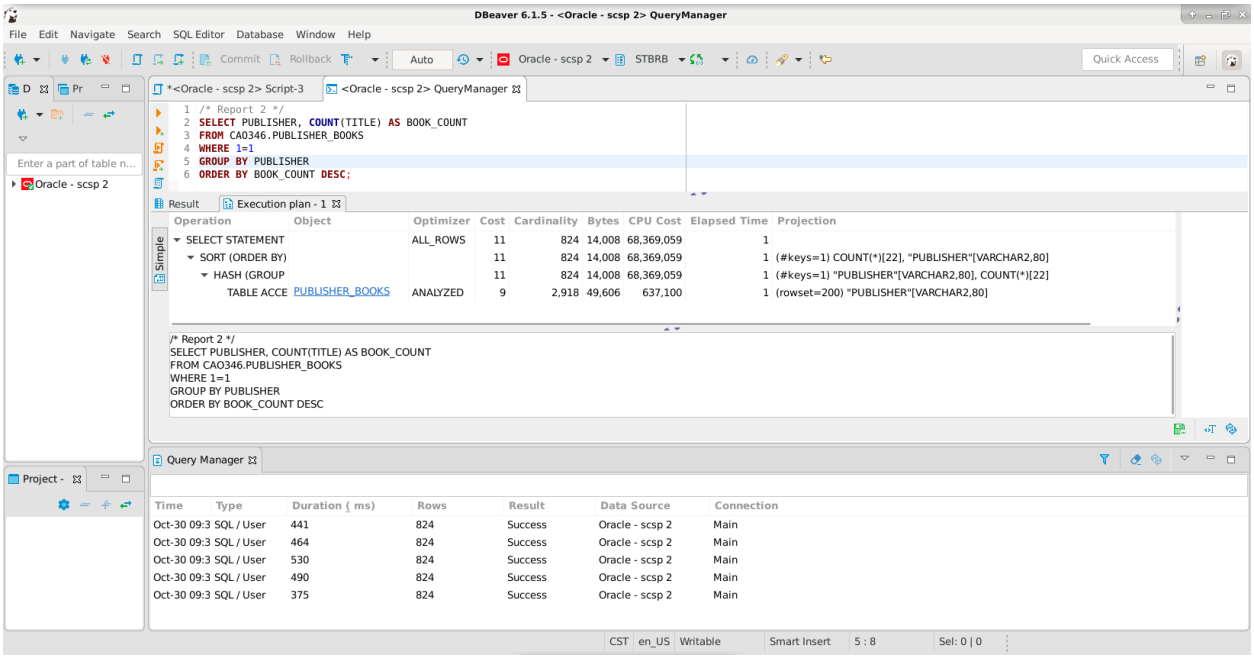
Report 1: A report that lists all the Authors and the number of books they have.



Trial #	Time (ms)
1	964
2	991
3	1071
4	1044
5	988
Average:	1011.6

During the trial runs of report 1, we retrieved a total of 1774 records listing the Authors with the total number of books they wrote. After a total of five trial runs, we calculated an average run time of 1011.6 milliseconds, or 1.01 seconds. The results signify that the SELECT and ORDER BY operations have the highest cost, requiring approximately 30k bytes to process. The second highest is the Hash operation, as it has a reduced cost due to our indexed primary key inside of the Author relation.

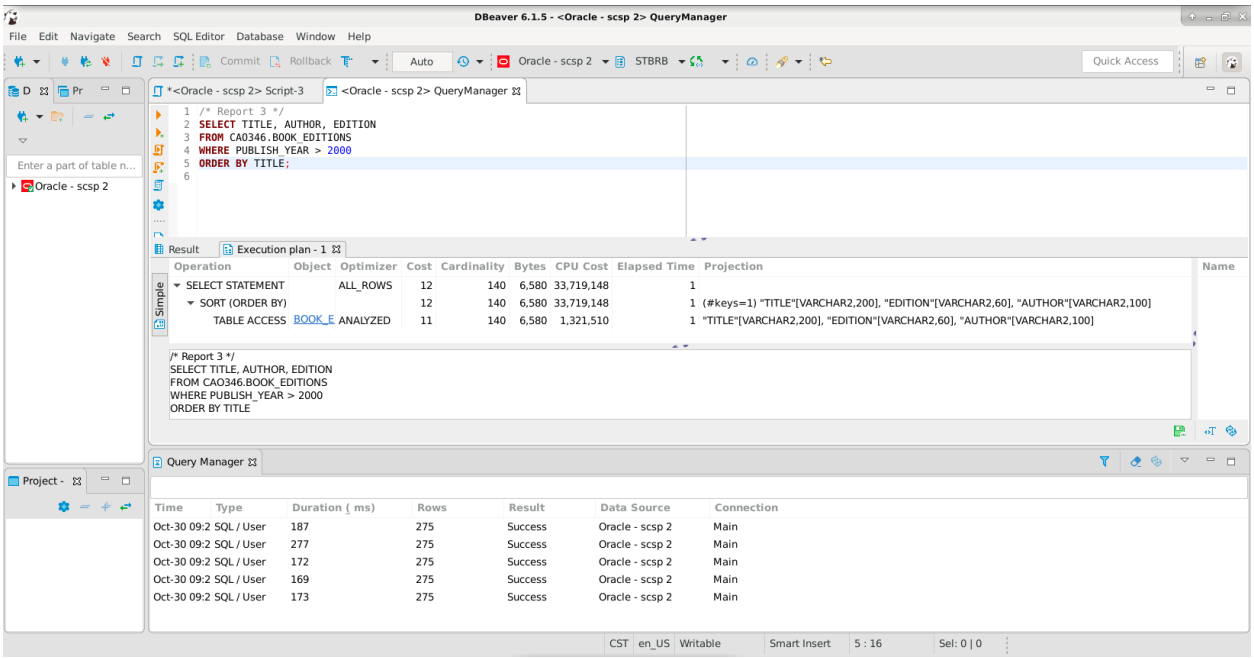
Report 2: A report that lists all the publishing companies and the number of books they have.



Trial #	Time (ms)
1	441
2	464
3	530
4	490
5	375
Average:	460

During the trial runs of report 2, we retrieved a total of 824 records listing publishing companies and their books in an average time of 460 milliseconds, or approximately half a second. The results of these trials were fairly similar to that of report 1, signifying that the SELECT and ORDER BY operations have the highest cost. The next highest was the hash operation, as it has a reduced cost to access the PUBLISHER_BOOK relation due to our indexed primary key.

Report 3: A report that lists books made after the year 2000.

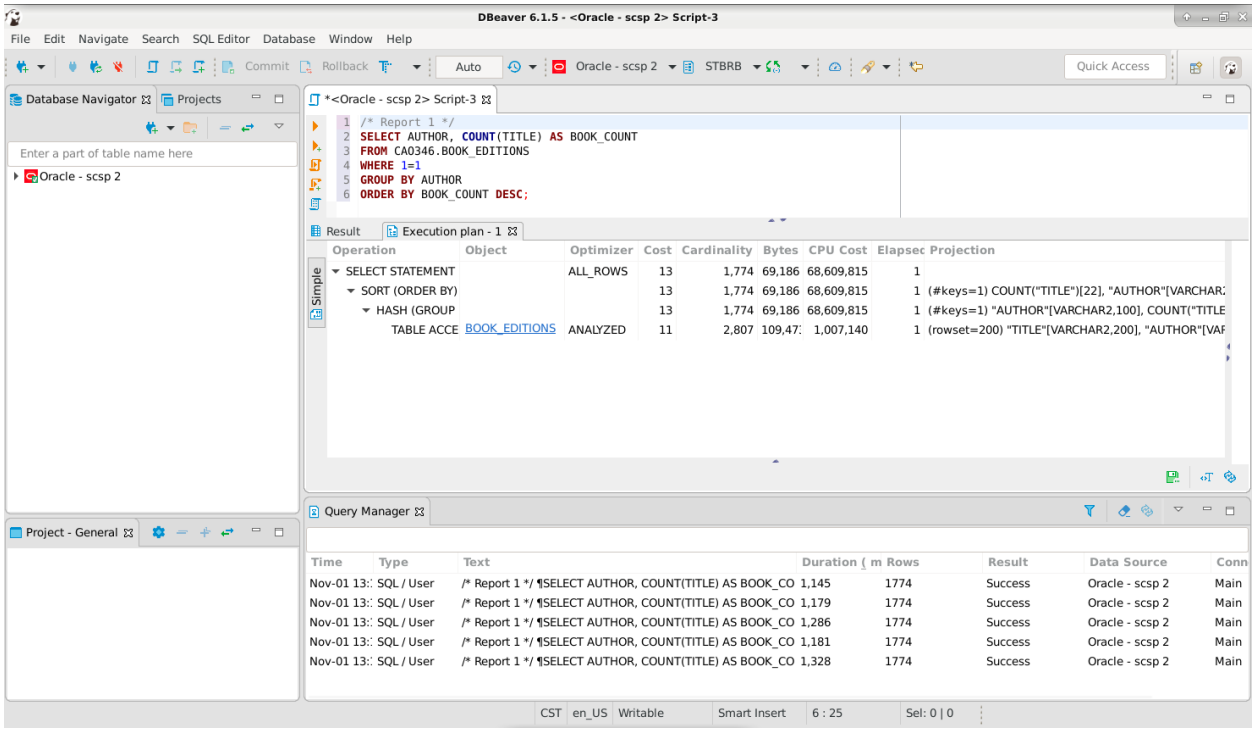


Trial #	Time (ms)
1	187
2	277
3	172
4	169
5	173
Average:	195.6

During the trial runs of report 3, we retrieved a total of 275 records listing the books with published in 2000 or later. The average retrieval time for these records was 195.6 milliseconds. The results of these trials showed the same pattern as the first two reports, being that the SELECT and ORDER BY operations had the greatest cost impact while the HASH operation was minimized by our indexed primary key.

Unoptimized Results (Removed Primary and Foreign Keys from all relations)

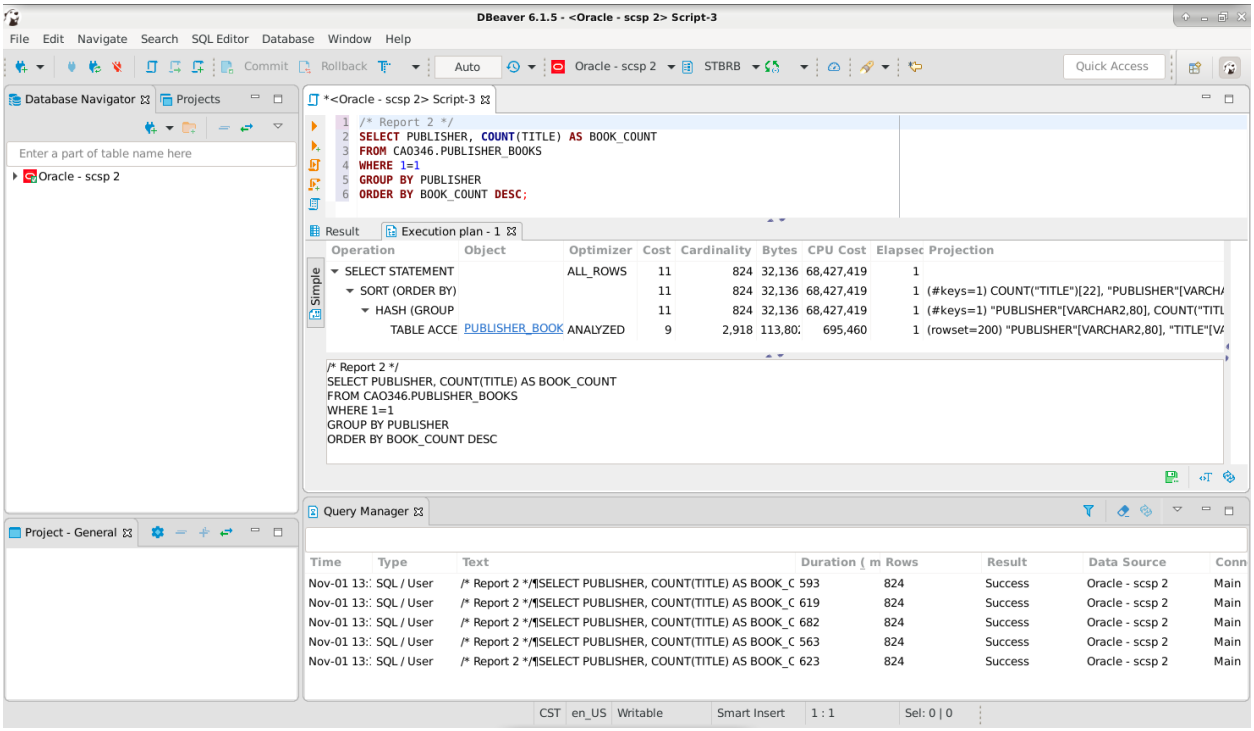
Report 1(Unoptimized):A report that lists all the Authors and the number of books they have.



Trial #	Time (ms)
1	1145
2	1179
3	1286
4	1181
5	1328
Average:	1223.8

After running report 1 with no primary or foreign keys, we found that the query took slightly longer to process at an average of 1.2 seconds and the overall costs were higher.

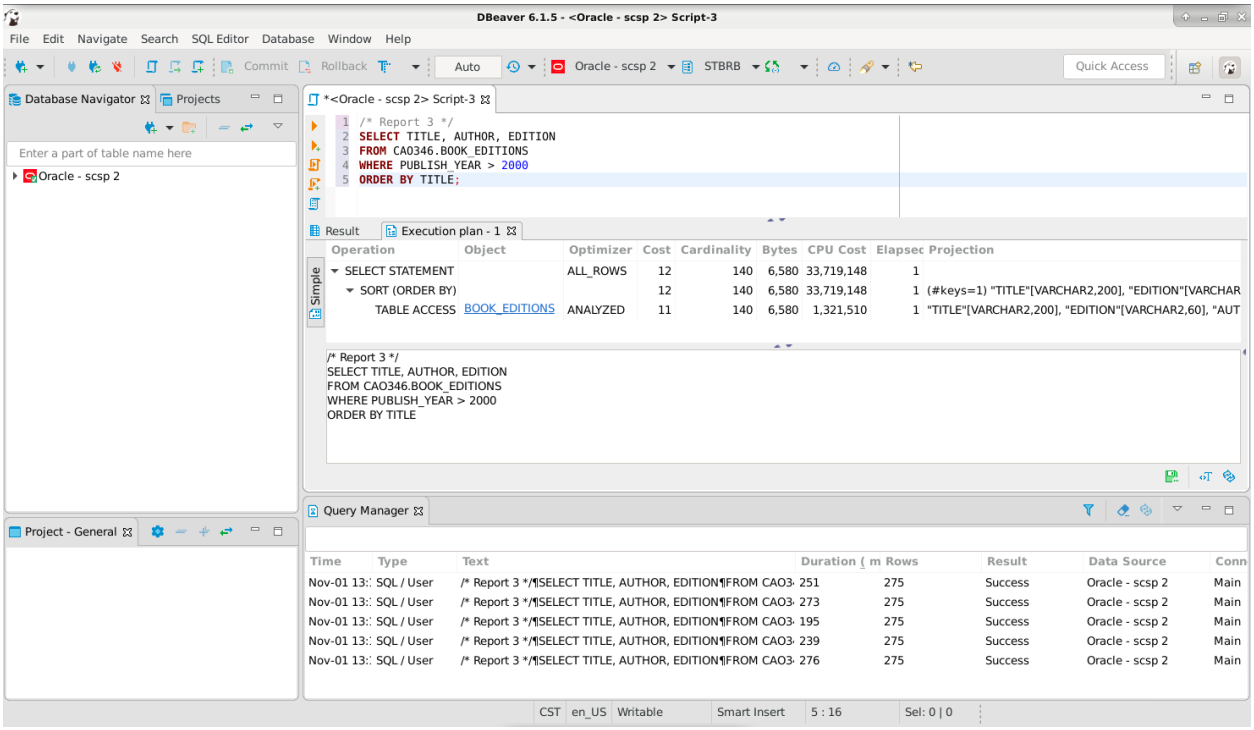
Report 2(unoptimized):A report that lists all the publishing companies and the number of books they have.



Trial #	Time (ms)
1	593
2	619
3	682
4	563
5	623
Average:	616

Running report 2 as an unoptimized query also returned results showing that it took more time to run. This report took approximately .15 more seconds to run that the optimized query, however, the costs remained the same.

Report 3(unoptimized): A report that lists books made after the year 2000.



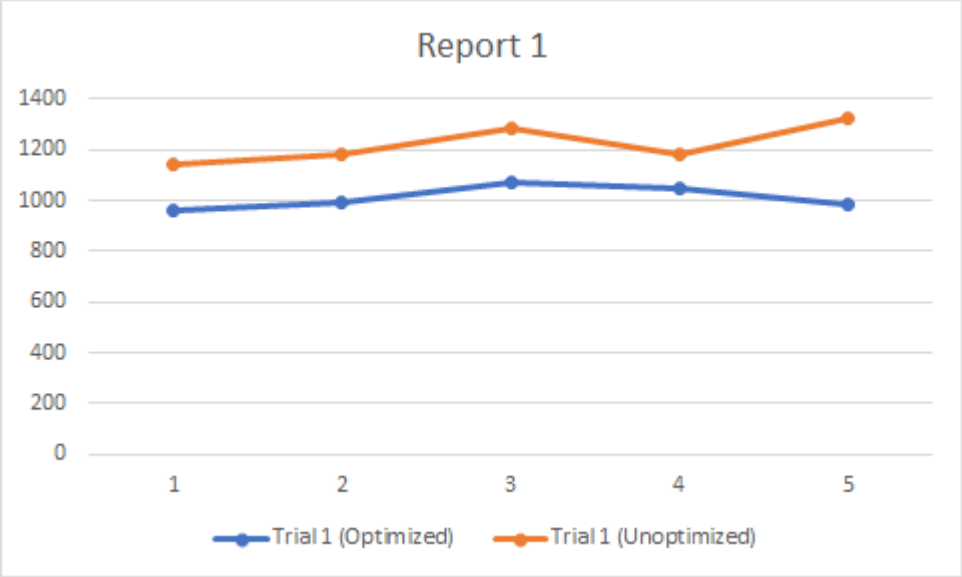
Trial #	Time (ms)
1	251
2	273
3	195
4	239
5	276
Average:	246.8

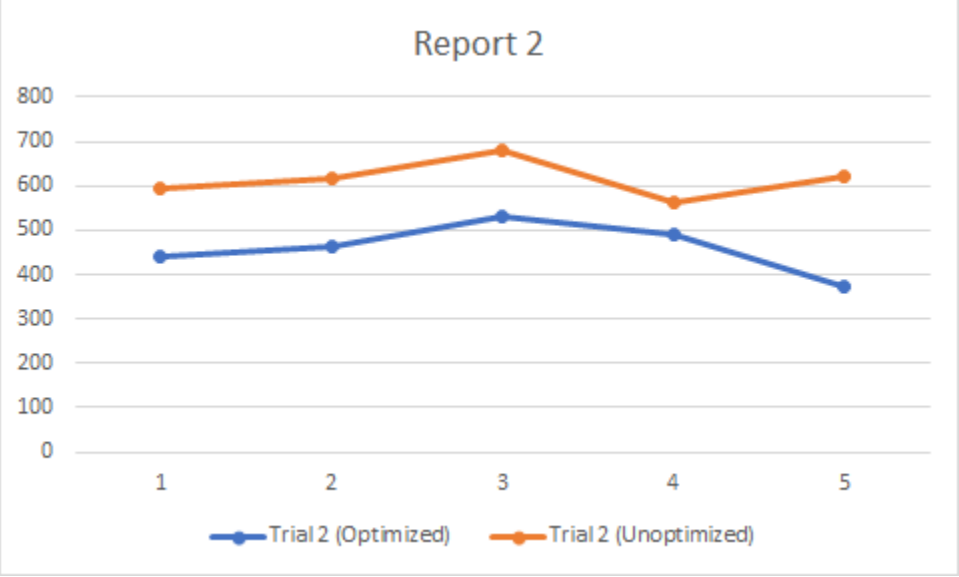
Report 3 without indexes followed the same pattern in which it's run time was slower than the optimized version. It averaged at a runtime of .25 seconds and had the same cost as the original.

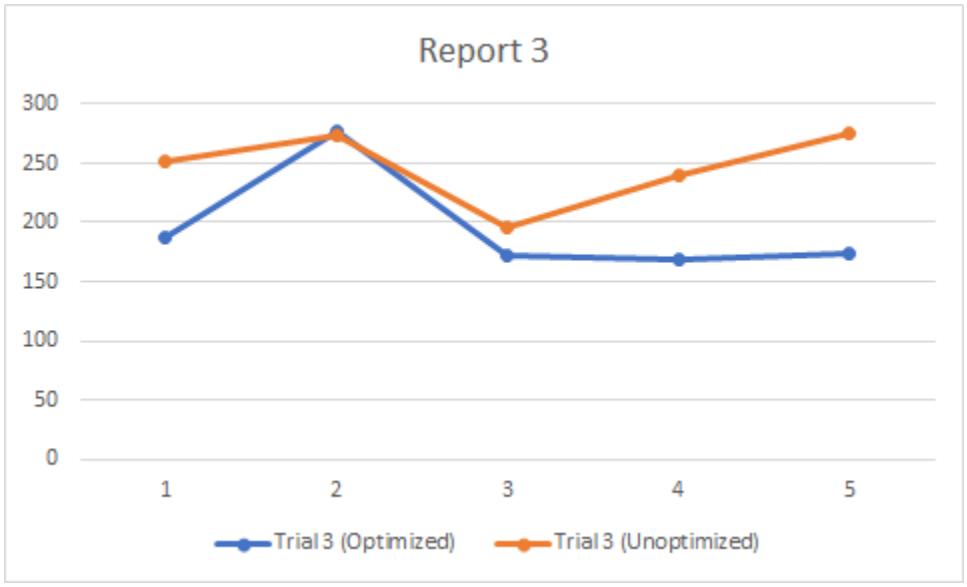
Data Analysis on Optimized Reports Vs Unoptimized Reports

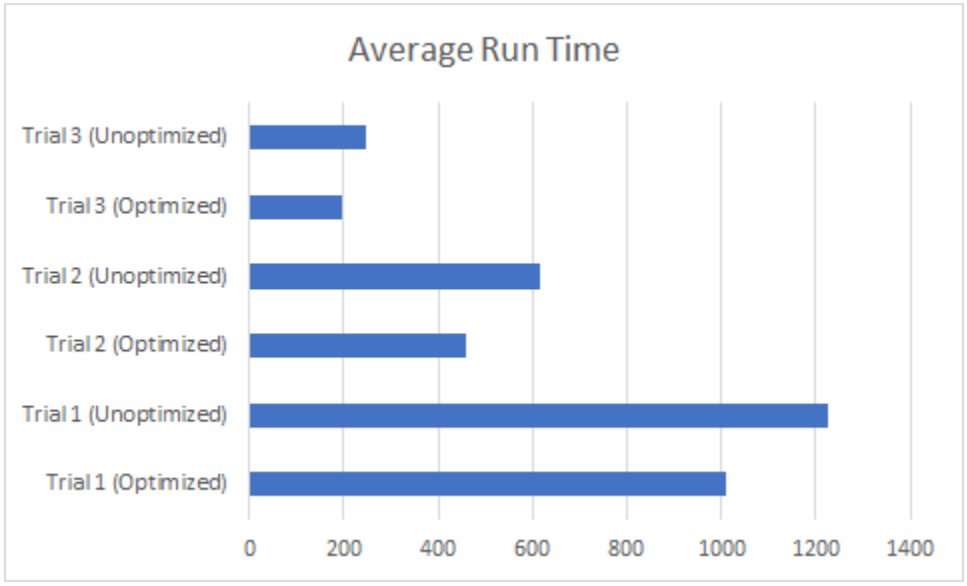
Calculation: ((Unoptimized - Optimized) / Unoptimized) * 100 %

Report 1: 1011.6ms; 1223.8ms --- 17.3% speed increase when optimized
Report 2: 460ms; 616ms --- 25.3% speed increase when optimized
Report 3: 195.6ms; 246.8ms --- 20.7% speed increase when optimized






Report 3
300
250
200
150
100
50
0
1
2
3
4
5
Trial 3 (Optimized)
Trial 3 (Unoptimized)


Average Run Time
Trial 3 (Unoptimized)
Trial 3 (Optimized)
Trial 2 (Unoptimized)
Trial 2 (Optimized)
Trial 1 (Unoptimized)
Trial 1 (Optimized)
0
200
400
600
800
1000
1200
1400

Conclusion on Optimized Reports Vs Unoptimized Reports.

Throughout the optimized reports we found that the ORDER BY and SELECT have the highest cost while the HASH has the lowest due to the configuration of our tables. In the unoptimized reports we found that, while the execution time took longer, the computational cost remained the same. Because we store values in the index tables that are all unique, the key we use to identify a particular value in an external table is just as unique as it's corresponding value. These particular tables are indexed to begin with; however, taking away the values corresponding primary key still leaves you with a unique value so indexed with a primary key or not the computational cost will remain the same. In conclusion, we can see from the graphs and the related tables that all reports ran faster when optimized. The optimized reports ran on average 21.1% faster than their unoptimized counterparts while the computational cost remains the same.

Bad Queries (Part II):

Our query returns the title, author and publisher from all books that have a specific jacket condition, binding type, and book type. The results are further ordered with respect to the selection order. For this example, we wanted to know how many mystery books were in “Good” condition with a hard cover. In order to do this, we joined on title from four tables and 100 book records are returned.

We ran three versions of our query to see if different types of joins would affect the execution time in a measurable or significant way as well. The exact times of each execution of the query are included in the sql comments.

The first tests were completed without primary keys or references in the tables:

```
1  --EXPLAIN PLAN SET statement_id = 'left_outer_query' FOR
2  SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
3  FROM CA0346.BOOK_EDITIONS
4  LEFT OUTER JOIN
5      CA0346.PUBLISHER_BOOKS
6      ON BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE
7  LEFT OUTER JOIN
8      CA0346.BOOK_BINDING_TYPES
9      ON BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE
10 LEFT OUTER JOIN
11     CA0346.BOOK_CONDITION
12     ON BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE
13 LEFT OUTER JOIN
14     CA0346.BOOK_TYPE
15     ON BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE
16 WHERE BOOK_CONDITION.JACKET_CONDITION = 'Good'
17 AND BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover'
18 AND BOOK_TYPE = 'Mystery'
19 ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;
20 -- 136ms without keys
21 --SELECT PLAN_TABLE_OUTPUT
22 --FROM TABLE(DBMS_XPLAN.DISPLAY());
23
24
```

Result | Execution plan - 1

Simple

Operation	Object	Optimizer	Cost	Cardinality
SELECT STATEMENT		ALL_ROWS	44	19
SORT (ORDER BY)			44	19
HASH JOIN (OUTER)			43	19
HASH JOIN			34	18
HASH JOIN			25	40
HASH JOIN			16	125
TABLE ACCESS (FULL)	BOOK_CONDIT	ANALYZED	9	120
INDEX (FAST FULL SCAN)	AUTHOR_TITLI	ANALYZED	7	2,807
TABLE ACCESS (FULL)	BOOK_TYPE	ANALYZED	9	872
TABLE ACCESS (FULL)	BOOK_BINDIN	ANALYZED	9	1,175
TABLE ACCESS (FULL)	PUBLISHER_B	ANALYZED	9	2,918

Natural Joins:

```
--EXPLAIN PLAN SET statement_id = 'implicit_join_query' FOR
SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
FROM
    CA0346.BOOK_EDITIONS,
    CA0346.PUBLISHER_BOOKS,
    CA0346.BOOK_BINDING_TYPES,
    CA0346.BOOK_CONDITION,
    CA0346.BOOK_TYPE
WHERE
    BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE AND
    BOOK_CONDITION.JACKET_CONDITION = 'Good'AND
    BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover' AND
    BOOK_TYPE = 'Mystery'
ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;

--SELECT PLAN_TABLE_OUTPUT
--FROM TABLE(DBMS_XPLAN.DISPLAY());

--168 ms without keys
```

```
--EXPLAIN PLAN SET statement_id = 'explicit_join_query' FOR
SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
FROM CA0346.BOOK_EDITIONS
JOIN CA0346.PUBLISHER_BOOKS ON BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE
JOIN CA0346.BOOK_BINDING_TYPES ON BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE
JOIN CA0346.BOOK_CONDITION ON BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE
JOIN CA0346.BOOK_TYPE ON BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE
WHERE BOOK_CONDITION.JACKET_CONDITION = 'Good'
AND BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover'
AND BOOK_TYPE = 'Mystery'
ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;
-- 132 ms without keys
--SELECT PLAN_TABLE_OUTPUT
--FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Result | Execution plan - 1

Simple

Operation	Object	Optimizer	Cost	Cardinality
SELECT STATEMENT		ALL_ROWS	44	19
SORT (ORDER BY)			44	19
HASH JOIN			43	19
HASH JOIN			34	18
HASH JOIN			25	40
HASH JOIN			16	125
TABLE ACCESS (FULL)	BOOK_CONDIT	ANALYZED	9	120
INDEX (FAST FULL SCAN)	AUTHOR_TITLI	ANALYZED	7	2,807
TABLE ACCESS (FULL)	BOOK_TYPE	ANALYZED	9	872
TABLE ACCESS (FULL)	BOOK_BINDIN	ANALYZED	9	1,175
TABLE ACCESS (FULL)	PUBLISHER_B	ANALYZED	9	2,918

The primary keys were reapplied to see any performance gains:

Left Outer Joins:

```
--EXPLAIN PLAN SET statement_id = 'left_outer_query' FOR
SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
FROM CA0346.BOOK_EDITIONS
LEFT OUTER JOIN
    CA0346.PUBLISHER_BOOKS
    ON BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE
LEFT OUTER JOIN
    CA0346.BOOK_BINDING_TYPES
    ON BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE
LEFT OUTER JOIN
    CA0346.BOOK_CONDITION
    ON BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE
LEFT OUTER JOIN
    CA0346.BOOK_TYPE
    ON BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE
WHERE BOOK_CONDITION.JACKET_CONDITION = 'Good'
AND BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover'
AND BOOK_TYPE = 'Mystery'
ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;
-- 134ms with keys
--SELECT PLAN_TABLE_OUTPUT
--FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Result | Execution plan - 1

Simple

Operation	Object	Optimizer	Cost	Cardinality
SELECT STATEMENT		ALL_ROWS	42	19
SORT (ORDER BY)			42	19
HASH JOIN (OUTER)			41	19
NESTED LOOPS (OUTER)			41	19
STATISTICS COLLECTOR			32	0
HASH JOIN			32	18
NESTED LOOPS			32	18
STATISTICS COLLECTOR			23	0
HASH JOIN			23	40
NESTED LOOPS			23	40
INDEX (FAST FULL SCAN)	BT_PK	ANALYZED	8	872
INDEX (RANGE SCAN)	BBT_PK	ANALYZED	9	1
INDEX (FAST FULL SCAN)	BBT_PK	ANALYZED	9	1,175
INDEX (RANGE SCAN)	PB_PK	ANALYZED	9	1
INDEX (FAST FULL SCAN)	PB_PK	ANALYZED	9	2,918

With the primary and foreign keys applied, the queries performed with similar cost. Execution time was not heavily impacted either, as subsequent runs only vary by about ~ +-50ms, on average. This may be attributed to the design of the tables. Even without keys, they could be considered as offering some sort of indexing due to the thorough normalization of the data. This seems to be the reason for the execution plans being similar.

Natural Joins:

```sql
--EXPLAIN PLAN SET statement_id = 'implicit_join_query' FOR
SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
FROM
    CA0346.BOOK_EDITIONS,
    CA0346.PUBLISHER_BOOKS,
    CA0346.BOOK_BINDING_TYPES,
    CA0346.BOOK_CONDITION,
    CA0346.BOOK_TYPE
WHERE
    BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE AND
    BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE AND
    BOOK_CONDITION.JACKET_CONDITION = 'Good'AND
    BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover' AND
    BOOK_TYPE = 'Mystery'
ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;

--SELECT PLAN_TABLE_OUTPUT
--FROM TABLE(DBMS_XPLAN.DISPLAY());

--194 ms with keys
```

```sql
--EXPLAIN PLAN SET statement_id = 'explicit_join_query' FOR
SELECT BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER
FROM CA0346.BOOK_EDITIONS
JOIN CA0346.PUBLISHER_BOOKS ON BOOK_EDITIONS.TITLE = PUBLISHER_BOOKS.TITLE
JOIN CA0346.BOOK_BINDING_TYPES ON BOOK_EDITIONS.TITLE = BOOK_BINDING_TYPES.TITLE
JOIN CA0346.BOOK_CONDITION ON BOOK_EDITIONS.TITLE = BOOK_CONDITION.TITLE
JOIN CA0346.BOOK_TYPE ON BOOK_EDITIONS.TITLE = BOOK_TYPE.TITLE
WHERE BOOK_CONDITION.JACKET_CONDITION = 'Good'
AND BOOK_BINDING_TYPES.BINDING_TYPE = 'Hard Cover'
AND BOOK_TYPE = 'Mystery'
ORDER BY BOOK_EDITIONS.TITLE, BOOK_EDITIONS.AUTHOR, PUBLISHER_BOOKS.PUBLISHER;
-- 120 ms with keys
--SELECT PLAN_TABLE_OUTPUT
--FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Result | Statistics | Execution plan - 1

Simple

Operation	Object	Optimizer	Cost	Cardinality
SELECT STATEMENT		ALL_ROWS	42	19
SORT (ORDER BY)			42	19
HASH JOIN			41	19
NESTED LOOPS			41	19
STATISTICS COLLECTOR			32	0
HASH JOIN			32	18
NESTED LOOPS			32	18
STATISTICS COLLECTOR			23	0
HASH JOIN			23	40
NESTED LOOPS			23	40
INDEX (FAST FULL SCAN)	BT_PK	ANALYZED	8	872
INDEX (RANGE SCAN)	BBT_PK	ANALYZED	9	1
INDEX (FAST FULL SCAN)	BBT_PK	ANALYZED	9	1,175
INDEX (RANGE SCAN)	PB_PK	ANALYZED	9	1
INDEX (FAST FULL SCAN)	PB_PK	ANALYZED	9	2,918

Some of the operations in the execution plans here have a slightly smaller cost compared to the non indexed tables, for instance the SELECT is improved by two. Differences in joins used do not affect the query cost or time to execute in a significant way. Overall, the execution times do not vary enough to claim a significant increase through the use of the indexes. The cost improves by just a few points over the queries ran on the tables without keys.