Documentation- The document includes 5 different Query optimization plans with the snipped pictures of time and Query plans. Then it included XML generation of 5 different query plans each with the Screen shots and some snipped pictures of times & plans. It has the Table that states the time of Query optimization with the Forced XML generation.

Query Optimizer Decisions/Results- In the descriptive cases for all the 5 different query plans, the query optimizer took **Right decision 4 times**. The time taken by query optimizer is less. Only 1 time it made a wrong decision when I forced **XML Plan2 in query 5- that is when the query plan includes 2000 rows generated by ran_dt and then tested the query.**

Time taken by the original Query plan 5 is 329ms while the time taken by the Forced XML is 312ms.

Dataset Description- I have created two tables and one database. The database name is player score. It is real data of players score. I have three tables one with the name records, house and ran_dt. The records include the data of player score it has 15 columns and 11151 rows. House dataset is the Kaggle real dataset it includes 18 different columns 21614 rows. Another table is the random generated table with the records. It has 8 different columns and 2000 rows.

Observations- While trying to get around 2000 records the random generation of data took half an hour for the data generation.

Generating Query-

```
CREATE Table ran_dt
(    col1 int,
col2 nvarchar(50),
col3 nvarchar(50),
col4 nvarchar(50),
col5 nvarchar(50),
col6 nvarchar(50),
col7 nvarchar(50),
col8 nvarchar(50)
);

Declare @Id int
Set @Id = 478
While @Id <= 2000
Begin
   Insert Into ran_dt values (@Id,'col2-' + CAST(Round(@Id*Rand(),0) as
nvarchar(10)),'col3-' + CAST(@Id*Rand() as nvarchar(10)),
              'col4-' + CAST(Round(@Id*Rand(),0) as nvarchar(10)),'col5-' +
CAST(@Id*Rand() as nvarchar(10)),
              'col6-' + CAST(Round(@Id*Rand(),0) as nvarchar(10)),'col7-' +
CAST(@Id*Rand() as nvarchar(10)),                'col8-' +
CAST(Round(@Id*Rand(),0) as nvarchar(10)))    Print @Id
   Set @Id = @Id + 1
End
```

Query Plan change in response to change in Cardinalities The query used to generate these 5 different plans-

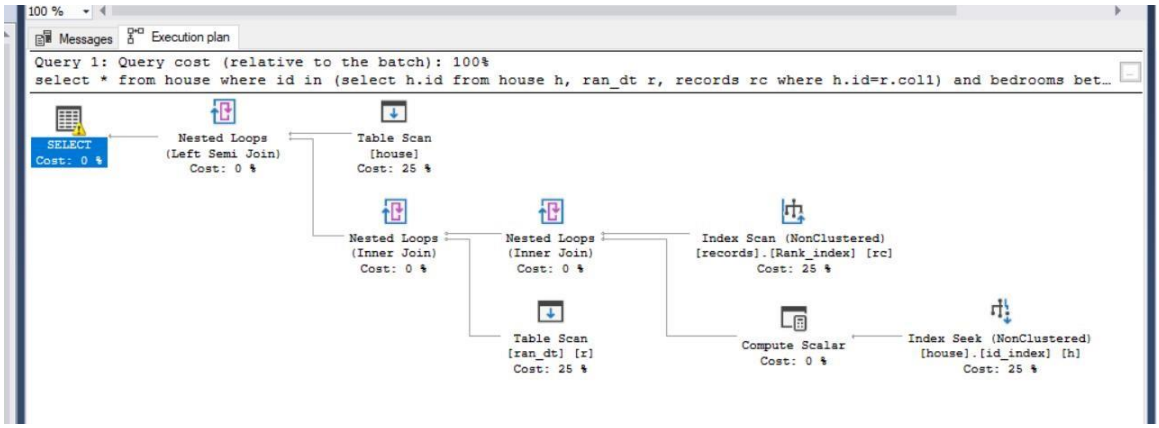
```
select * from house where id in (select h.id from house h, ran_dt r, records rc
where h.id=r.col1) and bedrooms between 1 and 33;
```

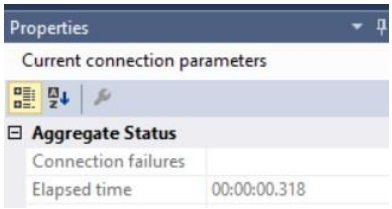
Below are the 5 different query optimizations with respect to change in cardinalities by adding and deleting number of rows.

1. **Query plan with one row in ran_dt and with one row in house.**

Observations- With only one row in two different tables it computes non-clustered index seek on one table and index scan on another table. Note- The one table having big columns(house) it performs index seek on that and the one having less columns it performs scan.

Query Optimization & Time-

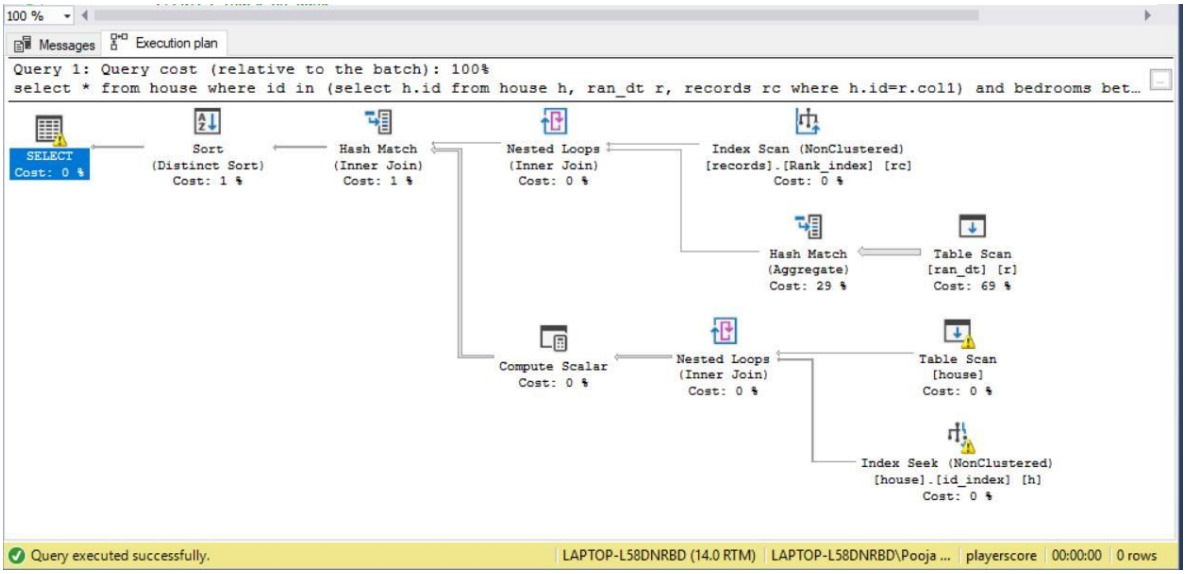


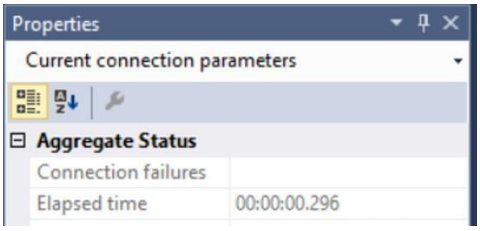


---

[1] **. With 10 rows in random dataset and 100 in house.**

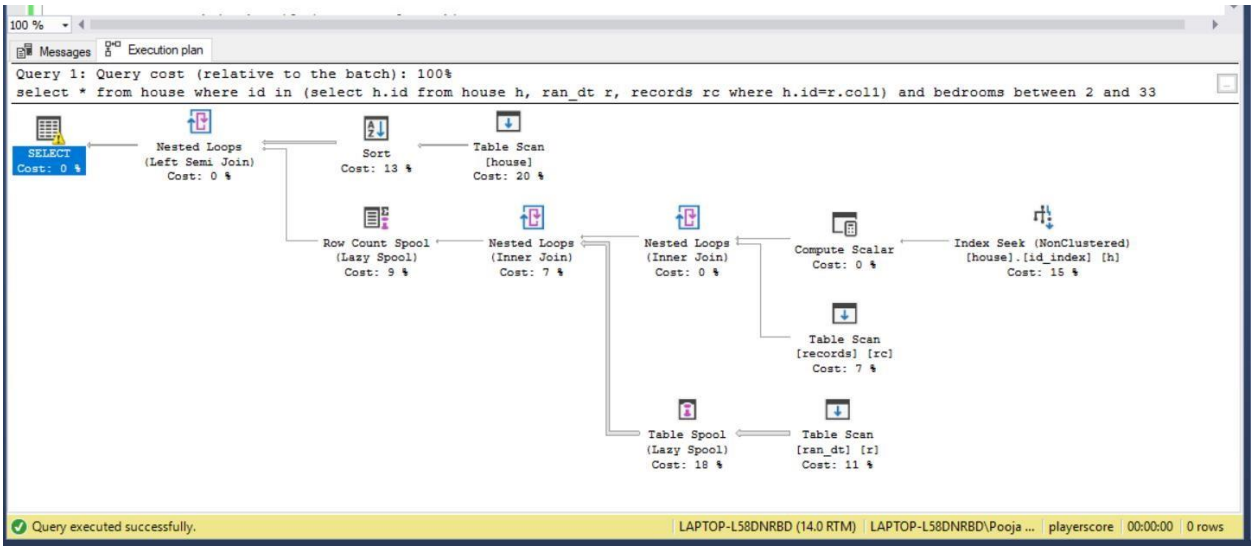
Observations- the query performs Index seek (non-clustered), table scan on house & ran_dt, Index scan (nonclustered) on records table, nested loops (inner join), and nested loops (inner join). Below are the Query plans-

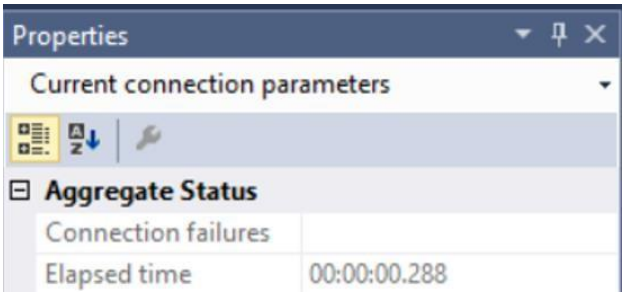




3. **This query plan has 999 rows on table house and 100 rows on ran_dt.**

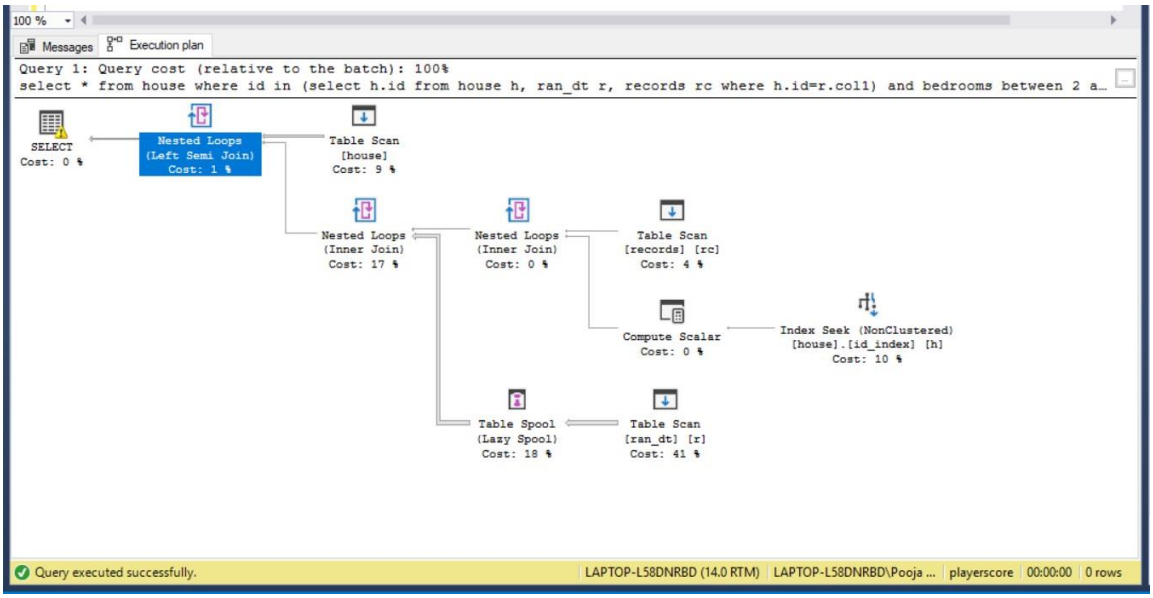
Observations- This query plan does table scan on ran_dt & records, index seek on house, Nested loop inner join & let semi join and sort. Below is the Query optimization plan and time-

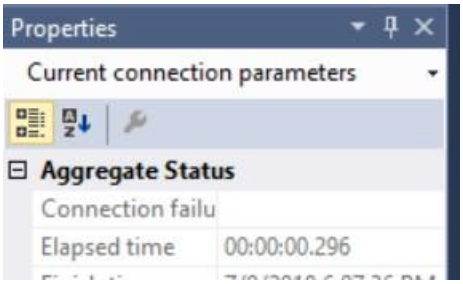




4. **This query plan has 999 rows on house and 477 Rows on ran_dt.**

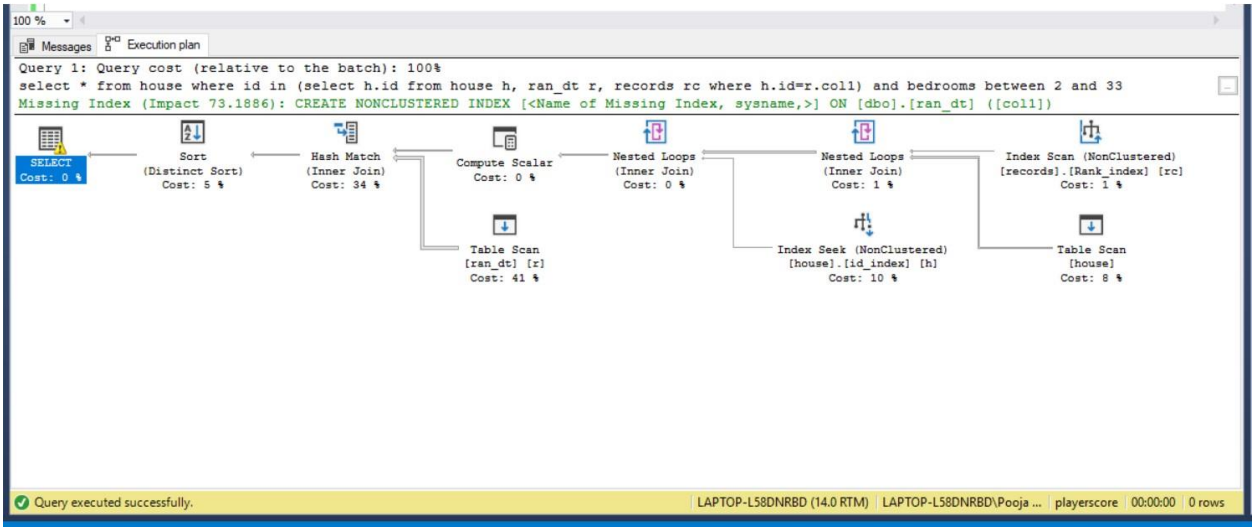
Observation- This query plan does table scan on ran_dt, Index seek(non-clustered) on house, table spool, compute scalar, Table scan on records, two Nested loops (inner joins) and Nested loops (left semi join).

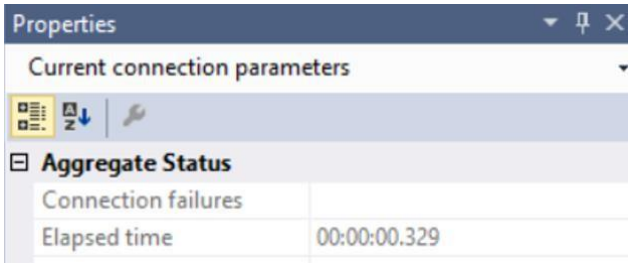




5. **This query plan includes 2000 rows generated by ran_dt and then tested the query.**

Observations- In this query plan it performs Table scan, Index scan non-Clustered , Index Scan on records, compute scalar, table scan on ra_dt, Nested loop (inner join), hash match (inner join), sort (distinct sort).

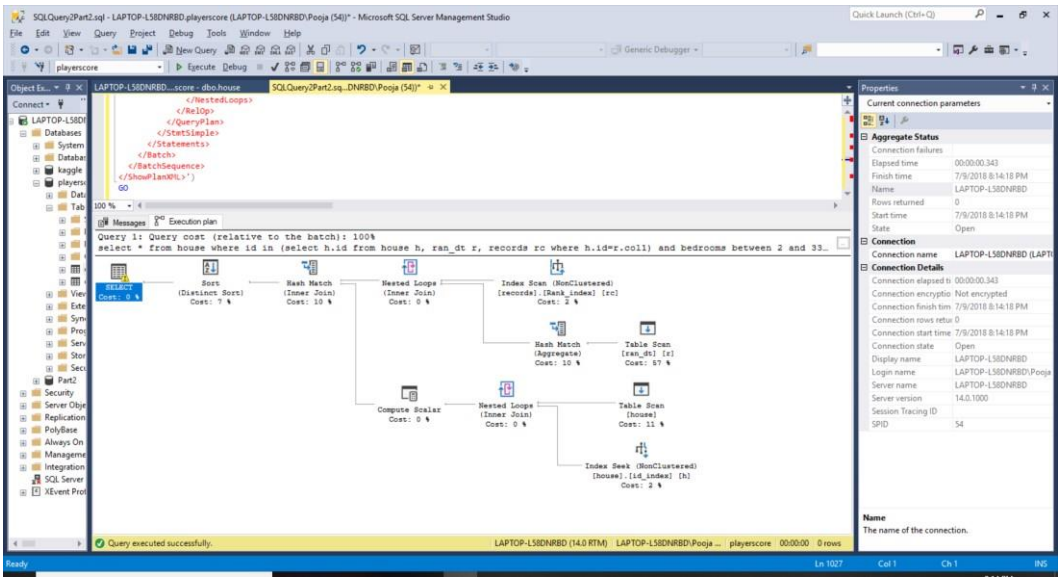


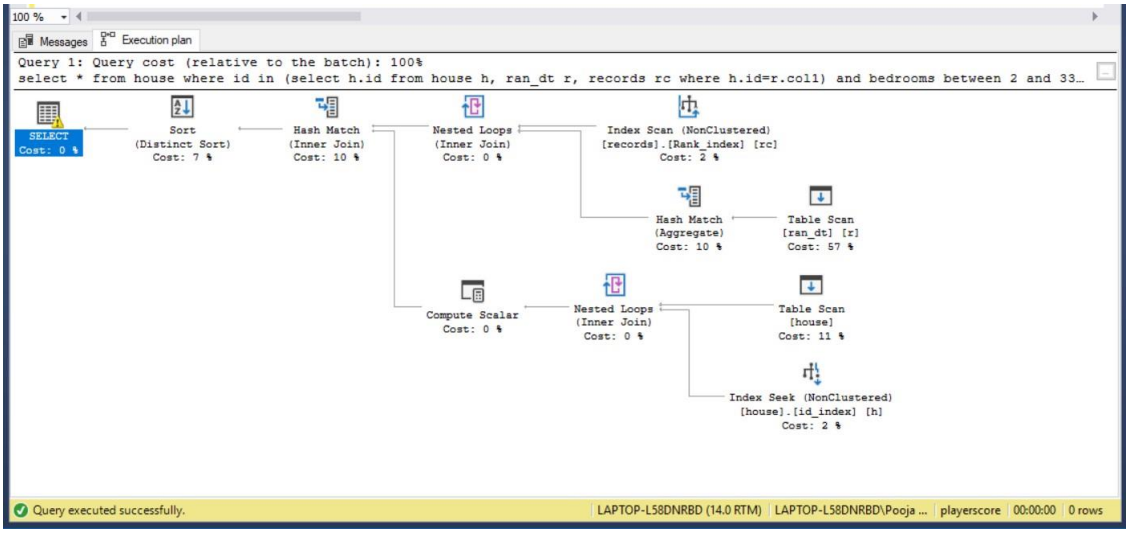


**********XML GENERATION**********

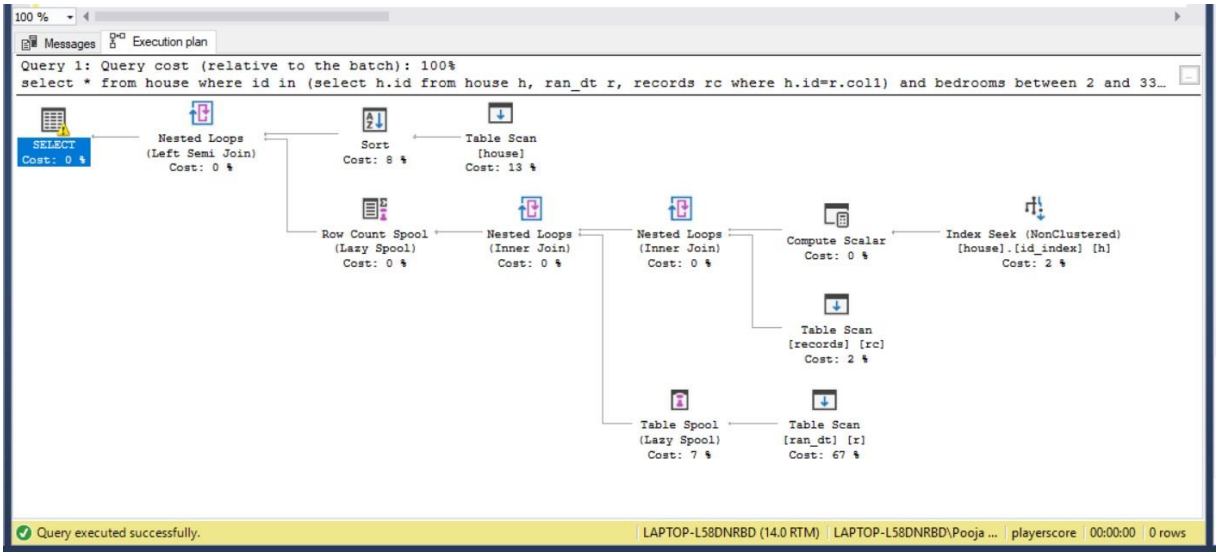
Query Plan 1- 1 row in house, records and ran_dt

XML of Query plan 2 in query plan 1-

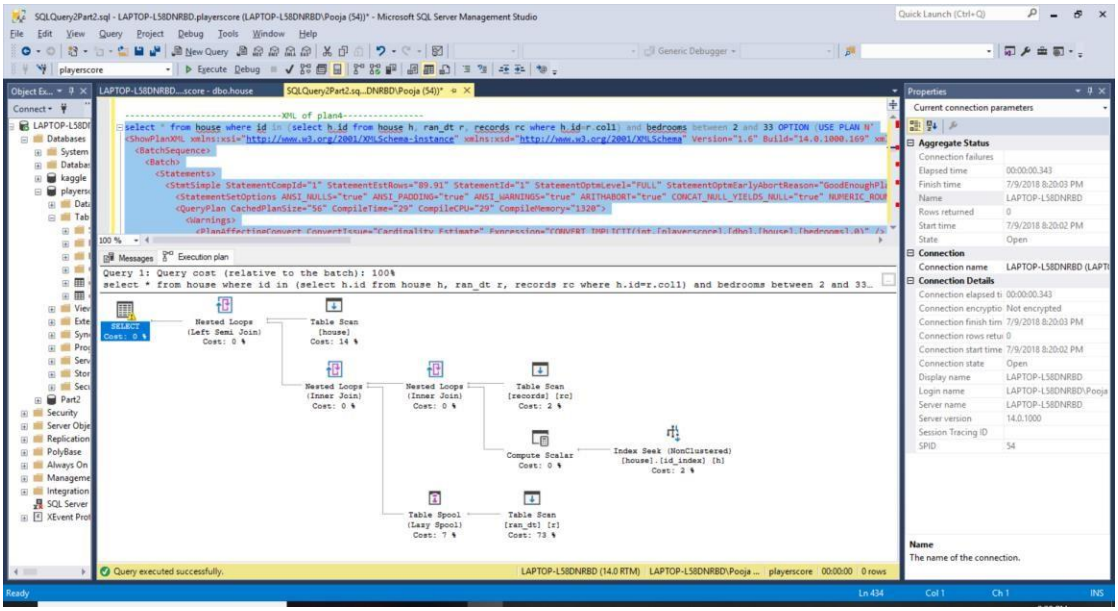


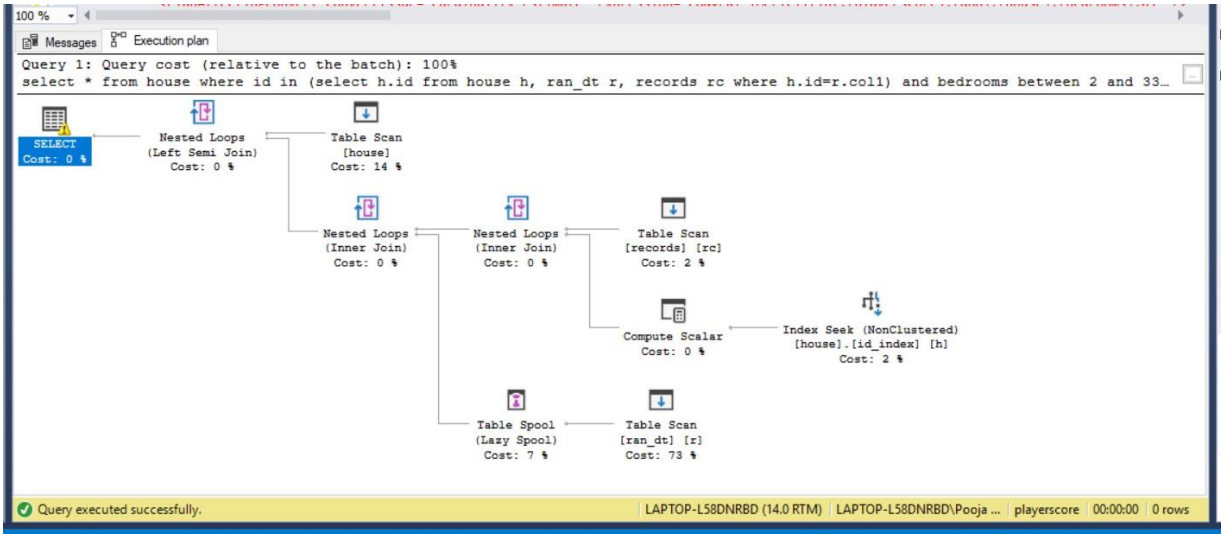


XML of plan 3 in query plan 1-

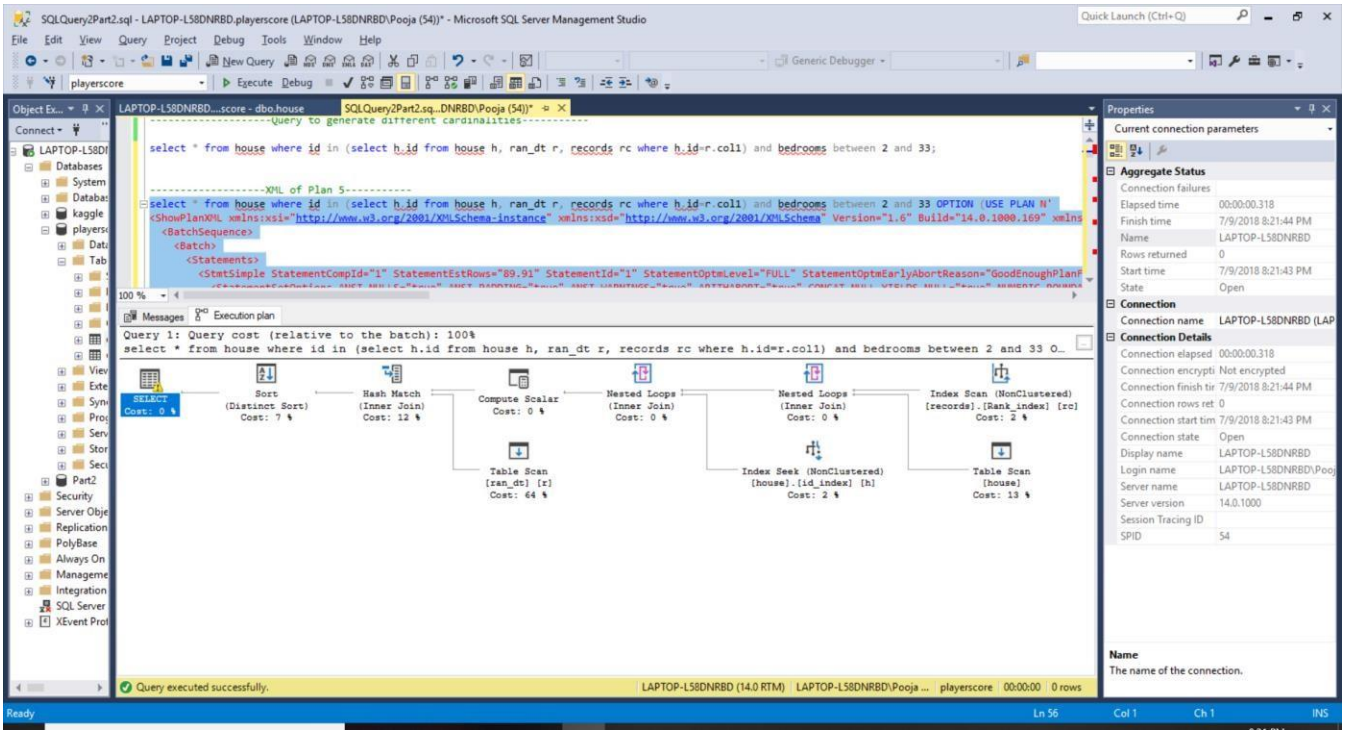


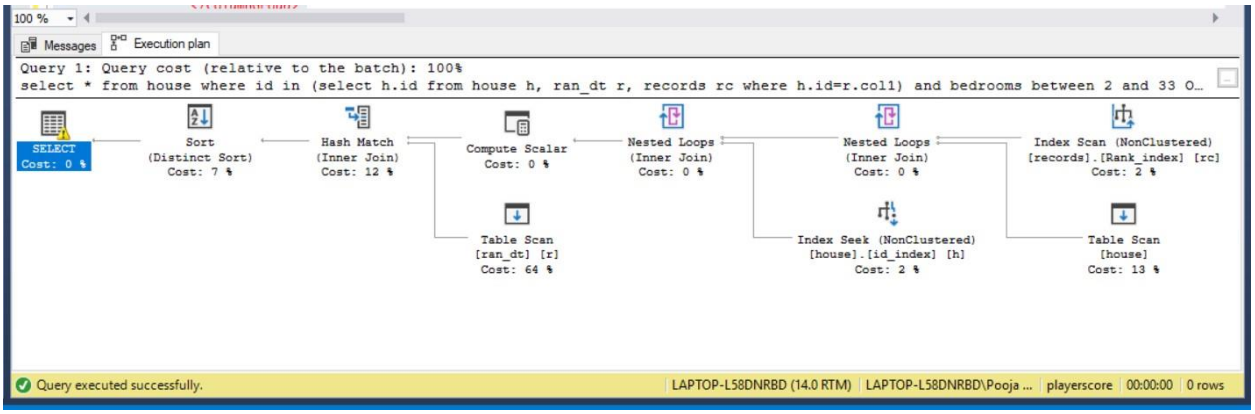
XML of 4 in query plan 1-



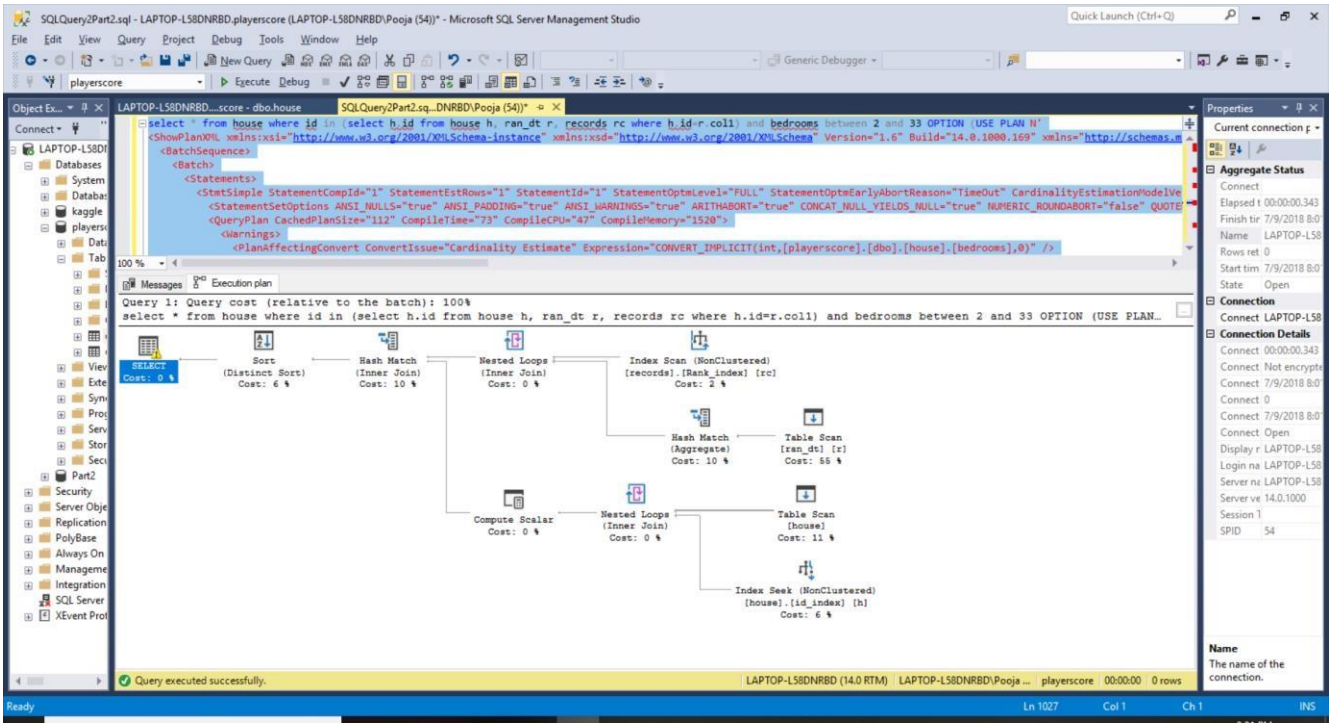


XML of plan 5 in query plan 1-

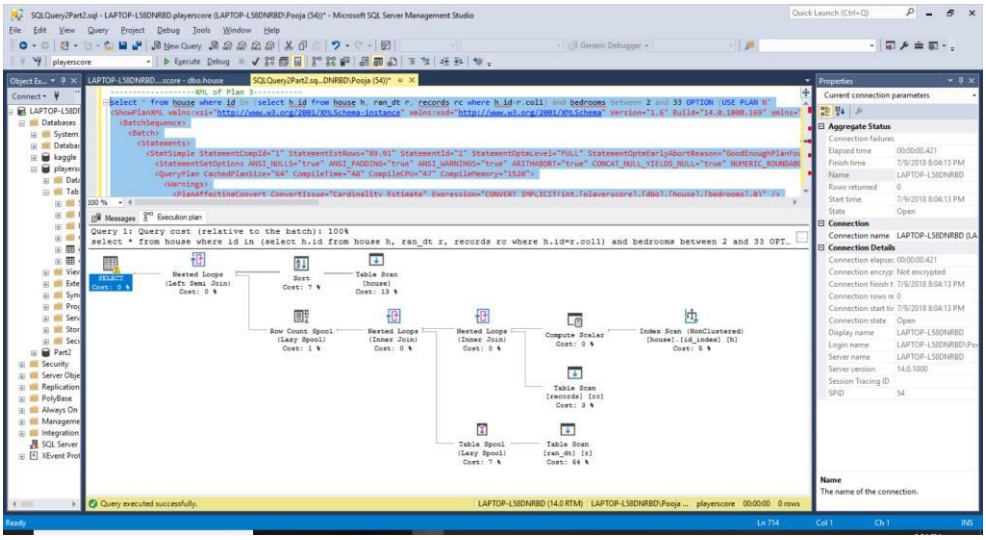


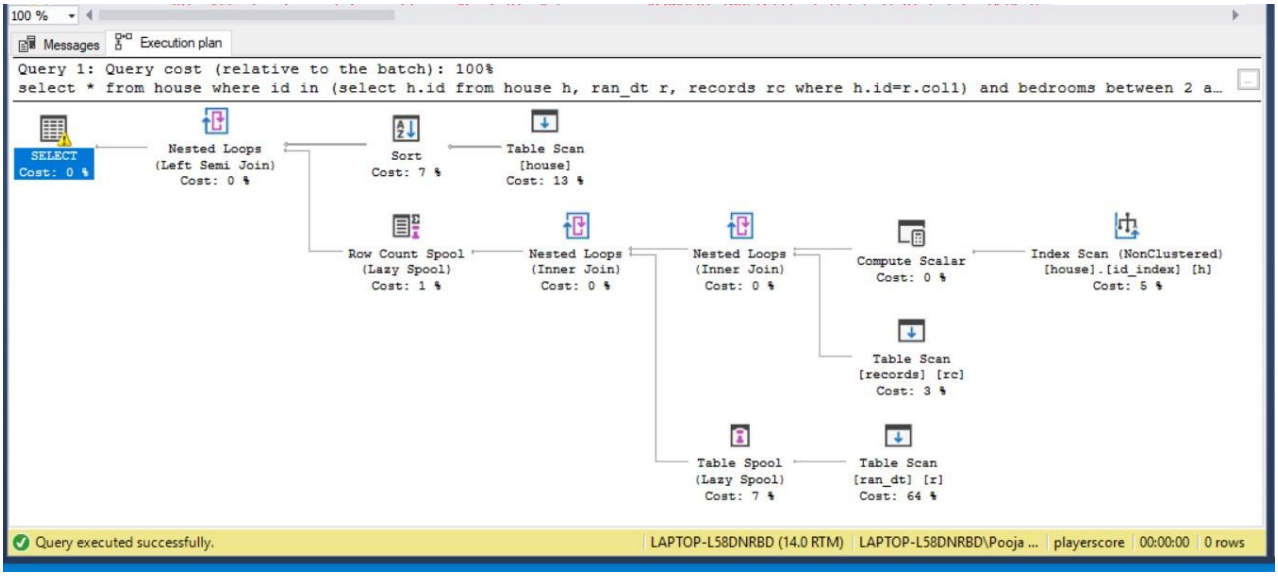


Query Plan 2- With 10 rows in random dataset and 100 in house Query plan 1 XML in Query plan2-

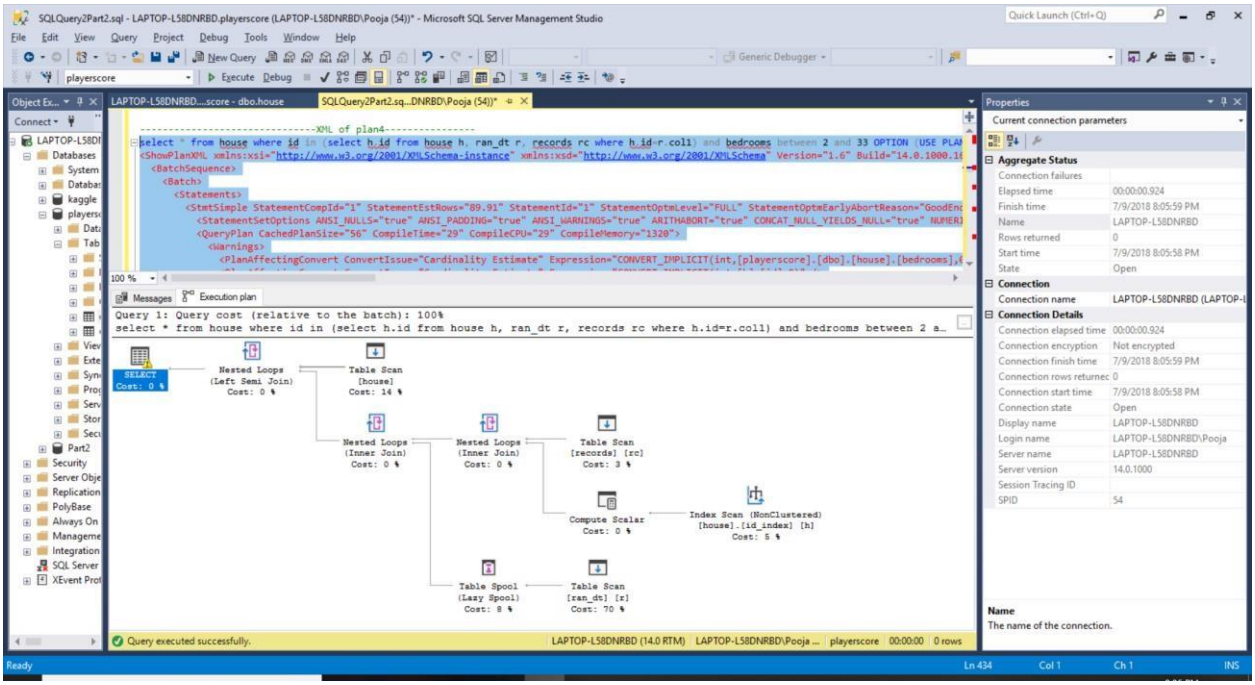


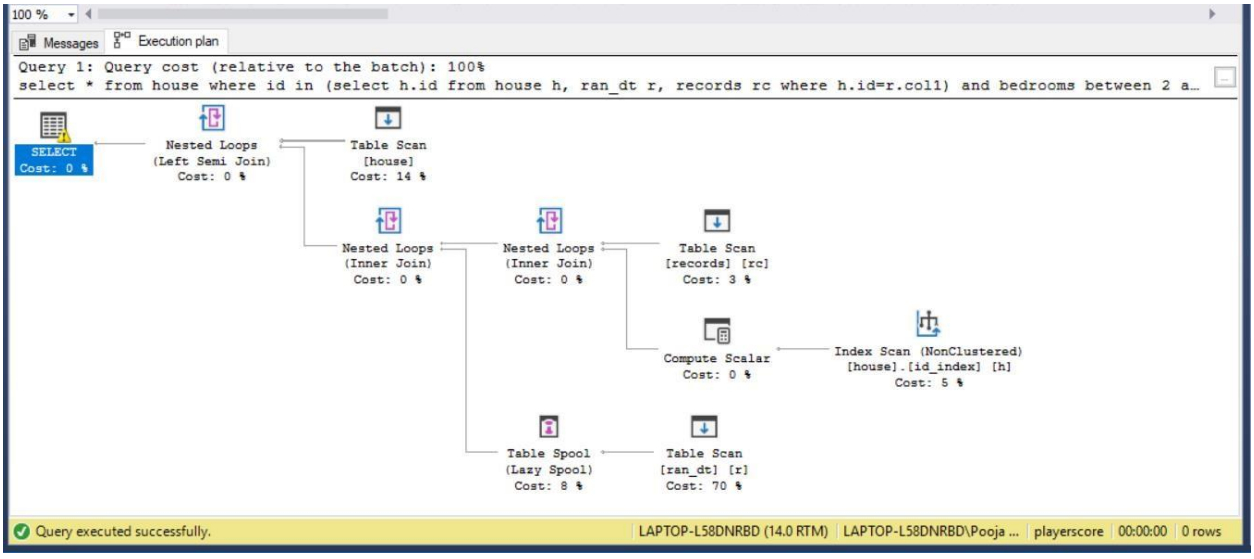
XML Query plan 3 in query plan2-



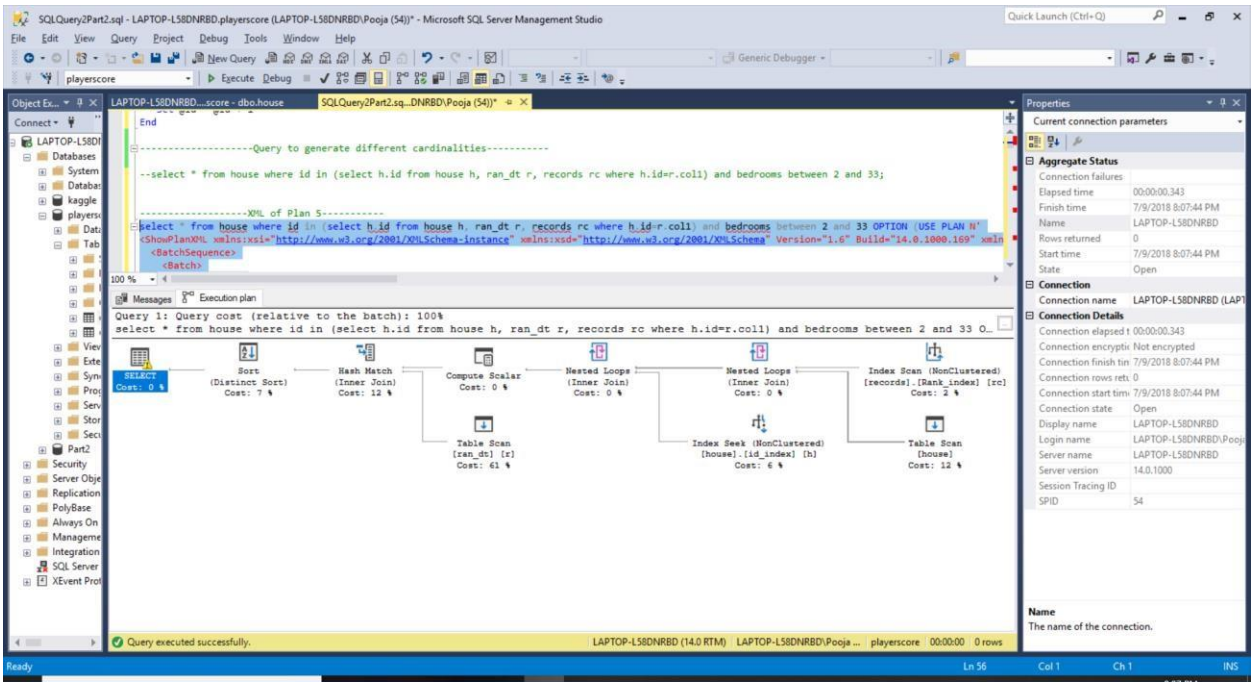


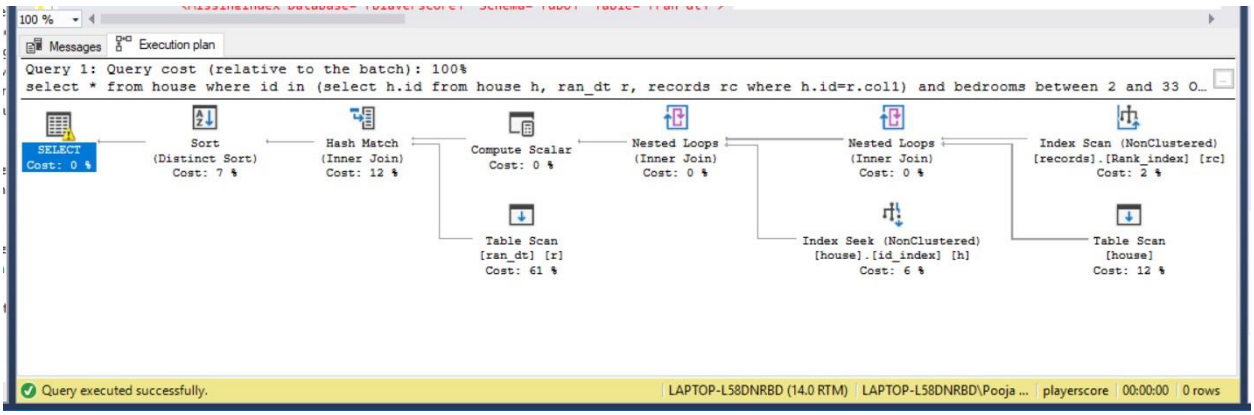
XML Query plan 4 in query plan 2-





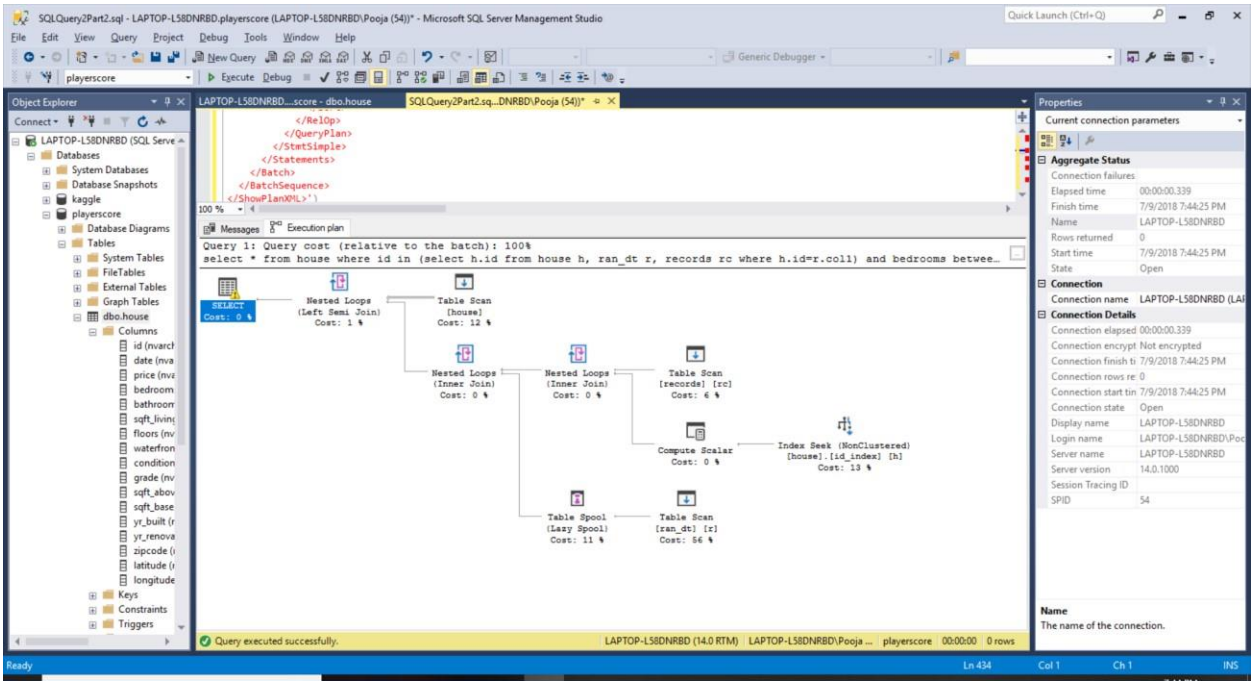
XML query plan 5 in query plan 2-





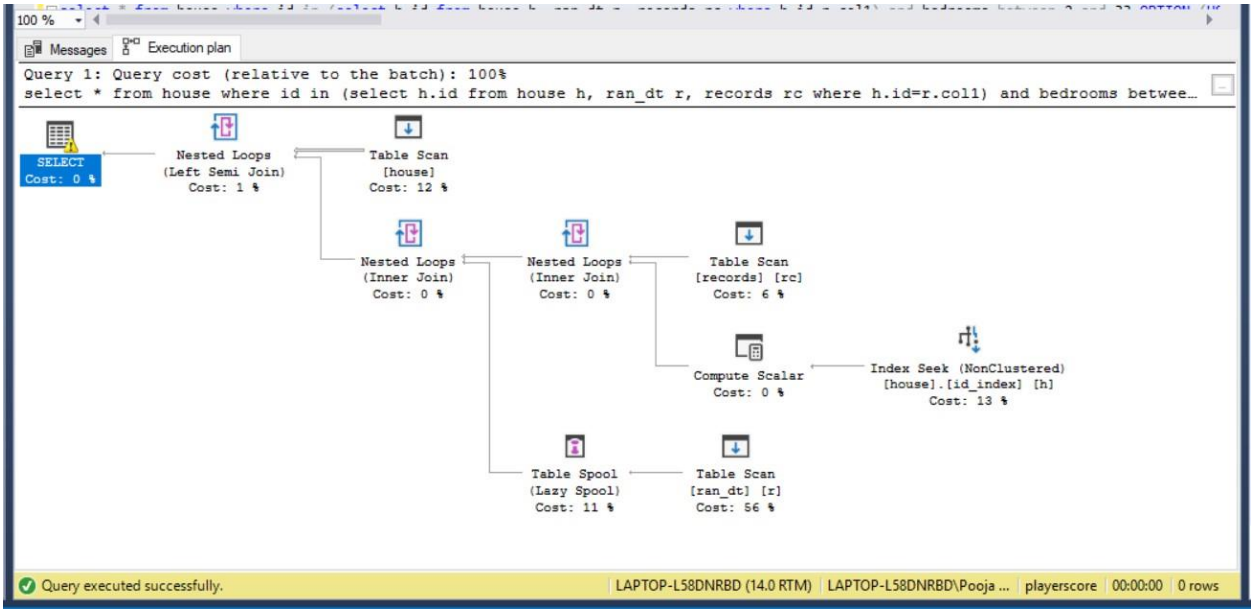
Query plan 3- This query plan has 999 rows on table house and 100 rows on ran dt

XML Query plan1 in query plan 3-

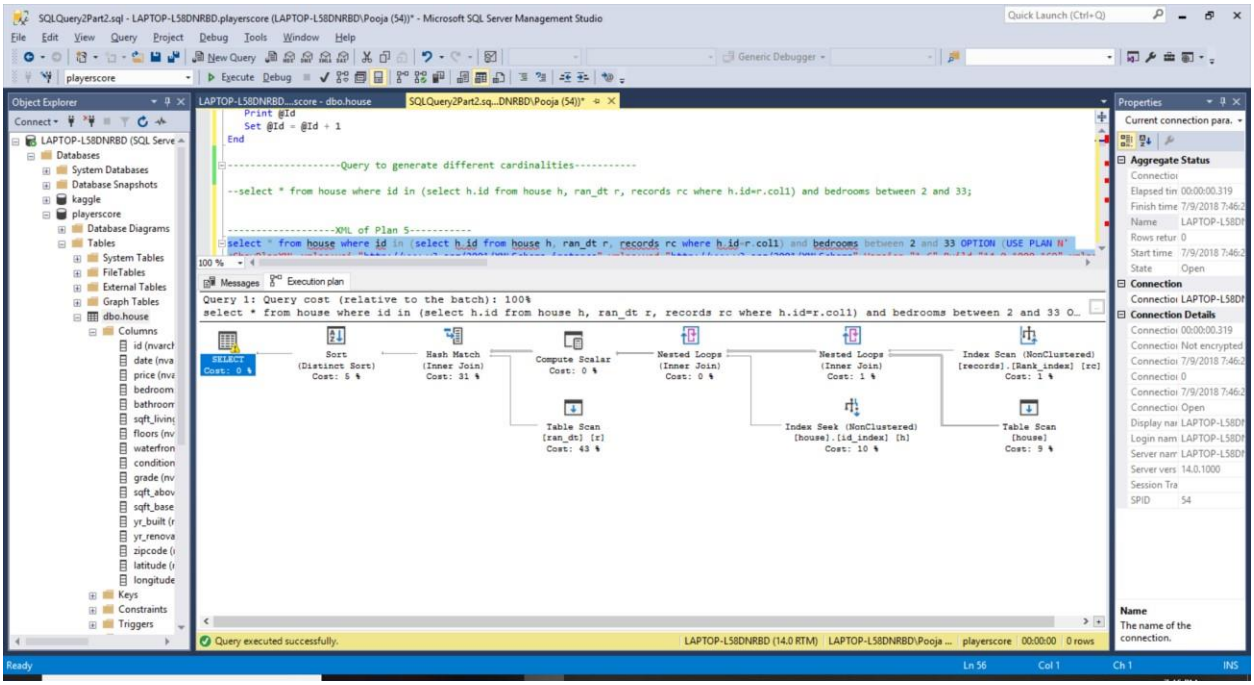


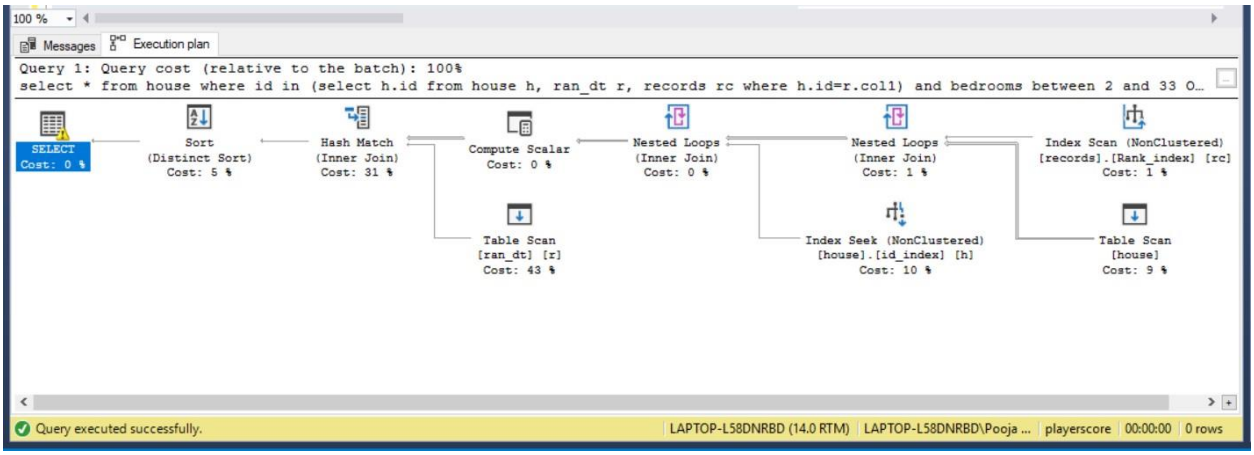
XML plan 2 in query plan 3-

XML of plan 4 in query plan 3-



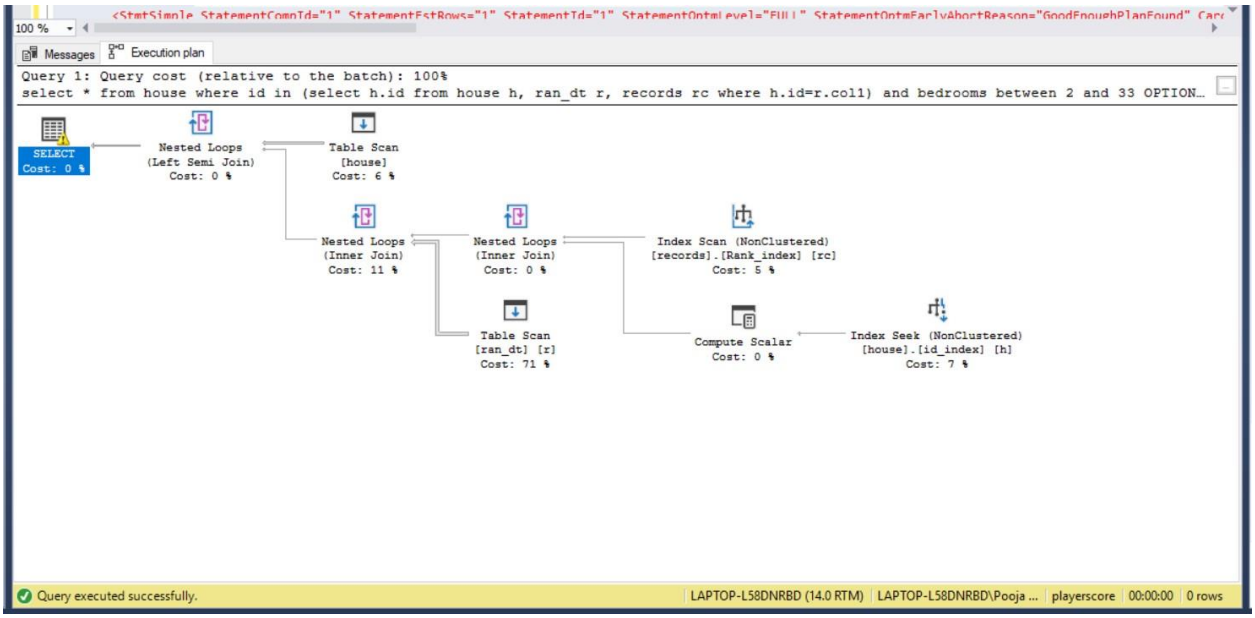
XML plan 5 in query plan 3-

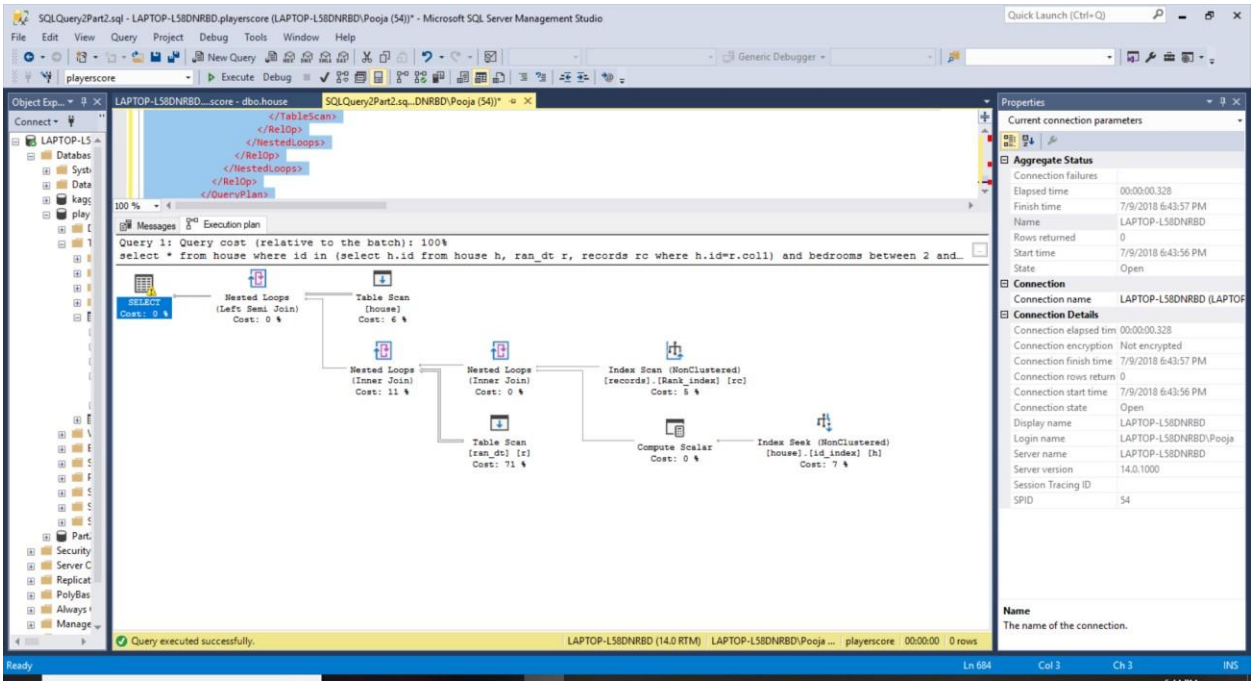




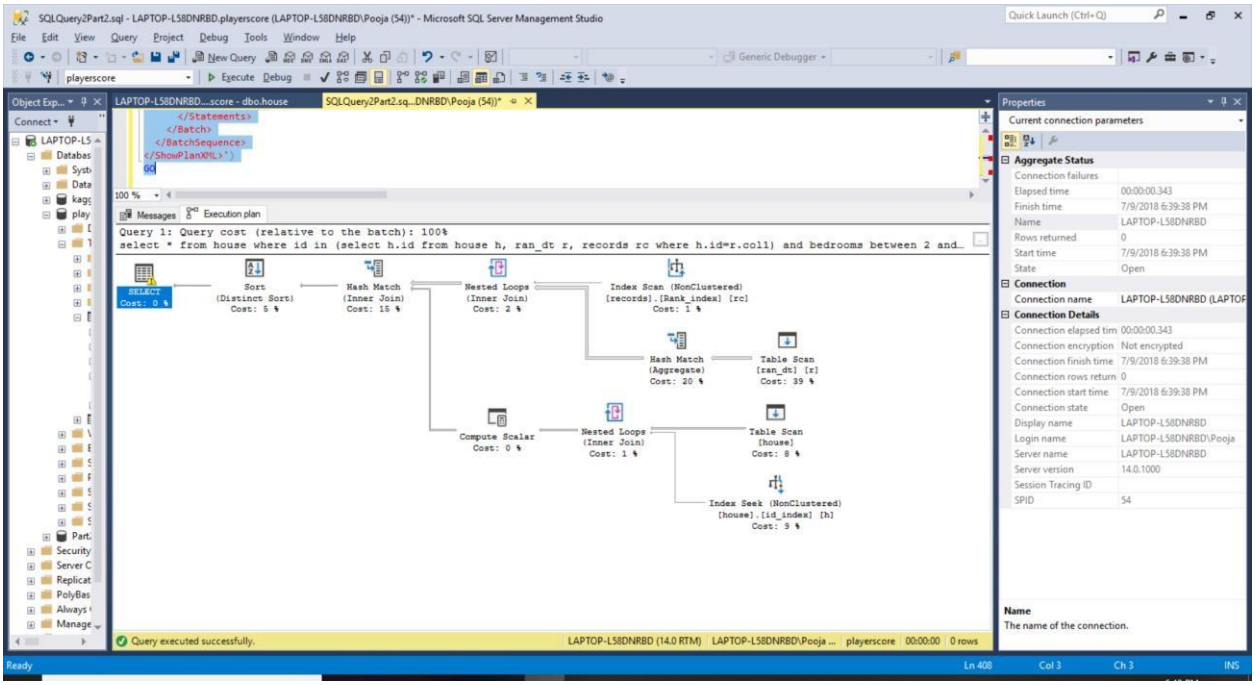
Query plan4- This query plan has 999 rows on house and 477 Rows on ran_dt

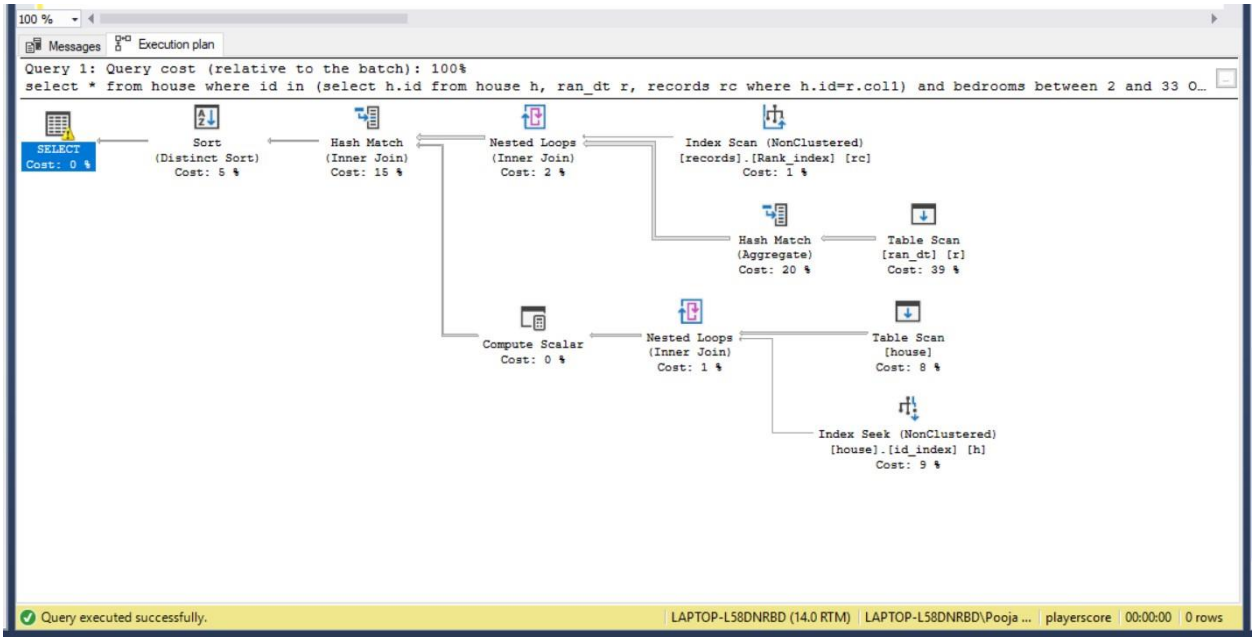
Query plan 1 XML on Query plan 4-



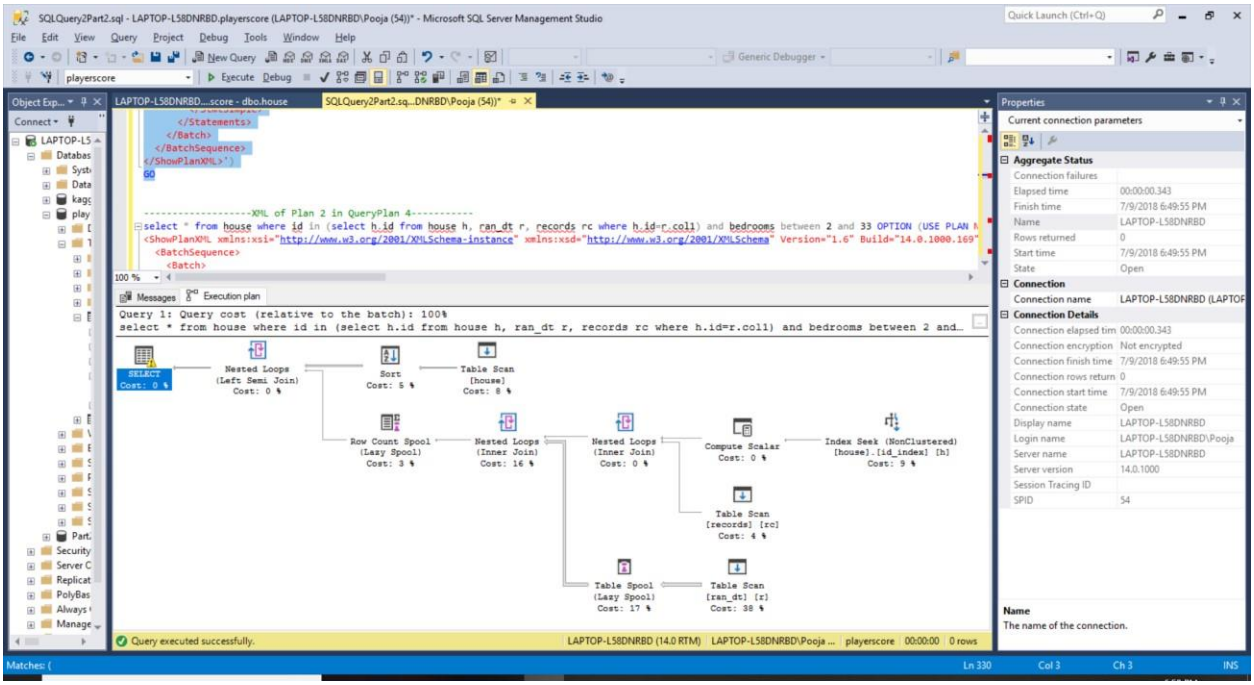


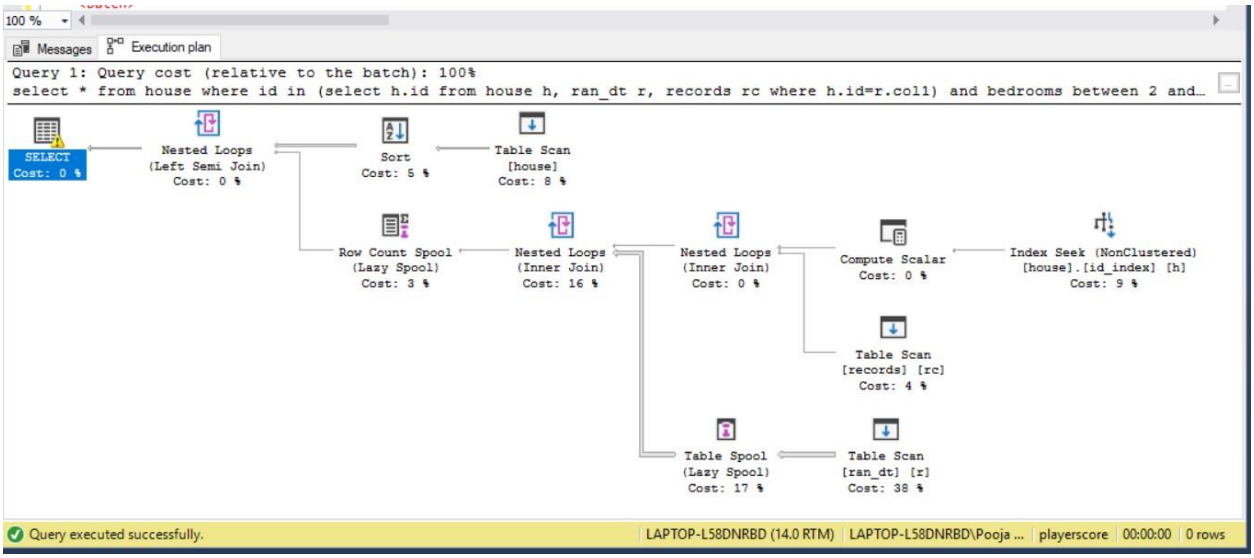
Query plan 2 XML on Query plan 4-



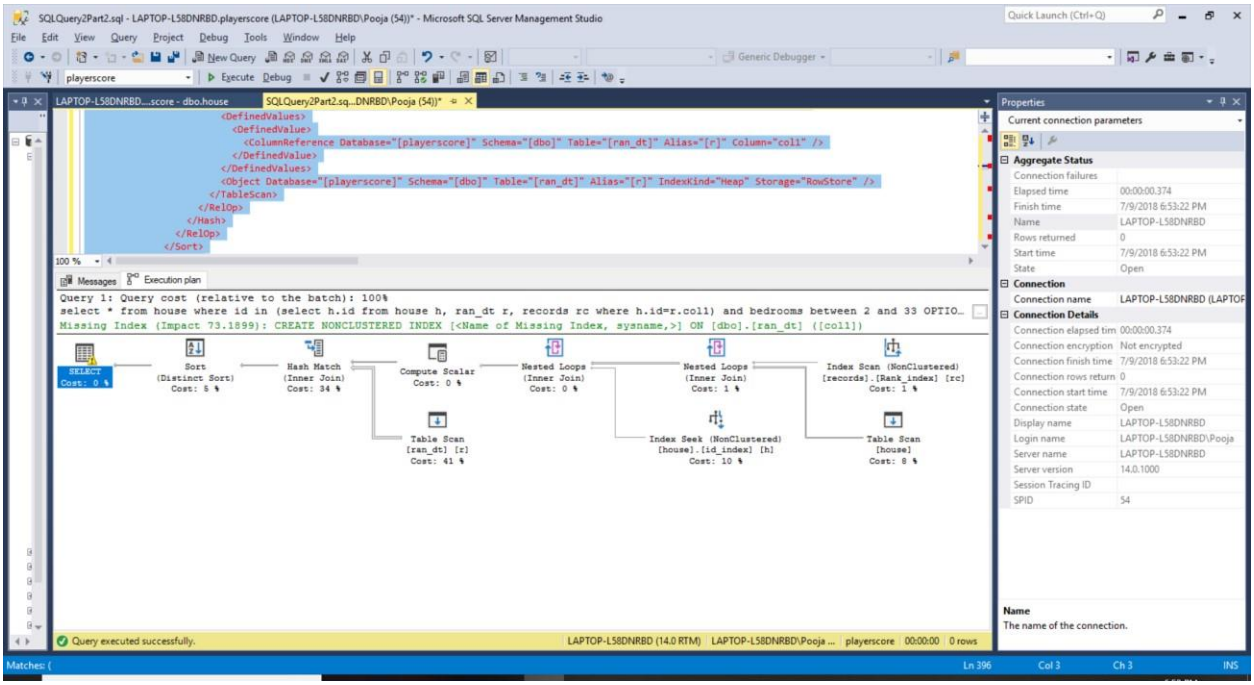


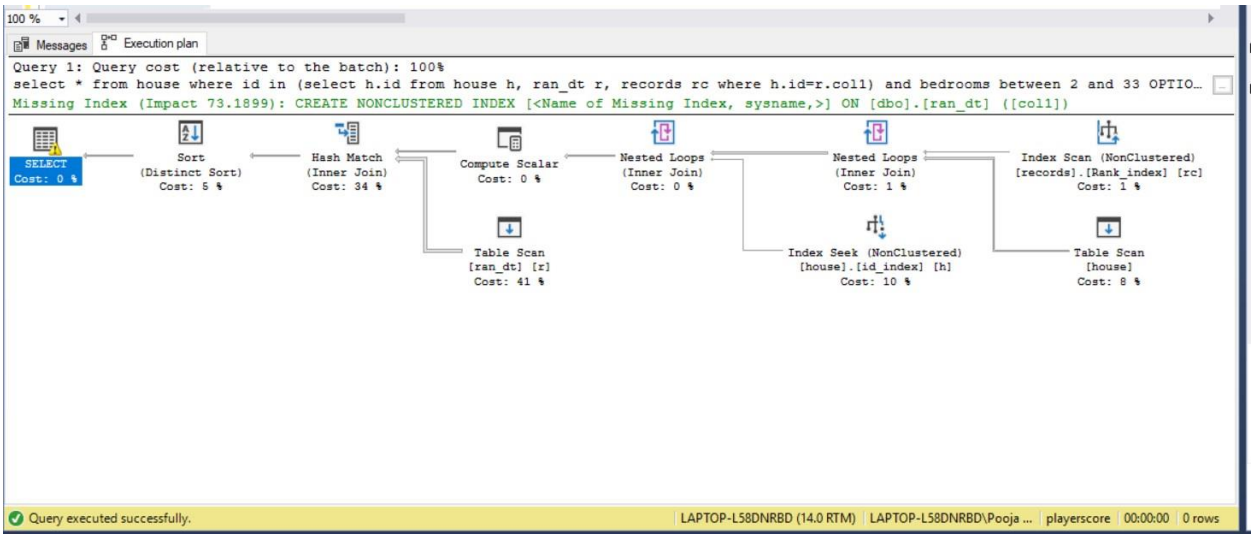
QUERY plan 3 XML on query plan 4-





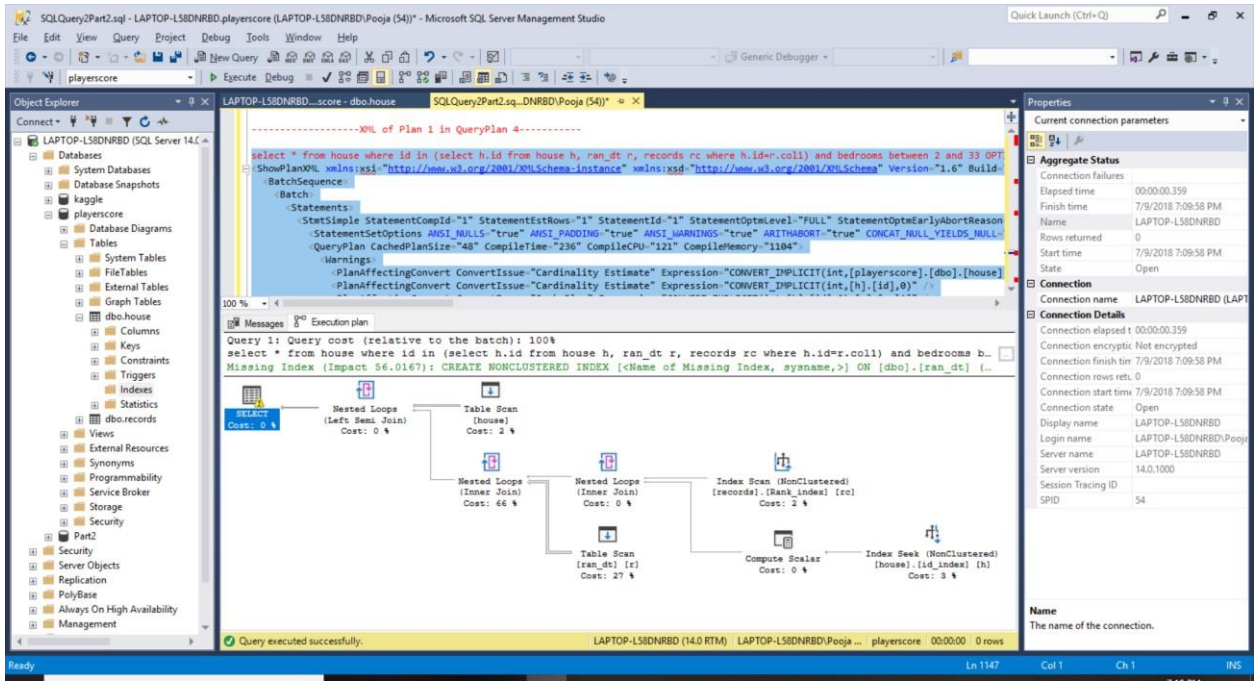
QUERY plan 5 on query plan 4-

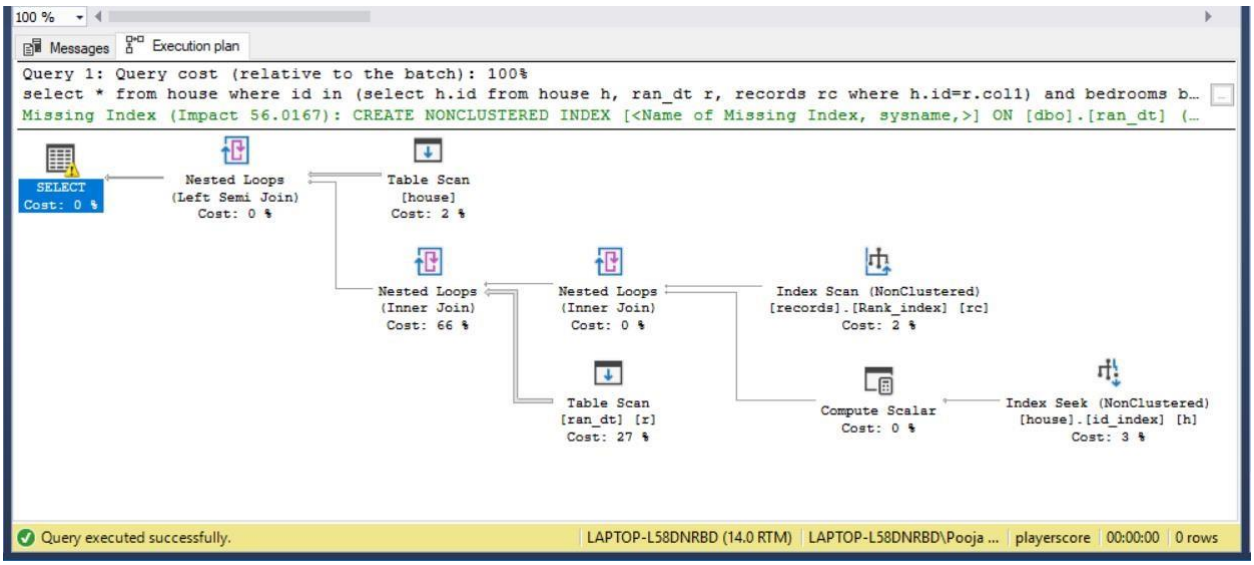




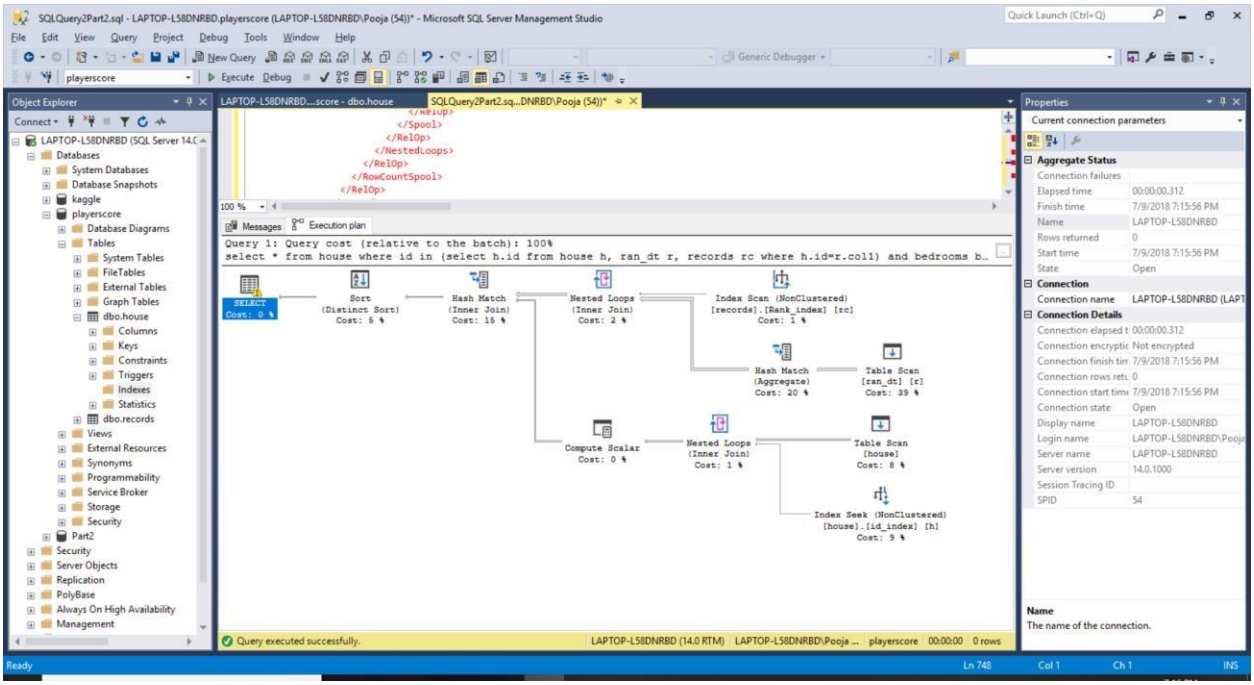
Query plan 5- This query plan includes 2000 rows generated by ran_dt and then tested the query.

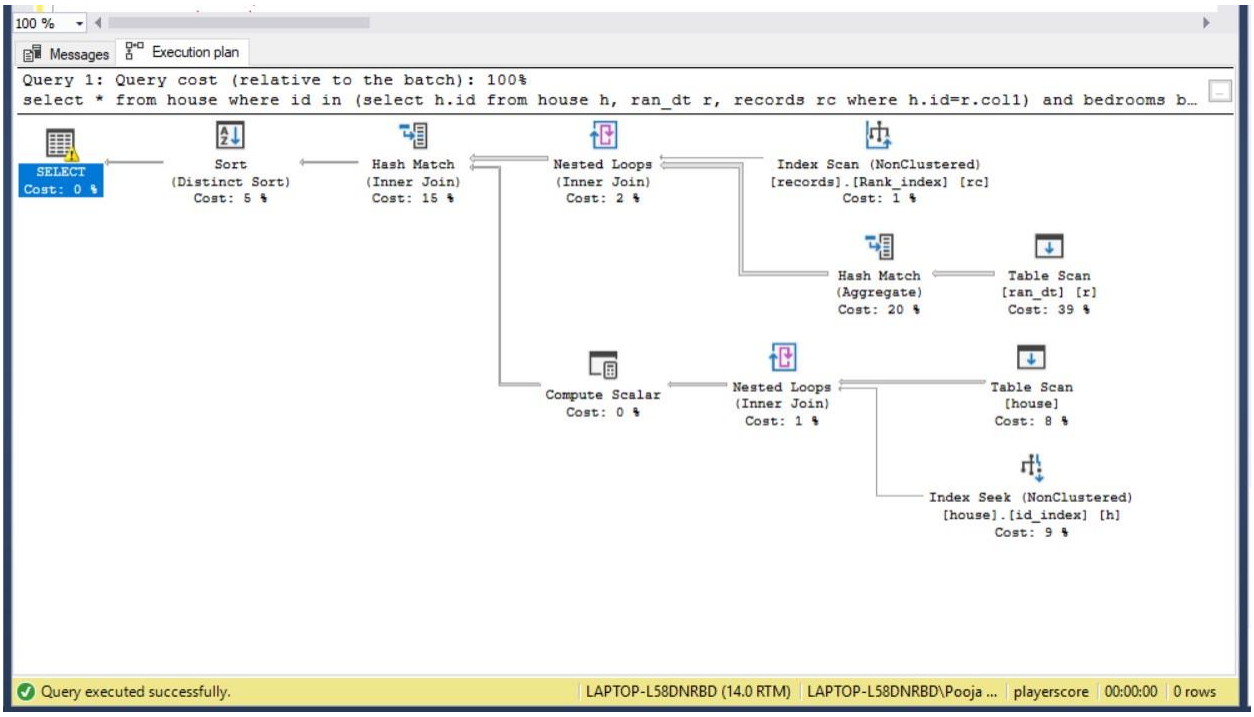
XML of Plan 1 in Query plan 5-



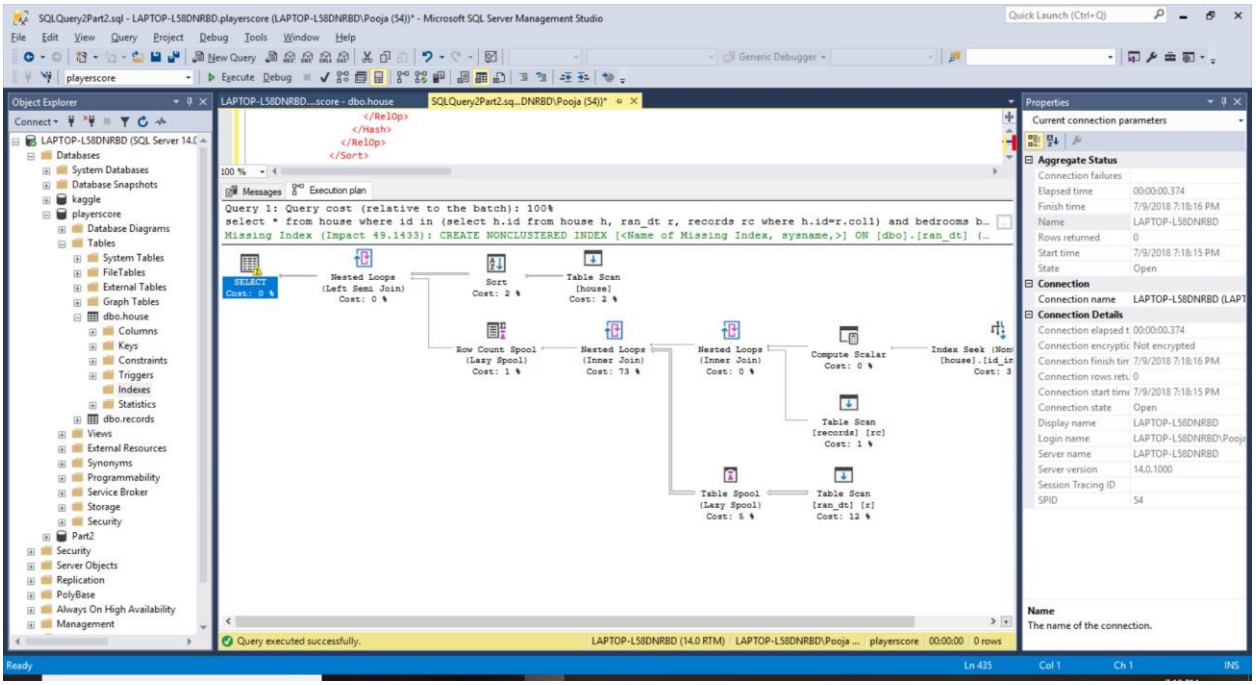


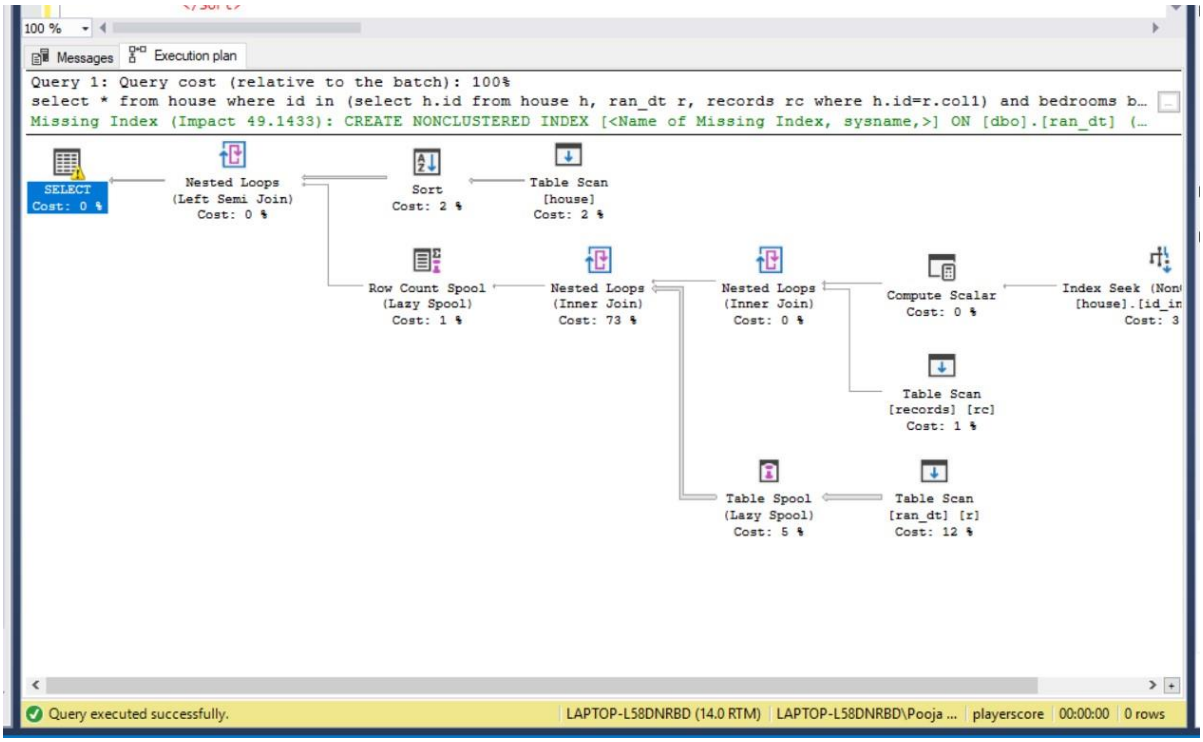
XML Plan2 in query 5-



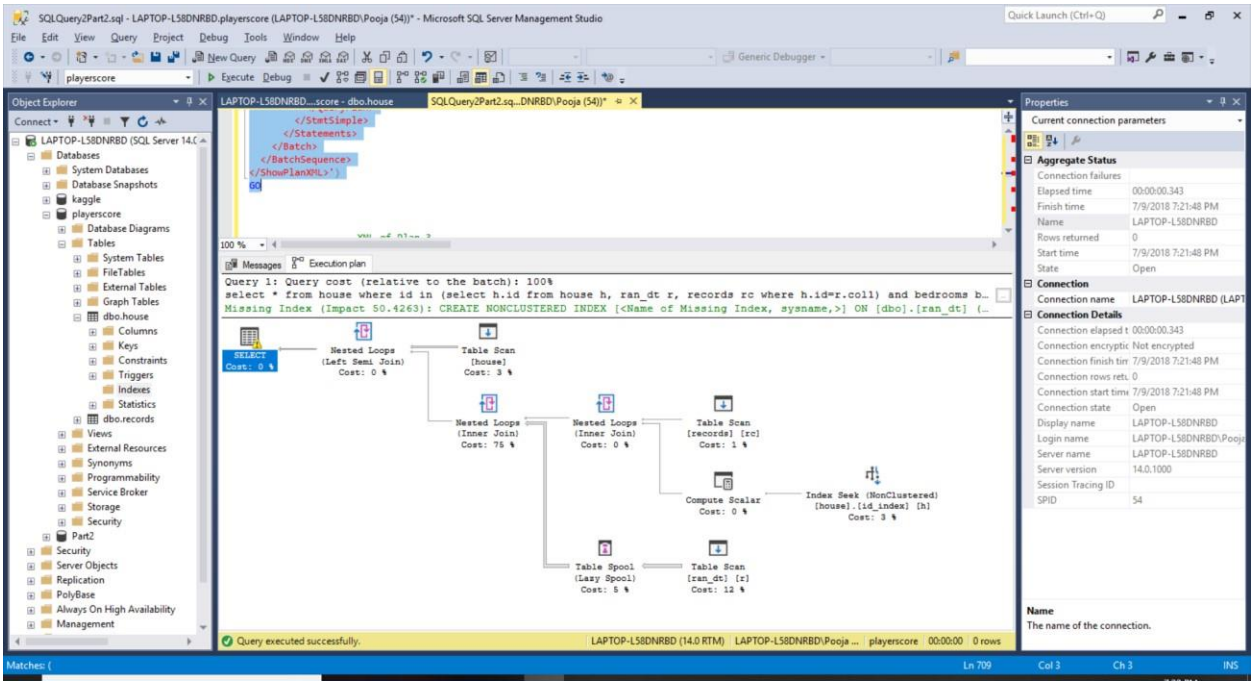


XML Plan 3 in query Plan5-





XML plan 4 in Query plan 5-



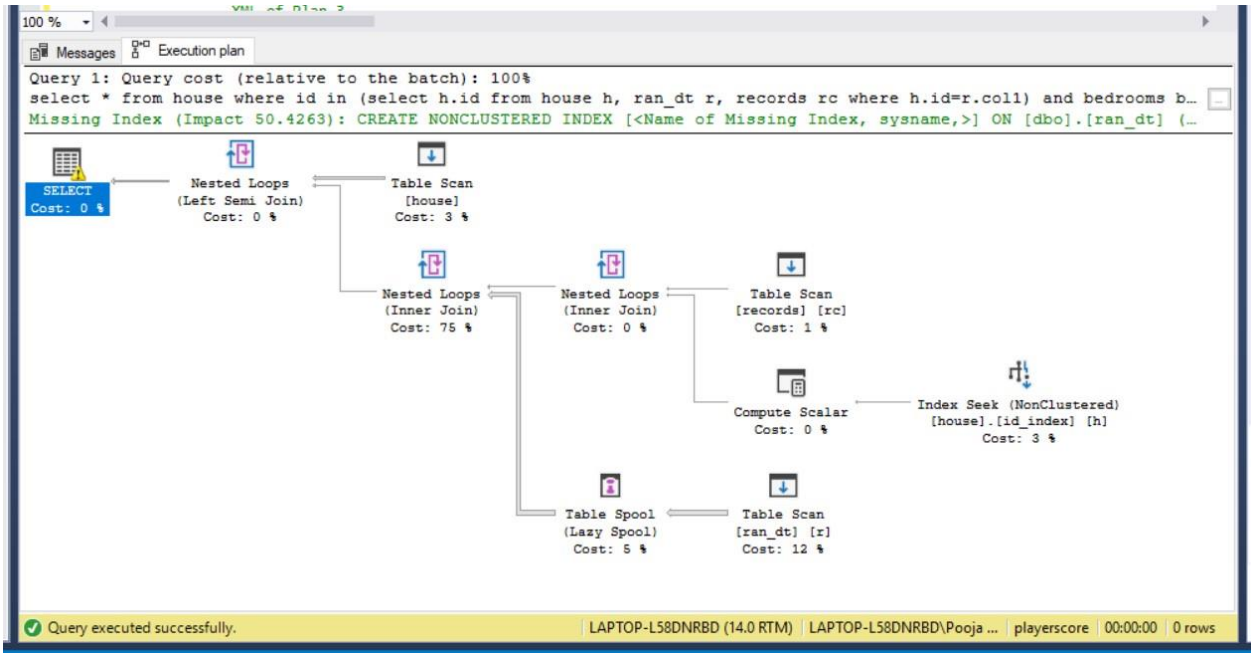


Table of all the XML and Query plans-

a. The table includes the query plans and XML. The query plans are the default query without xml generation with cardinalities changes.
b. The XML column is the xml generation of all the 5 different query plans.
c. The time highlighted in blue is the time taken by the query optimizer for the default runs.

	Query plan 1	Query plan 2	Query plan 3	Query plan 4	Query plan 5
XML 1	318ms	343ms	343ms	328ms	359ms
XML2	343ms	296ms	343ms	343ms	312ms
XML3	328ms	421ms	288ms	343ms	374ms
XML4	343ms	924ms	339ms	296ms	343ms
XML5	318ms	343ms	319ms	374ms	329ms