Dataset Description- I have created two tables and one database. The database name "kaggle'', it is real data of housing. I have a **table1** name **"house"** that includes 18 different Columns naming **"id, date, price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, condition, grade, sqft_above, sqft_basement, yr_built, yr_renovated, zipcode, latitude and longitude"**. The number of rows in the table house is **"2,1614"**. So, the dataset is **21614 * 18**.

Table 2- The second table is **"random_dataset"**, this table is generated by random number, it has 8 different columns naming **"col1, col2, col3, col3, col4, col5, col6, col7 and col8"** and number of rows are "1,165,680".

Observations- I was trying to generate 2,000,000 rows and it took more than 12 hours to generate. So, I stopped it and see the result it has generated **1,165,680** rows.

Generating Query-

```
CREATE Table random_dataset
(    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 = 1
While @Id <= 2000000 Begin
   Insert Into random_dataset 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 Schema

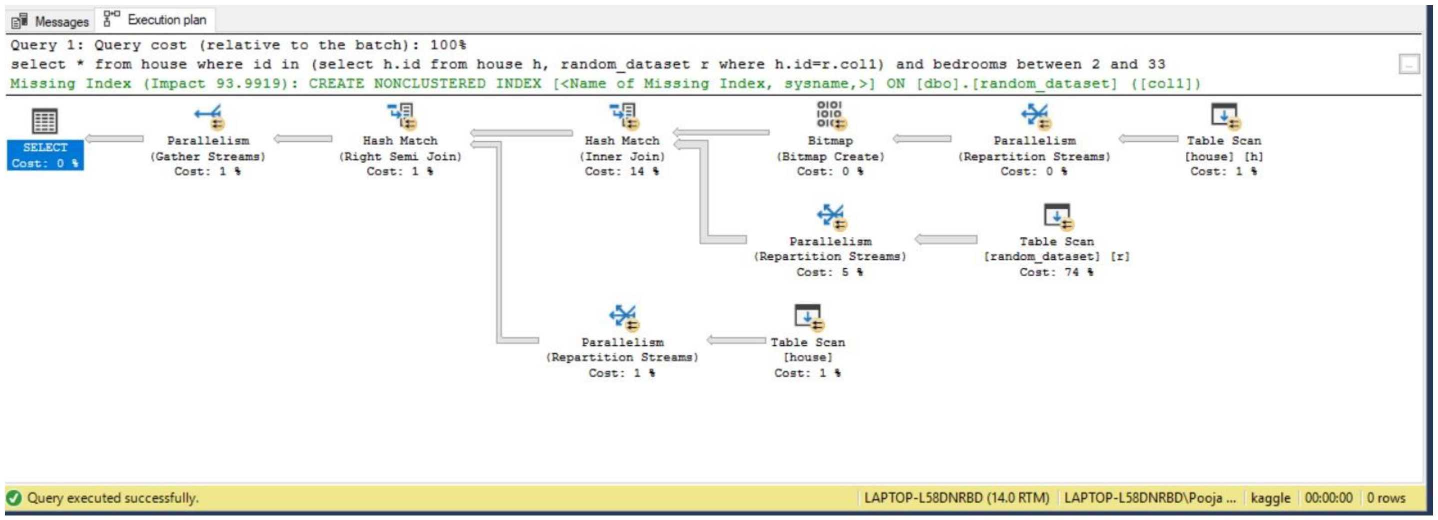
The Query used to generate these plans-

```
select * from house where id in (select h.id from house h, random_dataset r where
h.id=r.col1) and bedrooms between 2 and 33;
```

Below are the 5 different Query optimizations with respect to change in Schema-

1. Without index- Query plan without Index.

Observation- This query plans includes all the results of table house and random_dataset.

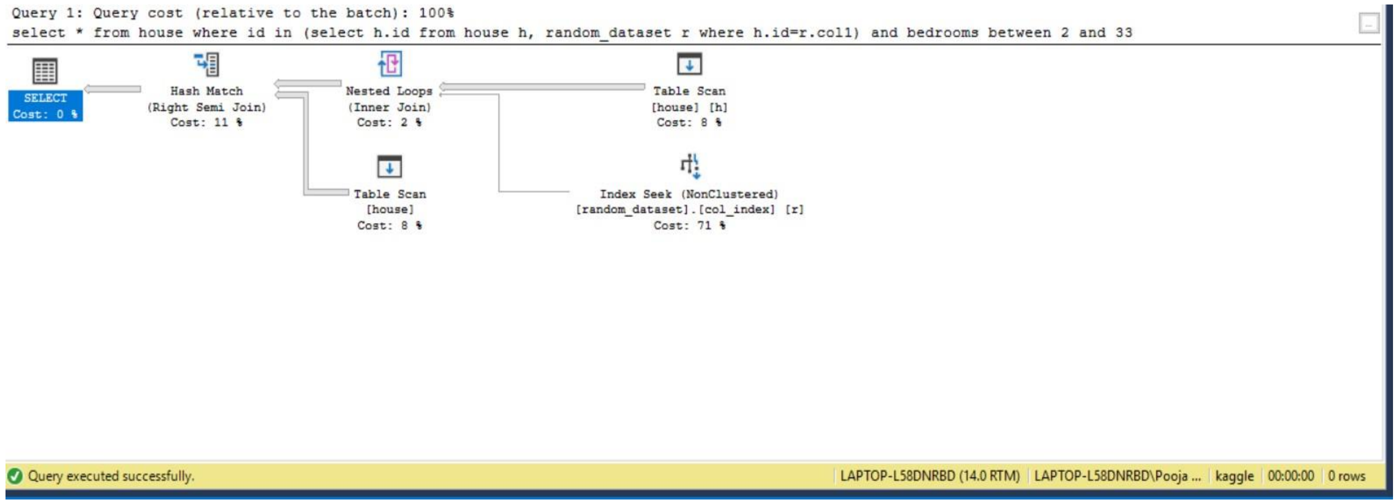


```
Query 1: Query cost (relative to the batch): 100%
select * from house where id in (select h.id from house h, random_dataset r where h.id=r.col1) and bedrooms between 2 and 33
Missing Index (Impact 93.9919): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[random_dataset] ([col1])
```

```
Query executed successfully.    LAPTOP-L58DNRBD (14.0 RTM) | LAPTOP-L58DNRBD\Pooja ... | kaggle | 00:00:00 | 0 rows
```

2. With non-clustered index on the Col1 in Random_dataset
    a. `CREATE INDEX col_index ON random_dataset(col1);`

Observation- In this query with non-clustered index it performs Index seek.

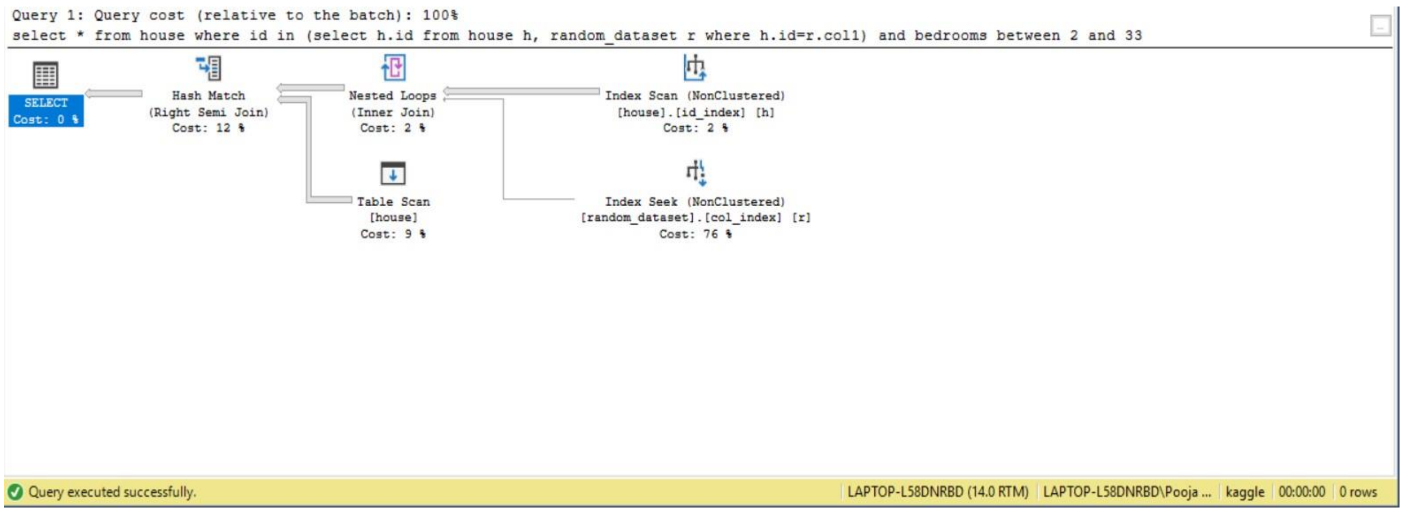


```
Query 1: Query cost (relative to the batch): 100%
select * from house where id in (select h.id from house h, random_dataset r where h.id=r.col1) and bedrooms between 2 and 33
```

```
Query executed successfully.    LAPTOP-L58DNRBD (14.0 RTM) | LAPTOP-L58DNRBD\Pooja ... | kaggle | 00:00:00 | 0 rows
```

3. With non-clustered index on the Col1 in Random_dataset and id in House.
    a. `CREATE INDEX id_index ON house(id);`
    b. `CREATE INDEX col_index ON random_dataset(col1);`

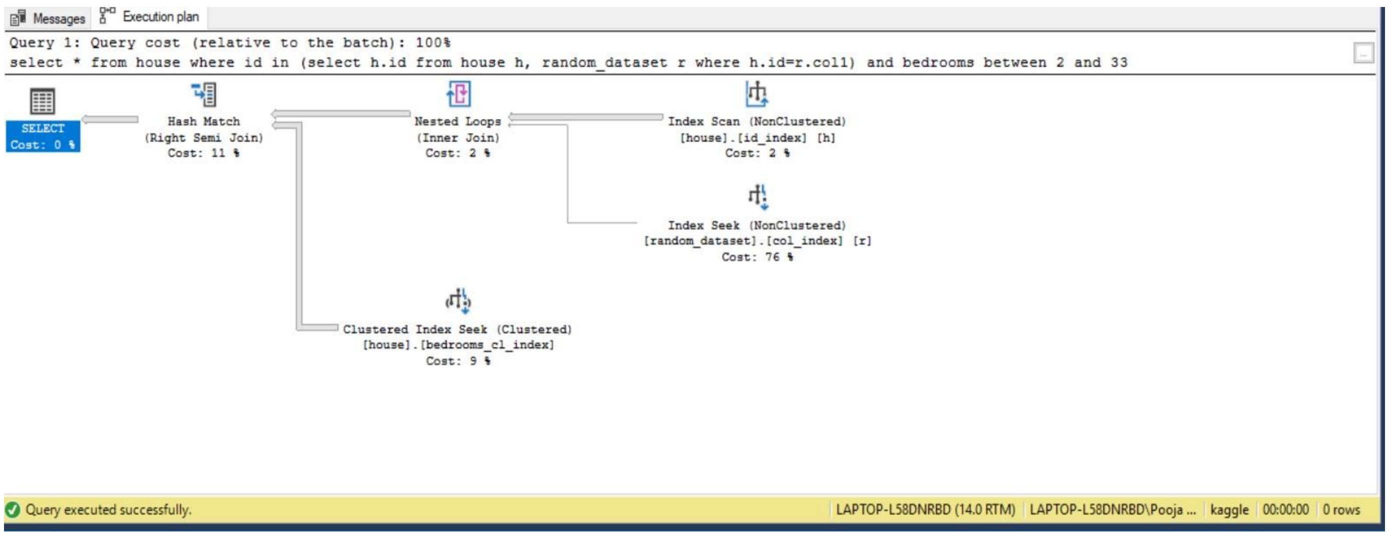
Observation- In this query for creating index it performs Index scan on house(id) & Index seek on random_dataset on col1.


```
Query 1: Query cost (relative to the batch): 100%
select * from house where id in (select h.id from house h, random_dataset r where h.id=r.col1) and bedrooms between 2 and 33
```

```
Query executed successfully.    LAPTOP-L58DNRBD (14.0 RTM) | LAPTOP-L58DNRBD\Pooja ... | kaggle | 00:00:00 | 0 rows
```

4. With non-clustered index on the Col1 in Random_dataset and id in House. Also having the Clustered index on the bedrooms in House.
   a. `CREATE INDEX col1_index ON random_dataset(col1);`
   b. `CREATE INDEX id_index ON house(id);`
   c. `CREATE clustered INDEX bedrooms_cl_index ON house(bedrooms);`

Observation- For creating Clustered Index it performs Index seek.



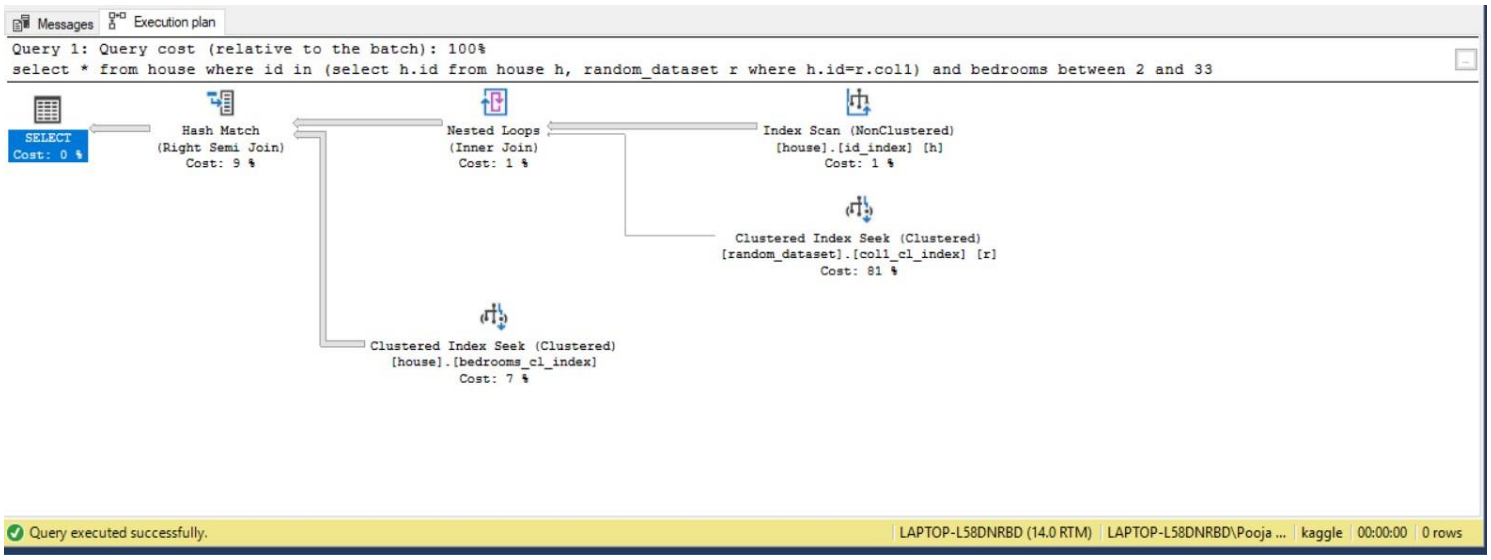
```
Query 1: Query cost (relative to the batch): 100%
select * from house where id in (select h.id from house h, random_dataset r where h.id=r.col1) and bedrooms between 2 and 33
```

SELECT
Cost: 0 %
Hash Match
(Right Semi Join)
Cost: 11 %
Nested Loops
(Inner Join)
Cost: 2 %
Index Scan (NonClustered)
[house].[id_index] [h]
Cost: 2 %
Index Seek (NonClustered)
[random_dataset].[col_index] [r]
Cost: 76 %
Clustered Index Seek (Clustered)
[house].[bedrooms_cl_index]
Cost: 9 %

```
Query executed successfully.    LAPTOP-L58DNRBD (14.0 RTM) | LAPTOP-L58DNRBD\Pooja ... | kaggle | 00:00:00 | 0 rows
```

5. With non-clustered index on id in House. And having the Clustered index on the bedrooms in House and clustered index on Col1 in Random dataset.
   a. `CREATE clustered INDEX col1_cl_index ON random_dataset(col1);`
   b. `CREATE INDEX id_index ON house(id);`
   c. `CREATE clustered INDEX bedrooms_cl_index ON house(bedrooms);`

Observation- Here it creates clustered Index Seek.



```
Query 1: Query cost (relative to the batch): 100%
select * from house where id in (select h.id from house h, random_dataset r where h.id=r.col1) and bedrooms between 2 and 33
```

SELECT
Cost: 0 %
Hash Match
(Right Semi Join)
Cost: 9 %
Nested Loops
(Inner Join)
Cost: 1 %
Index Scan (NonClustered)
[house].[id_index] [h]
Cost: 1 %
Clustered Index Seek (Clustered)
[random_dataset].[col1_cl_index] [r]
Cost: 81 %
Clustered Index Seek (Clustered)
[house].[bedrooms_cl_index]
Cost: 7 %

Query executed successfully. | LAPTOP-L58DNRBD (14.0 RTM) | LAPTOP-L58DNRBD\Pooja ... | kaggle | 00:00:00 | 0 rows