TCSS 564 Name- Blaise Dmello

Schema For the Analysis:

Address(A_ID, Apt_house_no, Street_Name, City, Zipcode)

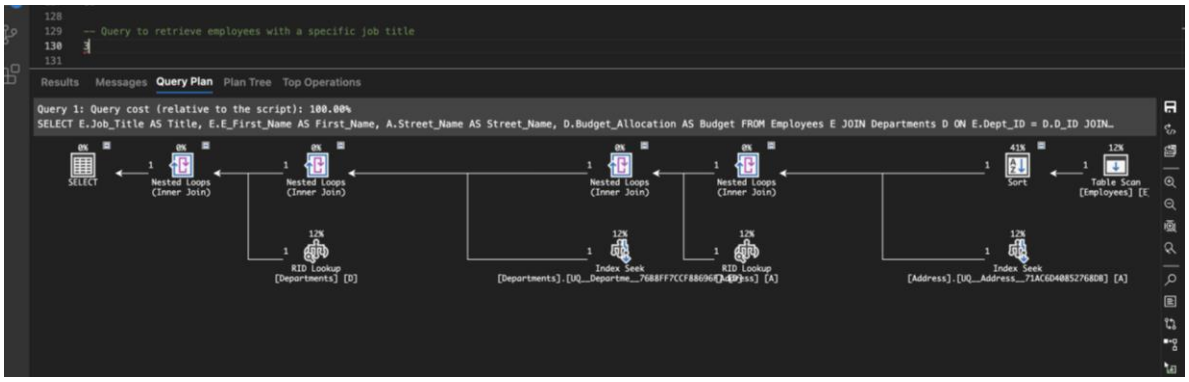
Departments(D_ID, D_Name, Office_Sec, Budget_Allocation)

Employees(EID, E_First_Name, E_Last_Name, Email, Phone, Job_Title, Dept_ID, Address_ID, Salary)

Query To Be Tested:

```
SELECT
        E.Job_Title AS Title,
        E.E_First_Name AS First_Name,
        A.Street_Name AS Street_Name,
        D.Budget_Allocation AS Budget
FROM
        Employees E
JOIN
        Departments D ON E.Dept_ID = D.D_ID
JOIN
        dbo.Address A ON E.Address_ID = A.A_ID
WHERE
        E.Job_Title = 'Software Engineer' OR
        D.D_ID < 5 OR
        A.Street_Name LIKE 'M%' OR
        D.Budget_Allocation > 1000000
ORDER BY
        E.E_First_Name ASC;
```

1. Initial Query Plan.



Here We can see the Query plan for the above-mentioned query on a plain table with no indexes although we haven't explicitly mentioned it, the ID fields of each of these table have a UNIQUE Constraint which acts as a form of index thus we can see that 2 of the iterators having index seeks instead of Table Scans. Which is due to the fact that scans take up O(n) time/operations whereas seeks take O(log(n)) time/operations. We are also performing RID lookups on tables Departments and Address, this is due to the lack of indexes on those tables and hence there not a B+ tree constructed being any particular order they are arranged in so seeking on indexes with the RID becomes the viable scenario in this instance.

2. After Creating Clustered Indexes On Each of The Tables (i.e. Primary Keys).

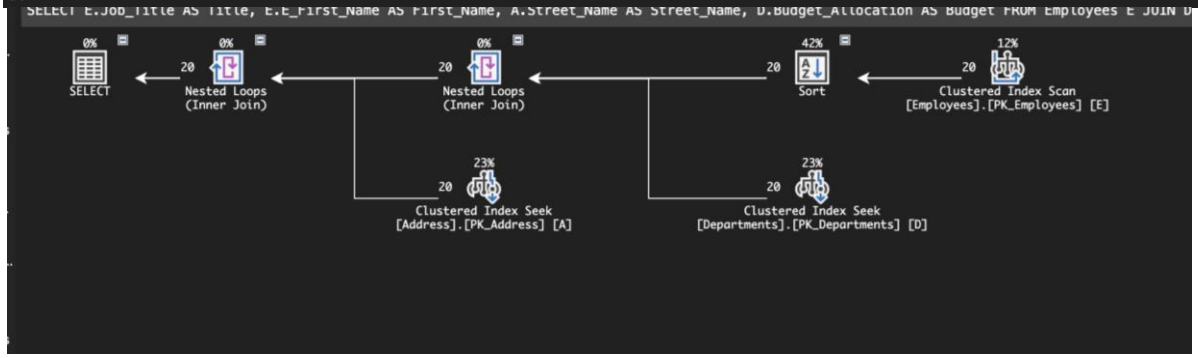
```
-- Adding Primary Key Constarints to Address Table
ALTER TABLE dbo.Address
ADD CONSTRAINT PK_Address PRIMARY KEY (A_ID);
GO

-- Adding Primary Key Constraint to Departments Table
ALTER TABLE Departments
ADD CONSTRAINT PK_Departments PRIMARY KEY (D_ID);
GO

-- Adding Primary Key Constraint to Employees Table
ALTER TABLE Employees
ADD CONSTRAINT PK_Employees PRIMARY KEY (EID);
GO

-- Linking Employees Table and Address Table Via Foreign Key In Employees Table
ALTER TABLE Employees
ADD CONSTRAINT FK_Address
FOREIGN KEY (Address_ID)
REFERENCES dbo.Address(A_ID);
GO

-- Linking Employees Table and Departments Table Via Foreign Key In Employees Table
ALTER TABLE Employees
ADD CONSTRAINT FK_Department
FOREIGN KEY (Dept_ID)
REFERENCES Departments(D_ID);
GO
```

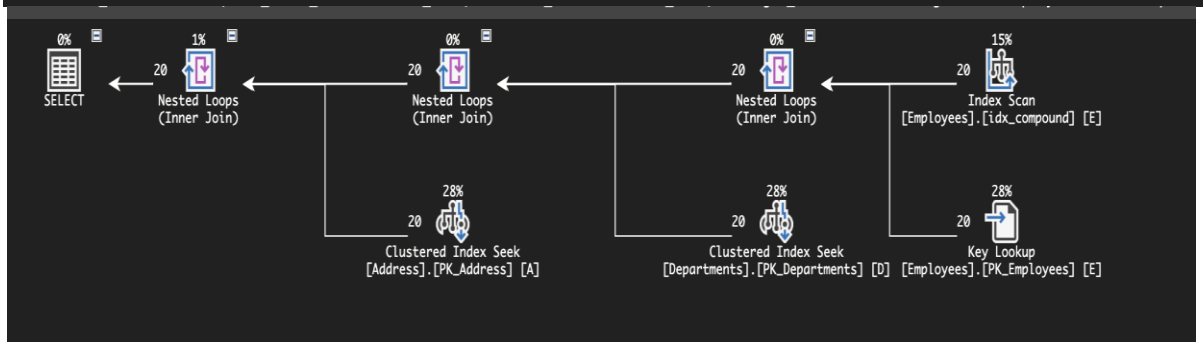


Above we've altered the tables to have primary keys which in turn creates Clustered Indexes on each of the IDs. The right most iterator which was previously a Table scan changed to a Clustered index scan as the Employees table will be physically sorted in accordance with the index created on E_ID. Furthermore, the 2 iterators at the bottom operating on Departments and Adress tables switched to a clustered index seek while also loss of the RID lookup iterators from the previous query plan. This is due to the fact that there is a clustered index physically sorting the Address and Departments table in ASC order and hence there being no need for a lookup just a Clustered Index Seek of O(log(n)) operation/time.

3. After Creating Non-Clustered Indexes On Employees Tables Column E_First_Name.

```
-- Create Unclustered Index On Employees Table
```

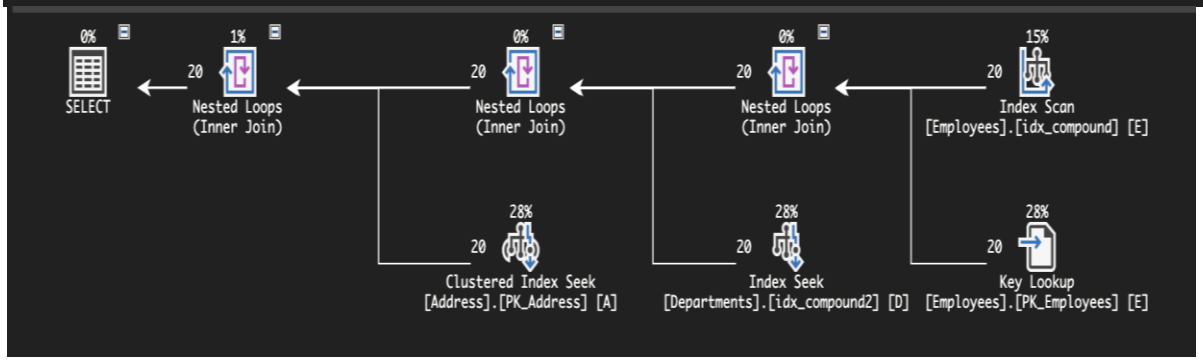
```
CREATE INDEX idx_compound ON Employees(E_First_Name);
GO
```



After creating a non-clustered index on Employees table, we can see that the clustered index scan from previous query plan changed to non-clustered index scan and a key lookup. This is due to the fact that scanning the entire Employees table is expensive given that it has a lot of columns. The fact that the query also doesn't require doesn't help its cause, so it makes sense that the Query optimizer chose to do a non-clustered index scan to select limited no. Of columns to reduce page/disk accesses and looking up any additional missing column required using the key lookup iterator.

4. After Creating Non-Clustered Indexes On Departments Tables Columns (D_ID, Budget_Allocation).

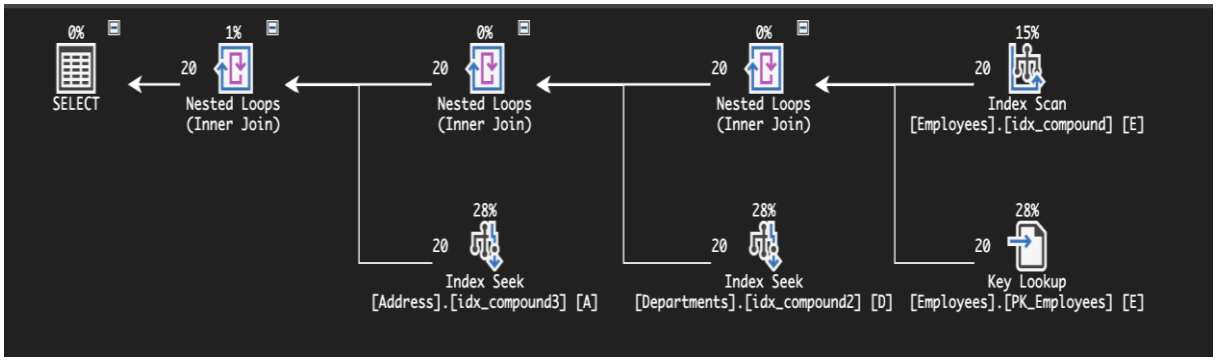
```
-- Create Unclustered Index On Departments Table
CREATE INDEX idx_compound2 ON Departments(D_ID,Budget_Allocation);
GO
```



Additionally, creating non clustered index on Departments table changes the respective iterator from Clustered Index seek to just index seek. This is due to the fact that even though both are O(log(n)) operation/time the 'n' of clustered index which we will denote as N is greater than the 'n' of non-clustered index i.e. N > n. This is due to the extra unecessary columns accessed in the clustered index thus requiring more resources.

5. After Creating Non-Clustered Indexes On Address Tables Columns (A_ID, Street_Name).

```
-- Create Unclustered Index On Address Table
CREATE INDEX idx_compound3 ON dbo.Address(A_ID,Street_Name);
GO
```



Here we can see a similar situation as before, the iterator that previously performing clustered index seek on Address table switched to non-clustered index seek due to the size difference of N > n and thus reducing the operation and time complexity from O(log(N)) to O(log(n)) where N > n due to the size of the columns and disk accesses required.

Estimated SubTree Cost & Elapsed Time

	Subtree Costs	Elapsed Time (ms)
No Indexes	**0.0836**	**1ms**
Clustered Indexes on Each of the Tables	**0.074**	**0.3ms**
Non-Clustered Index on Employees	**0.054**	**0.02ms**
Non-Clustered Index on Department	**0.077**	**1ms**
Non-Clustered Index on Address	**0.109**	**5ms**