1. Items: 160000 Tuples / 1600 Pages

Order: 200000 Tuples / 2000 Pages

Let R be the outer relation and S be the inner relation

1. Cost (Page-Oriented NLJ) = NPages(R) + NPages(R) \* NPages(S)

= 1600 + 1600 \* 2000

= 3201600 I/O

1. Cost (Block-Oriented NLJ) = NPages(R) + ceil(NPages(R) / (Buffer - 2)) \* NPages(S)

= 1600 + (1600/800) \* 2000

= 5600 I/O

1. Cost (SMJ) = NPages(R) + NPages(S) + 2 \* NPages(R) \* num\_passes(R) + 2 \* NPages(S) \*

num\_passes(S)

= 1600 + 2000 + 2 \* 1600 \* 2 + 2 \* 2000 \* 2

= 18000 I/O

1. Cost (Hash Join) = 3 \* (NPages(R) + NPages(S))

= 3 \* (1600 + 2000)

= 10800 I/O

1. The lowest possible cost to perform this query would be if each relation was read only once, and we could do this by storing the entire smaller relation in memory, and then reading the larger relation page by page, where for each tuple in the larger relation, we would search for the matching tuple in the smaller relation, which already exists on memory and would therefore not require additional costs.

Cost = NPages(R) + NPages(S)

= 1600 + 2000

= 3600 I/O

To do this, the buffer space must be big enough to hold the entire smaller relation as well as one additional page to hold the page we are reading from the larger relation and also another for the output buffer.

Buffer = min(NPages(R), NPages(S)) + 1 + 1

= min(1600, 2000) + 2

= 1602 Buffer pages

1. Student: 64000 Tuples / 800 Pages
2. RFfaculty = 1 / NKeys(Col)

= 1 / 10

= 0.1

RFwam = (High(Col) – val) / (High(Col) – Low(Col))

= (100 – 75) / (100 – 0)

= 0.25

Result size = π NTuples(Ri) \* π RFi

= 64000 \* (0.1 \* 0.25)

= 1600 Tuples

= 20 Pages

1. Cost (Clustered B+ Tree Index) = (NPages(I)+NPages(R)) \* π RFi

= (120 + 800) \* 0.1 \* 0.25

= 23 I/O

Cost (Full Table Scan) = NPages(R)

= 800 I/O

Therefore, cost of best plan would be to utilise the index, with cost of 23 I/O. This is because a clustered index is the best tool to use with a range query, and with the index being on both keys, we would be able to find the pages we need without needing to do any on-the-fly selections.

1. Cost (Unclustered B+ Tree Index) = (NPages(I)+NTuples(R)) \* π RFi

= (60 + 64000) \* 0.25

= 16015 I/O

Cost (Full Table Scan) = NPages(R)

= 800 I/O

Therefore, cost of best plan would be to perform a full table scan, with cost of 800 I/O. This is because an unclustered index is not optimal on range queries, as we would have to access every single tuple due to the lack of order among the tuples.

1. Cost (Unclustered Hash Index) = NTuples(R) \* π RFi \* 2.2

= 64000 \* 0.1 \* 2.2

= 14080 I/O

Cost (Full Table Scan) = NPages(R)

= 800 I/O

Therefore, cost of best plan would be to perform a full table scan, with cost of 800 I/O. This is because any unclustered index is not suitable for searching. Unclustered indexes should be used when there is more insertion required with little to no searching.

1. Cost (Full Table Scan) = NPages(R)

= 800 I/O

Therefore, cost of best plan would be to perform a full table scan, with cost of 800 I/O. This is because we cannot use the index available as range queries cannot be applied to hash indexes.

1. Customer: 2000 Tuples / 20 Pages

Order: 8000 Tuples / 80 Pages

OrderItem: 80000 Tuples / 800 Pages

1. RFcusid = 1 / max(NKeys(Col\_A), NKeys(Col\_B))

= 1 / 2000

= 0.0005

RForderid = 1 / max(NKeys(Col\_A), NKeys(Col\_B))

= 1 / 8000

= 0.000125

RFquantity = (val – Low(Col)) / (High(Col) – Low(Col))

= (10 – 0) / (50 – 0)

= 0.2

RFitemname = 1 / NKeys(Col)

= 1/400

= 0.0025

Result size = π NTuples(Ri) \* π RFi

= (2000 \* 8000 \* 80000) \* (0.0005 \* 0.000125 \* 0.2 \* 0.0025)

= 40 Tuples

= 1 Page

1. Cost (Customer JOIN Order) = NPages(R) + NPages(R) \* NPages(S)

= 20 + (20 \* 80)

= 1620 I/O

Result Size (Customer JOIN Order) = π NTuples(Ri) \* π RFi

= (2000 \* 8000) \* 0.0005

= 8000 Tuples

= 80 Pages

Cost (JOIN OrderItem) = NPages(R) \* NPages(S)

= 80 \* 800

= 64000 I/O

Total Cost = 1620 + 64000

= 65620 I/O

1. Cost (Order JOIN OrderItem) = 3 \* (NPages(R) + NPages(S))

= 3 \* (80 + 800)

= 2640 I/O

Result Size (Order JOIN OrderItem) = π NTuples(Ri) \* π RFi

= (8000 \* 80000) \* 0.000125

= 80000 Tuples

= 800 Pages

Cost (JOIN Customer) = NPages(S) + 2 \* NPages(R) \* num\_passes(R) + 2 \* NPages(S) \*

num\_passes(S)

= 20 + 2 \* 800 \* 2 + 2 \* 20 \* 2

= 3300 I/O

Total Cost = 2640 + 3300

= 5940 I/O

1. Cost (OrderItem JOIN Order) = NPages(R) + 2 \* NPages(R) \* num\_passes(R) +

(NPages(S) + NPages(I.S))

= 800 + 2 \* 800 \* 2 + 80 + 20

= 4100 I/O

Result Size (OrderItem JOIN Order) = π NTuples(Ri) \* π RFi

= (8000 \* 80000) \* 0.000125

= 80000 Tuples

= 800 Pages

Cost (JOIN Customer) = 2 \* NPages(R) + 3 \* NPages(S)

= 2 \* 800 + 3 \* 20

= 1660 I/O

Total Cost = 4100 + 1660

= 5760 I/O

1. Cost (SELECTION Quantity > 10) = (NPages(I)+NPages(R)) \* π RFi

= (200 + 800) \* 0.2

= 200 I/O

Result Size (SELECTION Quantity > 10) = NTuples(R) \* π RFi

= 80000 \* 0.2

= 16000 Tuples

= 160 Pages

Cost (JOIN Order) = NPages(R) \* NPages(S)

= 160 \* 80

= 12800 I/O

Result Size (JOIN Order) = π NTuples(Ri) \* π RFi

= (8000 \* 16000) \* 0.000125

= 16000 Tuples

= 160 Pages

Cost (JOIN OrderItems) = 2 \* NPages(R) + 3 \* NPages(S)

= 2 \* 160 + 3 \* 20

= 380 I/O

Total Cost = 200 + 12800 + 380 = 13380 I/O