

Sheet 6

Name: Asmaa Gamal Abdel-Haleem Mabrouk Nagy

Course: Database « Audience Course »

Department: Communications & Electronics

DBMS

Query Processing & External Sorting

1) a) 1- Conjunctive Selection:

Condition	AND	Condition
-----------	-----	-----------

ex

SELECT * FROM EMPLOYEE

WHERE Department = 'IT' AND Salary > 5000;

2- Disjunctive Selection:

{ }	OR	{ }
-----	----	-----

ex

SELECT * FROM PRODUCT

WHERE Category = 'Electronics' OR Price < 1000;

b) Multiple options for Execution:

* index Scan] or [both
* table "] or [both

∴ while < conjunctive > Selection queries
disjunctive

express logical conditions to filter data, the actual execution can vary based on the data engine's optimization, indices, & information.

2 Alternative ways:

① * "GROUP BY": This query groups the data by the specified attribute and provides the count of occurrences for each unique value.

② * Aggregate Fun:

ex: MIN(), MAX(), AVG()
to get info about each unique value.

③ * window Fun: ex: ROW-NUMBER()
to assign a unique num to each row within a Partition, then, you can filter based on this num.

④ * JOIN.

3 Implementation Steps:

- ① Scan the Data
- ② Grouping (if required)
- ③ initialization of aggregate variables
- ④ iterating
- ⑤ update the " "
- ⑥ Finalizing & outputting Results.
- ⑦ Optimization using
 - index usage
 - caching
 - Parallel Processing
 - Query optimization

4 * Outer Join implementation:

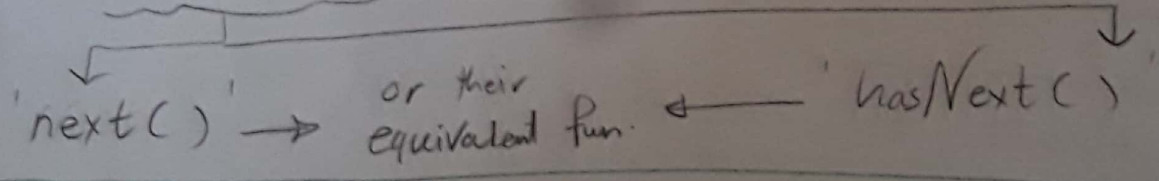
- 1- Nested Loop Join.
- 2- Hash Join.
- 3- Merge Join.
- 4- Left & right outer Join.

* Non-Equi Join:

- 1- theta Join
 - 2- Nested Loop Join with Non-Equi Condition.
 - 3- Hash-Join
 - 4- Merge-Join
 - 5- Filtering Post Join
- With " " "

5 * iterator concept: is a Programming Pattern to traverse a Container without exposing the underlying details of its implementation

* iterator Methods:



6 * sparse index "non-dense" can be used to implement aggregate operator, but there are

Some issues:

- ① incomplete coverage
- ② Multiple index look ups [I/O] ↑
- ③ Optimization Challenges.

Steps of LEFT OUTER JOIN using Sort-Merge Join algorithm

4

7

- 1- Sort the left & right table based on 'join-key'
- 2- initialize Pointers. 'left Pointer = 0', 'right Pointer = 0'
- 3- Perform merge
- 4- Process Remaining Rows in the Left table

Using external Sorting :

a 10,000 Page & 3 buffers

$$(i) \# \text{ runs}_{1st \text{ Pass}} = \left\lceil \frac{\text{tot Pages}}{\text{Buffer Pages}} \right\rceil = \left\lceil \frac{10,000}{3} \right\rceil = 3,333 \text{ runs}$$

$$(ii) \# \text{ Passes} = \log_{B-1} R = \log_2 3333 \approx 11 \text{ Passes}$$

$\begin{array}{c} B-1 \\ \downarrow \\ \text{buffer} \end{array}$
 $\begin{array}{c} R \\ \downarrow \\ \text{Runs} \end{array}$

$$(iii) \text{ tot I/O Cost} = \# \text{ Pages}_{\text{tot}} * \# \text{ Passes} = 10,000 * 11 = 110,000 \text{ I/O}$$

$$(iv) \# \text{ Buffer Pages in } 2 \text{ Passes} = \# \text{ buffer}_{\text{@each Pass}} * 2 = 6 \text{ buffer Pages}$$

b 20,000 Page & 5 buffers :

$$\# \text{ runs} = \frac{20,000}{5} = 4,000 \text{ runs}$$

$$\# \text{ Passes} = \log_4 4000 \approx 6 \text{ Passes}$$

$$\# \text{ I/O} = 20,000 * 6 = 120,000 \text{ I/O}$$

$$\# \text{ Buffers in } 2 \text{ Passes} = 5 * 2 = 10 \text{ buffers}$$

Continue of Q ⑧:

⑤

2 Million Pages & 17 buffers :

$$\# \text{ runs} = \frac{2\,000\,000}{17} \approx 117647 \text{ runs}$$

$$\# \text{ Passes} = \log_{16} 117647 = 4 \text{ Passes}$$

$$\# \text{ I/O} = 2^M * 4 = 8,000,000 \text{ I/O}$$

$$\# \text{ Buffers in 2 Passes} = 17 * 2 = 34 \text{ buffer}$$