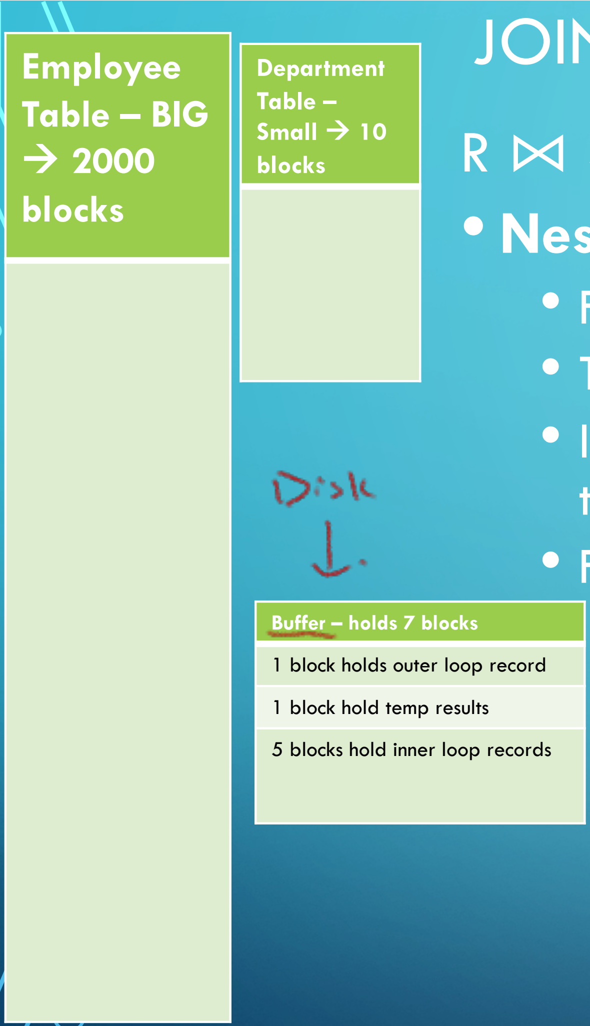
WEEK 9

QUERY OPTIMIZATIONS – JOINS, PROJECTIONS, CARTESIAN PRODUCT

STUDENT OBJECTIVES

- Upon completion of this video, you should be able to:
  - Recognize the order that queries are performed greatly affects the speed of the query.
  - Identify strategies to improve the speed with which joins are performed
  - Identify strategies to speed up projection, difference and aggregate functions
  - Differentiate between pipelining and materialization and list pros and cons of each method

JOIN OPTIMIZING STRATEGIES



Employee Table – BIG → 2000 blocks

Department Table – Small → 10 blocks

Disk

↓

Buffer – holds 7 blocks

1 block holds outer loop record

1 block hold temp results

5 blocks hold inner loop records

R ⋈ S

- **Nested Loop Join (Brute Force):**
  - For each record in R
  - Then check each record in S check to see if this is a match.
  - It makes a difference which table we start with, depending on the number of blocks available for buffering.
  - For example:
    - Suppose that we want the last name and the department name they work in for each of our employees.
    - All of the employee data fits into 2000 blocks and all of the department data fits into 10 blocks.
    - We have 7 blocks for buffering (nB)
    - We will need to use 1 block to hold the outer loop records and 1 block to hold temporary results to write back to the disk
    - So we have 5 blocks to hold the inner loop records.

Employee Table – BIG → 2000 blocks

Department Table – Small → 10 blocks

Buffer – holds 7 blocks

- 1 block holds outer loop record
- 1 block hold temp results
- 5 blocks hold inner loop records

Situation 1:

- Total number of blocks accessed for outer loop records is 2000 blocks

cache .

- Number of blocks for inner loop records is 10 blocks (Total Number of Accesses for inner loop → 10/5 * 2000 = 4000)
- TOTAL Number of block accesses is 2000 + ([10/2] * 2000) = **6000 blocks accesses**

Situation 2 → Switch around:

- Total number of blocks accessed for outer loop records is 10 blocks
- Number of blocks for inner loop 2000 blocks (Total Number of Accesses for inner loop→ 2000/5 * 10 = 4000)
- TOTAL Number of block accesses is 10 + (2000/5*10) = **4010 block accesses**

QUESTION: The above example shows that is it better to use the table that takes up less blocks as the outer loop

SORT OPTIMIZING STRATEGIES

- **Sort-Merge:** Sort both R and S on the join key, find matches in a final pass

- **Use index to find matches:** if index exists for one of two tables say S, then walk through R and use the join key to access S and see if S has a matching tuple for R's tuple (this is a version of nested loops).

- **Hash-Join**: Records of R and S are hashed to the same buckets, then do a single pass through the buckets (all matching values will be in the same buckets)

PROJECTION OPTIMIZING STRATEGIES

- If the key is one of the attributes asked to be retrieved then just write records to file (there will not be any duplicates)
- If the key is **not** one of the attributes and we don't want duplicates:

  (e.g. ***SELECT** salary **FROM employee***)

then do one of:

  - Write all the Salaries to a temp table
  - Sort
  - Scan and eliminate duplicates

OR

- Hash on returned value as inserting into temp table, check if there already, if it is drop duplicates

DIFFERENCE OPTIMIZING STRATEGIES

- Sort both tables, and detect differences on a final pass, e,g, show all the pets that aren’t cats or snakes.

PetType	PetName	PetID
Bird	Tweety	12
Cat	Tiger	33
Cat	Fluffy	44
Dog	Rover	22
Snake	Twisty	32

PetType	PetName	PetID
Cat	Tiger	33
Cat	Fluffy	44
Snake	Twisty	32

PetType	PetName	PetID
Bird	Tweety	12
Dog	Rover	22

- NOTE: you can use this method for UNION and INTERSECTION also!

AGGREGATE OPTIMIZING STRATEGIES

- Aggregate operators like: ***Min, Max, Count, Sum, Average***
- Example: Suppose you had:

SELECT MAX(salary) FROM employee

and we have an ascending index on ***salary***.

QUESTION: What is the fastest choice to answer this query?

ANSWER: Take the last value in the index (don't even need to look at the table!)

QUESTION: Suppose we have:

SELECT SUM(salary) FROM employee

and we have a ~~non-~~dense index on *salary*, can we use the index to work out the query?

Could we use the index if it was ~~dense~~ Nondense?

- If you have a GROUP BY clause in your SQL, like:

SELECT dno, AVG(salary) FROM employee GROUP BY dno

the usual technique is to first sort the table on the grouping attribute, then compute for each group (if we have a clustering index on the group attribute→ EVEN BETTER).

CARTESIAN PRODUCT OPTIMIZING STRATEGIES

- Just don't do it!

PIPELINING VS. MATERIALIZATION

- **Pipelining**: as the resulting tuples of operation are produced, they are forwarded directly to the next operation – LIMITED BY BUFFER SPACE but can improve performance.

  result is not on disk. it is on BUFFER.

- **Materialization**: the results of an operation are stored in a temporary relation - could be time consuming because you have to do a write to disk! (also, sometimes unnecessary as you are going to use this file immediately anyways)