# Query Processing Part 1: A Simplified Cost Model and the Scan Operator

**Abdu** Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

March 24, 2024

# Leaning Objectives

After this lecture, you should be able to:

- Explain the big picture of Query Processing and Optimization
- Discuss the difference between physical and logical operators
- Explain the basic model of costing for physical operators
- Reason about the different implementations of scan operators and their cost/memory requirements

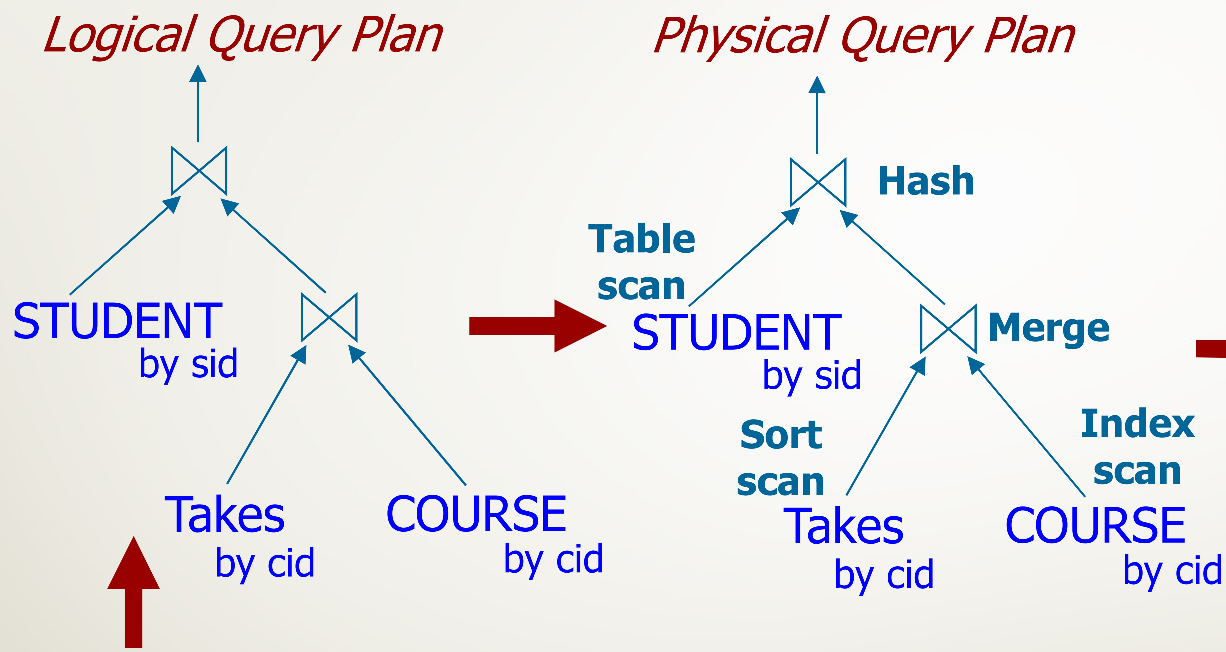
# Today's Lecture!

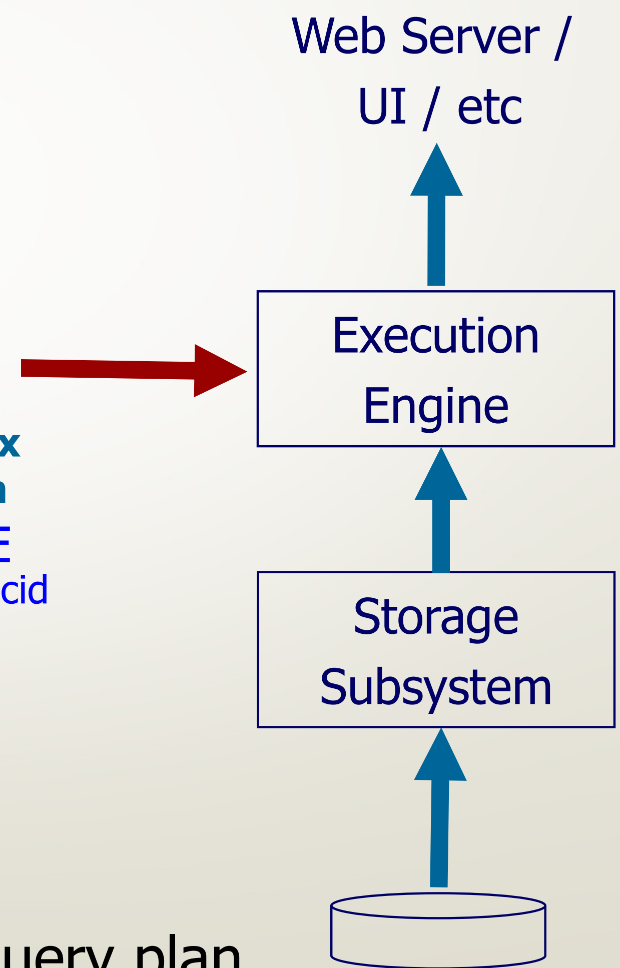
- Query Execution: The Big Picture
  - Basics of Costing
  - Classifications of Physical Operators
- Scan Operator
  - Table-scan, index-scan and sort-scan
  - Costs for Scans

## RECAP: Codd's Logical Operations: The Relational Algebra

- Six basic operations:
  - Projection $\pi_{\alpha}(R)$
  - Selection $\sigma_{\theta}(R)$
  - (Rename) $\rho_{\alpha}(R)$
  - Union $R_1 \cup R_2$
  - Difference $R_1 - R_2$
  - Product $R_1 \times R_2$
- And some other useful ones:
  - Join $R_1 \bowtie_{\theta} R_2$
  - Intersection $R_1 \cap R_2$

# QP: The Big Picture: SQL → Logical Query Plan → Physical Query Plan




```
SELECT *
FROM STUDENT, Takes, COURSE
WHERE STUDENT.sid = Takes.sid
            AND Takes.cid = COURSE.cid
```

Today's agenda: operations in the physical query plan

# Logical v.s. Physical Operators

- Logical operators
  - *what* they do
  - e.g., union, selection, project, join, grouping

- Physical operators
  - *how* they do it
  - e.g., nested loop/sort-merge/hash/index join
  - In other words, physical operators are particular implementations of relational algebra operations
  - Physical operators also pertain to none RA operations, such as “scanning” a table.

# Physical operators and costs

Before we describe implementations, we need to examine the *cost* involved

- Often, we have to make choices about which physical operators to use.
- For this, we need to estimate the "cost" of each physical operator.

# Cost Parameters (or “statistics”)

Estimating the cost:

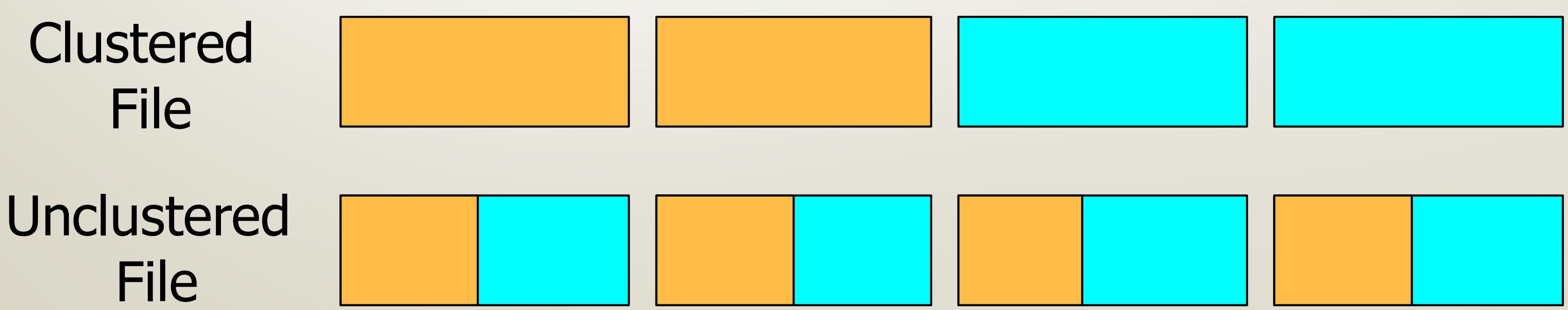
- Important in optimization (next topic: Qry Opt.)
  - Picking between plans
- Compute disk I/O cost only
  - Main memory operations are cheap
- We compute the cost to *read* the arguments of the operator
- We don’t compute the cost to *write* the “final” result
  - Same for all types of operators

# Cost Parameters (or "statistics")

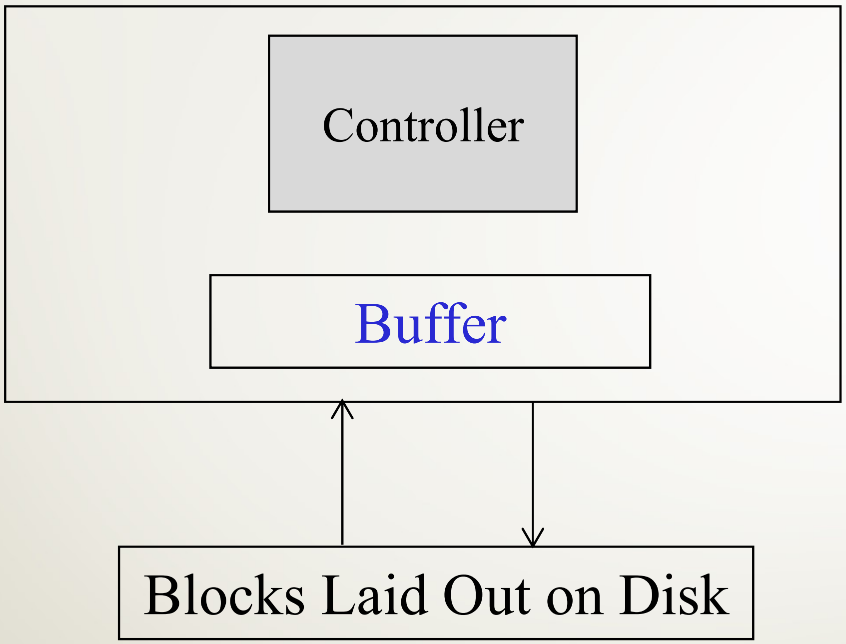
- Cost parameters
  - M = number of blocks that fit in main memory
  - B(R) = number of blocks needed to hold R (best case # blocks)
  - T(R) = number of tuples in R (worst case # of blocks)
  - V(R,a) = number of distinct values of the attribute a

T(R) / (number of tuples that fits in a block) = B (R)

Here T(R) = 4; B(R) = 2



# A Mental Model for Costing Operator Implementations




Temp computation space in main memory

We compute the cost to *read the arguments* of the operator
We don't compute the cost to *write the result*

# Think-Pair-Share

In general, which file organization would be more efficient in terms of disk I/Os

**A: Clustered file**

**B: Unclustered file**

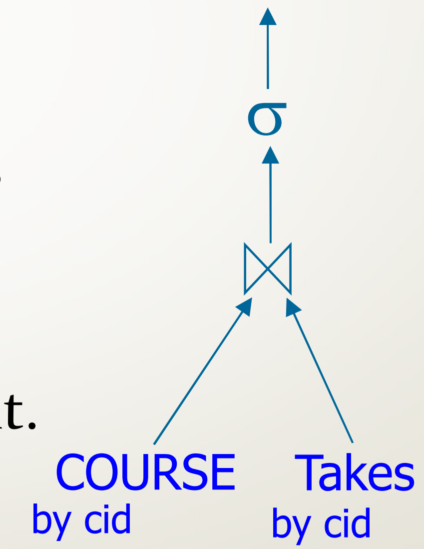
# Outline

- Query Execution: The Big Picture
  - ✓ Basics of Costing
  - Classifications of Physical Operators
- Scan Operator
  - Table-scan, index-scan and sort-scan
  - Costs for Scans
- Nested-loop joins

## The iterator model for implementing operators

Each (physical) operation is implemented by 3 functions:

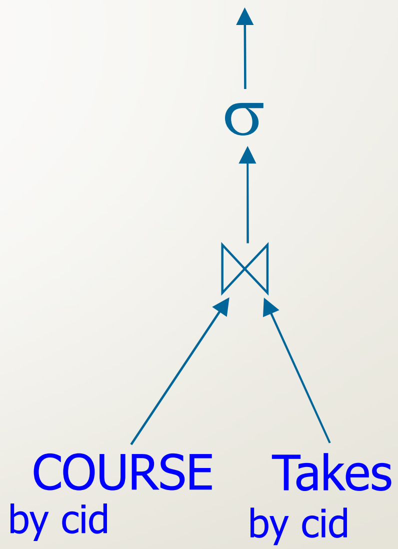
- **Open**: sets up the data structures and performs initializations
- **GetNext**: returns the the next tuple of the result.
- **Close**: ends the operations. Cleans up the data structures.



# The iterator model for implementing operators

- Enables pipelining!
  - As opposed to: execute each operator in entirety, store its results on disk or in main memory
  - Many operators can be "active" simultaneously

- Pipelining: Not always possible (or meaningful):
  - E.g., "sort scan".
  - "Blocking" operators

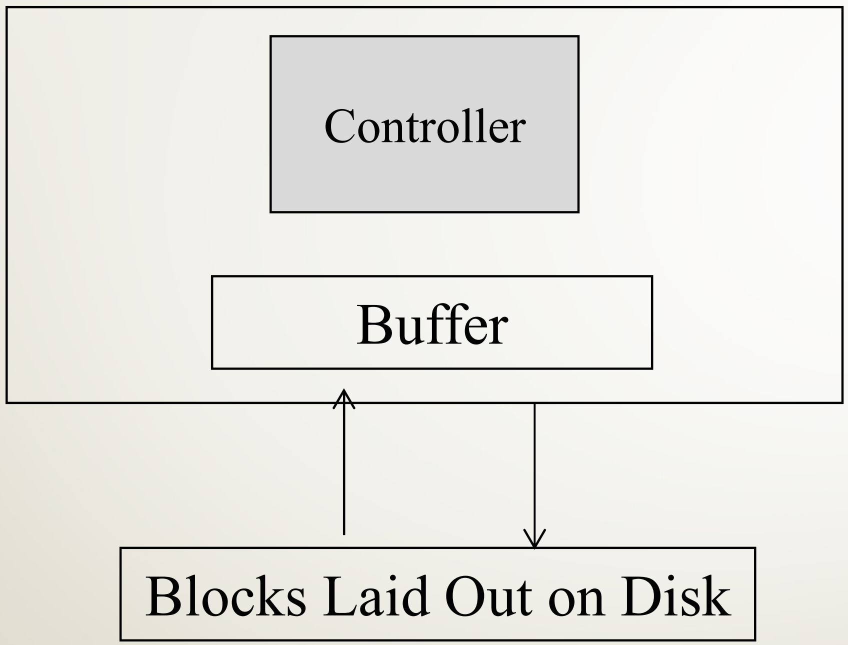


# Overview of operator implementations: first classification

Operator algorithms mostly of one of these types:

- Sorting-based
- Hash-based
- Index-based

# Another classification of operator implementations




***One pass*:**

- reading the data only once from disk.
- Typically, at least one of the arguments must fit in memory.

***Two pass*:**

- Data need not fit in memory but is not "too large".

***Multipass*:**

- No limit on data size.

# How to think about operators

- Classification 1: Hash/Index/Sort
- Classification 2: One/Two/Many pass

- Metric 1: I/O Cost
- Metric 2: Buffer requirements (how large should M be)

# Outline

- ✓ Query Execution: The Big Picture
  - ✓ Basics of Costing
  - ✓ Classifications of Physical Operators
- Scan Operator
  - Table-scan, index-scan and sort-scan
  - Costs for Scans
- Nested-loop joins

# Getting started: Scanning Tables

- Read the entire contents of a relation R
  - or partial contents: all tuples that satisfy a criterion

- Table-scan: R is stored in some area of secondary storage (disk), in blocks.
  - These blocks are known to the system.
  - Can get all blocks one by one.
- Index-scan: if we have index to find the blocks.
  - "Scan the index, grab the relevant blocks"
  - This can be particularly useful for getting tuples that satisfy a predicate

# Sorting while scanning tables

- May want to sort the tuples as we read them: "Sort-scan"
- Why?
  - ORDER BY in query
  - Some RA operations are implemented using sort
    - Union, Intersection, …
  - Future operations may be cheaper (e.g., GROUP BY)
- How?
  - If indexed, then trivial
  - If fits in main memory, then table-scan or index-scan and sort in memory
  - If too large, "multiway merge sort" (see later)

# Sorting with Scanning

*First cut: relation is small*

- Have M main memory blocks available for use
- Assumption: $B(R) <= M$
- Method:
  - Read all $B(R)$ blocks, sort, write
- Cost: $B(R)$

# Sorting with Scanning

*Second cut: Relation is large*

- Have M main memory blocks available for use
- How would you do this?

# Sorting with Scanning

*Second cut: Two pass "multi-way merge sort"*

- Have M main memory blocks available for use
- Step 1:
  - Read M blocks at a time, sort, write
  - Result: have runs of length M on disk
- Step 2:
  - Merge M-1 runs at a time, construct output, write to disk
  - Result: have runs of length $M(M-1) \approx M^2$

# Example

M = 3, each block holds two values

Data on disk: [3,9] [8,1] [7,6] [9,5] [2,0] [3,8]

# Example, cont.

*M = 3, each block holds two values*

*Data on disk: [3,9] [8,1] [7,6] [9,5] [2,0] [3,8]*

Step 1: Sort Runs

[3,9] [8,1] [7,6] => [1,3] [6,7] [8,9]

[9,5] [2,0] [3,8] => [0,2] [3,5] [8,9]

Step 2: Merge Sorted Runs

[1,3][0,2][_,_] => [1,3][0,2][0,1] => output [0,1]

[1,3][0,2][_,_] => [1,3][0,2][2,3] => output [2,3]

[6,7][3,5][_,_] => [6,7][3,5][3,5] => output [3,5]

[6,7][8,9][_,_] => [6,7][8,9][6,7] => output [6,7]

[8,9][8,9][_,_] => [8,9][8,9][8,8] => output [8,8]

# Think-Pair-Share

Given this query plan →

B(R)=20; M = 22; R is sorted on b

which of the following implementation of table scan would be used?

Project a,b

Group By a

Scan R (a,b,c,d)

**A: Sort-Scan**

**B: Table-Scan**

**C: Multiway merge sort**

# Outline

- ✓ Query Execution: The Big Picture
  - ✓ Basics of Costing
  - ✓ Classifications of Physical Operators
- Scan Operator
  - ✓ Table-scan, index-scan and sort-scan
  - Costs for Scans

# Multi-way merge sort cost

Cost: $3B(R)$,

Assumption: $B(R) < M^2$ ; *why?*

$B(R) / M <= M-1 \Leftrightarrow B(R) <= M(M-1) < M^2$

# Cost of the Scan Operator

- Table scan: B(R); Sort-scan: 3B(R)

  [assuming B(R) <= M(M-1)]
- Index scan: B(R) + #blocks of the index

  ≈ B(R); Sort-scan: B(R)

  (assuming index on sorting key)

# Cost of the Scan Operator (Cont.)

- Unclustered relation:
  - we have assumed so far that all tuples of R are "clustered", i.e., stored in ~ B blocks.
  - If tuples of R are interspersed with tuples of other relations, then cost:
  - scan: T(R); sort: T(R) + 2B(R)

# Now Generalizing…

- Done with costing of scan operators
- Next lecture we will generalize to other operators
  - A lot more complex than scan
  - Many different classifications/tradeoffs

# Outline

- ✓ Query Execution: The Big Picture
  - ✓ Basics of Costing
  - ✓ Classifications of Physical Operators
- ✓ Scan Operator
  - ✓ Table-scan, index-scan and sort-scan
  - ✓ Costs for Scans