Query Optimization II - Physical Planning I

Lecture 6

ICS502 - Database Programming

Dr. Eng. Amal Yassien

German International University in Cairo

Office: A.216

Office Hours: Monday 2nd (or by appointment via email)

Email: amal.walied@giu-uni.de

Acknowledgment: These slides are based on the slides of Dr. Caroline Sabty, Dr. Wael Abouelsaadat and "Databases: The Complete Book - 2nd edition" by Prof. Gracia-Moline, Prof. Ullman, and Prof. Widom.

October 23, 2025

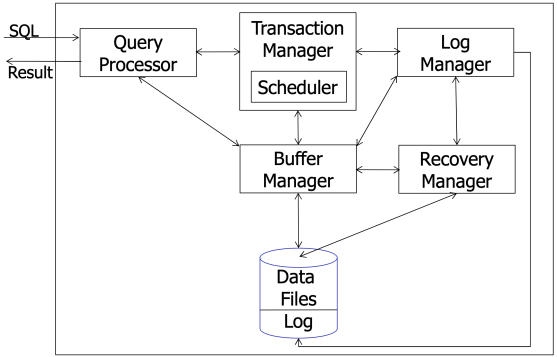
Today's Agenda

- **Physical Planning - A Conceptual Overview**
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- Physical Plan Operators based on Data Access Method
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

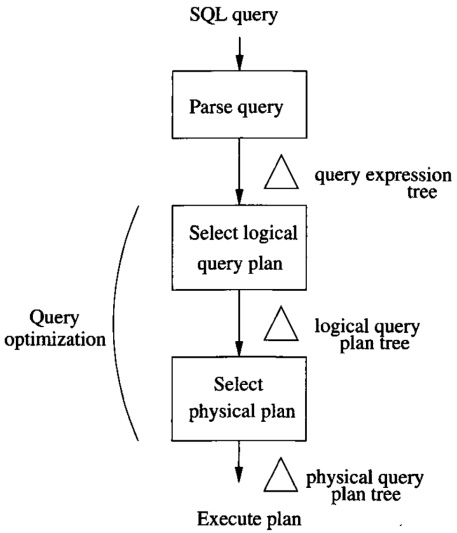
RDBMS Query Processing: Overview

The Query Processing Module

The query processor is the group of components of a DBMS that turns user queries and data-modification commands into a sequence of database operations and executes those operations.



Query Processor: The Components



Query Processing Phase II: Query Optimization

Logical Planning

The parse tree is converted to an initial query plan, which is usually an algebraic representation of the query. This initial plan is then transformed into an equivalent plan that is expected to require less time to execute.

Physical Planning

A physical query plan selects algorithms to implement each of the operators of the logical plan, and the order of execution for these operators. It also includes details such as how the queried relations are accessed, and when and if a relation should be sorted.

Aspects Considered While Optimizing SQL Query Execution

Query Optimization Considerations

1. Which of the algebraically equivalent forms of a query leads to the most efficient algorithm for answering the query? - Logical Plan
2. For each operation of the selected form, what algorithm should we use to implement that operation? - Physical Plan
3. How should the operations pass data from one to the other, e.g., in a pipelined fashion, in main-memory buffers, or via the disk? - Buffer Management

Aspects Considered While Optimizing SQL Query Execution

Query Optimization Considerations

1. Which of the algebraically equivalent forms of a query leads to the most efficient algorithm for answering the query? - Logical Plan
2. For each operation of the selected form, what algorithm should we use to implement that operation? - Physical Plan
3. How should the operations pass data from one to the other, e.g., in a pipelined fashion, in main-memory buffers, or via the disk? - Buffer Management

How to Decide about these Considerations?

Each of these choices depends on the metadata about the database. Typical metadata that is available to the query optimizer includes: the size of each relation; statistics; the existence of certain indexes; and the layout of data on disk.

Query Processing Phase II: Query Optimization

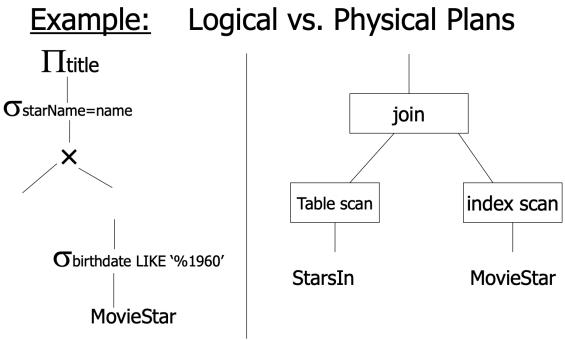


Figure: Logical vs. Physical Planning

Today's Agenda

- Physical Planning - A Conceptual Overview
- **Physical Plan Operators that are not Relational Algebra Ones**
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- Physical Plan Operators based on Data Access Method
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

Physical Operators that are not Relational Algebra: Scanning Tables I

Table Scan

The relation R is stored in an area of secondary memory, with its tuples arranged in blocks. The blocks containing the tuples of R are known to the system, and it is possible to get the blocks one by one.

Index Scan

If there is an index on any attribute of R, we may be able to use this index to get all the tuples of R. For example, a sparse index on R, as discussed in Lecture 1, can be used to lead us to all the blocks holding R, even if we don't know otherwise which blocks these are.

Physical Operators that are not Relational Algebra: Scanning Tables II

Sort Scan

a query could include an ORDER BY clause, requiring that a relation be sorted. Sort Scan takes a relation R and a specification of the attributes on which the sort is to be made, and produces R in that sorted order.

Possible Ways to Implement Sort Scan

If we need to sort relation R on attribute a, then sort scan be made using the following operators.

- index scan on R would produce R in the correct order, if there is a b+ tree index on R.a
- table scan or index scan on R would produce R in the correct order, if
  1. R is small enough to be fully loaded in main memory, then sort in main memory
  2. R is too large to be fully loaded in main memory, then use multi-pass sort (we will see this later ;))

Physical Plan: The possibilities

Physical Plan Cost Estimation

We will use the number of disk I/Os to measure the cost of a physical operator. This way, we are able to decide regarding which algorithm, operator, or access method would be the most efficient.

Cost Estimation Parameters

- M - The # of main-memory buffers available to an execution of a particular operator, where main memory buffers has the same size as disk block
- T(R) - The # of tuples in relation R
- B(R) - The # of blocks that hold relation R
- S(R) - The # of bytes each tuple in relation R occupy
- V(R,a) - The # of distinct values in relation R for attribute a

I/O Cost Estimation for Scanning Tables

Table Scan

The cost varies between B(R) and T(R) I/Os based on:

- If Relation R is contiguous, (i.e., a block is stored in the same cylinder on the hard disk), then it takes $B(R)$ I/Os to fetch relation R
- If Relation R is non-contiguous, then it takes $T(R)$ I/Os to fetch relation R

Index Scan - *to fetch the whole table*

same as table scan above

Index Scan - *to fetch part of the table*

We will see the I/O cost analysis of this later ;)

Table Scan Implementation using Iterators

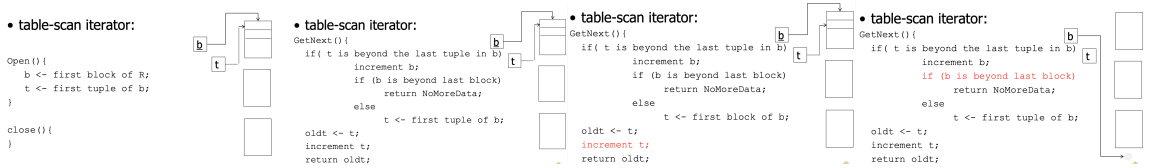


Figure: An Exemplar Iterator Implemention of Table Scan

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- **Physical Plan Operators based on Input Type**
- Physical Plan Operators based on Algorithm Type
- Physical Plan Operators based on Data Access Method
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

Physical Operators based on Input Type

Tuple-at-a-time

These operations (e.g. selection and projection) do not require an entire relation, or even a large part of it, in memory at once. Thus, we can read a block at a time, use one main-memory buffer, and produce our output.

Relation-at-a-time

These one-argument operations (e.g. duplicate elimination, grouping) require seeing all or most of the tuples in memory at once. For these operators, we might use different algorithm types depending on the relation size and available main memory blocks (e.g. One-Pass or Two-Pass).

Binary Relation-at-a-time

All other operations are in this class: set and bag versions of union, intersection, difference, joins, and products. Also, different algorithm types can be used based on main memory availability.

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- **Physical Plan Operators based on Algorithm Type**
- Physical Plan Operators based on Data Access Method
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

Physical Plan Operators based on Algorithm Type

One-Pass Algorithms

Some methods involve reading the data only once from disk. Usually, they require at least one of the arguments to fit in main memory, although there are exceptions, especially for selection and projection.

Two-Pass Algorithms

Some methods work for data that is too large to fit in available main memory but not for the largest imaginable data sets. These two-pass algorithms are characterized by reading data a first time from disk, processing it in some way, writing all, or almost all, of it to disk, and then reading it a second time for further processing during the second pass.

Multi-Pass Algorithms

Some methods work without a limit on the size of the data. These methods use three or more passes to do their jobs, and are natural, recursive generalizations of the two-pass algorithms.

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- **Physical Plan Operators based on Data Access Method**
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

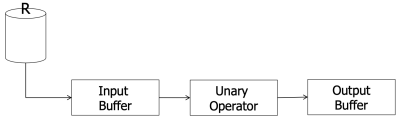
Physical Plan Operators: Putting it all together

Tuple-at-a-time Operators

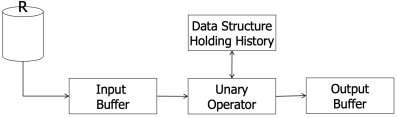
Since Tuple-at-a-time operate only on a single row, then generally One-Pass Algorithms are sufficient to do the job.

Full-Relation-at-a-time, Unary and Binary

These operators require most of the relation (at least first argument in case of binary) to be available in the main memory to be able to produce an output. Therefore, the algorithm type to be used depends on the available memory and the operated-on relation(s) size.



(a) Flow of Information in Tuple-at-a-time Operators



(b) Flow of Information in Relation-at-a-time Unary Operators

Unary Relation-at-a-time Operators: One-Pass Algorithms I

Grouping γ_L

A grouping operation γ_L gives us zero or more grouping attributes and presumably one or more aggregated attributes. If we create in main memory one entry for each group, then we can scan the tuples of R, one block at a time. The entry for a group consists of values for the grouping attributes and an accumulated value or values for each aggregation, as follows:

MAX and MIN

Record the minimum or maximum value, respectively, of attribute a seen for any tuple in the group so far. Change this minimum or maximum, if appropriate, each time a tuple of the group is seen.

COUNT

Add one for each tuple of the group that is seen.

Unary Relation-at-a-time Operators: One-Pass Algorithms I (Cont'd)

Grouping γ_L

A grouping operation γ_L gives us zero or more grouping attributes and presumably one or more aggregated attributes. If we create in main memory one entry for each group — that is, for each value of the grouping attributes — then we can scan the tuples of R, one block at a time. The entry for a group consists of values for the grouping attributes and an accumulated value or values for each aggregation, as follows:

SUM

Add the value of attribute a to the accumulated sum for its group, provided that it is not NULL.

AVG

Keep track of *SUM* and *COUNT* in each group.

Unary Relation-at-a-time Operators: One-Pass Algorithms II

Duplicate Elimination δ

To eliminate duplicates, we can read each block of R one at a time, but for each tuple we need to make a decision as to whether it the first time to see this tuple, or we have seen it before.

Duplicate Elimination δ Steps

To support this decision, we need to keep in memory one copy of every tuple we have seen, where one memory buffer holds one block of R 's tuples, and the remaining M —1 buffers can be used to hold a single copy of every tuple seen so far.

It is not always about I/O Cost

Typically $\delta(R)$ costs $B(R)$ or $T(R)$. However, since we must check that the outputted tuple have not been seen before, we need to compare it to the tuples outputted. If we use main memory linear data structure, the we get time complexity of $O(n^2)$

Unary Relation-at-a-time Operators: One-Pass Algorithms II (Cont'd)

Duplicate Elimination δ

To eliminate duplicates, we can read each block of R one at a time, but for each tuple we need to make a decision as to whether it the first time to see this tuple, or we have seen it before.

Duplicate Elimination δ Steps

To support this decision, we need to keep in memory one copy of every tuple we have seen, where one memory buffer holds one block of R 's tuples, and the remaining M —1 buffers can be used to hold a single copy of every tuple seen so far.

δ can't always be performed in One-Pass

For δ to be performed in One-Pass, then $B(\delta(R)) \leq M$

Binary Relation-at-a-time Operators: One-Pass Algorithms I

In one-pass binary operators, we assume that $min(B(R), B(S)) \leq M$, and that S is the smaller relation.

Set Union

We read S into M^-1 buffers of main memory and build a search structure whose search key is the entire tuple. All these tuples are also copied to the output. Then read each block of R into the M^{th} buffer, one at a time. For each tuple t of R, we see if t is in S, and if not, we copy t to the output. If t is also in S, we skip t.

Set Intersection

Read S into M^-1 buffers and build a search structure with full tuples as the search key. Read each block of R, and for each tuple t of R, see if t is also in 5. If so, copy t to the output, and if not, ignore t.

Binary Relation-at-a-time Operators: One-Pass Algorithms II

Set Difference

Assuming S is the smaller relation, we read S into M^-1 buffers of main memory and build a search structure whose search key is the entire tuple.

Set Difference: $S - R$

read the blocks of R and examine each tuple t in turn. If t is in S, then we delete t from the copy of S in main memory, while if t is not in S we do nothing. After considering each tuple of R, we copy to the output those tuples of S that remain.

Set Difference: $R - S$

Read each block of R and examine each tuple t on that block. If t is in S, then ignore t; if it is not in S then copy t to the output.

Binary Relation-at-a-time Operators: One-Pass Algorithms III

We assume that S is the smaller relation.

Product: $R \times S$

Read S into $M - 1$ buffers of main memory; no special data structure is needed. Then read each block of R, and for each tuple t of R concatenate t with each tuple of 5 in main memory. Output each concatenated tuple as it is formed.

Join: $R(X, Y) \bowtie S(Y, Z)$

Read all the tuples of S and form them into a main-memory search structure with the attributes of Y as the search key. Use $M - 1$ blocks of memory for this purpose. Read each block of R into the one remaining main-memory buffer. For each tuple t of R, find the tuples of S that agree with t on all attributes of Y, using the search structure. For each matching tuple of 5, form a tuple by joining it with t, and move the resulting tuple to the output.

Physical Operators Classifications

Irrespective of the input type or algorithm type of a give physical plan operator, there are *four* different approaches or methods that are used by RDBMS to access the required data:

Physical Plan Operators Access Methods

1. Iterative Based Access Method
2. Sort-based Access Method
3. Hash-based Access Method
4. Index-based Access Method

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- **Physical Plan Operators based on Data Access Method**
  - **Iterative Based Access Method**
  - Sort-based Access Method
  - Hash-based Access Method
  - Index-based Access Method

Binary Relation-at-a-time: One-Pass-Iteration-based Algorithm for Bag Union

```
Open() {
    R.Open();
    CurRel := R;
}

GetNext() {
    IF (CurRel = R) {
        t := R.GetNext();
        IF (t <> NotFound) /* R is not exhausted */
            RETURN t;
        ELSE /* R is exhausted */ {
            S.Open();
            CurRel := S;
        }
    }
    /* here, we must read from S */
    RETURN S.GetNext();
    /* notice that if S is exhausted, S.GetNext()
       will return NotFound, which is the correct
       action for our GetNext as well */
}

Close() {
    R.Close();
    S.Close();
}
```

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- **Physical Plan Operators based on Data Access Method**
  - Iterative Based Access Method
  - **Sort-based Access Method**
  - Hash-based Access Method
  - Index-based Access Method

Two-Pass-Sort-based Algorithm: Merge Sort (2TPMMS)

2TPMMS: The Rationale

It is possible to sort very large relations in two passes Two-Phase, Multiway Merge-Sort (TPMMS). Suppose we have M main- memory buffers to use for the sort. TPMMS sorts a relation R as follows:

Phase I: 1^{st} *Pass*

Repeatedly fill the M buffers with new tuples from R and sort them, using any main-memory sorting algorithm. Write out each sorted sublist to secondary storage.

Phase II: 2^{nd} *Pass*

Merge the sorted sublists. For this phase to work, there can be at most M —1 sorted sublists, which limits the size of R. We allocate one input block to each sorted sublist and one block to the output.

TPMMS: I/O Cost

Phase I To perform the first pass, we need to read all relations blocks ($B(R)$), then write all relations blocks ($B(R)$) after partial sorting which results in $2B(R)$ I/O

Phase II To perform the second pass, we need to read all partially sorted relations blocks ($B(R)$), then write all relations blocks ($B(R)$) after the final sorting which results in $2B(R)$ I/O

Therefore, sorting using two-pass algorithm takes $4B(R)$ I/O to sort relation R

Duplicate Elimination δ using TPMMS: Example

Example Setting

Suppose that tuples are integers, and only two tuples can fit inside a block. Also, $M = 3$ and $T(R) = 17$ following rows:

2,5,2,1,2,2,4,5,4,3,4,2,1,5,2,1,3

Pass 1 Output

- Example

	In Memory	Waiting on Disk		
1	Sublist R1	1,2	2,2	2,5
2	Sublist R2	2,3	4,4	4,5
3	Sublist R3	1,1	2,3	5

Duplicate Elimination δ using TPMMS: Example (Cont'd)

- Example

	In Memory		Waiting on Disk		
1	Sublist R1	1,2	2,2	2,5	
2	Sublist R2	2,3	4,4	4,5	
3	Sublist R3	1,1	2,3	5	

(a) 1st Pass Output

- Example

	In Memory	Waiting on Disk		
1	1,2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	1,1	2,3	5	Sublist R3

(b) The beginning of Pass 2

- Example

	In Memory	Waiting on Disk		
1	1,2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	1,1	2,3	5	Sublist R3
Output 1, now;				
1	2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	2,3	5		Sublist R3

(c) Displaying the first Output in Pass 2

Duplicate Elimination δ using TPMMS: Example (Cont'd)

- Example

	In Memory	Waiting on Disk		
1	1,2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	1,1	2,3	5	Sublist R3

(a) The beginning of Pass 2

- Example

	In Memory	Waiting on Disk		
1	1,2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	1,1	2,3	5	Sublist R3

Output 1, now;

1	2	2,2	2,5	Sublist R1
2	2,3	4,4	4,5	Sublist R2
3	2,3	5		Sublist R3

(b) Displaying the first Output in Pass 2

- Example

Output 2

	In Memory	Waiting on Disk		
1	5			Sublist R1
2	3	4,4	4,5	Sublist R2
3	3	5		Sublist R3

(c) Displaying the second Output in Pass 2

Other Binary and Unary Relation-at-time Operators using 2TPMMS

$\gamma_L, \bowtie, \cup, \cap, -$

Read sections 15.4.3 - 15.4.6 of the textbook!

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- **Physical Plan Operators based on Data Access Method**
  - Iterative Based Access Method
  - Sort-based Access Method
  - **Hash-based Access Method**
  - Index-based Access Method

Hash-based Access Method: The Main Idea

1. Read relation R block by block
2. Take each tuple in a block and hash it to one of a set of buckets of a hash file
3. All "similar" tuples should hash to the same bucket
4. Examine each bucket in isolation to produce final result

Hash-based Duplicate Elimination δ

1. Read relation R block by block
2. Take each tuple in a block and hash it to one of a set of buckets of a hash file
3. Visit each bucket and eliminate duplicates (if the bucket is too large to fit in memory, use 2TPMMS to perform δ within each bucket!)

Hash-based $\delta(R)$ I/O Cost

In order to hash relation R, we need to read each block of R $B(R)$, then hash it into a buckets, then write all blocks of R after hashing $B(R)$. So hashing takes: $2B(R)$. Now to eliminate duplicates within each bucket, we need to read the buckets again resulting in another $B(R)$. So in total, hash-based $\delta(R)$ takes $3B(R)$ I/Os.

Hash-based Join Algorithm $R \bowtie S$

1. Read Relation R in memory block by block
2. Hash all tuples in R using hash function h
3. Read Relation S in memory block by block
4. Hash all tuples in S using the same hash function h
5. Since the same hash function h is used for R and S, corresponding buckets can be compared

I/O Cost of Hash-based Join?

Think about this and tell me next time. Hint: look at the analysis in the δ ;)

Other Binary and Unary Relation-at-time Operators using Hash Methods

$\gamma_L, \cup, \cap, -$

Read sections 15.5.3 - 15.5.4 of the textbook!

Today's Agenda

- Physical Planning - A Conceptual Overview
- Physical Plan Operators that are not Relational Algebra Ones
- Physical Plan Operators based on Input Type
- Physical Plan Operators based on Algorithm Type
- **Physical Plan Operators based on Data Access Method**
  - Iterative Based Access Method
  - Sort-based Access Method
  - Hash-based Access Method
  - **Index-based Access Method**

Clustered Index

Clustered Index General Definition

clustering indexes, which are indexes on an attribute or attributes such that all the tuples with a fixed value for the search key of this index appear on roughly as few blocks as can hold them.

Index-based Method Main Usage

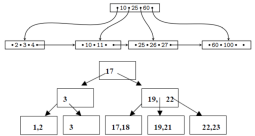
Index-based Selection

If there is an index on the attribute used to extract tuples, then index-scan might be useful - *refer to practical part made during Lecture 2*

Zig-Zag Joins

Zig-Zag Join is an algorithm that uses b+ trees created on the common attributes of the relations to be joined. It basically goes back and forth between the 2 b+ trees to find the matching rows to join

- R(X,Y) * S(Y,Z)
- On 2 B+ trees together



Takeaways

- Iterators are the basic building blocks of a physical plan. Table scan (also called SEQ scan) and index scan are the most common ones
- Relational Algebra operations are called operators in the physical plan
- Physical plan operators are classified according to how many rows or relations they act on (row at a time, relation at a time, binary relation at a time), and according to how many times they pass on the rows of the relation (single pass or multi-pass)
- There are several techniques to access a relation content to perform a specific SQL operation: linear access methods, sort based methods, hash-based methods and index-based methods. Typically, all of them are implemented in a database engine

Next Time

- Physical Planning II

Thank You
Vielen Dank

Dr. Eng. Amal Yassien

amal.walied@giu-uni.de

A.216