Carnegie Mellon University

Query Processing

Lecture #10

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

Project #1 is due TODAY @ 11:59pm

Homework #3 is due Wednesday October 4th @ 11:59pm

Mid-term Exam is on Wednesday October 18th (in class)

Project #2 is due Wednesday October 25th @ 11:59am

LECTURE #08 CORRECTION

Nasty Deez Nutz In Yo Moth 2 months ago

why we do have to suffer with these bad lectures? professor pavlo sucks straight up.

REPLY 11

The14thChapter 9 hours ago

Yo i herd that andy pushed this old lady down the stairs. hes awful. databases are tight and all but he needs to stop with dez bad lectures. santa monica out!

REPLY 53

DaOldSchoolRapJiveTurkey94 2 weeks ago

Andy is awful. He speaks so fast that I get headaches. I wish somebody that was at CMU would stab him.

REPLY 139

View all 16 replies

LECTURE #08 CORRECTION

Highlighted comment   Pinned by CMU Database Group

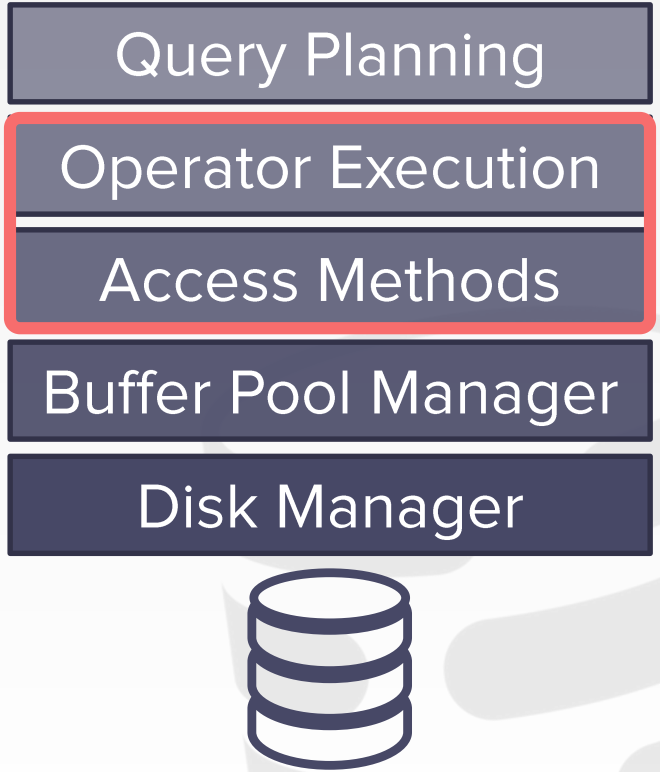
William Cody Laeder 2 days ago

City/Farm don't use SIMD this hurts portability (Google ships farmhash in Chrome). They use a small buffer internally (normally 64, XXHash uses 256 for larger mode). If the hash internally tries to fill this buffer before it computes a digest (and XOR the old digest with the new 64bytes digest), and if it that buffer isn't full it does a unique

REPLY

STATUS

We are now going to talk about how the DBMS execute queries that retrieve data from the system's access methods.



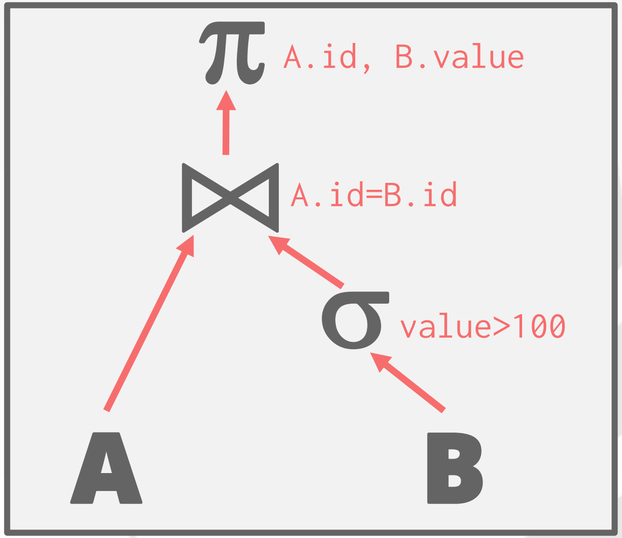
CARNEGIE MELLON DATABASE GROUP

QUERY PLAN

The operators are arranged in a tree. Data flows from the leaves toward the root.

The output of the root node is the result of the query.

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



CARNEGIE MELLON DATABASE GROUP

TODAY'S AGENDA

Processing Models

Access Methods

Expression Evaluation

Project #2

CARNEGIE MELLON DATABASE GROUP

Processing Model

A DBMS's **processing model** defines how the system executes a query plan.

→ Different trade-offs for different workloads.

Three approaches:

→ Iterator Model
→ Materialization Model
→ Vectorized / Batch Model

ITERATOR MODEL

Each query plan operator implements a `next` function.

→ On each invocation, the operator returns either a single tuple or a null marker if there are no more tuples.
→ The operator implements a loop that calls next on its children to retrieve their tuples and then process them.

Top-down plan processing.

Also called **Volcano** or **Pipeline** Model.

CARNEGIE MELLON
DATABASE GROUP

ITERATOR MODEL

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```

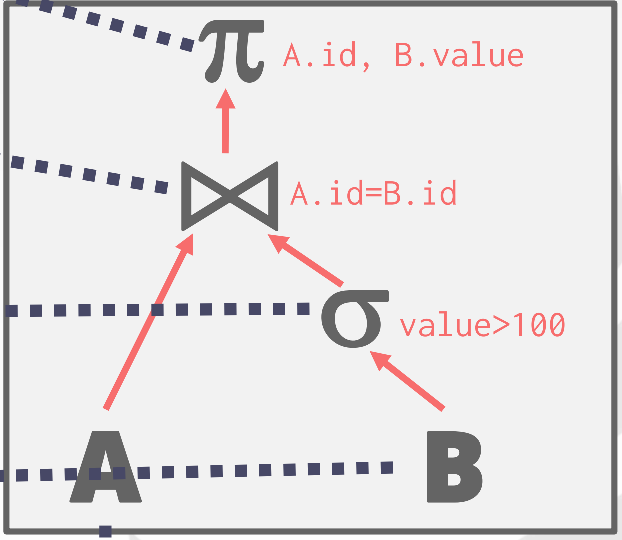
```
for t in child.Next():
  emit(projection(t))
```

```
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): emit(t1⋈t2)
```

```
for t in child.Next():
  if evalPred(t): emit(t)
```

```
for t in A:
  emit(t)
```

```
for t in B:
  emit(t)
```



ITERATOR MODEL

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```

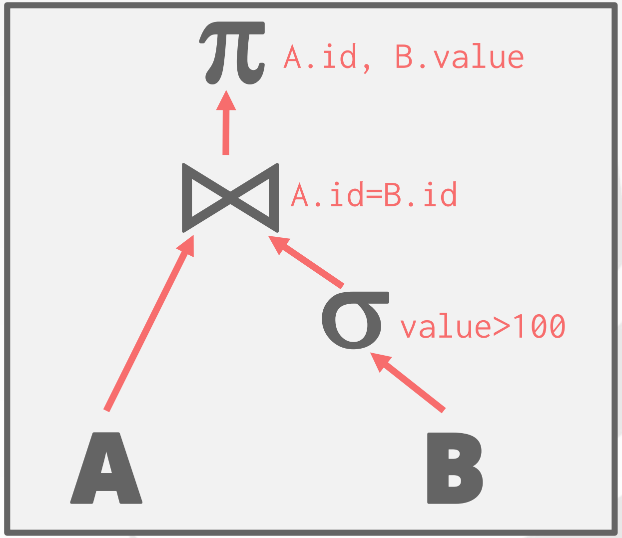
```
for t in child.Next():
  emit(projection(t))
```

```
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): emit(t1⋈t2)
```

```
for t in child.Next():
  if evalPred(t): emit(t)
```

```
for t in A:
  emit(t)
```

```
for t in B:
  emit(t)
```



CARNEGIE MELLON
DATABASE GROUP

ITERATOR MODEL

1

```
for t in child.Next():
  emit(projection(t))
```

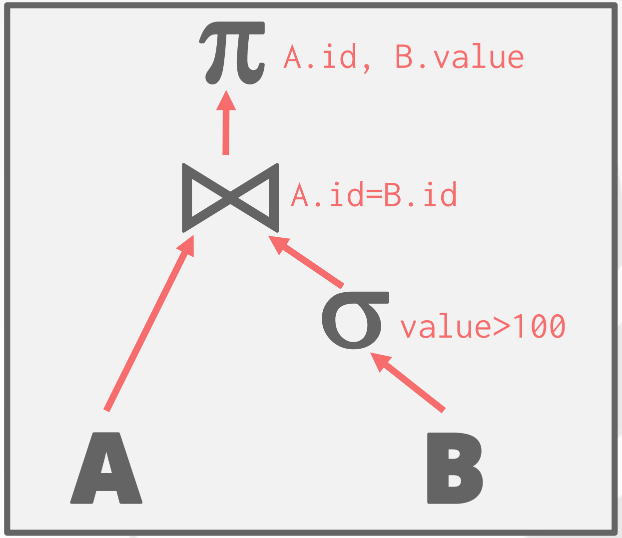
```
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): emit(t1⋈t2)
```

```
for t in child.Next():
  if evalPred(t): emit(t)
```

```
for t in A:
  emit(t)
```

```
for t in B:
  emit(t)
```

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



CARNEGIE MELLON DATABASE GROUP

ITERATOR MODEL

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```

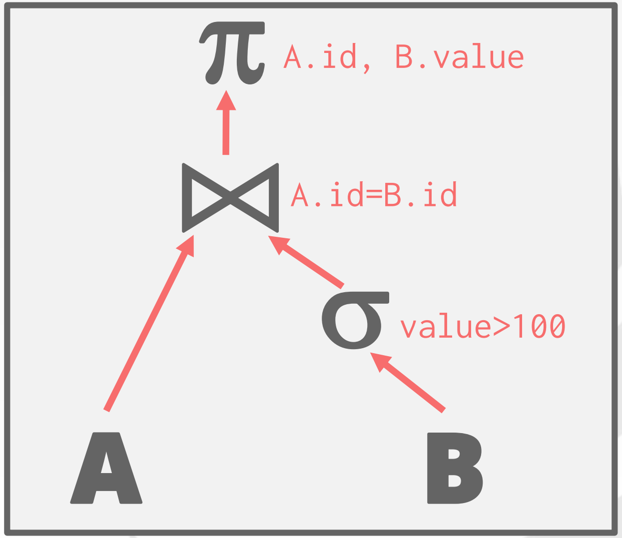
1
```
for t in child.Next():
  emit(projection(t))
```

2
```
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): emit(t1⋈t2)
```

```
for t in child.Next():
  if evalPred(t): emit(t)
```

3
```
for t in A:
  emit(t)
```

```
for t in B:
  emit(t)
```



CARNEGIE MELLON DATABASE GROUP

ITERATOR MODEL

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```

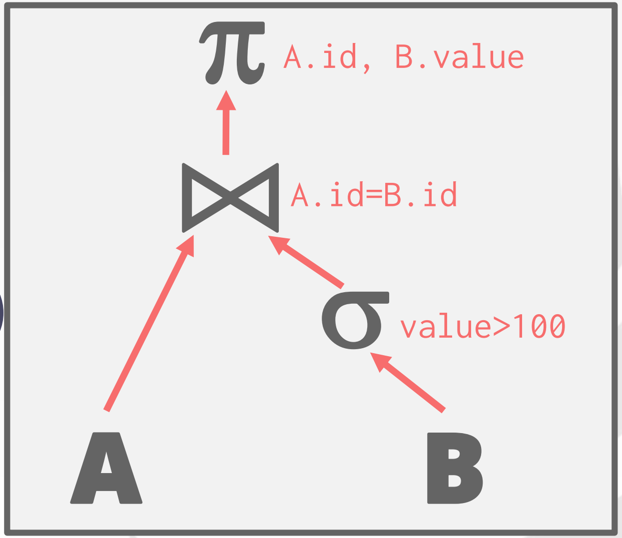
1
```
for t in child.Next():
  emit(projection(t))
```

2
```
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): emit(t1⋈t2)
```

3
```
for t in A:
  emit(t)
```

4
```
for t in child.Next():
  if evalPred(t): emit(t)
```

5
```
for t in B:
  emit(t)
```



CARNEGIE MELLON
DATABASE GROUP

ITERATOR MODEL

This is used in almost every DBMS. Allows for tuple **pipelining**.

Some operators will block until children emit all of their tuples.
→ Joins, Subqueries, Order By

Output control works easily with this approach.
→ LIMIT

MATERIALIZATION MODEL

Each operator processes its input all at once and then emits its output all at once.

→ The operator "materializes" it output as a single result.

→ The DBMS can push down hints into to avoid scanning too many tuples.

Bottom-up plan processing.

CARNEGIE MELLON DATABASE GROUP

MATERIALIZATION MODEL

```
out = { }
for t in child.Output():
  out.add(projection(t))
```

```
out = { }
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⋈t2)
```

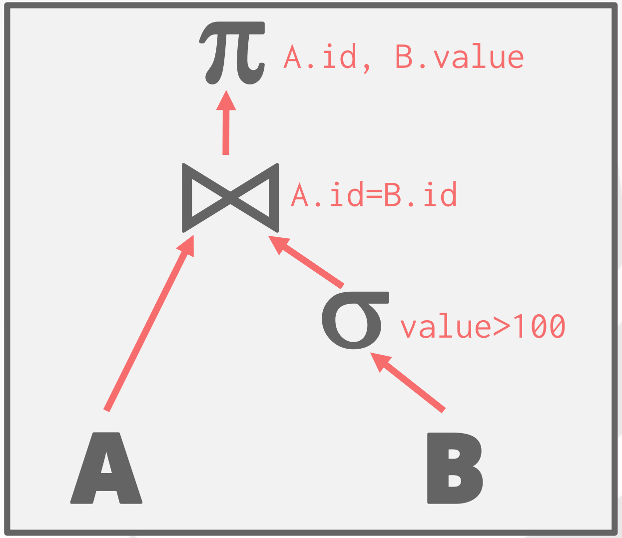
```
out = { }
for t in child.Output():
  if evalPred(t): out.add(t)
```

1

```
out = { }
for t in A:
  out.add(t)
```

```
out = { }
for t in B:
  out.add(t)
```

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



CARNEGIE MELLON DATABASE GROUP

MATERIALIZATION MODEL

```
out = { }
for t in child.Output():
  out.add(projection(t))
```

```
out = { }
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⋈t2)
```

3

```
out = { }
for t in child.Output():
  if evalPred(t): out.add(t)
```

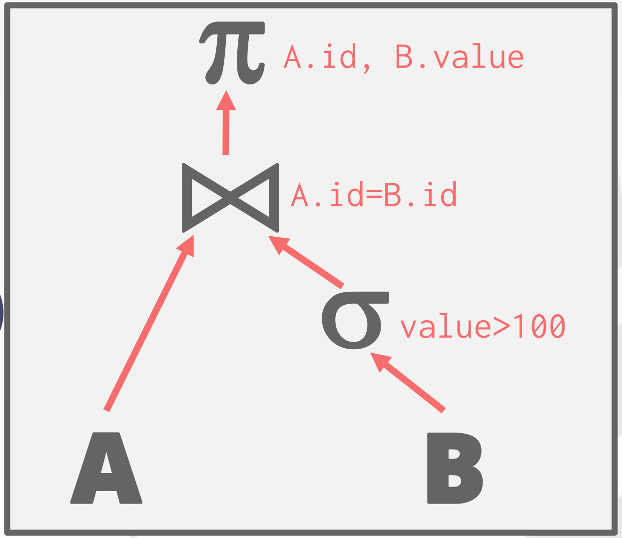
1

```
out = { }
for t in A:
  out.add(t)
```

2

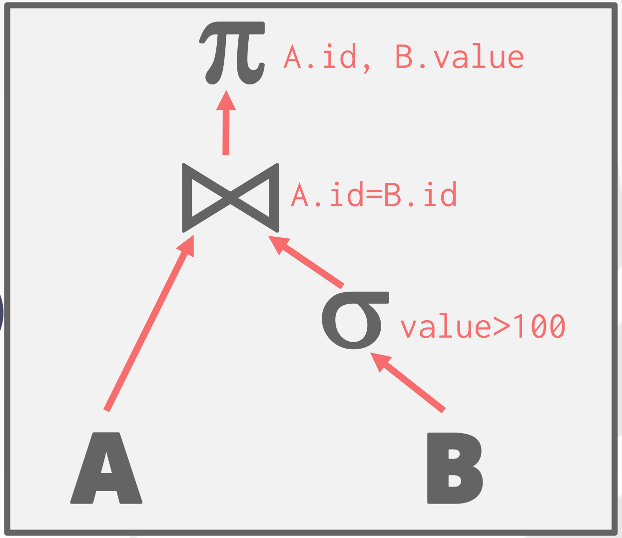
```
out = { }
for t in B:
  out.add(t)
```

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



MATERIALIZATION MODEL

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



5
```
out = { }
for t in child.Output():
  out.add(projection(t))
```

4
```
out = { }
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⨝t2)
```

3
```
out = { }
for t in child.Output():
  if evalPred(t): out.add(t)
```

1
```
out = { }
for t in A:
  out.add(t)
```

2
```
out = { }
for t in B:
  out.add(t)
```


CARNEGIE MELLON DATABASE GROUP

MATERIALIZATION MODEL

Better for OLTP workloads because queries typically only access a small number of tuples at a time.

→ Lower execution / coordination overhead.
→ More difficult to parallelize.

Not good for OLAP queries with large intermediate results.

CARNEGIE MELLON DATABASE GROUP

VECTORIZATION MODEL

Like Iterator Model, each operator implements a `next` function.

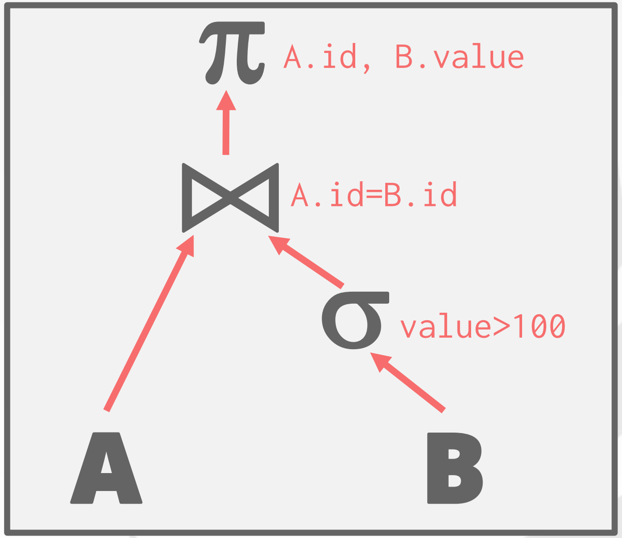
Each operator emits a **batch** of tuples instead of a single tuple.

→ The operator's internal loop processes multiple tuples at a time.
→ The size of the batch can vary based on hardware or query properties.

CARNEGIE MELLON DATABASE GROUP

VECTORIZATION MODEL

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



1

```
out = { }
for t in child.Output():
  out.add(projection(t))
  if |out|>n: emit(out)
```

2

```
out = { }
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⋈t2)
  if |out|>n: emit(out)
```

```
out = { }
for t in child.Output():
  if evalPred(t): out.add(t)
  if |out|>n: emit(out)
```

3

```
out = { }
for t in A:
  out.add(t)
  if |out|>n: emit(out)
```

```
out = { }
for t in B:
  out.add(t)
  if |out|>n: emit(out)
```


CARNEGIE MELLON DATABASE GROUP

VECTORIZATION MODEL

1
```
out = { }
for t in child.Output():
  out.add(projection(t))
  if |out|>n: emit(out)
```

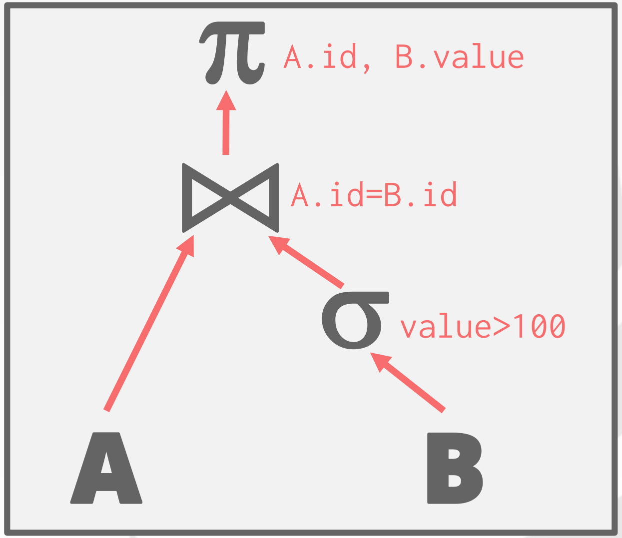
2
```
out = { }
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⋈t2)
  if |out|>n: emit(out)
```

4
```
out = { }
for t in child.Output():
  if evalPred(t): out.add(t)
  if |out|>n: emit(out)
```

3
```
out = { }
for t in A:
  out.add(t)
  if |out|>n: emit(out)
```

5
```
out = { }
for t in B:
  out.add(t)
  if |out|>n: emit(out)
```

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



CARNEGIE MELLON DATABASE GROUP

VECTORIZATION MODEL

Ideal for OLAP queries

→ Greatly reduces the number of invocations per operator.

→ Allows for operators to use vectorized (SIMD) instructions to process batches of tuples.

PROCESSING MODELS SUMMARY

Iterator / Volcano
- → Direction: Top-Down
- → Emits: Single Tuple
- → Target: General Purpose

Materialization
- → Direction: Bottom-Up
- → Emits: Entire Tuple Set
- → Target: OLTP

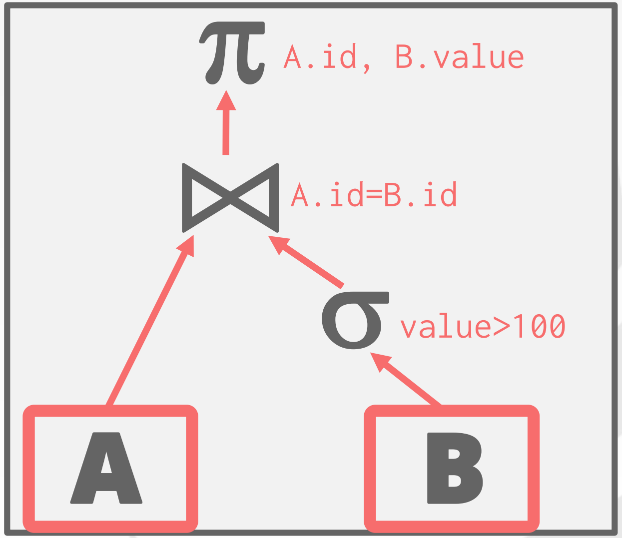
Vectorized
- → Direction: Top-Down
- → Emits: Tuple Batch
- → Target: OLAP

ACCESS METHODS

An **access method** is a way that the DBMS can access the data stored in a table.
→ Not defined in relational algebra.

Three basic approaches:
→ Sequential Scan
→ Index Scan
→ Multi-Index / "Bitmap" Scan

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.value > 100
```



CARNEGIE MELLON DATABASE GROUP

SEQUENTIAL SCAN

For each page in the table:
→ Retrieve it from the buffer pool.
→ Iterate over each tuple and check whether to include it.

```
for page in table.pages:
  for t in page.tuples:
    if evalPred(t):
      // Do Something!
```

The DBMS maintains an internal **cursor** that tracks the last page / slot it examined.

CARNEGIE MELLON DATABASE GROUP

SEQUENTIAL SCAN: OPTIMIZATIONS

This is almost always the worst thing that the DBMS can do to execute a query.

Sequential Scan Optimizations:

→ Prefetching
→ Parallelization
→ Zone Maps
→ Buffer Pool Bypass
→ Heap Clustering

CARNEGIE MELLON
DATABASE GROUP

ZONE MAPS

Pre-computed aggregates for the attribute values in a page.

DBMS can check the zone map first to decide whether it wants to access the page.

```
SELECT * FROM table
 WHERE val > 600
```

Original Data

val
100
200
300
400
400

Zone Map

type	val
MIN	100
MAX	400
AVG	280
SUM	1400
COUNT	5

BUFFER POOL BYPASS

The sequential scan operator will not store fetched pages in the buffer pool to avoid overhead.

→ Memory is local to running query.

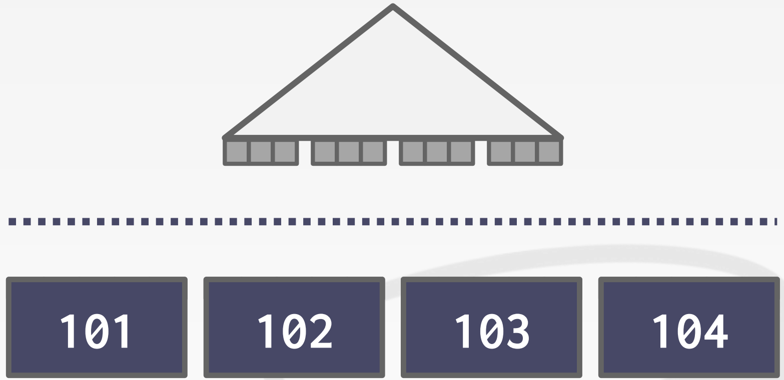
→ Works well if operator needs to read a large sequence of pages that are contiguous on disk.

Called "Light Scans" in Informix.

HEAP CLUSTERING

Tuples are sorted in the heap's pages using the order specified by a clustering index.

If the query accesses tuples using the clustering index's attributes, then the DBMS can jump directly to the pages that it needs.

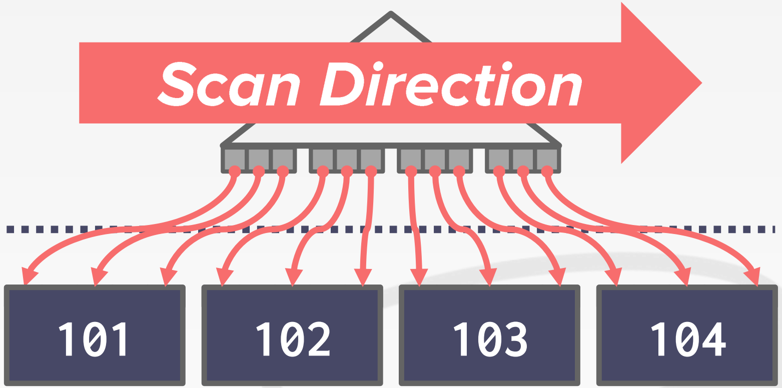


CARNEGIE MELLON DATABASE GROUP

HEAP CLUSTERING

Tuples are sorted in the heap's pages using the order specified by a clustering index.

If the query accesses tuples using the clustering index's attributes, then the DBMS can jump directly to the pages that it needs.



CARNEGIE MELLON DATABASE GROUP

INDEX SCAN

The DBMS picks an index to find the tuples that the query needs.

Which index to use depends on:
- → What attributes the index contains
- → What attributes the query references
- → The attribute's value domains
- → Predicate composition
- → Whether the index has unique or non-unique keys

Later: Query Optimization

CARNEGIE MELLON DATABASE GROUP

INDEX SCAN

Suppose that we a single table with 100 tuples and two indexes:

→ Index #1: **age**
→ Index #2: **dept**

```
SELECT * FROM students
 WHERE age < 30
   AND dept = 'CS'
   AND country = 'US'
```

Scenario #1

There are 99 people under the age of 30 but only 2 people in the CS department.

Scenario #2

There are 99 people in the CS department but only 2 people under the age of 30.

CARNEGIE MELLON
DATABASE GROUP

MULTI-INDEX SCAN

If there are multiple indexes that the DBMS can use for a query:

→ Compute sets of record ids using each matching index.
→ Combine these sets based on the query's predicates (union vs. intersect).
→ Retrieve the records and apply any remaining terms.

Postgres calls this "Bitmap Scan"

CARNEGIE MELLON DATABASE GROUP

MULTI-INDEX SCAN

With an index on `age` and an index on `dept`,

- → We can retrieve the record ids satisfying `age<30` using the first,
- → Then retrieve the record ids satisfying `dept='CS'` using the second,
- → Take their intersection
- → Retrieve records and check `country='US'`.

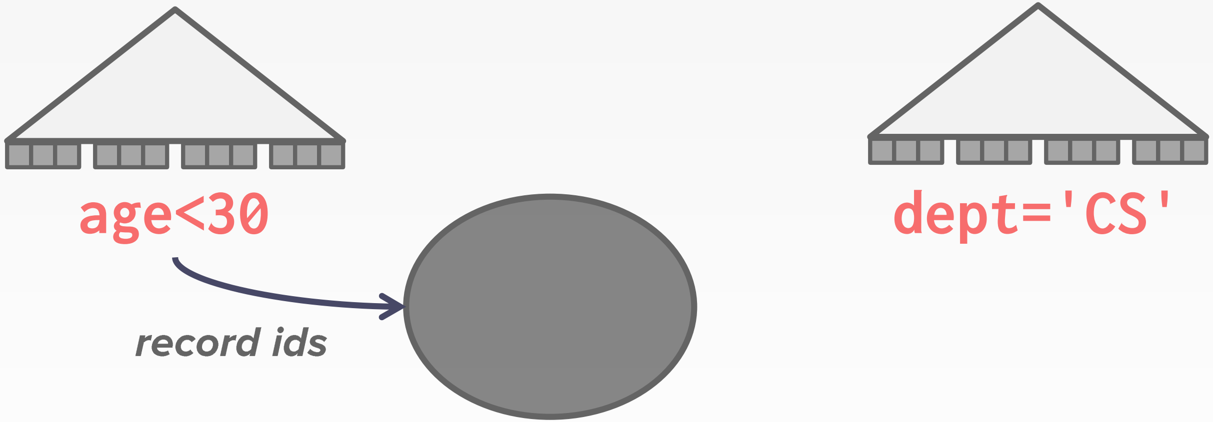
```
SELECT * FROM students
 WHERE age < 30
   AND dept = 'CS'
   AND country = 'US'
```


CARNEGIE MELLON
DATABASE GROUP

MULTI-INDEX SCAN

Set intersection can be done with bitmaps, hash tables, or Bloom filters.

```
SELECT * FROM students
 WHERE age < 30
   AND dept = 'CS'
   AND country = 'US'
```

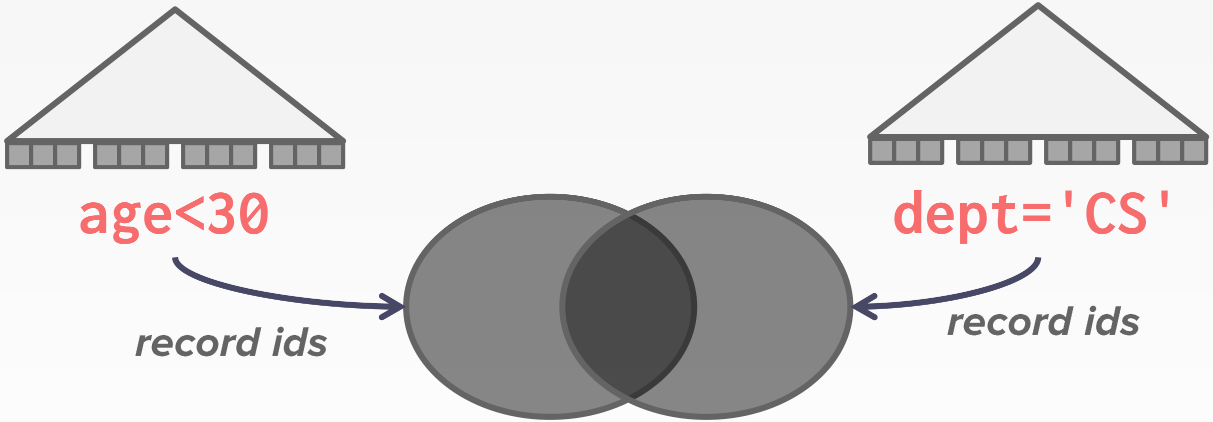


CARNEGIE MELLON DATABASE GROUP

MULTI-INDEX SCAN

Set intersection can be done with bitmaps, hash tables, or Bloom filters.

```
SELECT * FROM students
 WHERE age < 30
   AND dept = 'CS'
   AND country = 'US'
```

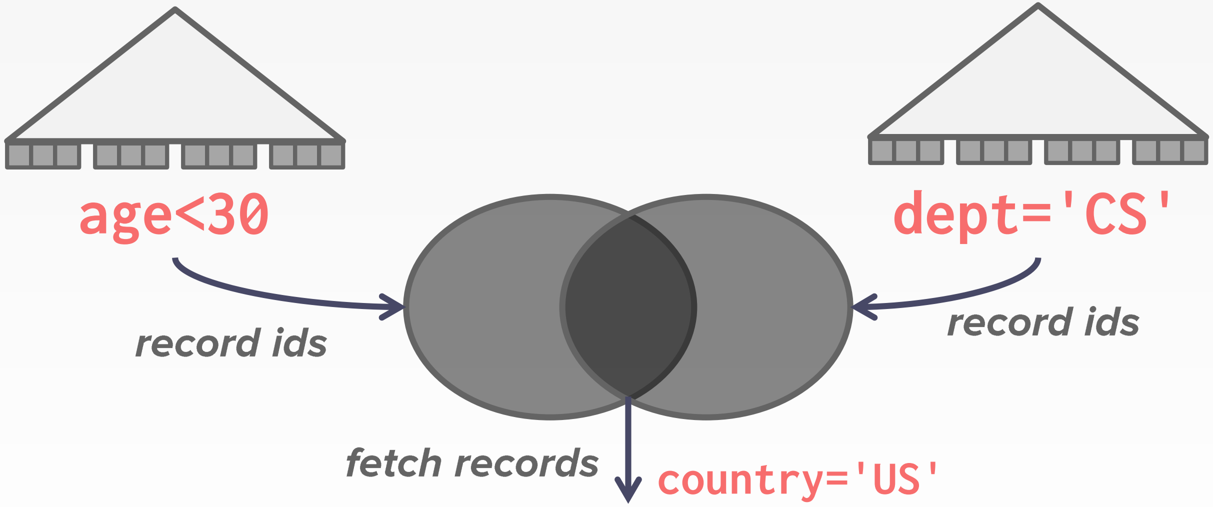


CARNEGIE MELLON DATABASE GROUP

MULTI-INDEX SCAN

Set intersection can be done with bitmaps, hash tables, or Bloom filters.

```
SELECT * FROM students
 WHERE age < 30
   AND dept = 'CS'
   AND country = 'US'
```

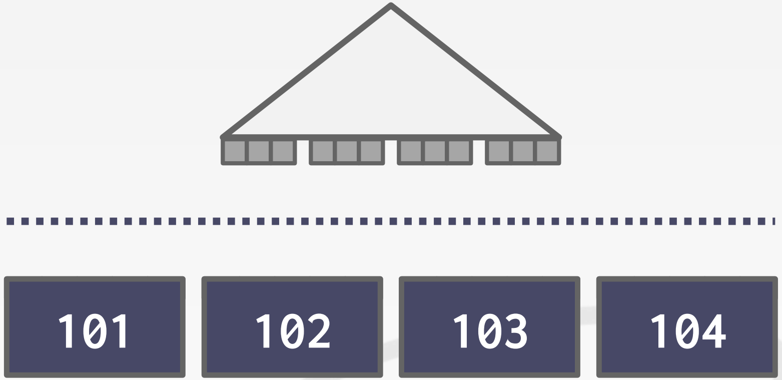


CARNEGIE MELLON DATABASE GROUP

INDEX SCAN PAGE SORTING

Retrieving tuples in the order that appear in an unclustered index is inefficient.

The DBMS can first figure out all the tuples that it needs and then sort them based on their page id.

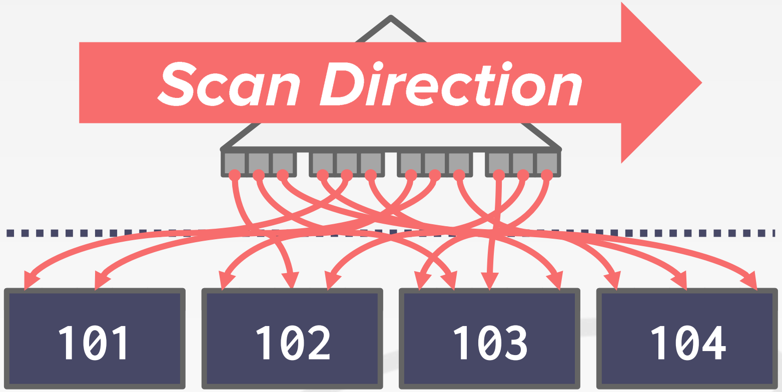


CARNEGIE MELLON DATABASE GROUP

INDEX SCAN PAGE SORTING

Retrieving tuples in the order that appear in an unclustered index is inefficient.

The DBMS can first figure out all the tuples that it needs and then sort them based on their page id.

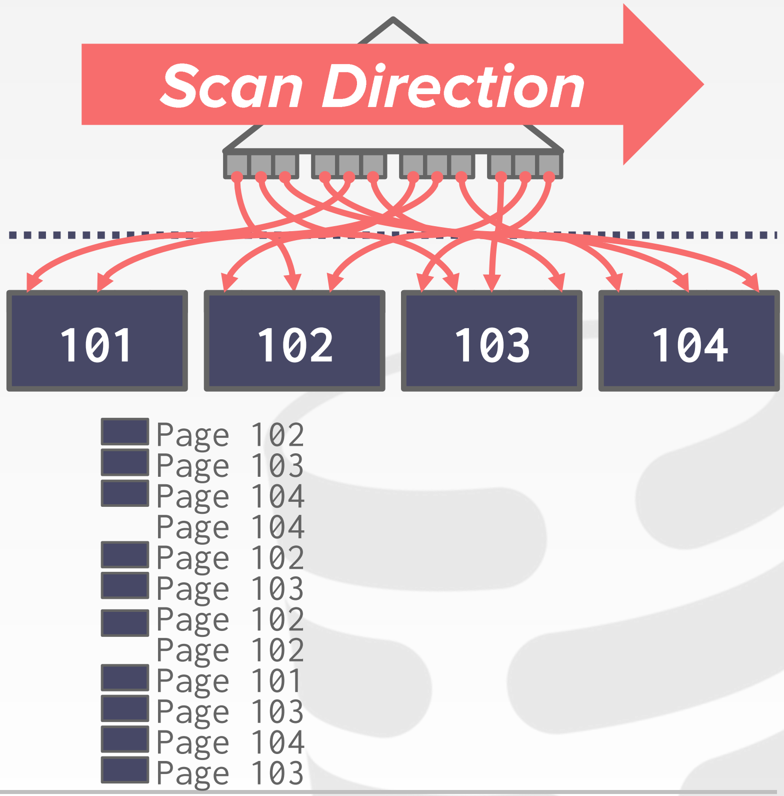


CARNEGIE MELLON DATABASE GROUP

INDEX SCAN PAGE SORTING

Retrieving tuples in the order that appear in an unclustered index is inefficient.

The DBMS can first figure out all the tuples that it needs and then sort them based on their page id.

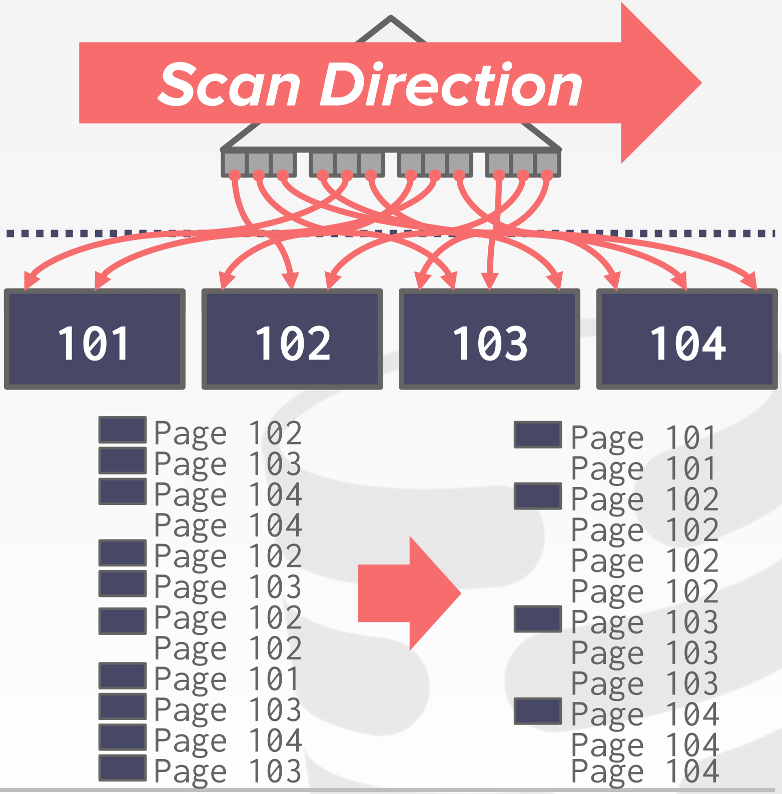


CARNEGIE MELLON DATABASE GROUP

INDEX SCAN PAGE SORTING

Retrieving tuples in the order that appear in an unclustered index is inefficient.

The DBMS can first figure out all the tuples that it needs and then sort them based on their page id.



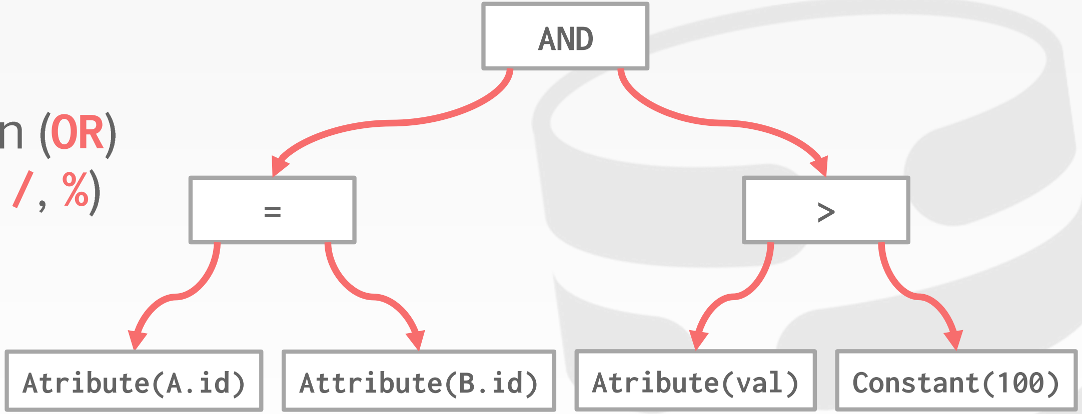
EXPRESSION EVALUATION

The DBMS represents a WHERE clause as an **expression tree**.

The nodes in the tree represent different expression types:

→ Comparisons (=, <, >, !=)
→ Conjunction (AND), Disjunction (OR)
→ Arithmetic Operators (+, -, *, /, %)
→ Constant Values
→ Tuple Attribute References

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND B.val > 100
```



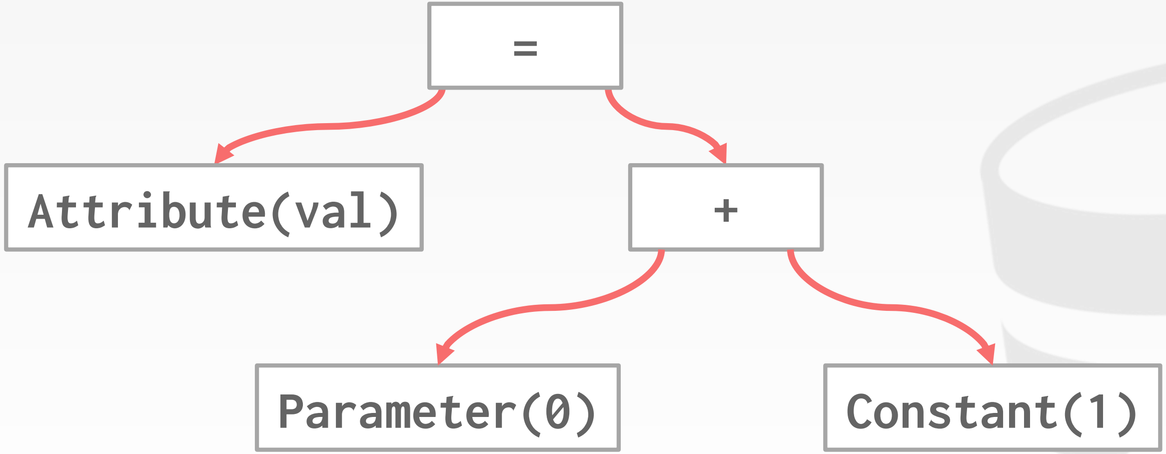
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

```
SELECT * FROM B
 WHERE B.val = ? + 1
```

Execution Context

Current Tuple	Query Parameters	Table Schema
(123, 1000)	(int:999)	B→(int:id, int:val)



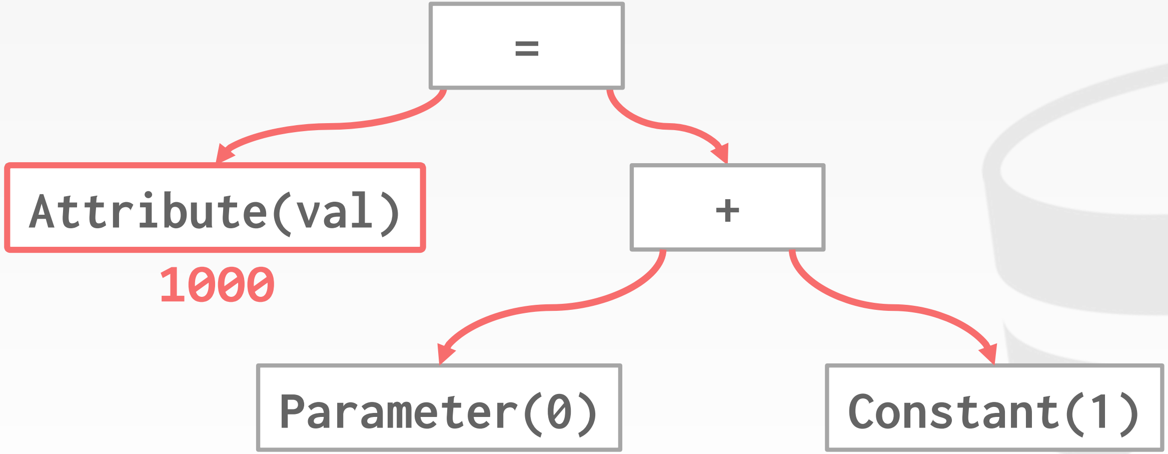
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

```
SELECT * FROM B
 WHERE B.val = ? + 1
```

Execution Context

Current Tuple	Query Parameters	Table Schema
(123, 1000)	(int:999)	B→(int:id, int:val)



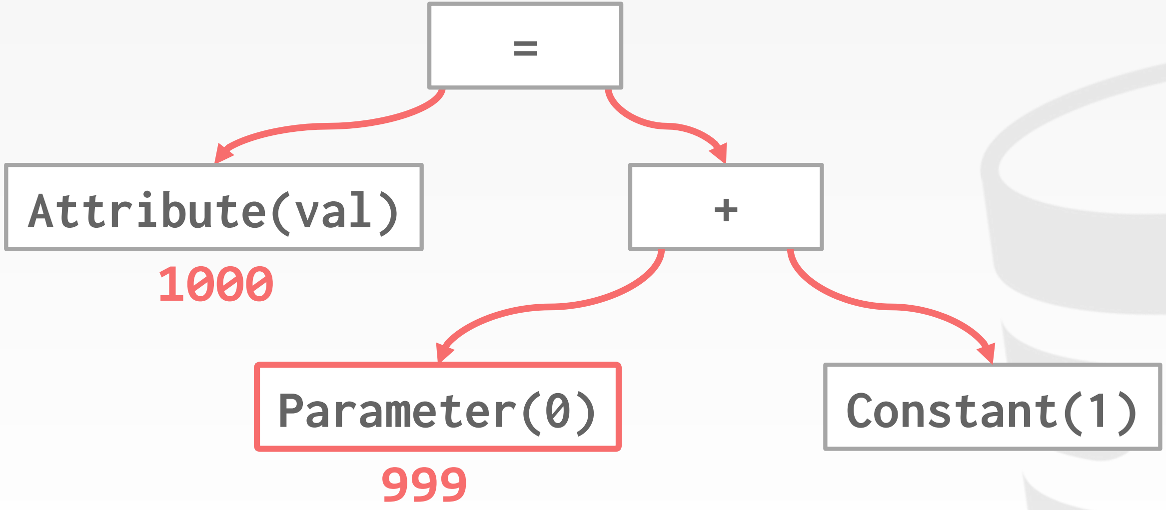
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

```
SELECT * FROM B
 WHERE B.val = ? + 1
```

Execution Context

Current Tuple	Query Parameters	Table Schema
(123, 1000)	(int:999)	B→(int:id, int:val)



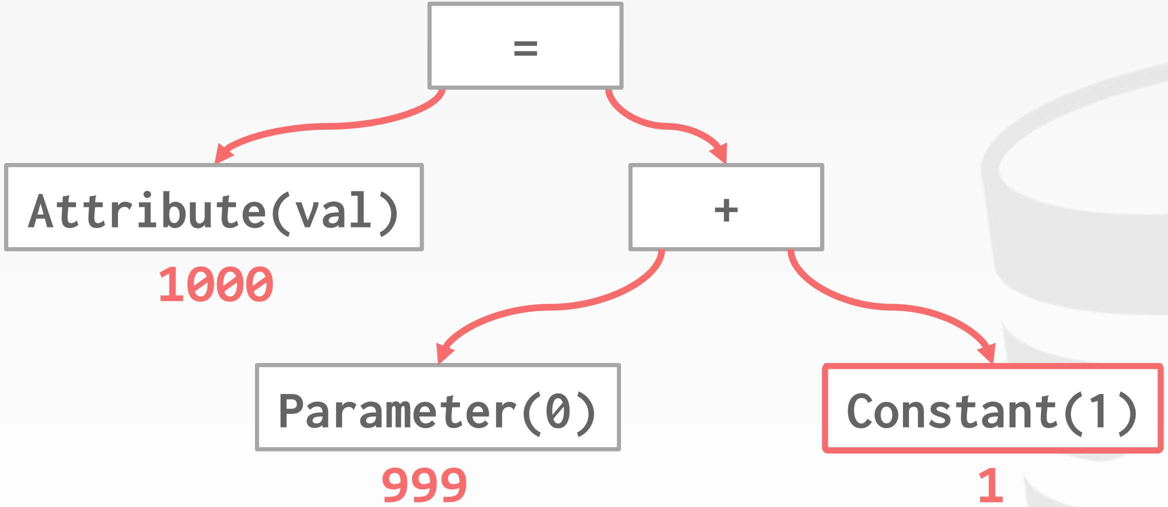
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

```
SELECT * FROM B
 WHERE B.val = ? + 1
```

Execution Context

Current Tuple	Query Parameters	Table Schema
(123, 1000)	(int:999)	B→(int:id, int:val)



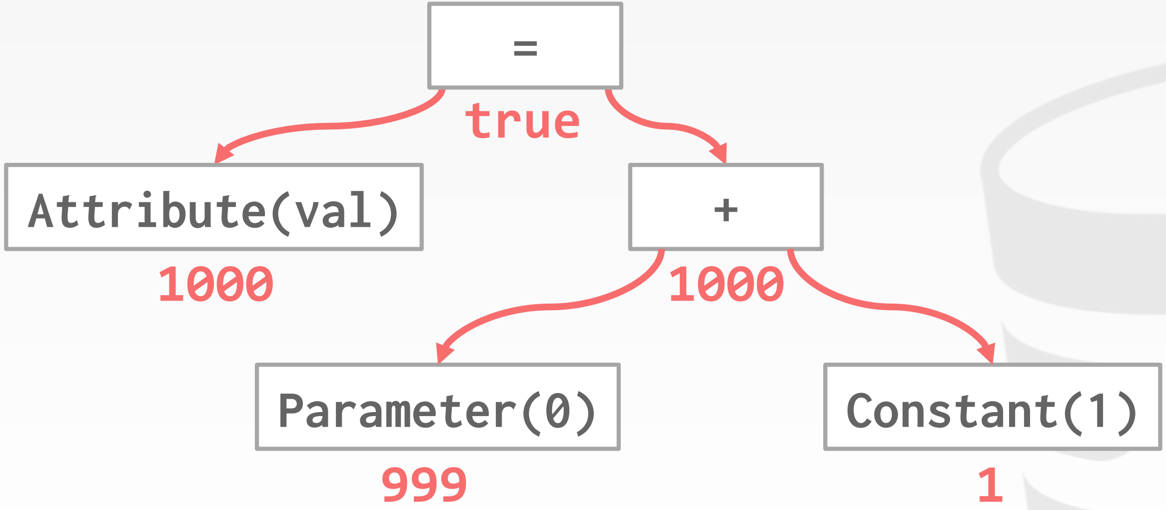
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

```
SELECT * FROM B
 WHERE B.val = ? + 1
```

Execution Context

Current Tuple	Query Parameters	Table Schema
(123, 1000)	(int:999)	B→(int:id, int:val)



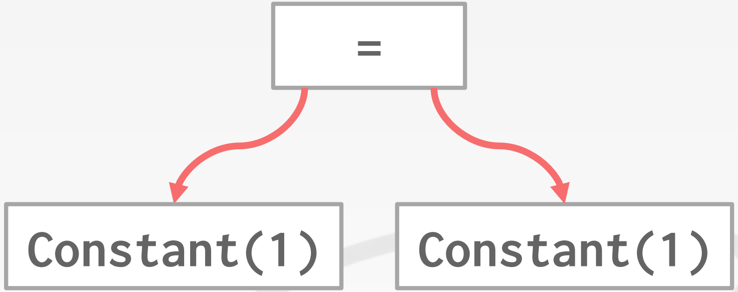
CARNEGIE MELLON DATABASE GROUP

EXPRESSION EVALUATION

Evaluating predicates in this manner is slow.

→ The DBMS traverses the tree and for each node that it visits it has to figure out what the operator needs to do.

Consider **WHERE 1=1**



A better approach is to just evaluate the expression directly.

→ Think JIT compilation

CARNEGIE MELLON DATABASE GROUP

CONCLUSION

The same query plan be executed in multiple ways.

(Most) DBMSs will want to use an index scan as much as possible.

Expression trees are flexible but slow.

CARNEGIE MELLON
DATABASE GROUP

PROJECT #2

You will build a **single-threaded** B+tree index.

- → Page Layout
- → Data Structure
- → Iterator.

We define the API for you. You need to provide the method implementations.

Due Date:
Wednesday Oct 25th

http://15445.courses.cs.cmu.edu/fall2017/project2/

CARNEGIE MELLON
DATABASE GROUP

THINGS TO NOTE

Do **not** change any file other than the six that you have to hand it.

We will provide an updated source tarball. You will need to copy over your files from Project #1.

Post your questions on Canvas or come to TA office hours.
→ We will **not** help you debug.

CARNEGIE MELLON DATABASE GROUP

PLAGIARISM WARNING

Your project implementation must be your own work.

→ You may **not** copy source code from other groups or the web.

→ Do **not** publish your implementation on Github.

Plagiarism will **not** be tolerated. See CMU's Policy on Academic Integrity for additional information.

NEXT CLASS

More query execution
→ External Merge Sort
→ Join Algorithms

CARNEGIE MELLON DATABASE GROUP