**Iterators, Relational Operators and Joins**

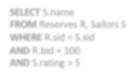
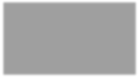
Alvin Cheung

Aditya Parameswaran 

R&G Chapters 12 & 14

Recall from Last Lecture

SQL Query

**SELECT** S.name 

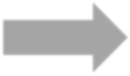
**FROM** Reserves R, Sailors S **WHERE** R.sid = S.sid

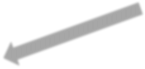
**AND** R.bid = 100

**AND** S.rating > 5

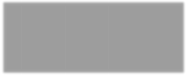
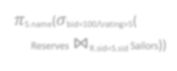
**(Logical) Query Plan:**

!S.name

Query Parser & Optimizer 

Equivalent to…

Relational Algebra

!S.name("bid=100⋀rating>5( 

Reserves⋈R.sid=S.sid Sailors))

**Optimized (Physical) Query Plan:** On-the-fly Project Iterator 

!S.name

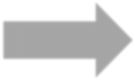
"S.rating>5

On-the-fly

"R.bid=100 ⋀ S.rating > 5

But actually will Select Iterator

⋈R.sid=S.sid

produce…⋈R.sid=S.sid

Indexed Nested Loop Join Iterator 

Operator Code

"R.bid=100

Sailors

Heap Scan

Reserves Sailors

B+-Tree Iterator Indexed Scan 

Iterator

Reserves

Relational Operators and Query Plans !sname(!sid(!bid("color=‘red’(Boats)) ⋈ Res) ⋈ Sailors)

• Expression Tree Representation = Query plan • Edges encode “flow” of tuples

• Vertices = Relational Alg Operators

"sname ⋈

• Source vertices = table access operators

• Also called dataflow graph

• Here, “flow of tuples”

• Not specific to DBMSs

• E.g., “big data systems”, ML systems

"bid 

!color=‘red Boats 

"sid 

⋈ 

Reserves Sailors

Query Executor Instantiates Operators !sname(!sid(!bid("color=‘red’(Boats)) ⋈ Res) ⋈ Sailors)

• Query optimizer selects operators to run in

sequence (i.e., the query plan)

On-the-fly 

Project Iterator

"sname

• Query executor runs these operators by creating instances thereof

• Each operator instance:

• Implements **iterator interface**

****⋈ On-the-fly 

On-the-fly

Project Iterator "sid 

⋈

Index NL Join Iterator

• Efficiently executes operator logic forwarding tuples

to next operator

Project Iterator Index NL Join Iterator 

"bid 

On-the-fly

Select Iterator 

Operator Code B+-Tree

!color=‘red

B+-Tree 

Indexed

Scan Iterator

B+-Tree 

Indexed

Scan Iterator

Indexed 

Scan Iterator

Boats

Reserves Sailors

Iterator Interface

The relational operators implemented as subclasses of the class Iterator:

abstract class iterator { // Invoked when “wiring” dataflow

void setup(List<Iterator> inputs); // configuring the input (children) args void init(args); // Invoked before calling next: sets up state tuple next(); // Invoked repeatedly: return another tuple

void close(); // Invoked when finished }

• **Pull-**based computation model

• e.g., Console calls **init** on root operator of query plan, and then **next**

• If tuple is not ready, this **next** request propagates down the query plan recursively • init/next can result in either *streaming (“on-the-fly”)* or *blocking (“batch”)* algorithm for that operator**:** • streaming: small, constant amount of work per call

• blocking: does not produce output until it consumes its entire input, i.e., all rows from children! • Q: examples?

• Any iterator can be input to any other, since they all implement the same interface (composability) • **State:** iterators may maintain substantial private “internal” state

• e.g., hash tables, running counts, large sorted files …

Example: Select (on-the-fly)

• A streaming operator: small amount of work per tuple produced

• init(predicate):

child.init()

pred = predicate; // local variable storing state current = NULL; // local cursor

• next():

while (current != EOF && !pred(current))

current = child.next(); // give us another tuple } // exit if pred is satisfied or EOF

return current; // return current tuple or EOF • close():

child.close()

Example: Heap Scan

• Leaf of the query plan

• init(relation):

heap = open heap file for this relation; // file handle

cur\_page = heap.first\_page(); // first page

cur\_slot = cur\_page.first\_slot(); // first slot on that page • next():

if (cur\_page == NULL) return EOF;

current = [cur\_page, cur\_slot]; // we will return this recordId cur\_slot = cur\_slot.advance(); // advance the slot for subseq. calls if (cur\_slot == NULL) { // advance to next page, first slot cur\_page = cur\_page.advance();

if (cur\_page != NULL)

cur\_slot = cur\_page.first\_slot();

}

return current;

• close():

heap.close() // close file

Example: Sort (2-pass)

• init(keys):

// input keys to sort by

// first, all of pass 0, a blocking call

child.init()

repeatedly call child.next() and generate the sorted runs on disk, until child gives EOF

// second, set up for pass 1, assumes enough buffers to merge

open each sorted run file and load one page per run into input buffer for pass 1 • next(): // pass 1 merge (assumes enough buffers to merge) output = min tuple across all buffers

if min tuple was last one in its buffer, fetch next page from that run into buffer return output // (or EOF if no tuples remain)

• close():

deallocate the runs files

child.close()

Example: Group By on Sorted input

**agg\_type state init merge(x) final**

**COUNT** count 0 count ++ count

**SUM** sum 0 sum += x sum

**AVG** ? ? ? ?

**MIN** ? ? ? ?

• Say input is sorted, and we want to do a group by • Not necessary, can also do group by without sorting (see later)



GroupBy Sort 



• Similar approach for “merging” tuples per group to form a result tuple per group across aggregates

• Initialize some state per group

• Operate one tuple at a time, and do the ”merge” with existing state • Return result tuple when done with group

Example: Group By on Sorted input

**agg\_type state init merge(x) final**

**COUNT** count 0 count ++ count

**SUM** sum 0 sum += x sum



GroupBy Sort 



**AVG** [count, sum]

[0, 0] [count++, sum+=x]

sum

/ count

**MIN** min +infinity min > x ? x : min

min

• Say input is sorted, and we want to do a group by

• Not necessary, can also do group by without sorting (see later)

• Similar approach for “merging” tuples per group to form a result tuple per group across aggregates

• Initialize some state per group

• Operate one tuple at a time, and do the ”merge” with existing state • Return result tuple when done with group

Example: Group By on Sorted input **agg\_type state init merge(x) final**

**COUNT** count 0 count ++ count



GroupBy

• init(group\_keys, aggs):

**SUM** sum 0 sum += x sum

Sort

child.init()

cur\_group = NULL;

**AVG** [count, sum]

[0, 0] [count++, sum+=x]

sum 

/ count

• next():

result = NULL

do {

tup = child.next();

**MIN** min +infinity min > x ? x : min

min

if (group(tup) != cur\_group) { // New group!

if (cur\_group != NULL) // Have we seen a group previously? result = [cur\_group, final() of all aggs] // Form result for that cur. group cur\_group = group(tup); // Initialize new group

call init() on all the aggs

}

call merge(tup) on all the aggs

} while (!result); // Exit if cur. grp result is formed return result;

• close():

child.close()

Neat: only maintains one tuple of partial results in memory at any time!

A Full (Single Table) Query Plan

• A Query Plan is Single-threaded!

• Exercise: trace the calls that lead to flow of tuples:

• Call init() on the root GroupBy

• How does init() recurse down the chain and return?

• Call next() on root

• How does next() recurse down the chain and return a tuple?

• Note how the blocking operator (sort) interacts with the other, streaming operators • Select and GroupBy are essentially streaming operators

• Note how we don’t store each operator output on disk;

tuples stream through the plan’s call stack



GroupBy 

Sort Select 



HeapScan

• Some operators like Sort use disk internally – but not exposed outside the operator • The iterator framework itself is lightweight

• Next: Binary Iterators

Join Operators

R&G 14.4



Schema & Costing for Examples

• Cost Notation

• [R] : the number of pages to store R

• pR : number of records per page of R

• |R| : the cardinality (number of records) of R

• |R| = pR\*[R]

• Reserves (sid: int, bid: int, day: date, rname: string) • [R]=1000, pR=100, |R| = 100,000

• Sailors (sid: int, sname: string, rating: int, age: real) • [S]=500, pS=80, |S| = 40,000

Simple Nested Loops **θ** Join

foreach **record** r in R do

foreach **record** s in S do

if **θ(r, s)** then add <r, s> to result buffer R

*Note: for simplicity we do not present iterator implementations for the join algorithms*

*Note: ignore cost of writing out; it is constant across approaches; plus in many cases data streams through them via next()*

**S 

[R]=1000, pR=100, |R| = 100,000

[S]=500, pS=80, |S| = 40,000

Cost: scan R once + scan S once per R tuple = [R] + |R|[S]

= 50,001,000

Changing the Join Order

foreach **record** s in S do

foreach **record** r in R do

if **θ(r, s)** then add <r, s> to result buffer R





[R]=1000, pR=100, |R| = 100,000 [S]=500, pS=80, |S| = 40,000 

Cost: scan S once + scan R once per S tuple

= [S] + |S|[R] = 40,000,500 S 

vs. 50,001,000

Join orders matter! Q: Can we improve this?

Page Nested Loop Join

**Idea: previous algo was inefficient w.r.t. I/O: operate at granularity of pages!** for each rpage in R:



for each spage in S:

for each rtuple in rpage:

for each stuple in spage:

if **θ**(rtuple, stuple):









add <rtuple, stuple> to result buffer

Cost = Scan R once, and scan S per page of R = [R] + ([R] \* [S]) = 501,000 vs. ~40M

SQ: Can we improve this?

**“Chunk”**

“Block” Nested Loop Join

**Idea: Extending even further using a “block” or a “chunk” of S pages at a time**

for each rchunk of B-2 pages of R:

for each spage of S:

for all matching tuples in spage and rchunk:

add <rtuple, stuple> to result buffer







Cost = Scan R once, plus scan S as many times as there are chunks = [R] + ⌈ [R]/(B-2) ⌉ \* [S]

= 1000 + ⌈ 1000/(B-2) ⌉ \* 500

= 6,000 for B=102 (~100x better than Page NL!)

S Overall, a frequently used join algorithm, esp. for non-eq. predicates

Index Nested Loops Join Consider when we have equijoin on ri = sj

foreach **tuple** r in R do

foreach **tuple** s in S where **ri == sj** do add <r, s> to result buffer

R

**Leaf**

**Data entries**

**s S:**

lookup(ri) INDEX on S

**Data Records**

Index Nested Loops Join Cost

foreach **tuple** r in R do

foreach **tuple** s in S where **ri == sj** do

add <r, s> to result

Cost = [R] + |R| \* cost to find matching S tuples

• If index uses Alt. 1 🡪 cost to traverse tree from root to leaf. (e.g., 2-4 IOs) • For Alt. 2 or 3:

• Cost to lookup RID(s); typically 2-4 IOs for B+Tree.

• Cost to retrieve records from RID(s)

• Clustered index: 1 I/O per *page of matching S tuples*.

• Unclustered index: up to 1 I/O per matching S tuple

Sort-Merge Join

• Requires equality predicate:

• Equi-Joins & Natural Joins

• Two Stages:

• Sort tuples in R and S by join key

• All tuples with same key in consecutive order

• Input might already be sorted … maybe result of another sort merge/index scan? • Join Pass: Merge-scan the sorted partitions and emit tuples that match • Challenge is that each tuple in R may match multiple tuples in S

• We keep track of the start of each block of S tuples with a “mark”

• That way, we know where to return for the next tuple of R

• R is “outer loop”, advances forward; S is “inner loop” forward + back to mark

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid bid** 28 103 28 104 31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname**

****22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid** 28 103 28 104

31 101 31 102 

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid** 28 103 28 104

31 101 31 102 

42 142 58 107

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid** 28 103 28 104 31 101 31 102 42 142 58 107 

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid** 28 103 28 104 31 101 31 102 42 142 58 107 

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

**sid bid**

28 103

28 104

31 101

31 102

42 142

58 107 

Sort-Merge Join

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid**

28 yuppy 103

28 yuppy 104

31 lubber 101

31 lubber 102

31 lubber2 101

31 lubber2 102

58 rusty 107

**sid bid**

28 103

28 104

31 101

31 102

42 142

58 107 

Sort-Merge Join, Part 1

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

****22 dustin 28 yuppy

31 lubber 31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 2

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

****22 dustin 28 yuppy

31 lubber 31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 3

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 4

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 5

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 6

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid**

****28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 7

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname** 22 dustin 28 yuppy 31 lubber

31 lubber2 44 guppy 58 rusty

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join, Part 8

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join, Part 9

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join, Part 10

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join, Part 11

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103

**sid bid** 28 103 28 104 31 101

31 102 42 142 58 107

Sort-Merge Join, Part 12

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 13

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 14

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 15

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 16

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy 31 lubber

31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 17

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 18

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 19

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 20

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid **28 103 28 104

31 101 31 102 42 142 58 107

Sort-Merge Join, Part 21

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 22

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 23

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 24

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 25

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join, Part 26

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join, Part 27

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join, Part 28

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join, Part 29

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102 42 142

58 107

Sort-Merge Join, Part 30

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 31

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 32

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 33

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 34

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber 31 lubber2

44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 35

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2 44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 36

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2 44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 37

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2 44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107

Sort-Merge Join, Part 38

**do {**

if (!mark) {

while (r < s) { advance r } while (r > s) { advance s } // mark start of “block” of S mark = s

}

if (r == s) {

result = <r, s>

advance s

return result

}

else {

reset s to mark

advance r

mark = NULL

}

}

**sid sname**

22 dustin

28 yuppy

31 lubber

31 lubber2 44 guppy

58 rusty

**sid sname bid** 28 yuppy 103 28 yuppy 104 31 lubber 101 31 lubber 102

**sid bid** 28 103 28 104

31 101 31 102

42 142 58 107