

RELATIONAL OPERATORS: SELECTION

CS 564- Fall 2021

ACKs: Jeff Naughton, Jignesh Patel, AnHai Doan

WHAT IS THIS LECTURE ABOUT?

Algorithms for the **selection operator**

LOGICAL VS PHYSICAL OPERATORS

- Logical operators
 - *what* they do
 - e.g., union, selection, project, join, grouping
- Physical operators
 - *how* they do it
 - e.g., nested loop join, sort-merge join, hash join, index join

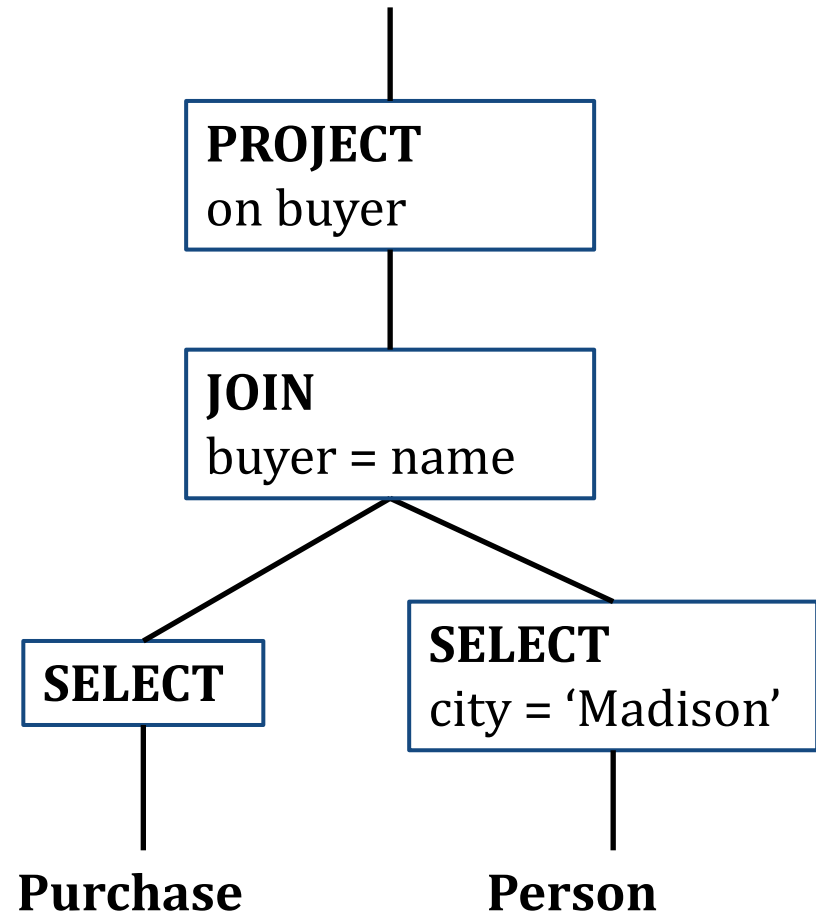
EXAMPLE QUERY

```
SELECT P.buyer
FROM   Purchase P, Person Q
WHERE  P.buyer=Q.name
AND    Q.city='Madison'
```

- Assume that Person has a B+ tree index on city

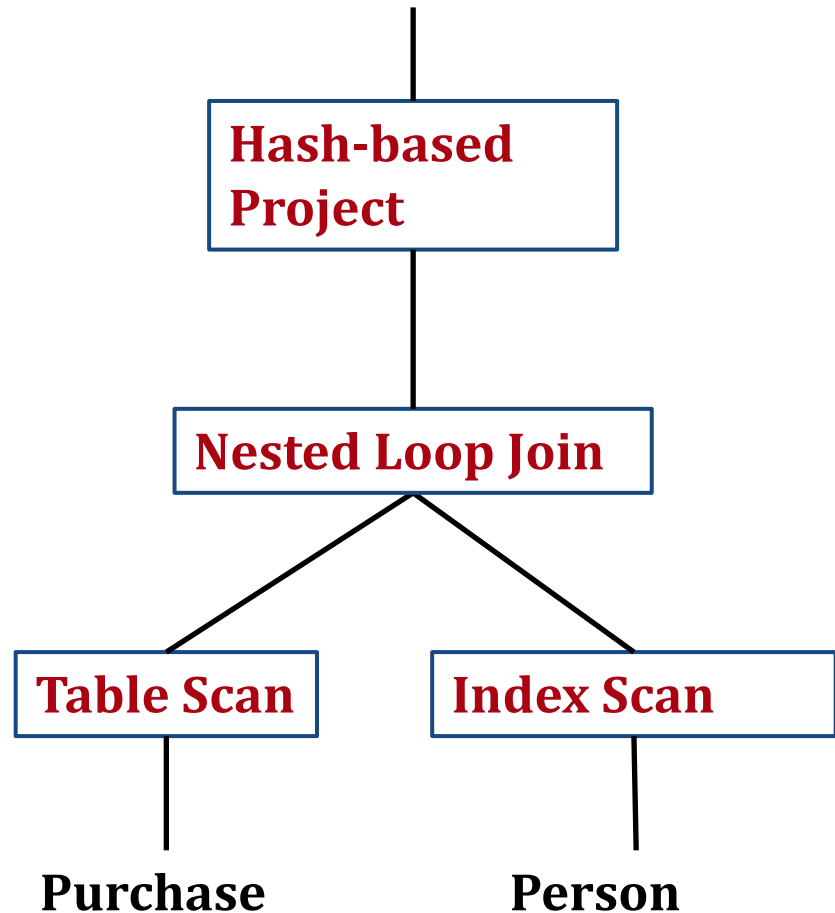
EXAMPLE: LOGICAL PLAN

SELECT P.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name
AND Q.city='Madison'



EXAMPLE: PHYSICAL PLAN

```
SELECT P.buyer
FROM   Purchase P, Person Q
WHERE  P.buyer=Q.name
AND    Q.city='Madison'
```



SELECTION

SELECT OPERATOR

access path = way to retrieve tuples from a table

- **File Scan:**

- scan the entire file
- I/O cost: number of pages N

- **Index Scan:**

- use an index available on some predicate
- I/O cost: it varies depending on the index

INDEX SCAN COST

- **Hash index:** $O(1)$
 - but we can only use it with equality predicates!
- **B+ tree index:** the cost depends on whether the index is clustered or not:
 - *unclustered*: we may need to access as many pages from the file as the (#selected tuples)
 - *clustered*: we need to access (#selected tuples) / (#tuples per page) pages in the file
 - **optimization**: we can sort the rids by page number before we retrieve them from the unclustered index

B+ TREE SCAN: EXAMPLE

- A relation with **1,000,000** records
- **100** records on a page (of the heap file)
- **500** (key, rid) pairs on a leaf page
- height of B+ tree = **3**

selectivity = percentage of tuples that satisfy the selection condition

Clustered index + 1% selectivity

- #tuples to retrieve: $1\% (1,000,000) = 10,000$
- #leaf pages = $10,000 / 500 = \mathbf{20}$
- #pages in the heap file = $10,000 / 100 = \mathbf{100}$
- I/O cost = $3 + 20 + 100 = \mathbf{123 \text{ I/Os}}$

B+ TREE SCAN: EXAMPLE

- A relation with **1,000,000** records
- **100** records on a page (of the heap file)
- **500** (key, rid) pairs on a leaf page
- height of B+ tree = **3**

selectivity = percentage of tuples that satisfy the selection condition

Unclustered index + 1% selectivity

- #tuples to retrieve: $1\% (1,000,000) = 10,000$
- #leaf pages = $10,000 / 500 = \mathbf{20}$
- #pages in the heap file = **10,000** (in the worst case)
- I/O cost = $3 + 20 + 10,000 = \mathbf{10,023 \text{ I/Os}}$

B+ TREE SCAN: EXAMPLE

- A relation with **1,000,000** records
- **100** records on a page (of the heap file)
- **500** (key, rid) pairs on a leaf page
- height of B+ tree = **3**

selectivity = percentage of tuples that satisfy the selection condition

	1% selectivity	10% selectivity
clustered	3+20+100	3+200+1,000
unclustered	3+20+10,000	3+200+100,000
unclustered + sorting	3+20+(~10,000)	3+200+(~10,000)

if we first sort, we will read at most all the pages in the B+ tree

GENERAL SELECTIONS

- So far we studied selection on a single attribute
- How do we use indexes when we have multiple selection conditions?
 - $R.A = 10 \text{ AND } R.A > 10$
 - $R.A = 10 \text{ OR } R.B < 20$

INDEX MATCHING

We say that an index *matches* a selection predicate if the index can be used to evaluate it efficiently

- relation $R(A, B, C, D)$
- hash index on composite key (A, B)

```
SELECT *  
FROM R  
WHERE A = 10 AND B = 5 ;
```

match

```
SELECT *  
FROM R  
WHERE A = 5 ;
```

no match

INDEX MATCHING: HASH INDEX

- **selection** = $\text{pred}_1 \text{ AND } \text{pred}_2 \text{ AND } \dots$
- $\text{pred}_i = (\text{attribute}) \{<, \leq, \geq, >, =, \neq\}$ (constant)

A hash index on (A, B, \dots) **matches** the selection condition if *all* attributes in the index search key appear in a predicate with equality (=)

EXAMPLE

relation $R(A, B, C, D)$

selection condition	hash index on (A,B,C)	hash index on (B)
$A=5 \text{ AND } B=3$	no	yes
$A>5 \text{ AND } B<4$	no	no
$B=3$	no	yes
$A=5 \text{ AND } C>10$	no	no
$A=5 \text{ AND } B=3 \text{ AND } C=1$	yes	yes
$A=5 \text{ AND } B=3 \text{ AND } C=1 \text{ AND } D > 6$	yes	yes

INDEX MATCHING: B+ TREE

- **selection** = $\text{pred}_1 \text{ AND } \text{pred}_2 \text{ AND } \dots$
- $\text{pred}_i = (\text{attribute}) \{<, \leq, \geq, >, =, \neq\} (\text{constant})$

A B+ tree index on (A, B, \dots) matches the selection condition if:

- the attributes in the predicates form a prefix of the search key of the B+ tree
- any operators can be used (not only equality!)

EXAMPLE

relation $R(A, B, C, D)$

selection condition	B+ tree on (A,B,C)	B+ tree on (B,C)
$A=5 \text{ AND } B=3$	yes	yes
$A>5 \text{ AND } B<4$	yes	yes
$B=3$	no	yes
$A=5 \text{ AND } C>10$	yes	no
$A=5 \text{ AND } B=3 \text{ AND } C=1$	yes	yes
$A=5 \text{ AND } B=3 \text{ AND } C=1 \text{ AND } D > 6$	yes	yes

MORE ON INDEX MATCHING

A predicate can match *more than one* index

- hash index on (A) and B+ tree index on (B, C)
- selection: A=7 **AND** B=5 **AND** C=4

Which index should we use?

1. use the hash index, then check the conditions B=5, C=4 for every retrieved tuple
2. use the B+ tree, then check the condition A=7 for every retrieved tuple
3. use both indexes, intersect the rid sets, and only then fetch the tuples

SELECTION WITH DISJUNCTION

- **selection** = $\text{pred}_1 \text{ OR } \text{pred}_2 \text{ OR } \dots$
- $\text{pred}_i = (\text{attribute}) \{<, \leq, \geq, >, =, \neq\} (\text{constant})$

The available indexes need to match **every** predicate in the disjunction!

DISJUNCTION: EXAMPLE

- hash index on (A) + hash index on (B)
- selection: $A=7$ **OR** $B>5$
- Only the first predicate matches an index
- The only option is to do a file scan

DISJUNCTION: EXAMPLE

- hash index on (A) + B+ tree on (B)
- $A=7$ OR $B>5$
- One solution is to do a file scan
- A second solution is to use both indexes, fetch the rids, and then do a union, and only then retrieve the tuples

Why do we need to perform the union before fetching the tuples?

GENERAL FORMULA

- hash index on (A) + B+ tree on (B)
- **(A=7 OR C>5) AND B > 5**
- We can use the B+ tree to fetch the tuples that satisfy the second predicate (B >5), then filter according to the first

CHOOSING THE RIGHT INDEX

Selectivity of an access path = *fraction* of tuples that need to be retrieved

- We want to choose the *most selective* path!
- Estimating the selectivity of an access path is generally a hard problem

ESTIMATING SELECTIVITY (1)

- selection: $A=3$ AND $B=4$ AND $C=5$
- hash index on (A,B,C)

The selectivity can be approximated by the formula:

$$1/(\# \text{ search keys})$$

- $\#keys$ is known from the index
- this assumes that the values are distributed *uniformly* across the tuples

EXAMPLE

- selection: $A=3$ **AND** $B=4$ **AND** $C=5$
- *clustered* hash index on (A,B,C)
- $\#pages = 10,000$
- $\#keys\ in\ hash\ index = 100$

- selectivity = 1%
- number of pages retrieved = $10,000 * 1\% = 100$
- I/O cost = $100 + (\text{a small constant})$

ESTIMATING SELECTIVITY (2)

- selection: $A=3$ AND $B=4$ AND $C=5$
- hash index on (B,A)

If we don't know the *#keys* for the index, we can estimate selectivity as follows:

- multiply the **selectivity** for each primary conjunct
- If *#keys* is not known for an attribute, use 1/10 as default value
- this assumes independence of the attributes!

ESTIMATING SELECTIVITY (3)

- Selection: $A > 10$ AND $A < 60$
- If we have a range condition, we assume that the values are uniformly distributed
- The selectivity will be approximated by $\frac{\text{interval}}{\text{High} - \text{Low}}$

Example: if A takes values in $[0, 100]$ then the selectivity will be $\sim \frac{60 - 10}{100 - 0} = 50\%$

ESTIMATING SELECTIVITY (3)

- Selection: $A > 10$ AND $A < 60$
- If we have a range condition, we assume that the values are uniformly distributed
- The selectivity will be approximated by $\frac{\text{interval}}{\text{High} - \text{Low}}$

Example: if A takes values in $[0, 100]$ then the selectivity will be $\sim \frac{60-10}{100-0} = 50\%$