Relational Operators #1

CS 564- Fall 2018

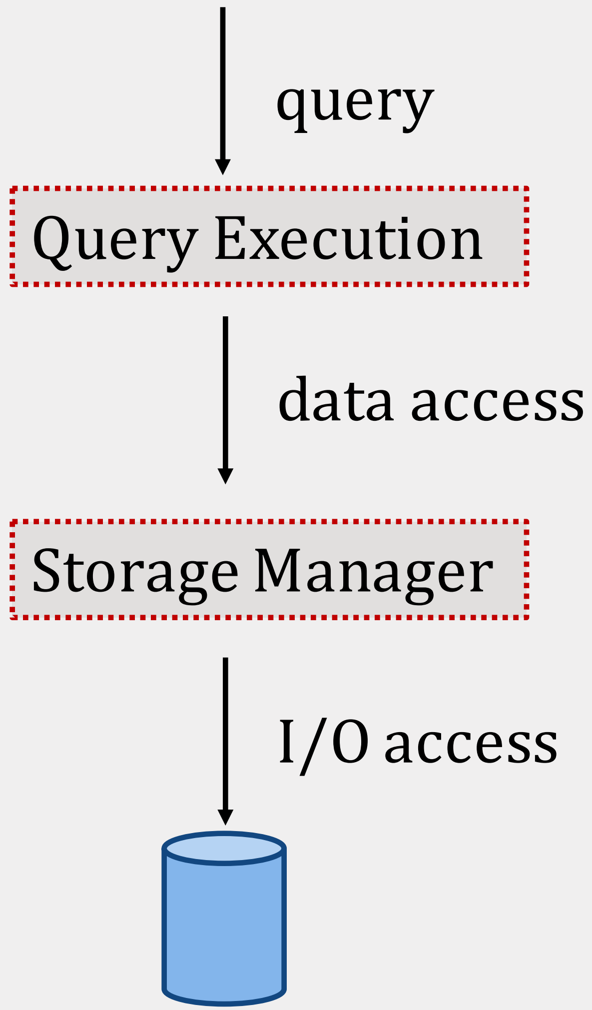
ACKs: *Jeff Naughton, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT?

Algorithms for relational operators:

- select
- project

ARCHITECTURE OF A DBMS



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

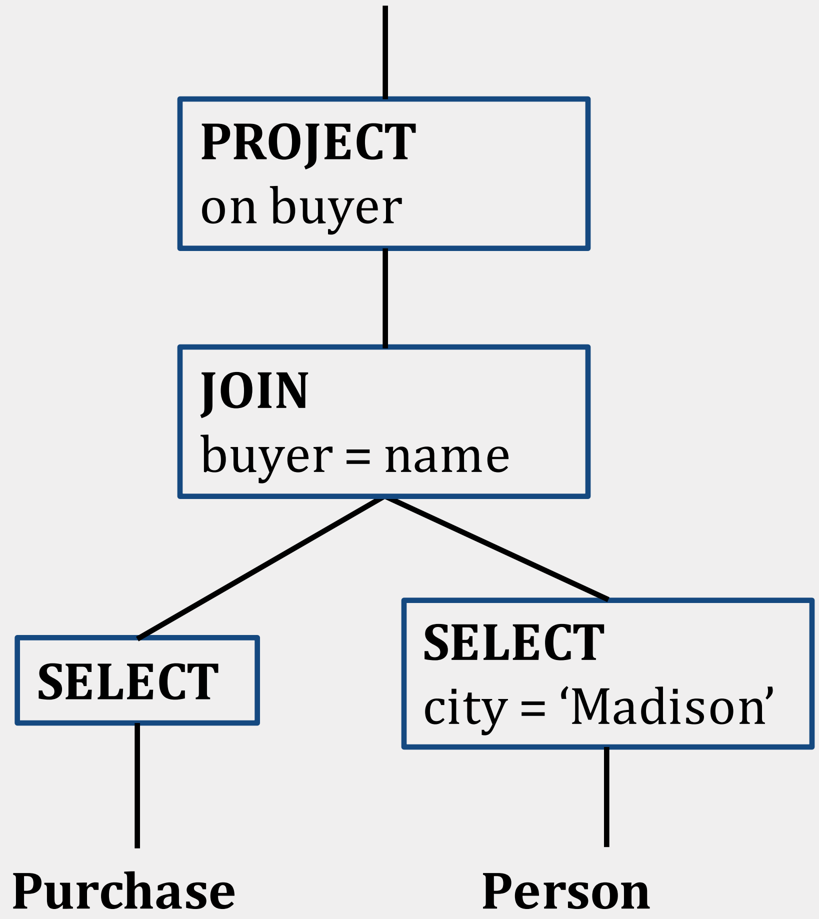
EXAMPLE QUERY

```
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

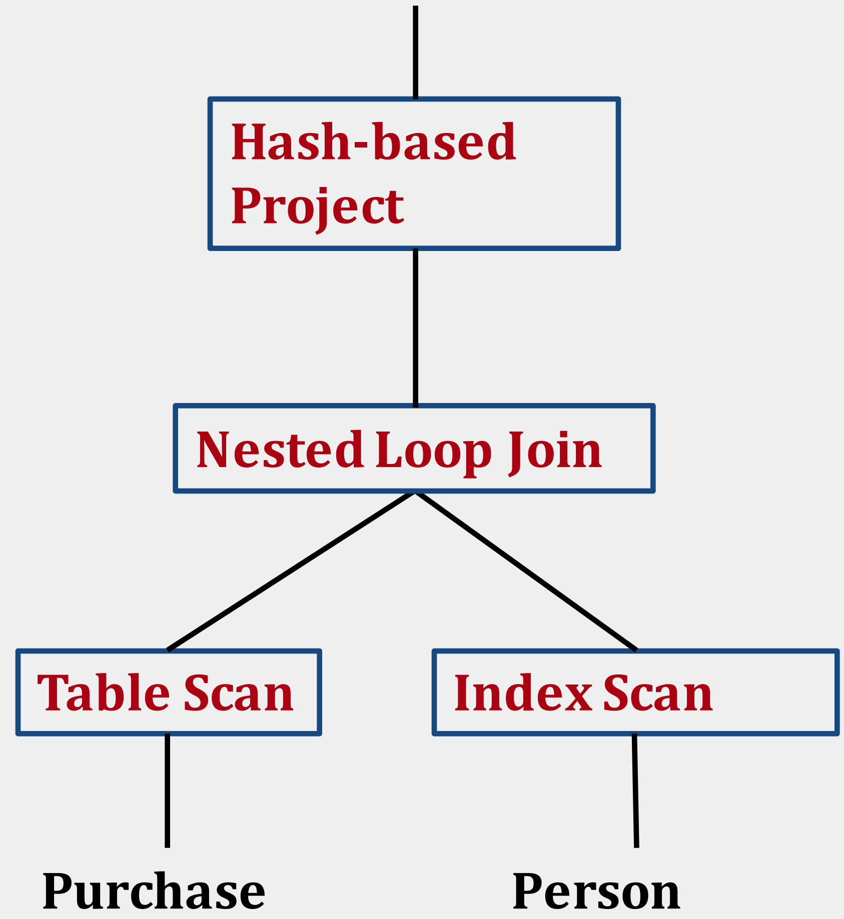
EXAMPLE: LOGICAL PLAN

```
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: O(N), where N = #pages
- **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: height – L_B + 1 + X
  - X depends on whether the index is clustered or not:
    - *unclustered*: X = # selected tuples in the worst case
    - *clustered*: X = (#selected tuples)/ (#tuples per page)
    - **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
- 500 (key, rid) pairs on a page
- height of B+ tree = 3

<u>selectivity</u> = percentage of tuples that satisfy the selection condition

	1% selectivity	**10% selectivity**
clustered	3+100	3+1000
unclustered	3+10,000	3+100,000
unclustered + sorting	3+(~10,000)	3+(~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` **`AND`** `R.A > 10`
  - `R.A = 10` **`OR`** `R.B < 20`

INDEX MATCHING

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

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

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

matches the index!

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

does not match the index!

INDEX MATCHING: HASH INDEX

selection = $\texttt{pred}_1$ AND $\texttt{pred}_2$ AND …

A hash index on $(A, B, \ldots)$ 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 **AND** B=3	no	yes
A>5 **AND** B<4	no	no
B=3	no	yes
A=5 **AND** C>10	no	no
A=5 **AND** B=3 **AND** C=1	yes	yes
A=5 **AND** B=3 **AND** C=1 **AND** D >6	yes	yes

The predicates A=5, B=3, C=1 that match the index are called **primary conjuncts**

INDEX MATCHING: B+ TREE

selection = `pred`$_1$ `AND` `pred`$_2$ `AND` …

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

- the attributes in the predicates form a prefix of the search key of the B+ tree
- any operations can be used (=, < , > , …)

EXAMPLE

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

selection condition	**B+ tree on (A,B,C)**	**B+ tree on (B,C)**
A=5 **AND** B=3	yes	yes
A>5 **AND** B<4	yes	yes
B=3	no	yes
A=5 **AND** C>10	yes	no
A=5 **AND** B=3 **AND** C=1	yes	yes
A=5 **AND** B=3 **AND** C=1 **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 (1)

- 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

SELECTION WITH DISJUNCTION

- 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?

SELECTION WITH DISJUNCTION

- 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: ***1/#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 + (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 $\frac{interval}{High-Low}$

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

PROJECTION

PROJECT OPERATOR

Simple case: `SELECT R.A, R.D`

- scan the file and for each tuple output R.A, R.D

Hard case: `SELECT DISTINCT R.A, R.D`

- project out the attributes
- eliminate *duplicate tuples* (this is the difficult part!)

PROJECT: SORT-BASED

Naïve algorithm:

1. scan the relation and project out the attributes
2. sort the resulting set of tuples using all attributes
3. scan the sorted set by comparing only adjacent tuples and discard duplicates

RUNNING EXAMPLE

`R(A,B,C,D,E)`

- $N = 1000$ pages
- $B = 20$ buffer pages
- Each field in the tuple has the same size
- Suppose we want to project on attribute `A`

SORT-BASED COST ANALYSIS

We will generally ignore the cost of writing the final result to disk, since it will be the same for every algorithm!

- initial scan = *1000 I/Os*
- after projection T =(1/5)*1000 = 200 pages
- cost of writing T = *200 I/Os*
- sorting in 2 passes = 2 * 2 * 200 = *800 I/Os*
- final scan = *200 I/Os*

total cost = 2200 I/Os

PROJECT: SORT-BASED

We can improve upon the naïve algorithm by modifying the sorting algorithm:

1. In Pass **0** of sorting, project out the attributes
2. In subsequent passes, eliminate the duplicates while merging the runs

SORT-BASED COST ANALYSIS

- we can sort in 2 passes
- pass **0** costs *1000 + 200 = 1200 I/Os*
- pass **1** costs *200 I/Os* (not counting writing the result to disk)

total cost = 1400 I/Os

PROJECT: HASH-BASED

2-phase algorithm:

- **partitioning**
  - project out attributes and split the input into B-1 partitions using a hash function *h*
- **duplicate elimination**
  - read each partition into memory and use an in-memory hash table (with a *different* hash function) to remove duplicates

PROJECT: HASH-BASED

When does the hash table fit in memory?

- size of a partition = $T \,/\, (B - 1)$, where T is #pages after projection
- size of hash table = $f \cdot T \,/\, (B - 1)$, where f is a fudge factor (typically ~ 1.2)
- So, it must be $B > f \cdot T \,/\, (B - 1)$, or approximately $B > \sqrt{f \cdot T}$

HASH-BASED COST ANALYSIS

- T = 200 so the hash table fits in memory!
- partitioning cost = 1000 + 200 = 1200 I/Os
- duplicate elimination cost = 200 I/Os

total cost = 1400 I/Os

COMPARISON

- Benefits of sort-based approach
  - better handling of skew
  - the result is sorted

- The I/O costs are the same if $B^2 > T$
  - 2 passes are needed by both algorithms

PROJECT: INDEX-BASED

- Index-only scan
  - projection attributes subset of index attributes
  - apply projection algorithm only to data entries
- If an *ordered index* contains all projection attributes as prefix of search key:
  1. retrieve index data entries in order
  2. discard unwanted fields
  3. compare adjacent entries to eliminate duplicates