CS 564: Database Management Systems

Lecture 21: Relational Operators I

Xiangyao Yu

3/11/2024

Midterm Exam Logistics

Mid-term exam

- March 20th, Wednesday
- 2:30–3:45pm (in-class)
- Please arrive 5 min early
- Paper-based, closed-book
- Cheat sheet allowed, US letter size (8.5 × 11 inches), double-sided

Previous years' exam questions have been released

Final exam is cumulative

- You will be tested on everything you have learned so far
- More focus on the second half of class

Module B3 Query Processing

Relational operators I

Relational operators II

Query optimization I

Query optimization II

Column Store

Outline

Selection

- Access path
- Clustered vs. unclustered
- Index matching

Projection

- Sort-based
- Hash-based

Logical vs. Physical Operators

Logical operators

- *what* they do
- e.g., selection, project, join, grouping, union

Physical operators

- *how* they are implemented
- e.g., nested-loop join, sort-merge join, hash join, index join

Outline

Selection

- Access path
- Clustered vs. unclustered
- Index matching

Projection

- Sort-based
- Hash-based

Selection Operator

Access path: way to retrieve tuples from a table

File Scan:

- Scan the entire file
- I/O cost: number of pages **N** in the file

Index Scan:

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

Selectivity: fraction of tuples that satisfy the selection condition

Outline

Selection

- **Access path**
- Clustered vs. unclustered
- Index matching

Projection

- Sort-based
- Hash-based

Access Path Example

```sql
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid
  AND  R.bid = 103;
```

Assuming we have a B+ tree index on R.bid

How to access relation R?

Access Path Example

```
SELECT S.sname
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid
  AND  R.bid = 103;
```

Assuming we have a B+ tree index on R.bid

Two ways to access relation R

- Access path 1: scan all pages in relation R
- Access path 2: use the index to access records satisfying R.bid = 103

Index Scan Cost

Hash index: O(1) IOs

- But we can only use it with equality predicates!

Index Scan Cost

Hash index: O(1) IOs

– But we can only use it with equality predicates!

B+ tree index: (height + **X**) IOs

– Clustered: **X** = (#selected records) / (#records per page)

– Unclustered: **X** = # selected tuples in the worst case

Outline

Selection

- Access path
- **Clustered vs. unclustered**
- Index matching

Projection

- Sort-based
- Hash-based

Clustered vs. Unclustered B+ Tree

A relation with 1,000,000 records

100 records on a page

height of B+ tree = 3

IO cost	0.1% selectivity	1% selectivity	10% selectivity
Clustered			
Unclustered			

Clustered vs. Unclustered B+ Tree

A relation with 1,000,000 records

100 records on a page

height of B+ tree = 3

IO cost	0.1% selectivity	1% selectivity	10% selectivity
Clustered	3+10	3+100	3+1,000
Unclustered	3+1,000	3+10,000	3+100,000

Outline

Selection

- Access path
- Clustered vs. unclustered
- **Index matching**

Projection

- Sort-based
- Hash-based

General Form of Selection

Simple selection

- attribute **op** constant
- attribute1 **op** attribute2
- **op**: $<, \leq, =, \neq, \geq, >$

More complex selection examples

- bid=5 $\wedge$ sid=3
- (day < ‘8/9/21’ $\wedge$ rname = ‘Joe’) $\vee$ bid=5 $\vee$ sid=3

Conjunctive Normal Form (CNF)

CNF: A collection of conjuncts that are connected through **and** (∧)

Each conjunct consists of one or more terms connected by **or** (∨)

- Conjuncts that contain ∨ are said to contain disjunction

Conjunctive Normal Form (CNF)

CNF: A collection of conjuncts that are connected through **and** (∧)

Each conjunct consists of one or more terms connected by **or** (∨)

- Conjuncts that contain ∨ are said to contain disjunction

Every selection condition can be expressed in CNF

For example:

- *(day < ‘8/9/21’ ∧ rname = ‘Joe’) ∨ bid=5 ∨ sid=3* **is equivalent to**
- *(day < ‘8/9/21’ ∨ bid=5 ∨ sid=3) ∧ (rname = ‘Joe’ ∨ bid=5 ∨ sid=3)*

Index Matching

An index matches a selection condition if the index can be used to retrieve tuples that satisfy the condition

Examples:

- Relation **R**(A,B,C,D)
- Hash index on composite key (A,B)

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

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

```
SELECT *
FROM   R
WHERE  A=5 AND B=5 AND C<4;
```

Index Matching

An index matches a selection condition if the index can be used to retrieve tuples that satisfy the condition

Examples:

- 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!

```
SELECT  *
FROM    R
WHERE   A=5 AND B=5 AND C<4;
```

matches the index!

Index Matching – Hash Index

A hash index matches a selection condition if there is an equality predicate for **each** attribute in the search key

Index Matching – Hash Index

A hash index matches a selection condition if there is an equality predicate for **each** attribute in the search key

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

Index Matching – B+ Tree

A B+ tree index matches a selection condition if there is a term of the form attribute **op** value for each attribute in a prefix of search key

Index Matching – B+ Tree

A B+ tree index matches a selection condition if there is a term of the form attribute **op** value for each attribute in a prefix of search key

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

Multiple Matches

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`

Multiple Matches

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`

We have multiple possible access paths

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

   (sorting the page number in rid can reduce #IOs)

Selection with Disjunction

If a subset of terms in a disjunction has matching indexes

- `A=7 **OR** B>5`
- Hash index on (A)
- We must perform a file scan. The index cannot be used.

If every term in a disjunction has a matching index

- `A=7 **OR** B>5`
- Hash index on (A), B+ tree on (B)
- We can retrieve candidate tuples using the indexes and take the union of rid

Selection with Disjunction – Exercise

What are the access paths for the following example?

- Hash index on (A), B+ tree on (B)
- `(A=7` **`OR`** `C>5)` **`AND`** `B > 5`

Selection with Disjunction – Exercise

What are the access paths for the following example?

- Hash index on (A), B+ tree on (B)
- `(A=7 OR C>5) AND B > 5`

Access path 1: file scan

Access path 2: index scan using B+ tree, then filter according to the first conjunct

The hash index cannot be used in this example

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

Selection: `A=3`

Hash index on (A)

If know the number of unique search keys in the index (**#keys**)

The selectivity can be approximated by: **1/#keys**

- Assuming that the values are distributed *uniformly* across the tuples

Estimating Selectivity

Selection: `A>10 AND A<60`

B+ tree index on (A)

If we have a range condition, a DBMS typically assumes 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

$$\sim \frac{60-10}{100-0} = 50\%$$

Exercise

1,000,000 records, 100 records per page

Selection: `A=3` **`AND`** `B=5` **`AND`** `C>10`

Hash index on (A)   #keys = 100,000

Clustered B+ tree index on (C)   max = 15, min=0

What are the access paths and the associated IO cost?

- Access path 1: Use hash index and then filter B=5 and C>10.

Exercise

1,000,000 records, 100 records per page

Selection: `A=3` **`AND`** `B=5` **`AND`** `C>10`

Hash index on (A)   #keys = 100,000

Clustered B+ tree index on (C)   max = 15, min=0

What are the access paths and the associated IO cost?

- Access path 1: Use hash index and then filter B=5 and C>10.
  Cost: 1000k records / #keys = 10 IOs
- Access path 2: Use B+ tree index and then filter A=3 and B=5

Exercise

1,000,000 records, 100 records per page

Selection: `A=3 AND B=5 AND C>10`

Hash index on (A)   #keys = 100,000

Clustered B+ tree index on (C)   max = 15, min=0

What are the access paths and the associated IO cost?

- Access path 1: Use hash index and then filter B=5 and C>10.
  Cost: 1000k records / #keys = 10 IOs
- Access path 2: Use B+ tree index and then filter A=3 and B=5
  Cost: (15-10)/15 * (1,000,000 / 100) = 3333 IOs

What if the B+ tree on (C) is unclustered?

Outline

Selection

- Access path
- Clustered vs. unclustered
- Index matching

Projection

- Sort-based
- Hash-based

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!)

Two approaches to eliminate duplicates

- Sort based
- Hash based

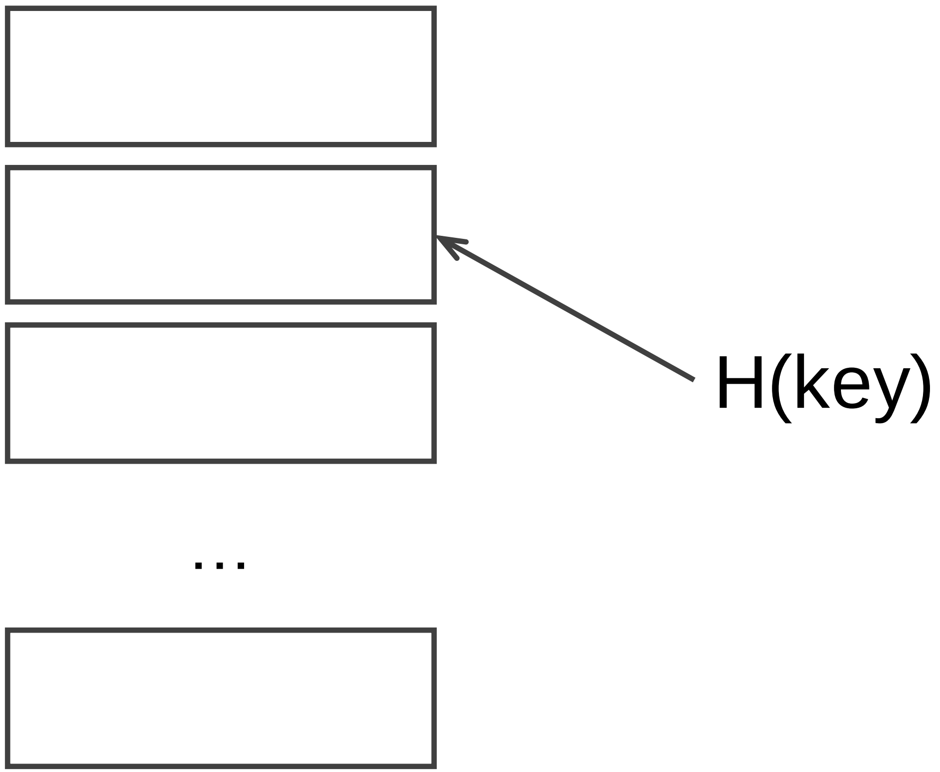
Project – Sort-Based

Step 1: Scan the relation and project out the attributes

Step 2: Sort the resulting set of tuples using projected attributes

Step 3: Scan the sorted set by comparing only adjacent tuples and discard duplicates

Project – Hash-Based



Duplicate rows are hashed to the same bucket

Remove the row when a duplicate is detected

Project – Hash-Based (Larger than Memory)

Phase 1: partitioning

- Project out attributes and split the input into B-1 partitions using a hash function *h*

Phase 2: duplicate elimination

- Read each partition into memory and use an in-memory hash table (with a *different* hash function) to remove duplicates
- If the partition does not fit into memory, run phase 1 recursively to partition into smaller pieces

Sort-Based vs. Hash-Based

Benefits of sort-based approach

- Better handling of skew (one hash partition is bigger than others)
- The result is sorted

Both algorithms may need multiple passes when data is bigger than memory

- Sort-based: 2-pass external merge sort
- Hash-based: partitioning + duplicate elimination

Project – Index-Only Scan

Index-only scan

- Projection attributes are a subset of index attributes
- Apply projection algorithm only to index entries, no need to access records

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

Summary

Selection

- Access path
- Clustered vs. unclustered
- Index matching

Projection

- Sort-based
- Hash-based