

# INFO20003 Database Systems

---

Week 7

# File Organisations Revision

- Heap File
- Sorted File
- Index:
  - Hash Index
  - B-Tree Index

# Q1) Index Selection

## 1. Question about the effect of index on selection:

Consider a relation  $R(a,b,c,d,e)$  containing 5,000,000 records, where each data page of the relation holds 10 records.  $R$  is organized as a sorted file with secondary indexes. Assume that  $R.a$  is a candidate key for  $R$ , with values lying in the range 0 to 4,999,999, and that  $R$  is stored in  $R.a$  order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file of  $R$  directly.
- Use a B+ tree index on attribute  $R.a$ .
- Use a hash index on attribute  $R.a$ .

### Queries:

- a.  $\sigma_{a < 50000}(R)$       sorted file over  $R$
- b.  $\sigma_{a = 50000}(R)$       hash index
- c.  $\sigma_{a > 50000 \wedge a < 50010}(R)$       B+ tree index

# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key

# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key
- Index on <a, b, c> will match predicates on <a, b, c>, <a, b>, <a>

Sorted primarily on a

a	b	c
1	3	1
2	1	1
2	3	1
3	2	1
3	6	1
3	6	4
6	3	1

# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key
- Index on <a, b, c> will match predicates on <a, b, c>, <a, b>, <a>

Break even by b

a	b	c
1	3	1
2	1	1
2	3	1
3	2	1
3	6	1
3	6	4
6	3	1

# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key
- Index on <a, b, c> will match predicates on <a, b, c>, <a, b>, <a>

a	b	c
1	3	1
2	1	1
2	3	1
3	2	1
3	6	1
3	6	4
6	3	1

Break even by c  
when a and b  
are the same

# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key
- Index on  $\langle a, b, c \rangle$  will match predicates on  $\langle a, b, c \rangle$ ,  $\langle a, b \rangle$ ,  $\langle a \rangle$ 
  - E.g. primary conjuncts can be  $(a=3 \wedge b>5)$
  - cannot be used to answer  $b=3$

a	b	c
1	3	1
2	1	1
2	3	1
3	2	1
3	6	1
3	6	4
6	3	1

$a=3$  and  $b>5$



# Primary Conjunct

- Predicate = selection condition

```
SELECT attribute list  
FROM relation list  
WHERE predicate1 AND ... AND predicate_k
```

- An index matches (a combination of) predicates that involve only attributes in a **prefix** of the search key
- Index on  $\langle a, b, c \rangle$  will match predicates on  $\langle a, b, c \rangle$ ,  $\langle a, b \rangle$ ,  $\langle a \rangle$ 
  - E.g. primary conjuncts can be  $(a=3 \wedge b>5)$
  - cannot be used to answer  $b=3$

a	b	c
1	3	1
2	1	1
2	3	1
3	2	1
3	6	1
3	6	4
6	3	1

$b=3$

# Q2) Matching Index

## 2. Matching index

Consider the following schema for the Sailors relation:

Sailors (sid INT, sname VARCHAR(50), rating INT, age DOUBLE)

For each of the following indexes, list whether the index matches the given selection conditions and briefly explain why.

- A B+ tree index on the search key (Sailors.sid)
  - a.  $\sigma_{\text{Sailors.sid} < 50,000}(\text{Sailors})$
  - b.  $\sigma_{\text{Sailors.sid} = 50,000}(\text{Sailors})$
- A hash index on the search key (Sailors.sid)
  - c.  $\sigma_{\text{Sailors.sid} < 50,000}(\text{Sailors})$
  - d.  $\sigma_{\text{Sailors.sid} = 50,000}(\text{Sailors})$
- A B+ tree index on the search key (Sailors.rating, Sailors.age)
  - e.  $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}(\text{Sailors})$
  - f.  $\sigma_{\text{Sailors.rating} = 8}(\text{Sailors})$
  - g.  $\sigma_{\text{Sailors.age} = 21}(\text{Sailors})$

- A B+ tree index on the search key (Sailors.sid)
  - a.  $\sigma_{\text{Sailors.sid} < 50,000} (\text{Sailors})$
  - b.  $\sigma_{\text{Sailors.sid} = 50,000} (\text{Sailors})$

- a) Match, primary conjuncts are: *Sailors.sid < 50,000*
- b) Match, primary conjuncts are: *Sailors.sid = 50,000*

- A hash index on the search key (Sailors.sid)
  - c.  $\sigma_{\text{Sailors.sid} < 50,000}(\text{Sailors})$
  - d.  $\sigma_{\text{Sailors.sid} = 50,000}(\text{Sailors})$

- c) No match, range queries cannot be applied to a hash index.
- d) Match, primary conjuncts are: *Sailors.sid = 50,000*

- A B+ tree index on the search key (Sailors.rating, Sailors.age)

e.  $\sigma_{\text{Sailors.rating} < 8 \wedge \text{Sailors.age} = 21}(\text{Sailors})$

f.  $\sigma_{\text{Sailors.rating} = 8}(\text{Sailors})$

g.  $\sigma_{\text{Sailors.age} = 21}(\text{Sailors})$

e) Match, primary conjuncts are *Sailors.rating* < 8 and *Sailors.rating* < 8  $\wedge$  *Sailors.age* = 21

f) Match, primary conjuncts are: *Sailors.rating* = 8

g) No match. The index on (Sailors.rating, Sailors.age) is primarily sorted on Sailors.rating, so the entire relation would need to be searched to find those sailors with a particular Sailors.age value.

# Q3) Cost of Joins

## 3. Question about the cost analysis of different joins:

Consider the join  $R \bowtie_{R.a = S.b} S$ , given the following information about the relations to be joined:

- Relation R contains 10,000 tuples and has 10 tuples/page.
- Relation S contains 2,000 tuples and also has 10 tuples/page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

The cost metric is the number of page I/Os unless otherwise noted and the cost of writing out the result should be uniformly ignored. **Use S as the outer relation**

- What is the cost of joining R and S using the **Page-oriented Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?
- What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?
- What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.
- What is the cost of joining R and S using the **Hash Join** algorithm?
- What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly. **Assuming infinite B**

## 5. Joins (between relations R and S, R = outer, S = inner) Cost

### a. NLJ

#### i. Tuple-oriented NLJ

$$\text{Cost} = \text{NPages}(\text{R}) + \text{NTuples}(\text{R}) * \text{NPages}(\text{S})$$

#### ii. Page-oriented NLJ

$$\text{Cost} = \text{NPages}(\text{R}) + \text{NPages}(\text{R}) * \text{NPages}(\text{S})$$

#### iii. Block-oriented NJL (for block\_size B)

$$\text{Cost} = \text{NPages}(\text{R}) + \text{ceil}(\text{NPages}(\text{R}) / (B - 2)) * \text{NPages}(\text{S})$$

### b. Hash Join

$$\text{Cost} = 3 * (\text{NPages}(\text{R}) + \text{NPages}(\text{S}))$$

 B = # buffer pages

### c. Sort-Merge Join

$$\begin{aligned} \text{Cost}_{\text{SMJ}} = & \text{NPages}(\text{R}) + \text{NPages}(\text{S}) + \\ & 2 * \text{NPages}(\text{R}) * \text{num\_passes}(\text{R}) + \\ & 2 * \text{NPages}(\text{S}) * \text{num\_passes}(\text{S}) \end{aligned}$$



Consider the join  $R \bowtie_{R.a = S.b} S$ , given the following information about the relations to be joined:

- Relation R contains 10,000 tuples and has 10 tuples/page.
- Relation S contains 2,000 tuples and also has 10 tuples/page.
- Attribute b of relation S is the primary key for S.
- Both relations are stored as simple heap files.
- Neither relation has any indexes built on it.
- 52 buffer pages are available.

R:

NT = 10,000

M = NP = 1,000

S:

NT = 2,000

N = NP = 200

Nkeys(b) = 2,000

B = 52

# Q3a)

$R.a = S.b$

R:

NT = 10,000

M = NP = 1,000

S:

NT = 2,000

N = NP = 200

Nkeys(b) = 2,000

B = 52

- a. What is the cost of joining R and S using the **Page-oriented Nested Loops** algorithm? What is the minimum number of buffer pages (in memory) required in order for this cost to remain unchanged?

$$\begin{aligned}\text{Total cost} &= (\# \text{ of pages in outer}) + (\# \text{ of pages in outer} \times \# \text{ of pages in inner}) \\ &= N + (N \times M) = 200 + (200 \times 1000) = 200,200\end{aligned}$$

3 buffer pages required: 1 input buffer to page through each relation; 1 output buffer to store output



## Q3b)

R.a = S.b

R:

NT = 10,000

M = NP = 1,000

S:

NT = 2,000

N = NP = 200

Nkeys(b) = 2,000

B = 52

- b. What is the cost of joining R and S using the **Block Nested Loops** algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

$$\# \text{ of blocks} = \text{ceil}\left(\frac{\# \text{ of pages in outer}}{B - 2}\right) = \text{ceil}\left(\frac{200}{50}\right) = 4$$

$$\begin{aligned} \text{Total cost} &= (\# \text{ of pages in outer}) + (\# \text{ of blocks} \times \# \text{ of pages in inner}) \\ &= 200 + (4 \times 1000) = 4200 \end{aligned}$$

If we have fewer buffers available, the cost will increase as the # of blocks will vary. The minimum number of buffer pages is 52 for this cost.

# Q3c)

$$R.a = S.b$$

R:

$$NT = 10,000$$

$$M = NP = 1,000$$

S:

$$NT = 2,000$$

$$N = NP = 200$$

$$Nkeys(b) = 2,000$$

$$B = 52$$

- c. What is the cost of joining R and S using the **Sort-Merge Join** algorithm? Assume that the external merge sort process can be completed in 2 passes.

$$\begin{aligned}\text{Cost of sorting R} &= 2 \times \# \text{ of passes} \times \# \text{ of pages of R} \\ &= 2 \times 2 \times 1000 = 4000\end{aligned}$$

$$\text{Cost of sorting S} = 2 \times 2 \times 200 = 800$$

$$\begin{aligned}\text{Cost of merging R and S} &= \# \text{ of pages read of R} + \# \text{ of pages read of S} \\ &= 1000 + 200 = 1200\end{aligned}$$

$$\begin{aligned}\text{Total cost} &= \text{Cost of sorting R} + \text{Cost of sorting S} + \text{Cost of merging R and S} \\ &= 4000 + 800 + 1200 = 6000\end{aligned}$$

## Q3d)

$$R.a = S.b$$

R:

$$NT = 10,000$$

$$M = NP = 1,000$$

S:

$$NT = 2,000$$

$$N = NP = 200$$

$$Nkeys(b) = 2,000$$

$$B = 52$$

d. What is the cost of joining R and S using the **Hash Join** algorithm?

In hash join, each relation is partitioned and then the join is performed by “matching” elements from corresponding partitions.

$$\begin{aligned} \text{Total cost} &= 3(M + N) \\ &= 3(1000 + 200) = 3600 \end{aligned}$$

# Q3e)

$R.a = S.b$

R:

NT = 10,000

M = NP = 1,000

S:

NT = 2,000

N = NP = 200

Nkeys(b) = 2,000

B = 52

e. What would the lowest possible I/O cost be for joining R and S using any join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

- Block-oriented nested loop
- Store the entire smaller relation in memory to have 1 block
- The larger relation will be read once
- Total cost =  $200 + 1000 = 1200$  I/O
- Minimum buffer page required =  $N_{\text{pages}}(\text{smaller relation}) + 2 = 202$