香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

CSC3170
Tutorial 9

School of Data Science
The Chinese University of Hong Kong, Shenzhen

Outline

- Join Algorithm
  - Block Nested Loop Join
  - Index Nested Loop Join
  - Sort-Merge Join
  - Hash Join
- Exercise

Block Nested Loop Join

- If we have B buffers available:
  - Use B-2 buffers for each block of the outer table.
  - Use one buffer for the inner table, one buffer for output.

```
foreach B - 2 pages p_R ∈ R:
  foreach page p_S ∈ S:
    foreach tuple r ∈ B - 2 pages:
      foreach tuple s ∈ p_s:
        if r and s match then emit
```

- **Cost:** $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

M pages
m tuples

S(id,value,cdate)

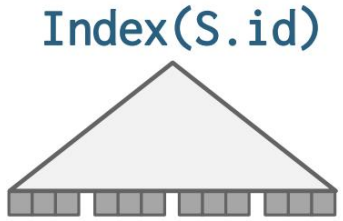
id	value	cdate
100	2222	11/4/2024
500	7777	11/4/2024
400	6666	11/4/2024
100	9999	11/4/2024
200	8888	11/4/2024

N pages
n tuples

Index Nested Loop Join

```
foreach tuple r ∈ R:
  foreach tuple s ∈ Index(r_i = s_j):
    if r and s match then emit
```

require an existing index of inner table



R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

M pages
m tuples

S(id,value,cdate)

id	value	cdate
100	2222	11/4/2024
500	7777	11/4/2024
400	6666	11/4/2024
100	9999	11/4/2024
200	8888	11/4/2024

N pages
n tuples

- Assume the cost of each index probe is some constant C per tuple.
- **Cost: $M + (m \cdot C)$**

Sort–Merge Join

- **Phase #1: Sort**
  - Sort both tables on the join key(s).
  - You can use any appropriate sort algorithm
  - These phases are distinct from the sort/merge phases of an external merge sort, from the previous class
- **Phase #2: Merge**
  - Step through the two sorted tables with cursors and emit matching tuples.
  - May need to backtrack depending on the join type.

Sort–Merge Join

```
sort R,S on join keys
cursor_R ← R_sorted, cursor_S ← S_sorted
while cursor_R and cursor_S:
  if cursor_R > cursor_S:
    increment cursor_S
  if cursor_R < cursor_S:
    increment cursor_R
    backtrack cursor_s (if necessary)
  elif cursor_R and cursor_S match:
    emit
    increment cursor_S
```

```
do{
    if (!mark){
        while(r<s){advance r}
        while(s<r){advance s}
        mark = s
    }
    if (r==s){
        emit
        advance s
    }
    else{
        reset s to mark
        advance r
        mark = NULL
    }
}
```

Sort–Merge Join

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
200	GZA
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	11/4/2024
500	7777	11/4/2024
400	6666	11/4/2024
100	9999	11/4/2024
200	8888	11/4/2024

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Sort–Merge Join

- **Sort Cost (R):** $2M \cdot (1 + \lceil \log_{B-1} \lceil M / B \rceil \rceil)$
- **Sort Cost (S):** $2N \cdot (1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil)$
- **Merge Cost:** $(M + N)$

- **Total Cost: Sort + Merge**

Worst Case: join attribute of all the tuples in both relations contains the same value
Merge Cost: $M \times N$

Hash Join

- **Phase #1: Build**
  - Scan the outer relation and populate a hash table using the hash function h_1 on the join attributes.
  - We can use any hash table that we discussed before but in practice linear probing works the best.

- **Phase #2: Probe**
  - Scan the inner relation and use h_1 on each tuple to jump to a location in the hash table and find a matching tuple.

Hash Join–Grace Hash Join

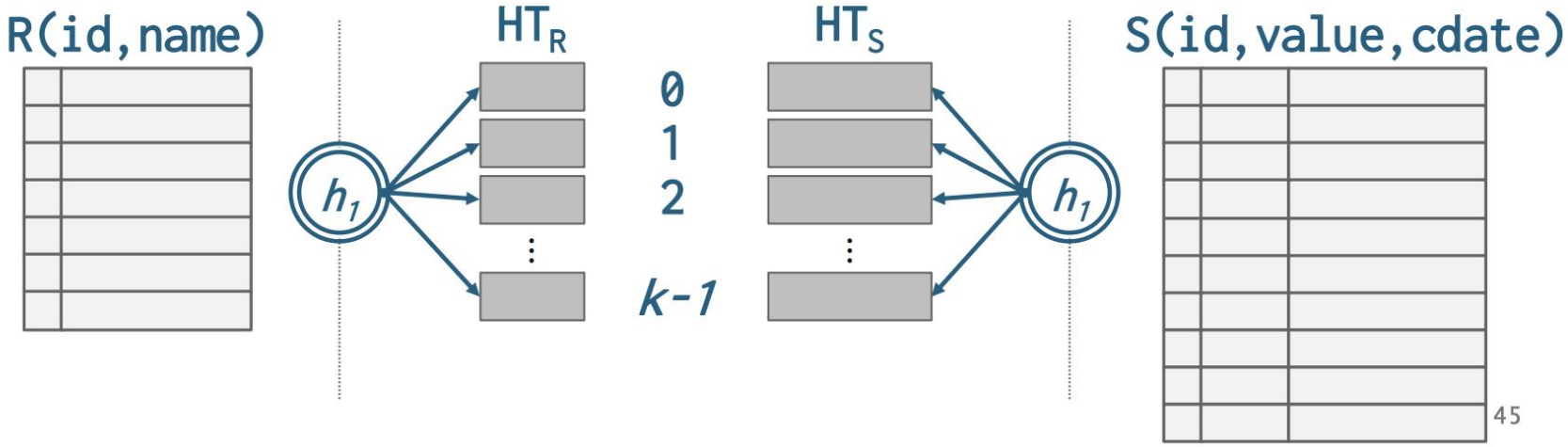
Deal the problem that the tables can not fit in the memory

- **Partition Phase:** Hash both tables on the join attribute into partitions.
- **Probe Phase:** Compares tuples in corresponding partitions for each table.

Hash Join–Grace Hash Join

Partition Phase

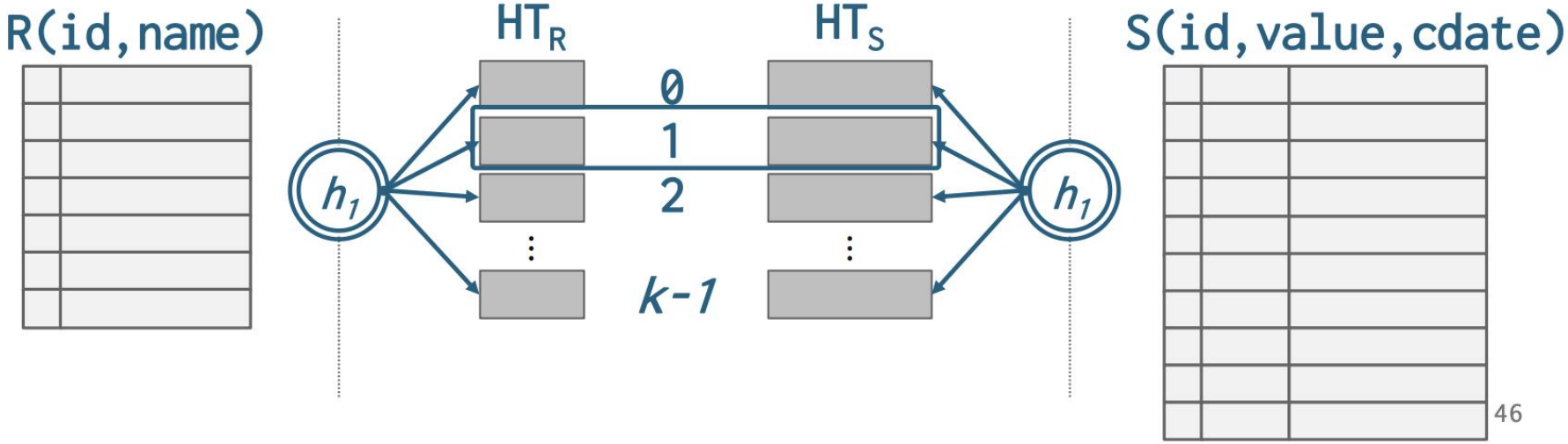
- Hash R into *k* buckets.
- Hash S into *k* buckets with same hash function.
- Write buckets to disk when they get full.



Hash Join–Grace Hash Join

Probe Phase

- Read corresponding partitions into memory one pair at a time, hash join their contents.



Hash Join–Grace Hash Join

- If we do not need recursive partitioning:
  - Cost: $3(M + N)$
- **Partition phase:**
  - Read+write both tables
  - $2(M+N)$ I/Os
- **Probe phase:**
  - Read both tables (in total, one partition at a time)
  - M+N I/Os

The algorithm is recursively applied when one or more of the partitions still does not fit into the available memory.

Exercise

Consider relations X and Y to be joined on the common attribute. There are

- B = 500 pages in the buffer
- Table X spans M = 2000 pages with 40 tuples per page
- Table Y spans N = 600 pages with 450 tuples per page

1. What is the I/O cost of a block nested loop join with Y as the outer relation and X as the inner relation?

Exercise

Consider relations X and Y to be joined on the common attribute. There are

- B = 500 pages in the buffer
- Table X spans M = 2000 pages with 40 tuples per page
- Table Y spans N = 600 pages with 450 tuples per page

1. What is the I/O cost of a block nested loop join with Y as the outer relation and X as the inner relation?

$$N + \lceil N/(B-2) \rceil \cdot M = 600 + 2 \times 2000 = 4600$$

Exercise

Consider relations X and Y to be joined on the common attribute. There are

- B = 500 pages in the buffer
- Table X spans M = 2000 pages with 40 tuples per page
- Table Y spans N = 600 pages with 450 tuples per page

2. What is the I/O cost of a Grace hash join with X as the outer relation and Y as the inner relation?

Exercise

Consider relations X and Y to be joined on the common attribute. There are

- B = 500 pages in the buffer
- Table X spans M = 2000 pages with 40 tuples per page
- Table Y spans N = 600 pages with 450 tuples per page

2. What is the I/O cost of a Grace hash join with X as the outer relation and Y as the inner relation?

$$3(M + N) = 3 \times 2600 = 7800$$

Q&A