Joins

CS165 - Section 10
The Join Operation

Example (SQL):

SELECT *  
FROM Employees, Department  
WHERE Employees.depid = Department.depid

Relational Algebra

Condition Join: \( R \bowtie_{(R\.sid<S\.sid)} S \)  
Equijoin: \( R \bowtie_{(R\.sid=S\.sid)} S \)  
Natural Join: \( R \bowtie S \)
Implementing Joins

Implementing join as cross-product followed by selection and projection is expensive, because cross-product result is normally much larger than join result

**Assumption For Following Slides**

Join Type: Equijoin between $R$ and $S$  \[ (R \bowtie_{(R_i=S_j)} S) \]

Size: $M$ pages in $R$; $p_R$ tuples per page in $R$
$N$ pages in $S$; $p_S$ tuples per page in $S$
$|S| > |R|$
Nested Loop Join

**Idea:** Try out all combinations of tuples, collect tuples with matching keys

\[
\begin{align*}
\text{foreach tuple } r & \in R: \\
& \quad \text{for each tuple } s \in S: \\
& \quad \quad \text{if } r_i = s_j \text{ then add } \langle r, s \rangle \text{ to result}
\end{align*}
\]
Nested Loop Join - I/O Cost

Cost in number of page I/O operations:

\[
\text{Cost} = \text{Cost}_{\text{Scan } R} + p_R \cdot M \cdot \text{Cost}_{\text{Scan } S}
\]

\[
\text{Cost}_{\text{Scan } R} = M
\]

\[
\text{Cost}_{\text{Scan } S} = N
\]

\[\Rightarrow \quad \text{Cost} = M + p_R \cdot M \cdot N\]
Block Nested Loop Join

**Problem:** If smaller relation R + 2 pages (input buffer for S + output buffer) does not fit into memory → high I/O cost

**Idea:** Similar to NLJ, but break outer relation up into parts

```plaintext
foreach block of B-2* pages of R:
  for each page of S:
    foreach tuple r ∈ R-block:
      for each tuple s ∈ S-page:
        if r_i == s_i then add ⟨r,s⟩ to result
```

* Assumes buffer pool size of B pages
Block Nested Loop Join - I/O Cost

Cost in number of page I/O operations:

\[
\text{Cost} = \text{Cost}_{\text{Scan } R} + \left\lceil \frac{M}{B-2} \right\rceil \cdot \text{Cost}_{\text{Scan } S}
\]

\[
\text{Cost}_{\text{Scan } R} = M
\]

\[
\text{Cost}_{\text{Scan } S} = N
\]

\[\Rightarrow \quad \text{Cost} = M + N \cdot \left\lceil \frac{M}{B-2} \right\rceil\]
Index Nested Loop Join

Idea: If index on one of the relations on the join attribute(s), make that relation the inner relation → do not compute cross-product $R \times S$

foreach tuple $r \in R$:
    foreach tuple $s \in S$ where $r_i == s_j$:
        add $\langle r, s \rangle$ to result
Index Nested Loop Join - I/O Cost

Cost in number of page I/O operations depends on kind of index and number of matching tuples!

\[
\text{Cost} = \text{Cost}_{\text{Scan } R} + p_R \cdot M \cdot \text{Cost}_{\text{Index Lookup}}
\]

\[
\text{Cost}_{\text{Scan } R} = M
\]

\[
\text{Cost}_{\text{Index Lookup}} = ? \quad \text{(e.g., hash table = 1-2 I/Os)}
\]

\[
\Rightarrow \quad \text{Cost} = M + p_R \cdot M \cdot \text{Cost}_{\text{Index Lookup}}
\]
Sort-Merge Join

**Idea:** Sort both relations on join attribute, then find matching tuples by merging the two relations

**Steps:**
- **Sort:** If not already sorted sort $R$ & $S$ on join attribute
- **Merging:**
  1. Scan through $R$ and $S$ in parallel
     - (advance $S$ if current $S$-tuple $<$ current $R$-tuple;
       advance $R$ if current $R$-tuple $<$ current $S$-tuple)
  2. If we find tuples $Tr$ and $Ts$ such that $Tr_i = Ts_j \rightarrow$ Add $\langle r, s \rangle$ to result

**Note:** If the key $s_j$ is not unique (i.e., there will be duplicates) in the inner relation, we need to remember the start of every $s_j$ partition!
Sort-Merge Join - Example

Step 1: Sort

\[
\begin{array}{c}
2 \\
4 \\
1 \\
2 \\
\end{array}
\]

\[
\begin{array}{c}
2 \\
3 \\
2 \\
1 \\
3 \\
2 \\
4 \\
\end{array}
\]
Sort-Merge Join - Example

Step 2: Merge
Sort-Merge Join - Example

Step 2: Merge (Select first key in $R$)
Sort-Merge Join - Example

Step 2: Merge (Select first key in S → match!)

\[ <R_0, S_0> \]
Sort-Merge Join - Example

Step 2: Merge (Advance main cursor in S)

\[<R_0, S_0>\]
Sort-Merge Join - Example

Step 2: Merge (Remember start of S; Advance cursor in S)
Sort-Merge Join - Example

Step 2: Merge (Advance main cursor in S → match!)
Sort-Merge Join - Example

Step 2: Merge (Advance main cursor in S → match!)

R

1
2
2
4

S

1
2
2
3
3
4

<R0,S0>
<R1,S1>
<R1,S2>
<R1,S3>
Sort-Merge Join - Example

Step 2: Merge (Reset S-cursor, Advance R-cursor → match!)

\[\text{R} \quad \begin{array}{c|c|c|c|c} 1 & 2 & 2 & 4 \\ \hline \end{array} \quad \text{S} \quad \begin{array}{c} 1 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 4 \\ \hline \end{array} \quad \text{match!} \]

\[\text{<R0,S0>, <R1,S1>, <R1,S2>, <R1,S3>, <R2,S1>} \]
Sort-Merge Join - Example

Step 2: Merge (Advance main cursor in S → match!)

\[
\begin{array}{c|c|c}
R & S \\
\hline 1 & 1 & \langle R0, S0 \rangle \\
2 & 2 & \langle R1, S1 \rangle \\
2 & 2 & \langle R1, S2 \rangle \\
4 & 2 & \langle R1, S3 \rangle \\
\end{array}
\]
And so on...
Sort-Merge Join - Code

// Sort input if necessary
if R not sorted on attribute i, sort it;
if S not sorted on attribute j, sort it

// Initialize variables (Tr/Ts = Main cursors for R and S, Ps = partition start in S)
Tr = first tuple in R; Ts = first tuple in S; Ps = first tuple in S

while Tr ≠ eof and Ps ≠ eof:
    // Continue scan of R
    while Tr_i < Ps_j:
        Tr = next tuple in R after Tr
    // Continue scan of S
    while Tr_i > Ps_j:
        Ps = next tuple in S after Ps
...

Sort-Merge Join - Code (cont’d)

while $Tr \neq \text{eof}$ and $Ps \neq \text{eof}$: (cont’d)

...  
$Ts = Ps$  
while $Tr_i = Ps_j$:  
    $Ts = Ps$  
    while $Ts_j = Tr_i$:  
        add $\langle r, s \rangle$ to result  
        $Ts =$ next tuple in $S$ after $Ts$  
        $Tr =$ next tuple in $R$ after $Tr$
$Ps = Ts$  

// Needed in case $Tr_i \neq Ps_j$
// Process $R$ partition
// Reset $S$ partition
// Compare join attribute
// Set start of $S$-scan to  
// end of $S$ partition
Sort-Merge Join - Cost

\[ \text{Cost} = \text{Cost}_{\text{Sort } R} + \text{Cost}_{\text{Sort } S} + \text{Cost}_{\text{Merge}} \]

\[ \text{Cost}_{\text{Sort } R} = O( M \cdot \log M ) \]

\[ \text{Cost}_{\text{Sort } S} = O( N \cdot \log N ) \]

\[ \text{Cost}_{\text{Merge}} = O( M \cdot N ) \text{ (Worst case); Practice: M+N}^* \]

* In reality S partitions rarely need to be scanned multiple times
  (Worst case: all tuples in R and S have same key)
Hash Join

**Idea:** Hash both relations on the join attribute to create partitions. Only join tuples in the same partition.

**Steps:**
- Partitioning Phase: Partition $R$ and $S$
- Probing Phase: Compare tuples in the same partition

**Note:** One of the partitioning phases can be streamed (i.e., partition/build hash table for smaller relation, scan larger one).
Hash Join

// Build Hash table
foreach tuple \( r \in R \):
    read \( r \) and insert into hash table using \( h(r_i) \)

// Probe hash table
for each tuple \( s \in S \):
    read \( s \) and probe hash table using \( h(s_j) \)
    for matching \( R \) tuples \( r \):
        add \( \langle r, s \rangle \) to result
Grace (*Partitioned*) Hash Join

// Partition \( R \) into \( k \) partitions
foreach tuple \( r \in R \):
  read \( r \) and add it to buffer page \( h(r_i) \)

// Partition \( S \) into \( k \) partitions
foreach tuple \( s \in S \):
  read \( s \) and add it to buffer page \( h(s_j) \)
for \( l = 1..k \):
  \( ht = \emptyset \) // Clear hashtable
  // Build Hash table for \( R_l \)
  foreach tuple \( r \in R_l \):
    read \( r \) and insert into hash table using \( h'(r_i) \)

  // Probe hash table for partition
  for each tuple \( s \in S \):
    read \( s \) and probe hash table using \( h'(s_j) \)
    for matching \( R \) tuples \( r \): add \( \langle r, s \rangle \) to result
Grace (*Partitioned*) Hash Join - Cost

Cost in number of page I/O operations depends on kind of hash table and number of collisions!

\[
\text{Cost} = \text{Cost}_{\text{Partition } R} + \text{Cost}_{\text{Partition } S} + \text{Cost}_{\text{Probing}}
\]

\[
\text{Cost}_{\text{Partition } R} = 2 \cdot M \quad \text{(Read input/Write output)}
\]

\[
\text{Cost}_{\text{Partition } S} = 2 \cdot N \quad \text{(Read input/Write output)}
\]

\[
\text{Cost}_{\text{Probing}} = M + N \quad \text{(Assuming no partition overflows)}
\]

\[\Rightarrow \quad \text{Cost} = 3 \cdot (M + N)\]