• Iteration Join (conceptual)
  For each $r \in R$ do
    For each $s \in S$ do
      if $r.C = s.C$ then
        output $r,s$ pair
• Merge Join (conceptual)
  (1) if R and S not sorted, sort them
  (2) i ← 1; j ← 1;
      While (i ≤ |R|) ∧ (j ≤ |S|) do
          if R[i].C = S[j].C then
              outputTuples
          else if R[i].C > S[j].C then
              j ← j+1
          else if R[i].C < S[j].C then
              i ← i+1
Output-Tuples (for duplicates)

While \((R[i].C = S[j].C) \land (i \leq |R|)\) do

\[jj \leftarrow j;\]

While \((R[i].C=S[j].C)\land(jj\leq|S|)\) do

output \((R[i], S[j])\);

\[jj \leftarrow jj+1;\]

\[i \leftarrow i+1\]
Merge Join

• Both R, S sorted by C
• Hash Join (conceptual)
  – Hash function \( h \), range \( 1 \rightarrow k \)
  – Buckets for \( R \): \( G_1, \ldots, G_k \)
  – Buckets for \( S \): \( H_1, \ldots, H_k \)

**Algorithm**

1. Hash \( R \) tuples into \( G_1, \ldots, G_k \) buckets
2. Hash \( S \) tuples into \( H_1, \ldots, H_k \) buckets
3. For \( i = 1 \) to \( k \) do
   match tuples in \( G_i, H_i \) buckets
Hash Join

• Step (1)

• Step (2)
• Index Join (conceptual)

(1) If not, create an index for S.C
(2) For each $r \in R$ do

$X \leftarrow \text{index-lookup}(S.C, r.C)$

For each $s \in X$ do

output $(r,s)$
Index Join

Memory

\[ R \]

\[ \ldots \]

\[ R \]

\[ \ldots \]

\[ S \]
Cost Analysis

- Cost model
  - # of disk IOs
- Assumption
  - 10 tuples/block
  - |R| = 10,000, Br =?
  - |S| = 5,000, Bs =?
  - Memory: buffers for 102 blks
Nested-Loop Join

For each \( r \in R \) do
  For each \( s \in S \) do
    if \( r.C = s.C \) then
      output \( r,s \) pair

– How many disk IOs?
Block Nested-Loop Join

- Read 100 blocks of R into memory
- Read all of S (using 1 block) and output joined tuples (using 1 block)
- Repeat until done

- How many disk IOs?
General Rules for Nested-Loop Join

- Use block nested-loop join
- If no table fits into memory
  - Smaller table on the left
- In each iteration
  - Read as many blocks in the left (outer) table as possible
    \[
    \left\lfloor \frac{B_R}{M-2} \right\rfloor \cdot B_S + B_R
    \]
  - What if a table fits into memory?
Merge Join

- Both R, S ordered by C; relations contiguous
• **Merge Join Algorithm**

\[ i \leftarrow 1; j \leftarrow 1; \]

\[ \text{While } (i \leq |R|) \land (j \leq |S|) \text{ do} \]

\[ \text{if } R[i].C = S[j].C \text{ then} \]

\[ \text{outputTuples} \]

\[ \text{else if } R[i].C > S[j].C \text{ then} \]

\[ j \leftarrow j+1 \]

\[ \text{else if } R[i].C < S[j].C \text{ then} \]

\[ i \leftarrow i+1 \]

– How many disk IOs if R and S have been already sorted?
Merge Sort Algorithm

(i) For each 102 blk chunk of R:
- Read chunk
- Sort in memory
- Write to disk
Merge Sort Algorithm

(2) Merge

- Read one block from each 102-blk chunk to memory buffer
- While (some buffer not empty)
  a) Output the smallest tuple
  b) If a buffer is empty
     - Read the next block from the chunk if not EOF

Sorted chunks

Sorted file

Memory

How many disk IOs for sorting?
Merge Join Cost

• Very efficient if tables are sorted: \( B_R + B_S \)

• Cost for sorting R

\[
2B_R\left(\left\lceil \log_{M-1}\left(\frac{B_R}{M}\right) \right\rceil + 1 \right)
\]
Further Improvement for Merge Join?

Hint: do we really need fully sorted tables?

R1

Join?

R2

sorted buckets
Hash Join

• Step (1)

• Step (2)
Hash Join

(1) Bucketize R into 101 buckets
  • Read R, hash, write

R → 

101 buckets

How many?

• Similarly for S
Hash Join

(2) Join each bucket
   a) Read one R bucket; build memory hash table
   b) Read corresponding S bucket; hash probe

How many disk IOs?
Recursive Partitioning

Basic idea: Two tuples can join only if they agree on every hash function

(1) Bucketize R into 101 buckets
  • Read R, hash, write
Recursive Partitioning

(2) Bucketize each Gi using another hash function

- Read Gi, hash, write

G1 \rightarrow \begin{array}{c}
\vdots \\
(990 \text{ blks}) \\
\end{array} \rightarrow G_{11} \rightarrow \vdots \rightarrow G_{12} \rightarrow \vdots

101 \text{ buckets}

98 \text{ blks}

Fits in memory?

Repeat until each bucket fits in memory
Hash Join Cost

• No recursive partitioning
  \[ - 3 \left( B_R + B_S \right) \]

• Recursive partitioning
  \[ - 2(B_R + B_S) \left[ \log_{(M-1)} \left( \frac{B_R}{M - 2} \right) \right] + (B_R + B_S) \]
Index Join

Memory

R

R

S

......
Index Join Algorithm

Read a block B from R

For each \( r \in B \) do

\[ X \leftarrow \text{index-lookup}(S.C, \ r.C) \]

For each \( s \in X \) do

output (\(r,s\))

Repeat to the end of R

– How many disk IOs?
Factors to Consider

- Index look up cost
  - How many blocks for index?
  - How many levels?
- Number of matching tuples
Partial Index Caching

– How many disk IOs for index lookup on average?

99 nodes

100 nodes
Summary of cost:

Block nested-loop join      5500
Merge join                  1500
Sort+merge joint            7500
Hash join                   4500
R1.C index                  2090–16100
Statistics for Tables

• DBMS has to collect statistics on tables/indexes
  – For optimal performance
  – Without stats, DBMS does stupid things...
• DB2
  – RUNSTATS ON TABLE <userid>.<table> (AND INDEXES ALL)
• Oracle
  – ANALYZE TABLE <table> COMPUTE STATISTICS
  – ANALYZE TABLE <table> ESTIMATE STATISTICS (cheaper than COMPUTE)
• Run the command after major update/index construction
Summary of Join Algorithms

- **Nested-loop join** ok for “small” relations (relative to memory size)
- **Hash join** usually best for *equi-join* if relations not sorted and no index
- **Merge join** for sorted relations
- **Sort merge join** good for *non-equijoin*
- Consider **index join** if index exists
- DBMS maintains *statistics* on data