Optimization of join operation

- Nested loop: one relation acts as outer loop relation (O), the other acts as inner loop relation (I). For every tuple in O, scan I one time to check join condition.

  Because the relation is accessed from disk in the unit of block, we can use block buffer to improve efficiency. For R $\bowtie$ S, if let R as O, S as I, b_R is physical block number of R, b_S is physical block number of S, there are n_B block buffers in system ($n_B>=2$), and n_B-1 buffers used for O, one buffer used for I, then the total disk access times needed to compute R $\bowtie$ S is:

  $b_R + \lceil b_R/(n_B-1) \rceil \times b_S$

Optimization of join operation

- Merge scan: order the relation R and S on disk in ahead, then we can compare their tuples in order, and both relation only need to scan one time. If R and S have not ordered in ahead, must consider the ordering cost to see if it is worth to use this method (p122)
- Using index or hash to look for mapping tuples: in nested loop method, if there is suitable access route on I (say B+ tree index), it can be used to substitute sequence scan. It is best when there is cluster index or hash on join attributes.
- Hash join: because the join attributes of R and S have the same domain, R and S can be hashed into the same hash file using the same hash function, then R $\bowtie$ S can be computed based on the hash file.