Project(R,A)

Create a new relation that retains only the attributes A taken from R.

Category: One-pass, tuple based

Notes: I/O cost depends on source.

Select(R,C)

Create a new relation including only tuples from R that satisfy C

Category: One-pass, tuple based

Notes: I/O cost depends on source.

DupElim(R)

Create a new relation from R by including each unique tuple exactly once

Category: One-pass, full relation

Notes: Set of unique tuples must fit in memory, $B(\delta(R)) \leq M$

GroupMin(R,A,B)

Create a new relation consisting of unique tuples of the attributes A and the minima of the attributes B over the corresponding grouped sets of tuples.

Category: One-pass, full relation

Notes: Keep the minimum per group in memory. I/O cost is B.

GroupMax(R,A,B)

Create a new relation consisting of unique tuples of the attributes A and the maxima of the attributes B over the corresponding grouped sets of tuples.

Category: One-pass, full relation

Notes: Keep the maximum per group in memory. I/O cost is B.

GroupCount(R,A)

Create a new relation consisting of unique tuples of the attributes A and counts of the sizes of corresponding grouped sets of tuples.

Category: One-pass, full relation

Notes: Keep a running count per group in memory. I/O cost is B.

GroupSum(R,A,B)

Create a new relation consisting of unique tuples of the attributes A and the sums of the attributes B over the corresponding grouped sets of tuples.

Category: One-pass, full relation

Notes: Keep a running sum per group in memory. I/O cost is B.

GroupAvg(R,A,B)

Create a new relation consisting of unique tuples of the attributes A and the averages of the attributes B over the corresponding grouped sets of tuples.

Category: One-pass, full relation

Notes: Keep a running sum and count per group in memory. I/O cost is B.

SetUnion(R,S)

Create a new relation containing each of the unique tuples found in either R or S.

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

BagUnion(R,S)

Create a new relation containing each of the tuples found in either R or S (including duplicates).

Category: One-pass, tuple-based

Notes: Minimal memory used. I/O cost is B(R)+B(S).

SetIntersection(R,S)

Create a new relation containing each of the unique tuples found in both R and S.

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

BagIntersection(R,S)

Create a new relation containing each tuple found in both R and S, repeated the lesser of their number of occurrences in each.

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

SetDifference(R,S)

Create a new relation containing each unique tuple found in R but not in S

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

BagDifference(R,S)

Create a new relation containing each unique tuple found in R more often than S, as many times as there are excess appearances in R

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

Product(R,S)

Create a new relation containing every possible concatenation of a tuple from R with a tuple from S.

Category: One-pass, tuple-based

Notes: S must fit in memory in memory. I/O cost is B(R)+B(S).

NaturalJoin(R,S)

Create a new relation containing concatenations of a tuple from R with a tuple from S, where the tuples match on shared attributes.

Category: One-pass, full relation

Notes: S must fit in memory in memory. I/O cost is B(R)xB(S)/M.

NestedLoopJoin(R,S)

Create a new relation containing concatenations of a tuple from R with a tuple from S, where the tuples match on shared attributes.

Category: One-and-a-half-pass, full relation

Notes: I/O cost is $B(R) \times \lceil B(S)/M \rceil$.

Sort (R)

Applies a two-phase multiway merge sort on R.

Category: Two pass, full relation

Notes: I/O cost is 3B. Size limit is $B < M^2$

SortDupElim(R)

Uses merge sort to eliminate duplicates in large relation R

Category: Two pass, full relation

Notes: I/O cost is 3B. Size limit is $B < M^2$

SortGroupAgg(R,A,G)

Uses merge sort to compute some aggregated property G of tuples from large relation R, as grouped by attributes A

Category: Two pass, full relation

Notes: I/O cost is 3B. Size limit is $B < M^2$

SortUnion(R,S)

Uses merge sort to take the union of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $B(R) + B(S) < M^2$

SortIntersection(R,S)

Uses merge sort to take the intersection of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $B(R) + B(S) < M^2$

SortDifference(R,S)

Uses merge sort to take the set difference of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $B(R) + B(S) < M^2$

SortJoin(R,S)

Uses merge sort to produce a join of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $B(R) + B(S) < M^2$

Simple version has I/O cost is 5(B(R)+B(S)). Size limit is $\max(B(R) + B(S)) < M^2$

HashDupElim(R)

Uses hashing to eliminate duplicates in large relation R

Category: Two pass, full relation

Notes: I/O cost is 3B. Size limit is $B < M^2$

HashGroupAgg(R,A,G)

Uses hashing to compute some aggregated property G of tuples from large relation R, as grouped by attributes A

Category: Two pass, full relation

Notes: I/O cost is 3B. Size limit is $B < M^2$

HashUnion(R,S)

Uses hashing to take the union of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $\min(B(R) + B(S)) < M^2$

HashIntersection(R,S)

Uses hashing to take the intersection of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $\min(B(R) + B(S)) < M^2$

HashDifference(R,S)

Uses hashing to take the set difference of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $\min(B(R) + B(S)) < M^2$

HashJoin(R,S)

Uses hashing to produce a join of large relations R and S

Category: Two pass, full relation

Notes: I/O cost is 3(B(R)+B(S)). Size limit is $\min(B(R) + B(S)) < M^2$

Fancier version has I/O cost (3-2M/B(S))(B(R)+B(S))

IndexSelect(R,A)

Uses an index to select tuples from R matching condition C on A

Category: Index-based

Notes: Average disk access is B(R)/V(R,A).

SortedIndexJoin(R,S)

Uses a sorted index to produce a join of large relations R and S

Category: Index-based

Notes: Optimistic case disk access is B(R)+B(S)