Relational Joins

Adam Ansari & Harsh Praharaj

Scope

- ➔ Theoretical underpinnings of Relational Joins
- ➔ Real world use-case of Cross Joins (Arbitrage)

How do Relational Joins work under the hood?

Physical Joins

Physical Joins

- ➔ These are the joins that users don’t use/write in their SQL queries.
- ➔ Instead these are implemented inside RDBMS as operators or algorithms to implement the Logical/Relational Joins.
- ➔ Three types :- Nested Loop, Merge and Hash.

Nested Loop Join

- ➔ Simplest form of physical join
- ➔ It’s the nesting of the for loops in this algorithm that gives nested loops join its name.
- ➔ Compares each row from one table (outer table) to each row from the other table (inner table) looking for rows that satisfy the join predicate.
- ➔ Cost grows quickly as the size of the input tables grow.

```
for each row R1 in the outer table
    for each row R2 in the inner table
        if R1 joins with R2
            return (R1, R2)
```

"Nested Loops" in action

Pseudo-Code Implementation

Merge Join

- ➔ Nested Joins okay for small datasets but we need something else for moderately sized (relatively bigger datasets).
- ➔ Simultaneously read and compare the two sorted inputs - one row at a time
- ➔ If the rows are equal, we output a joined row and continue. If the rows are not equal, we discard the lesser of the two inputs and continue. (Since the inputs are sorted, we know that we are discarding a row that is less than any of the remaining rows in either input and, thus, can never join.)
- ➔ Total cost is proportional to the sum of the number of rows in the inputs.

Hash Join

- ➔ Instead of iterating through each record in an inner table for each record in the outer, hash joins build a temporary hash table to index the values of the field whose equality is being tested.
- ➔ Executes in two stages :- ***Build Stage*** followed by the ***Probe Stage***
- ➔ Saves time at the cost of memory
- ➔ Unlike the nested loops and merge joins which immediately begin flowing output rows, the hash join is blocking on its build input.

for each row R1 in the build table

begin

calculate hash value on R1 join key(s)

insert R1 into the appropriate hash bucket

end

for each row R2 in the probe table

begin

calculate hash value on R2 join key(s)

for each row R1 in the corresponding hash bucket

if R1 joins with R2

return (R1, R2)

end

m = Number of entries in the Outer table (First deck) = 52
N = Number of entries in the Inner table (Second deck) = 52

Join	Complexity	Comparisons	Total
Nested Loop	O(m*n)	52*52	~2700
Merge Join	O(mlogm+nlogn)	Sorting(52log52+52log52) + comparisons(~100)	~280
Hash Join	O(m) + O(n)	Build Phase(52)+ Probe Phase(52)	104

Thank You!