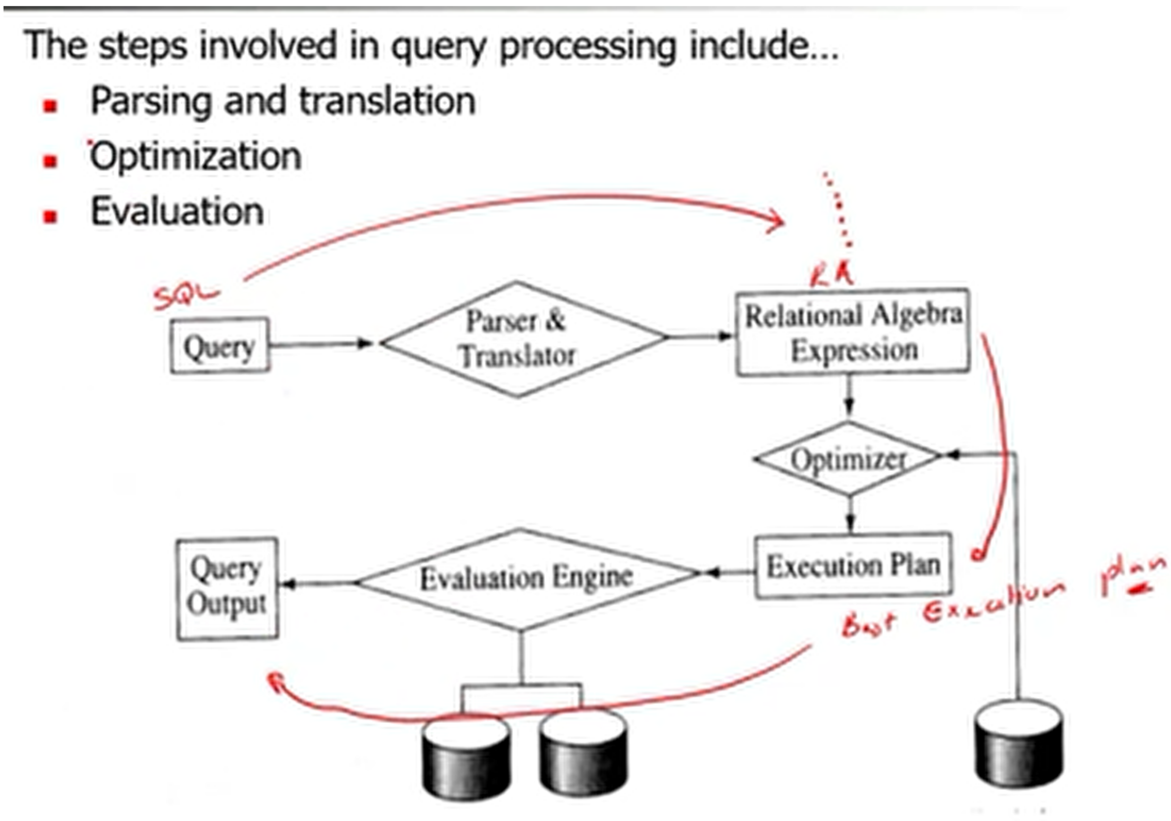
Query Processing is talking about what happens inside when we execute an SQL statement. There are 3 steps in Query Processing.

1. Parsing and Translation - parsing means checking syntactical errors in SQL query and stuff. Translation means converting SQL query into mathematical language like relational algebra.
2. Optimization - For one SQL query there are many relational algebra expressions, hence many execution plans. In this step, it will identify the best execution plan.
3. Evaluation - Now that we have an execution plan, the evaluation engine will execute that plan and will the output of the query.

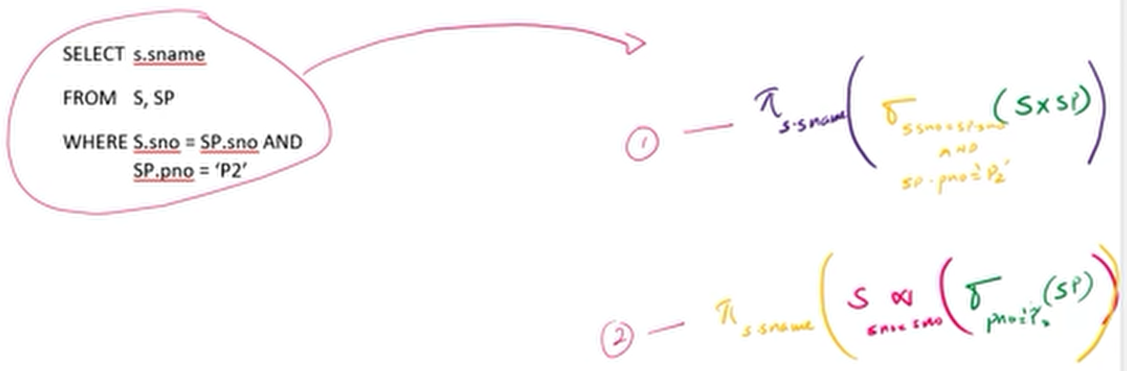
The steps involved in query processing include...

- Parsing and translation
- Optimization
- Evaluation



Example

1. Parsing and Translation



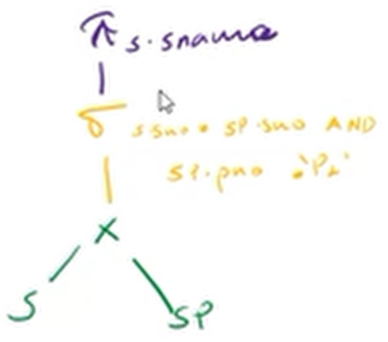
```
SELECT s.sname
FROM   S, SP
WHERE  S.sno = SP.sno AND
       SP.pno = 'P2'
```

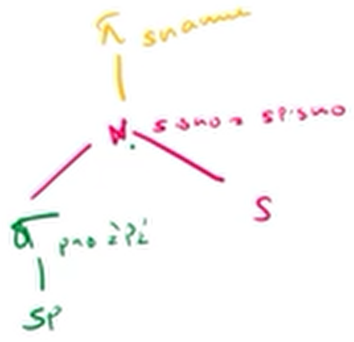
① — $\pi_{s.sname}\left(\sigma_{\substack{S.sno = SP.sno \\ AND \\ SP.pno = 'P_2'}}(S \times SP)\right)$

② — $\pi_{s.sname}\left(S \bowtie_{S.sno = S.sno}\left(\sigma_{pno = 'P_2'}(SP)\right)\right)$

This query does not have any syntactical errors Likewise we can have many algebra expressions for the same SQL query.

2. Optimization - Also we can have query trees or execution plans for the above algebra expressions.

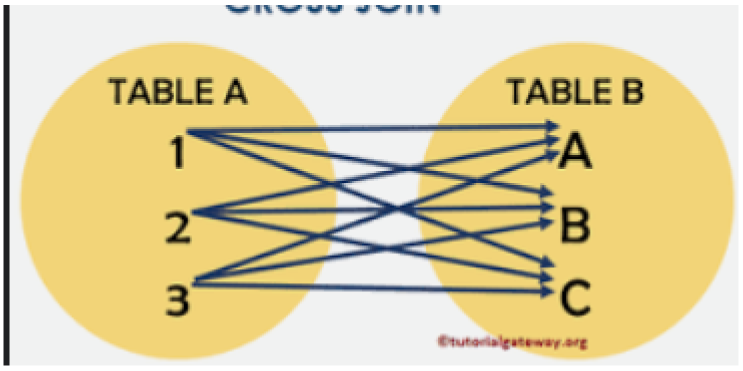




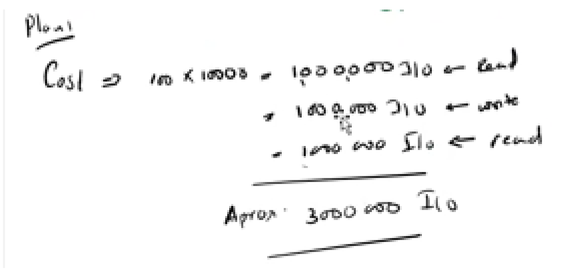
But still we need to find the best execution plan from other execution plans here. To measure the best execution plan we need some kind of measurement. Here we take I/O cost as the measurement.

a. Let’s take the first execution plan with the cartesian product.

Lets say we have 100 pages in the S table and 10000 pages in the SP table in the hard disk. As you can see we’re first joining these two tables together with cartesian product. When joining, one page of the S table is connected to every page of the SP table. Likewise every page of the S table is connected to SP table pages. Like in below picture.



So, to do this ultimately we have to take every page of the S table into memory once. And we will have to take every page of SP table into memory S times. So the total cost for the cartesian product is, 100 * 10000 = 1000000. After cartesian join, We just cant have 1000000 records in memory. So we will have to write them back to the hard disk, so that we can use them later. So to write them back, again we have to do 1000000 cost. Now we have the condition S.no = SP.no AND SP.no = ‘P2’. In order to check this condition we will have to retrieve 1000000 records back to memory (this process can happen block by block or anyhow but cant take all into memory at once) So, Again 1000000 cost. So the total cost is 3000000 I/O.



Plan 1

Cost ⇒ 100 × 10000 = 1000000 I/O ← read

+ 1000000 I/O ← write

+ 1000000 I/O ← read

Aprox: 3000000 I/O

b. Let’s take the next execution plan and see the cost.

Here you first take !0000 pages into memory and check the the condition SP.no = ‘p2’. Let’s say we have 50 records where SP.no = ‘p2’. But all these conditions checking and everything is happening when after pages are retrieved to memory from hard disk. So actually comparison does not affect to the cost directly. So for noe cost is only 10000 I/O (cost reading SP pages into memory) Then instead of cartesian we have inner join. Since we have 50 records where the SP.no = ‘p2’ condition satisfies, we get one page by one page of the S table, into memory and see if SP.no = S.no condition satisfies. Ultimately from this step we only have a cost of 100 I/O. After the comparison is finished we can give the output. Anyhow the total cost of the second execution plan is 10000 + 100 = 10100 I/O.

To get the best execution plan like this we can follow Heuristic rules. Some of them rules are,

1. Do equality operations first before joining 2 tables.
2. Avoid cartesian products.

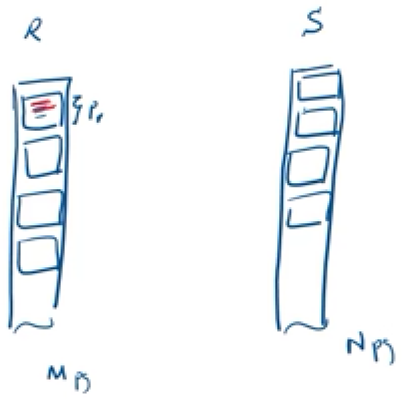
Also when we join tables we use several algorithms to do it.

Algorithms

1. Simple Nested Loop Join (SNLJ)

We have 2 tables. One is an outer table (R) and the other one is the inner table (S). Here, we take one record at a time of the R table into the memory and join it with all the records in the S table. You know we can’t just just take one record at a time into memory we will have to take the whole page into memory to get a record. Then after joining the first record of R table with all the records in S table, we need to get the second record of R table into memory to join it with all the records in S table. But we don’t have to do an I/O operation to get the 2nd record because we already have it in the memory because when we got the first record, we got the whole page which the 1st record resides in.So if we have,
Pr number of records in a page in R table,
We already have Pr number of records in memory. After this Pr is finished joining we go the the next page (next Pr). Likewise, if we have
P number of pages in R table,
P is the I/O cost to retrieve records to memory from R table.

If we talk about S table, we know for each record we have to join all the records in S table, like in above to fetch all the records in S to memory the I/O cost is the number of pages in S table. (not the number records in S table because we fetch pages directly) So if the number of Pages in S table is N, we have to fetch N pages for P*Pr times since we need to fetch N for each record in R.



R - outer table
S - inner table
Pr - no of tuples (records) in a page of the outer table
M - no of pages in outer table
N - no of pages in inner table

We know we have to take all outer table pages to the memory. The cost for that is M. We know for each tuple (record) in the R table we take all the records in the S into memory one page by one page. So the cost for that is N * no tuples in R table = N * (M*Pr)

So total cost = M + N*M*Pr

Here we assume 1 disk block = 1 page.

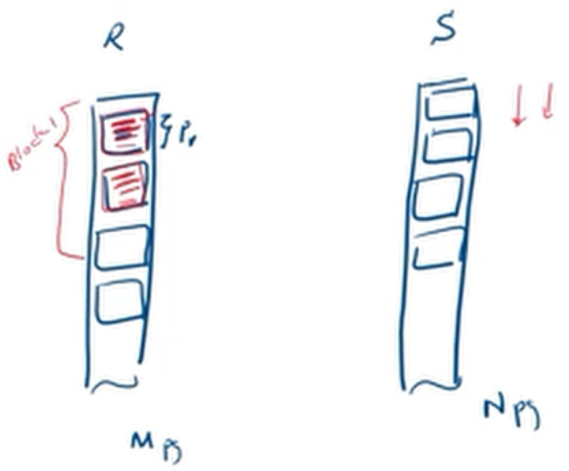
2. Page-Oriented Nested Loop Join (PNLJ)

Here instead of getting all the tuples in the inner table to memory per every tuple in the outer table, we take all the tuples in the inner table to memory per page in the outer table.

So total cost = M + N*M

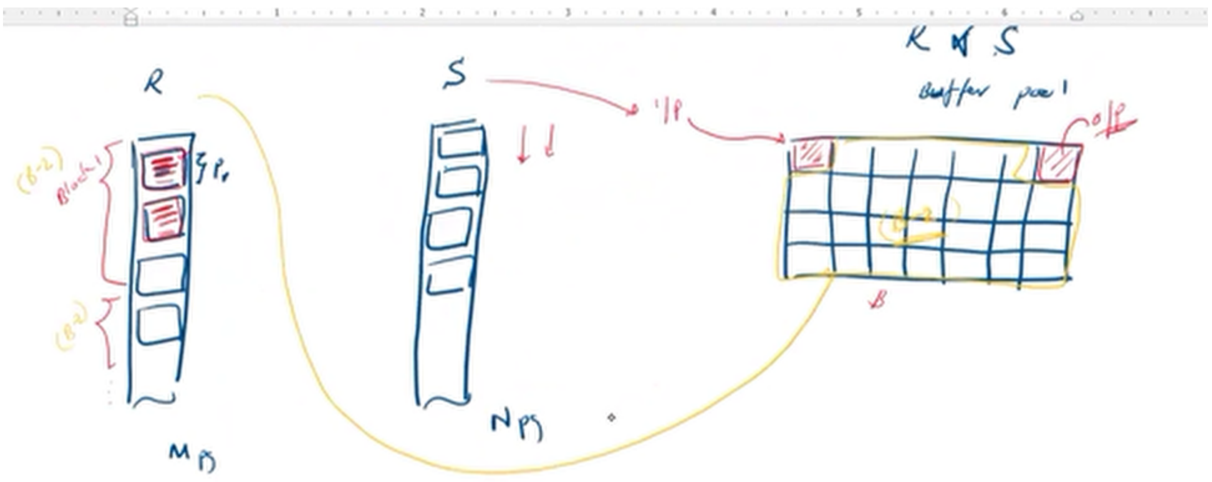
3. Block Nested Loop Join (BNLJ)

Here instead of getting all the tuples in inner table to memory per page in outer table, we take all the tuples in inner table to memory per a set of blocks/pages in outer table.



So total cost = M + N* no of sets of blocks in R table

Now lets see how to calculate no of blocks (here blocks mean, a set of pages) in R.

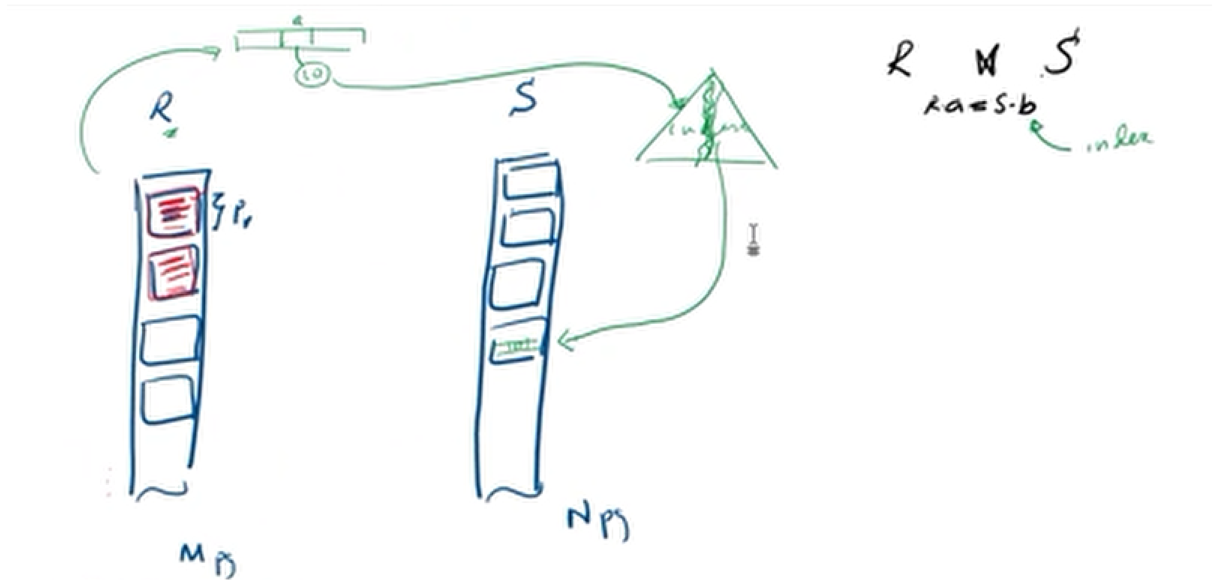


In memory we have a buffer. In this buffer we have a B number of frames. We can keep one page in one frame. So we allocate one frame to send pages back and forth in S table. Also we allocate one frame to get the output. Now we have B - 2 number of frames left. We have M number of pages in R table. So to we can keep B - 2 number pages of R table at once in the buffer. So we have to do this process, for ceil(M / (B - 2)) times to cover all pages in R table. Which means a block size of R table is ceil(M / (B - 2)).

So total cost of BNLJ = M + ceil(M / (B - 2))*N

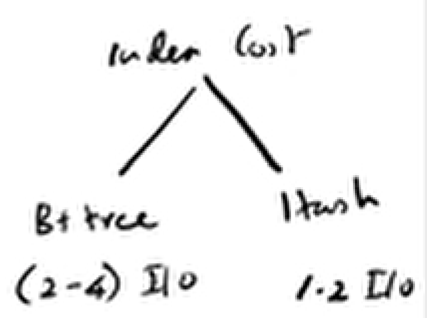
4. Indexed Nested Loop Join (INLJ)

In here for each tuple in the R table, we need to check every tuple in the S table. But here we S Table has indexes. So we dont need take every tuple of S into memory, We only have to go through the index. But we still have to get R tables tuples into memory.



So the total cost = M + (indexing cost) * no of records in R

Index cost ->


index cost
B+ tree
Hash
(2-4) I/O
1.2 I/O