TECH
VAULT

JOIN ALGORITHMS

- NESTED LOOP
- INDEX NESTED LOOP
- MERGE
- HASH

AMR ELHELW

All Here is about Inner join

Customers

cid	cname	email	Region
c1	Alice	alice@example.com	West
c2	Bob	bob@example.com	East
c3	Charlie	charlie@example.com	North
c4	Danny	danny@example.com	East

Orders

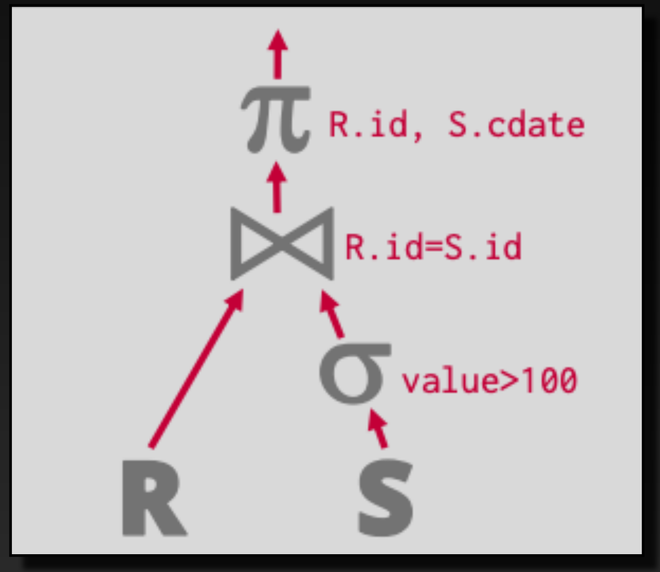
oid	odate	cid	amount
101	2022-04-10	c2	$150
102	2023-05-07	c1	$20
103	2021-10-04	c4	$37
104	2021-10-04	c1	$126
105	2023-06-20	c1	$30
106	2022-07-25	c4	$74

“Return all customer names and order amounts from the East region”

Order of joins matters in the cost.

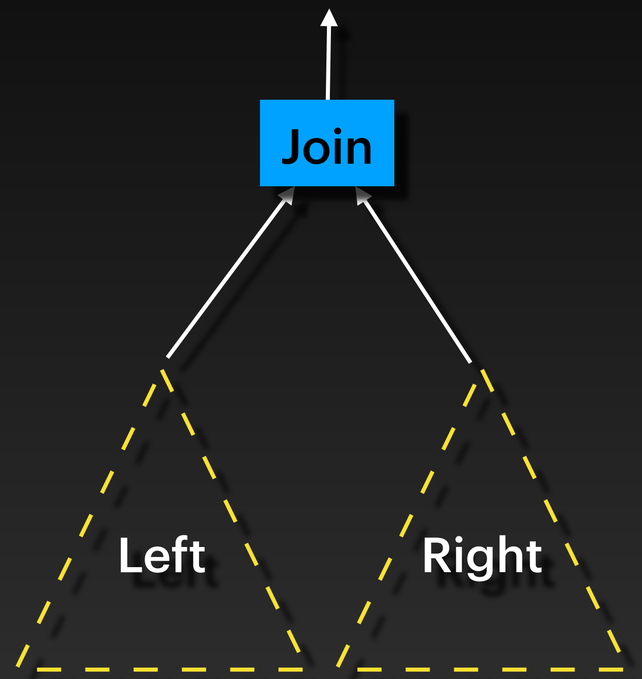
Join Query

```
SELECT R.id, S.cdate
FROM R JOIN S
ON R.id = S.id
WHERE S.value > 100
```



Join Plan

- Each input can be a subplan
  - Left (outer)
  - Right (inner)
- Each join algorithm has a different cost
- Cost can be estimated in terms of number of rows fetched from each side



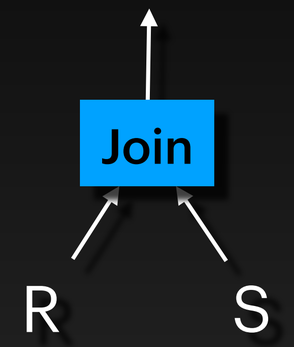
Nested Loop Join

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```

```
foreach tuple r in R:
  foreach tuple s in S:
    if (condition): output row
```

Outer

Inner



R

id	name
3	Alice
2	Bob
1	Charlie
4	Danny

S

id	Value
2	1000
3	2000
4	1300
2	5000
4	2500

Output

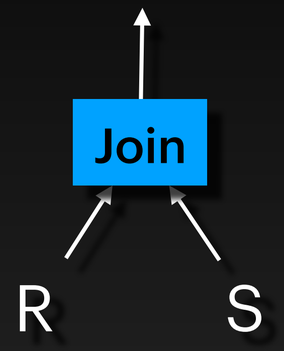
R.id	R.name	S.id	S.value
3	Alice	3	2000
2	Bob	2	1000
2	Bob	2	5000
4	Danny	4	1300
4	Danny	4	2500

Generally:

* nested loop join is the worst one.beacause of nested.
* is usually used in inequy joins.

Nested Loop Join

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```

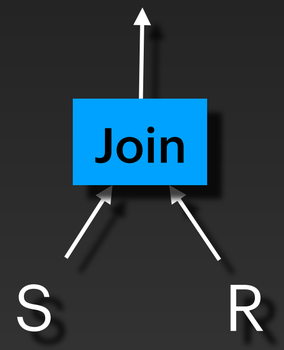


Cost = $N_R + (N_R * N_S)$

N_R: #rows in R

N_S: #rows in S

if Nr < Ns So 1st oneis better
else second one is better
because the nested is fixed in both equations



Cost = $N_S + (N_S * N_R)$

General Rule:
the lowest number of rows table
is better to be the outer table.
to minimize the left part.

Better to use the smaller input on the left side!

Nested Loop Join

```sql
SELECT *
FROM R JOIN S
ON R.id = S.id
```

```
foreach tuple r in R:
  foreach tuple s in S:
    if (condition): output row
```

Search for tuples in S with id = *r.id*

What if there was already an index on S.id?

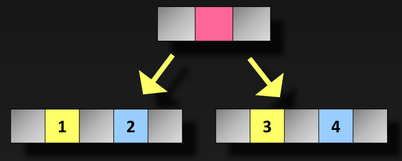
```
foreach tuple r in R:
  SS = Search Index for r.id
  foreach tuple s in SS:
    output row
```

Index Nested Loop Join

Index Nested Loop Join

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```

Index on S.id



Cost = N_outer + (N_outer * Cost_index_inner)

R

id	name
3	Alice
2	Bob
1	Charlie
4	Danny

S

id	Value
2	1000
3	2000
4	1300
2	5000
4	2500

Output

R.id	R.name	S.id	S.value
3	Alice	3	2000
2	Bob	2	1000
2	Bob	2	5000
4	Danny	4	1300
4	Danny	4	2500

Cost = N_outer + N_inner + Sort (if needed)

Merge Join

* works only with equi joins
* needs tables to be sorted on join key.
 other wise it sorts them which is more c.ost.

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```

perfect: if tables are sorted based on the join key.
how they can be sorted? if they comes from index scan, sorted before in a merge join, cached in a sort operation before

NOTE: the output is sorted which means it can be used later in another merge join.
and it is also useful if you need to order by that key.

R

id	name
1	Alice
2	Bob
3	Alice
4	Danny

S

id	Value
2	1000
2	5000
3	2000
4	1300
4	2500

Output

R.id	R.name	S.id	S.value
2	Bob	2	1000
2	Bob	2	5000
3	Alice	3	2000
4	Danny	4	1300
4	Danny	4	2500

HERE:
it sorts the 2 tables on id the:
it uses 2 pointers technique to move pointers
if both are queal. other wise move the less pointer and so on.

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```

If there was already an index on S.id

```
foreach tuple r in R:
  SS = Search Index for r.id
  foreach tuple s in SS:
    output row
```

If there was no index S.id

Build an “index” on the fly?

Focus on equi-joins —> Hash index

```
Build Hash table on S.id
foreach tuple r in R:
  SS = Search Hash table for r.id
  foreach tuple s in SS:
    output row
```

STEPS:
1. choose one table to build hash table on it.
2. iterate over the second table and lookup in the hash table.

Index Nested Loop Join

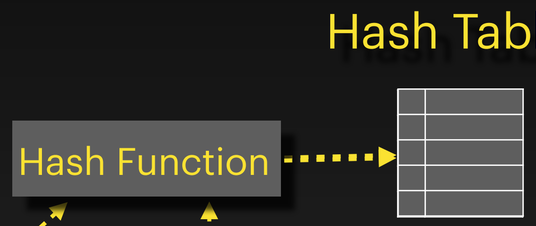
Cost = (N_inner + N_outer) * Cost_lookup_in_hash_table
consider outer, inner a big & small tables and optimizer choses the brst one to build a hash table for.

>> temp index dor this query inly.

Hash Join

Hash Join

```
SELECT *
FROM R JOIN S
ON R.id = S.id
```



R

id	name
3	Alice
2	Bob
1	Charlie
4	Danny

S

id	Value
2	1000
3	2000
4	1300
2	5000
4	2500

Output

R.id	R.name	S.id	S.value
3	Alice	3	2000
2	Bob	2	1000
2	Bob	2	5000
4	Danny	4	1300
4	Danny	4	2500

iterate over ids in R.
hash the current id and
lookup its value in hash table

create hash table for ids in S