MULTICOLUMN INDICES, JOINS & AGGREGATIONS

MULTICOLUMN INDICES

MULTICOLUMN INDEX (COMPOSITE INDEX)

```
create index index_documents_on_stp on
documents(supplier, total_amount,
published_on);
```

{ ANASOFT | 1000 | 12.9.2019 }

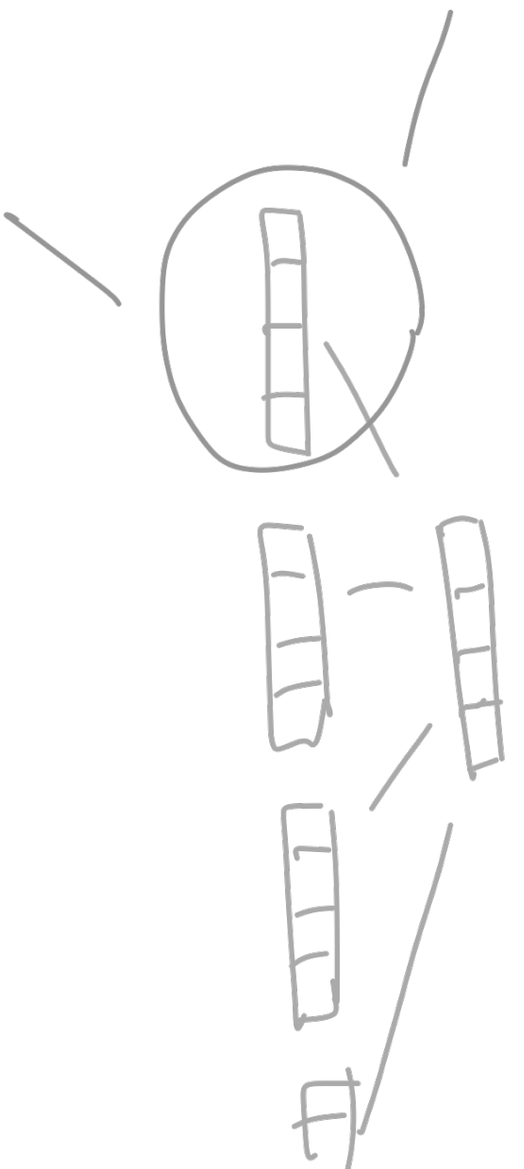
{ ANASOFT | 1000 | 12.10.2019 }

{ ANASOFT | 2000 | 1.1.2018 }

{ BETHESDA | 900 | 18.11.2009 }

{ BETHESDA | 1200 | 16.5.2010 }

{ BETHESDA | 1200 | 17.5.2010 }



```
explain select * from documents where supplier =
'SPP';


Bitmap Heap Scan on documents  (cost=4.48..35.99
rows=8 width=303)
   Recheck Cond: ((supplier)::text = 'SPP'::text)
   ->  Bitmap Index Scan on
index_documents_on_stp  (cost=0.00..4.48 rows=8
width=0)
         Index Cond: ((supplier)::text =
'SPP'::text)
```

```
explain select * from documents where supplier =
'SPP' and total_amount = 2000;
```

```
Index Scan using index_documents_on_stp on
documents  (cost=0.42..8.44 rows=1 width=303)
   Index Cond: (((supplier)::text = 'SPP'::text)
AND (total_amount = 2000::double precision))
```

```
explain select * from documents where supplier =
'SPP' and total_amount = 2000 and published_on =
now();
```

```
Index Scan using index_documents_on_stp on
documents  (cost=0.42..8.45 rows=1 width=303)
   Index Cond: (((supplier)::text = 'SPP'::text)
AND (total_amount = 2000::double precision) AND
(published_on = now()))
```

NON-LEADING COLUMN CONDITIONS

```
explain select * from documents where
total_amount = 2000;
```

```
 Seq Scan on documents  (cost=0.00..17398.30
rows=1276 width=303)
   Filter: (total_amount = 2000::double
precision)
```

NON-LEADING COLUMN CONDITIONS - INDEX SCAN ?!

```
explain select * from documents where total_amount
= 2000 and published_on > '2012-01-01';
```

```
Bitmap Heap Scan on documents
(cost=15348.86..17840.69 rows=779 width=303)
   Recheck Cond: ((total_amount = 2000::double
precision) AND (published_on > '2012-01-01
00:00:00'::timestamp without time zone))
   ->  Bitmap Index Scan on index_documents_on_stp
(cost=0.00..15348.66 rows=779 width=0)
         Index Cond: ((total_amount = 2000::double
precision) AND (published_on > '2012-01-01
00:00:00'::timestamp without time zone))
```

BENCHMARK

```
explain analyze select * from documents where
total_amount = 2000 and published_on > '2012-01-01';

Bitmap Heap Scan on documents
(cost=15348.86..17840.69 rows=779 width=303) (actual
time=19.163..19.663 rows=840 loops=1)
   Recheck Cond: ((total_amount = 2000::double
precision) AND (published_on > '2012-01-01
00:00:00'::timestamp without time zone))
   ->  Bitmap Index Scan on index_documents_on_stp
(cost=0.00..15348.66 rows=779 width=0) (actual
time=19.076..19.076 rows=840 loops=1)
         Index Cond: ((total_amount = 2000::double
precision) AND (published_on > '2012-01-01
00:00:00'::timestamp without time zone))
 Total runtime: 19.739 ms
```

BENCHMARK

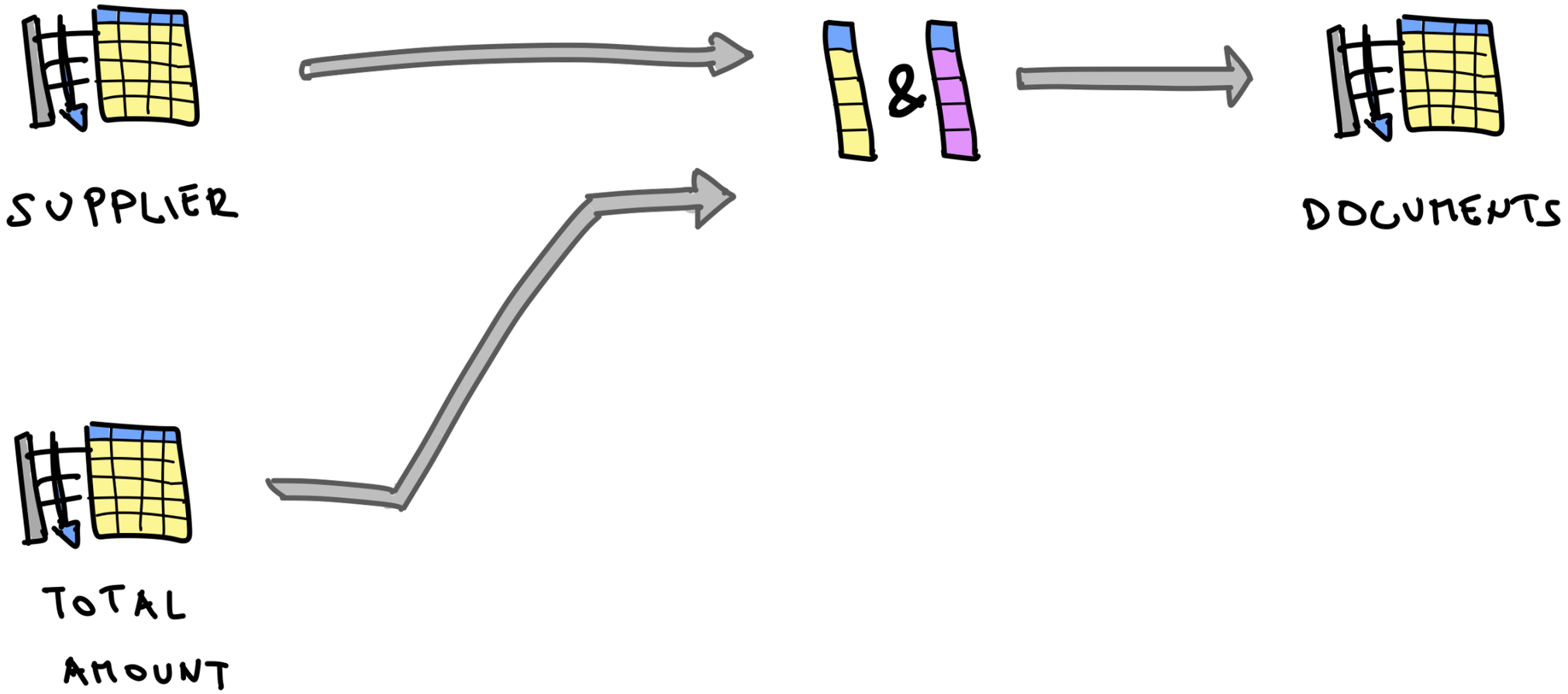
```
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze select * from documents where
total_amount = 2000 and published_on >
'2012-01-01';
```

```
Seq Scan on documents  (cost=0.00..18276.36
rows=779 width=303) (actual time=29.384..78.961
rows=840 loops=1)
   Filter: ((published_on > '2012-01-01
00:00:00'::timestamp without time zone) AND
(total_amount = 2000::double precision))
   Rows Removed by Filter: 350384
 Total runtime: 79.021 ms
```

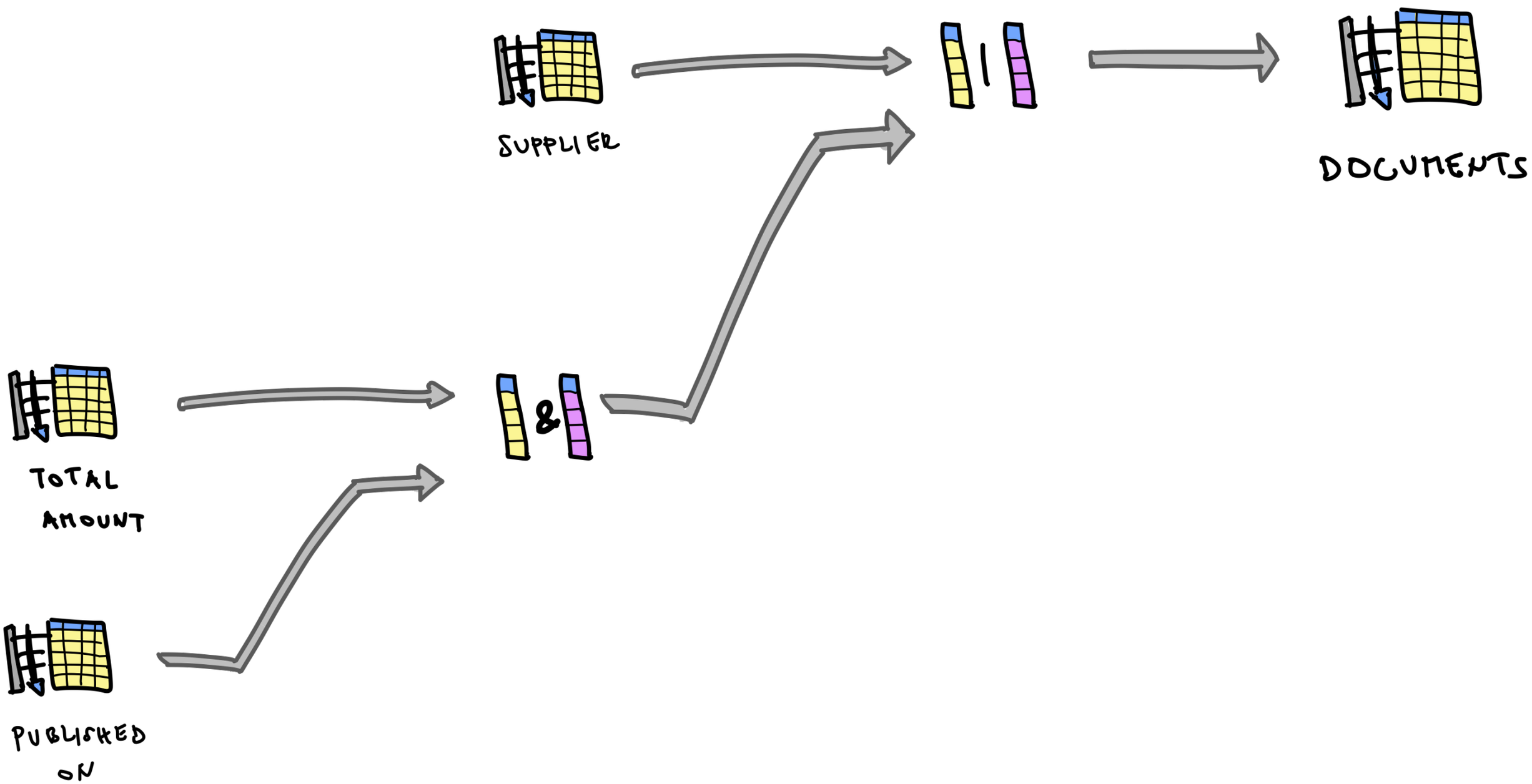
3 SEPARATE SINGLE-COLUMN INDICES

```
drop index index_documents_on_stp;

create index index_documents_on_supplier on
documents(supplier);

create index index_documents_on_total_amount on
documents(total_amount);

create index index_documents_on_published_on on
documents(published_on);
```

```
explain select * from documents where supplier = 'SPP'
and total_amount = 2000;

Bitmap Heap Scan on documents  (cost=30.73..34.74 rows=1
width=303)
   Recheck Cond: (((supplier)::text = 'SPP'::text) AND
(total_amount = 2000::double precision))
   ->  BitmapAnd  (cost=30.73..30.73 rows=1 width=0)
         ->  Bitmap Index Scan on
index_documents_on_supplier  (cost=0.00..4.48 rows=8
width=0)
               Index Cond: ((supplier)::text =
'SPP'::text)
         ->  Bitmap Index Scan on
index_documents_on_total_amount  (cost=0.00..25.99
rows=1276 width=0)
               Index Cond: (total_amount = 2000::double
precision)
```


SUPPLIER
&
DOCUMENTS
TOTAL
AMOUNT

```
select *
  from documents
 where supplier = 'SPP'
    or total_amount = 2000
   and published_on > '2013-01-01';
```


SUPPLIER
DOCUMENTS
TOTAL AMOUNT
PUBLISHED ON
&

COMPOSITE INDEX VS. PER-COLUMN INDEX

- Per-column indices:
  - more flexible
  - support any column combinations in the *where* clause

- Use composite index for:
  - efficient *order by* on 2 and more columns
  - covering index - piggybacking data (add data to index to leverage index-only scan and avoid heap scans)

COVERING INDEX

```
create index index_documents_on_dc on
documents(department, customer);

vacuum analyze;

explain select customer from documents where
department = 'Rozhlas a televizia Slovenska';
```

```
Index Only Scan using index_documents_on_dc on
documents  (cost=0.55..1200.50 rows=16683
width=41)

   Index Cond: (department = 'Rozhlas a televizia
Slovenska'::text)
```

JOINS

JOINS

- Nested loop
- Hash join
- Merge join

NESTED LOOP

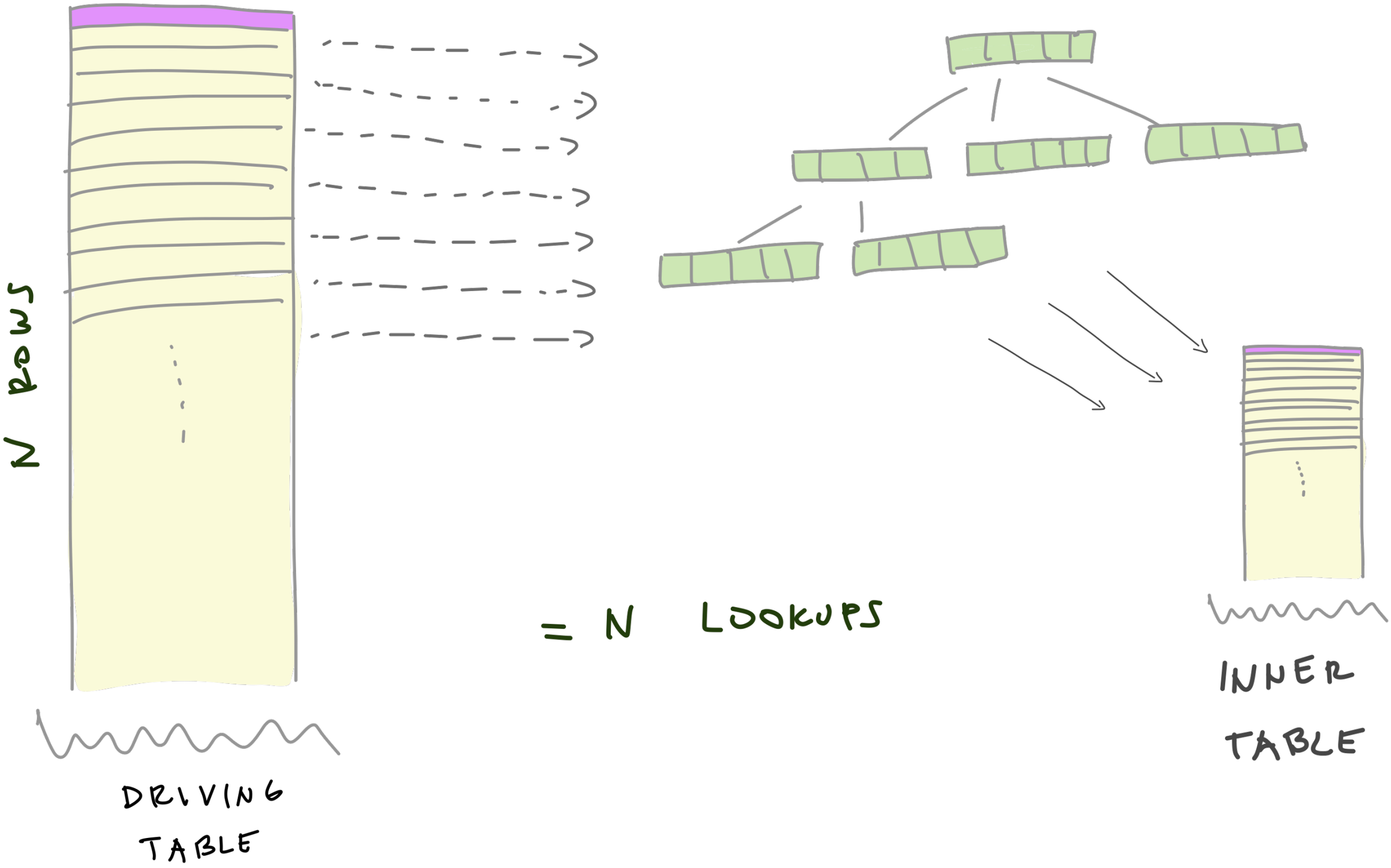
- For each row in table A
  - For each row in table B
    - If join condition is true, emit row (e.g., A.id = B.a_id)

- Most generic join algorithm, has no expectations

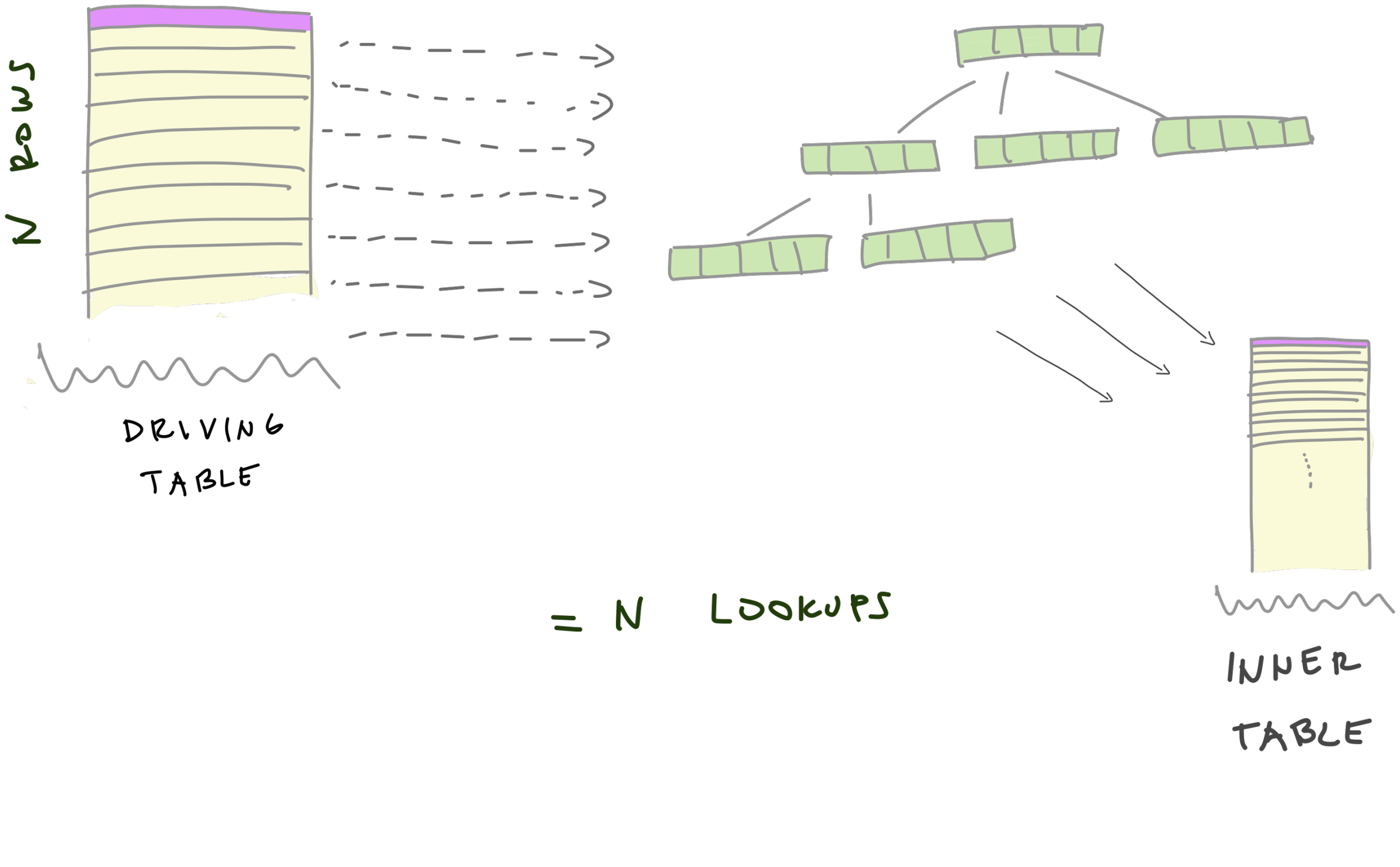
NESTED LOOP WITH INDEX

```
set enable_hashjoin=off;

explain select * from character c
   join paragraph p on p.charid = c.charid;

Nested Loop  (cost=0.28..13080.14 rows=35465
width=416)
   ->  Seq Scan on paragraph p
(cost=0.00..2094.65 rows=35465 width=361)
   ->  Index Scan using character_pkey on
"character" c  (cost=0.28..0.30 rows=1 width=55)
         Index Cond: ((charid)::text =
(p.charid)::text)
```

NESTED LOOPS WITH INDEX



NESTED LOOPS WITH INDEX



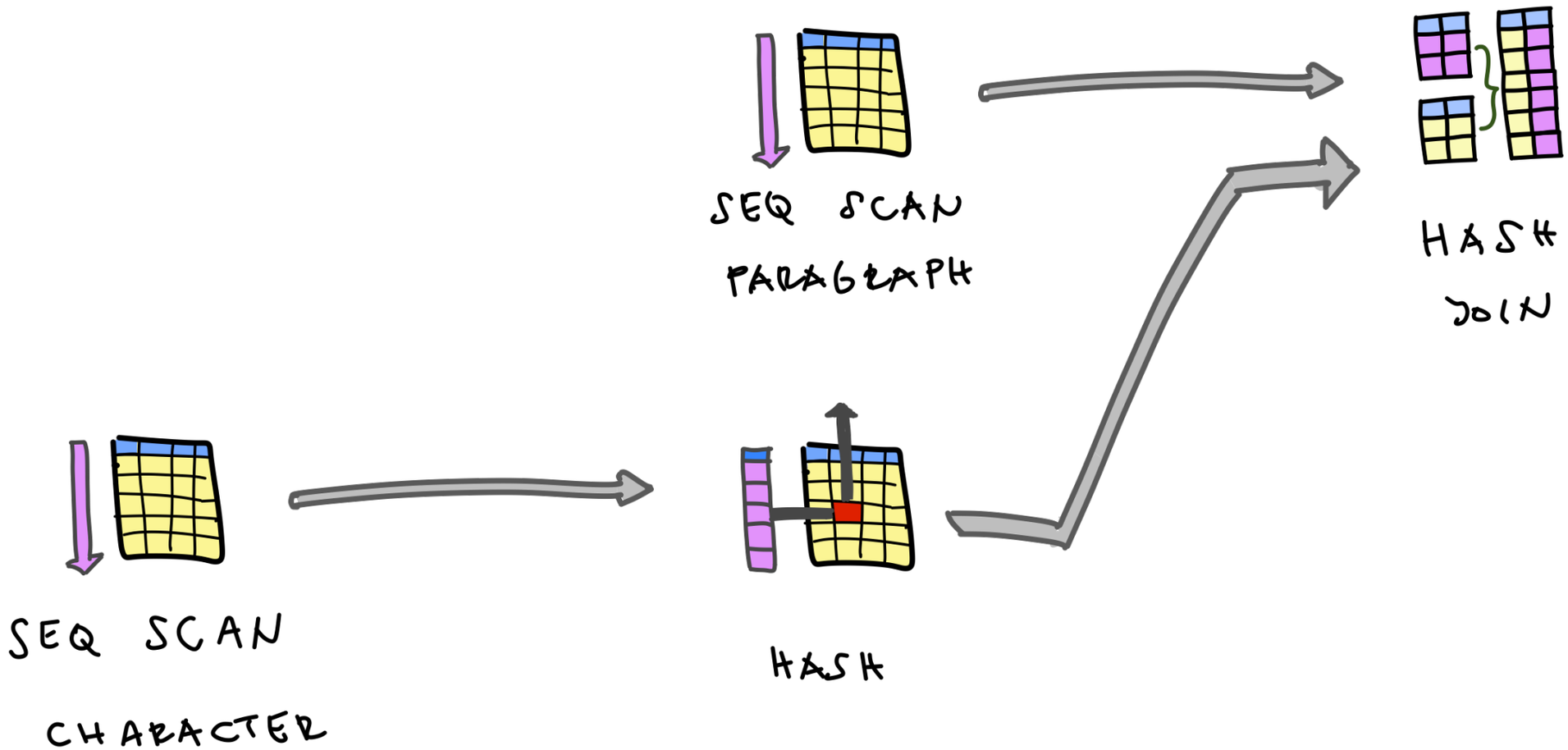
= N LOOKUPS

NESTED LOOPS – SUMMARY

- The smaller the driving table, the less lookups
- Query planner will make the correct decision based on the statistics. Ignore advice that you need to handle this yourself. (You do, if your database is using RBO, though).
- Nested loops are feasible with index
  - Seq scan for each row of the inner table is.. inefficient
    - CBO will try to avoid it

HASH JOIN

```
set enable_hashjoin=on;

explain select * from character c
   join paragraph p on p.charid = c.charid;

Hash Join  (cost=41.48..2623.78 rows=35465
width=416)
   Hash Cond: ((p.charid)::text = (c.charid)::text)
   ->  Seq Scan on paragraph p  (cost=0.00..2094.65
rows=35465 width=361)
   ->  Hash  (cost=25.66..25.66 rows=1266 width=55)
         ->  Seq Scan on "character" c
(cost=0.00..25.66 rows=1266 width=55)
```

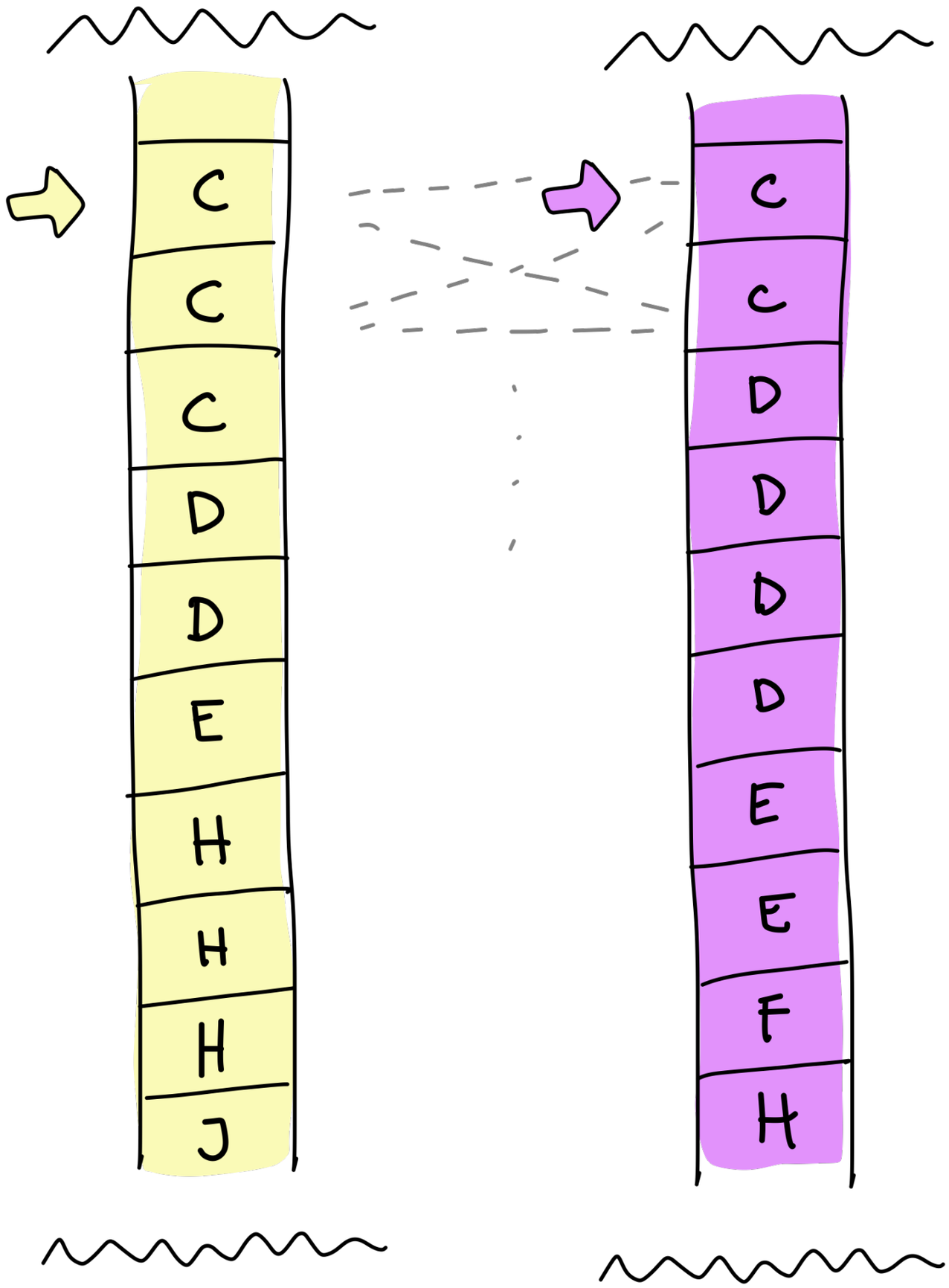

SEQ SCAN
PARAGRAPH
HASH
JOIN
SEQ SCAN
CHARACTER
HASH

HASH JOIN

- Compute hash table from inner table (hash key = join column)
- For each row in driving table, lookup value in the hash table (lookup key = the other join column)
- Only works for simple equality joins

MERGE JOIN

- Zip algorithm
- Requires both tables to be sorted (only viable via index scan)


C
C
C
D
D
E
H
H
H
J
C
C
D
D
D
D
E
E
F
H

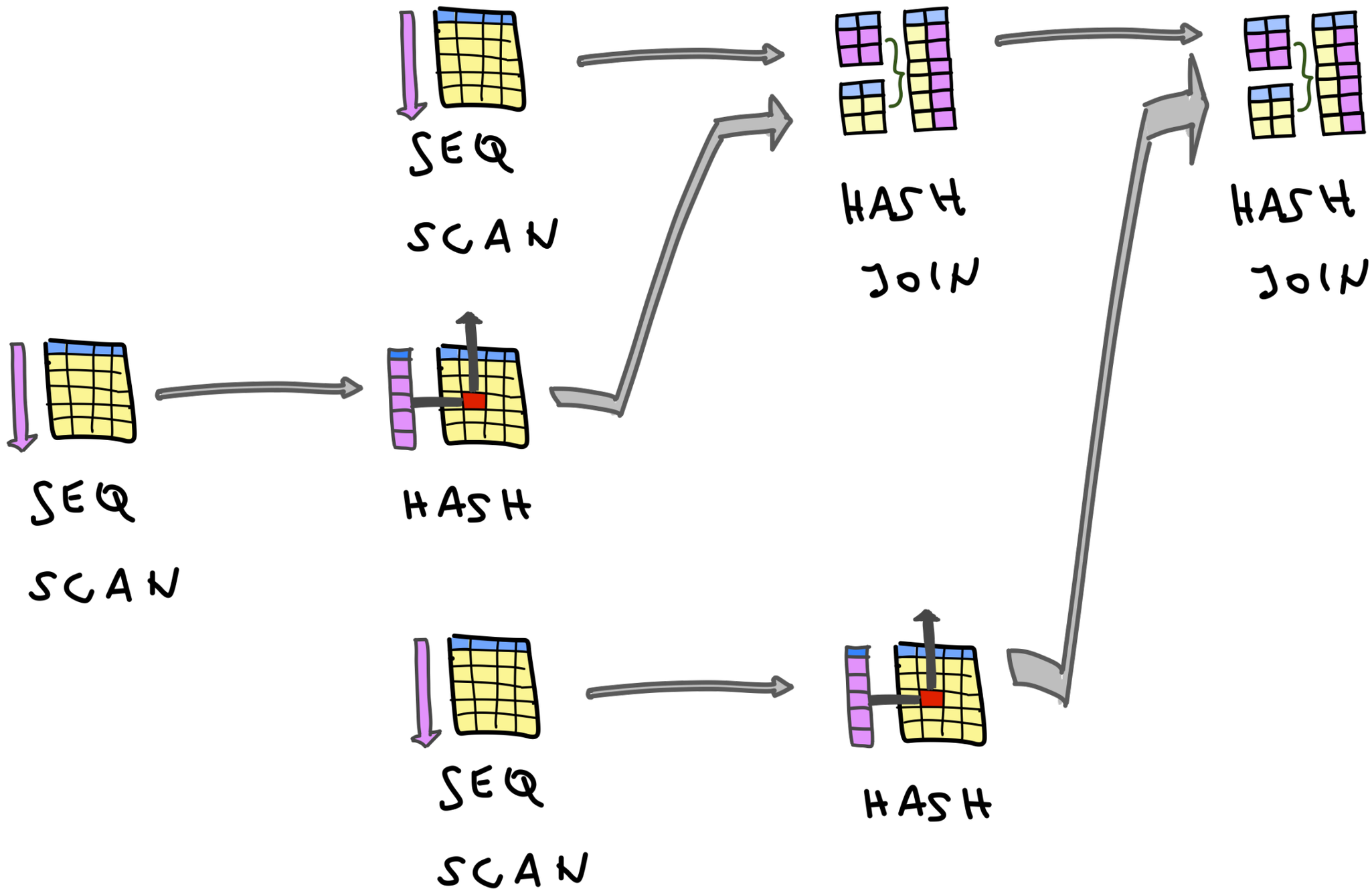
MERGE JOIN

```
create index idx_paragraph_on_charid on
paragraph(charid);

explain select * from character c
   join paragraph p on p.charid = c.charid;

Merge Join  (cost=0.57..8423.73 rows=35465
width=416)
   Merge Cond: ((c.charid)::text =
(p.charid)::text)
   ->  Index Scan using character_pkey on
"character" c  (cost=0.28..63.36 rows=1266
width=55)
   ->  Index Scan using index_paragraph_on_charid
on paragraph p  (cost=0.29..7913.89 rows=35465
width=361)
```

MULTI JOIN

```
select *
   from character c
   join character_work cw on c.charid = cw.charid
   join work w on w.workid = cw.workid;
```

```
Hash Join  (cost=43.45..102.93 rows=1346 width=173)
   Hash Cond: ((cw.workid)::text = (w.workid)::text)
   ->  Hash Join  (cost=41.48..82.45 rows=1346 width=74)
         Hash Cond: ((cw.charid)::text =
(c.charid)::text)
         ->  Seq Scan on character_work cw
(cost=0.00..22.46 rows=1346 width=19)
         ->  Hash  (cost=25.66..25.66 rows=1266 width=55)
               ->  Seq Scan on "character" c
(cost=0.00..25.66 rows=1266 width=55)
   ->  Hash  (cost=1.43..1.43 rows=43 width=99)
         ->  Seq Scan on work w  (cost=0.00..1.43 rows=43
width=99)
```


SEQ
SCAN
SEQ
SCAN
HASH
HASH
JOIN
HASH
JOIN
SEQ
SCAN
HASH

QUERY PLANS EXPLOSION

- Each join:
  - 3 join strategies
  - x2 driving/inner table combinations
  - seq scan vs. index access (3 types)
    - table permutations
    - ...

- Explosion of possible query plans

GENETIC OPTIMIZER

1. Generate a set of initial population - query plans
2. Cross and mutate
3. Evaluate fitness (plan cost) and select the most fit plans (lowest cost)
4. Continue at 2. until stop condition

- Activated based on the number of joined tables (`show geqo_threshold;`)
- For further reading https://www.postgresql.org/docs/current/geqo.html

OTHER TYPES OF JOINS

```
select * from paragraph p
   left join character c on p.charid = c.charid
```

```
Hash Left Join  (cost=41.48..2623.78 rows=35465
width=416)
   Hash Cond: ((p.charid)::text =
(c.charid)::text)
   ->  Seq Scan on paragraph p
(cost=0.00..2094.65 rows=35465 width=361)
   ->  Hash  (cost=25.66..25.66 rows=1266
width=55)
         ->  Seq Scan on "character" c
(cost=0.00..25.66 rows=1266 width=55)
```

OTHER TYPES OF JOINS

- Hash Left Join, Hash Right Join, Merge Left Join, Merge Right Join, Nested Loop Left Join, Nested Loops Right Join...
- Semijoins, Antijoins, ...

AGGREGATIONS

HASHAGGREGATE

```
explain select department, count(*) from
documents group by department;


HashAggregate  (cost=18276.36..18277.14 rows=78
width=42)
   ->  Seq Scan on documents
(cost=0.00..16520.24 rows=351224 width=42)
```

D

A

B

A

C

D

B

HASH(COLUMN)

GROUP DATA

GROUP DATA

GROUP DATA

B

D

A

Z

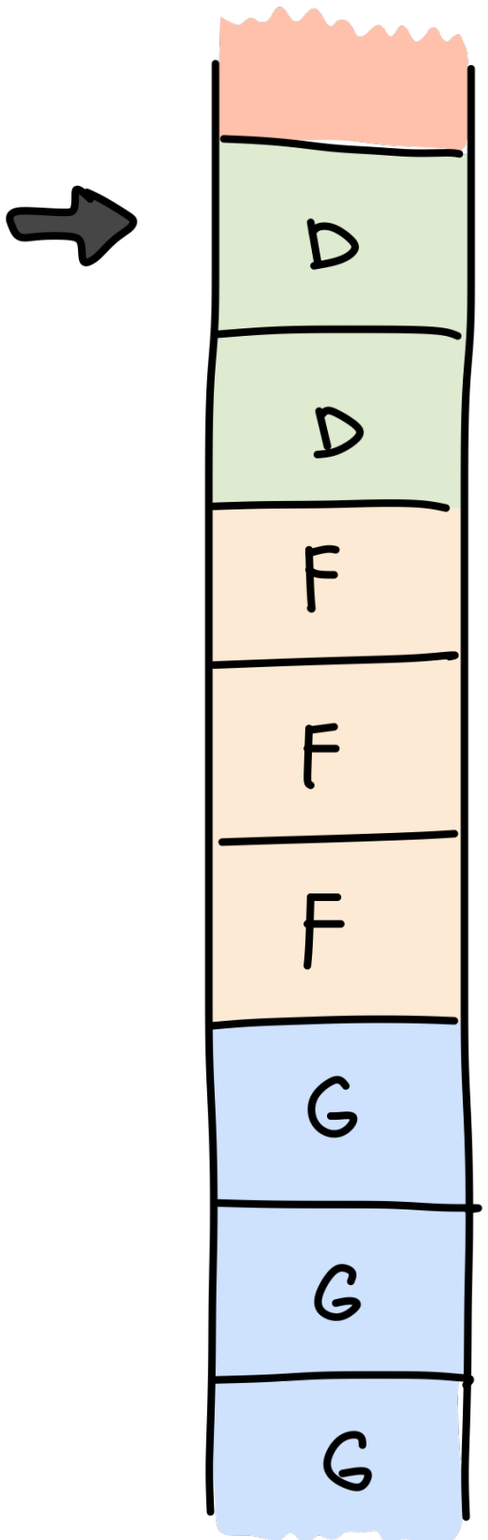
O

(COLLISIONS)

GROUP DAT

GROUP/GROUPAGGREGATE

```
create index index_documents_on_dc on
documents(department, customer); vaccum analyze;

explain select customer, count(*) from documents
where department = 'Rozhlas a televizia
Slovenska' group by customer;

GroupAggregate  (cost=0.55..1287.26 rows=335
width=41)

   ->  Index Only Scan using
index_documents_on_dc on documents
(cost=0.55..1200.50 rows=16683 width=41)

         Index Cond: (department = 'Rozhlas a
televizia Slovenska'::text)
```


D
D
F
F
F
G
G
G

SUMMARY (WHAT YOU SHOULD NOW)

- Understand the difference between 1 composite index and several single-column indices, draw a query plan for simple query with bitmap index scans (bitmap and/or).
- Understand why filtering on non-leading column in composite index is not efficient. Understand why like '%something%' is inefficient.
- Understand 3 join types and how they work. Estimate for a given query and schema, which join type would be most efficient.