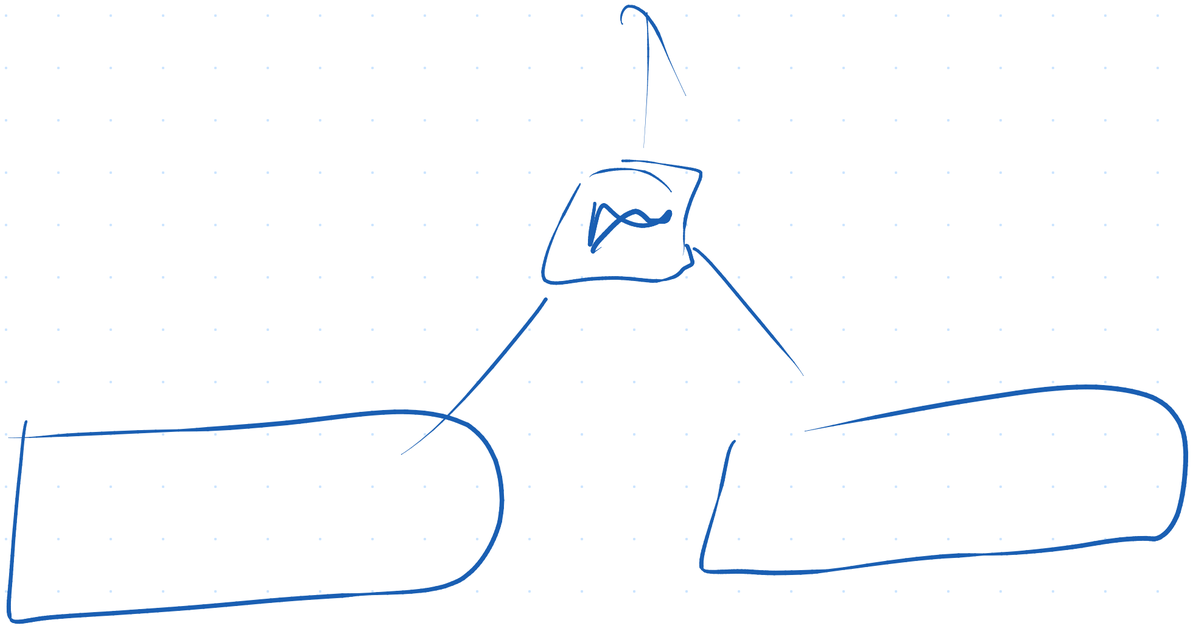


R	A	B	C
	1	2	3
	4	5	6
	7	8	9

SELECT A FROM R

↳ <1>

<4>

<7>

A B C

SELECT * FROM R

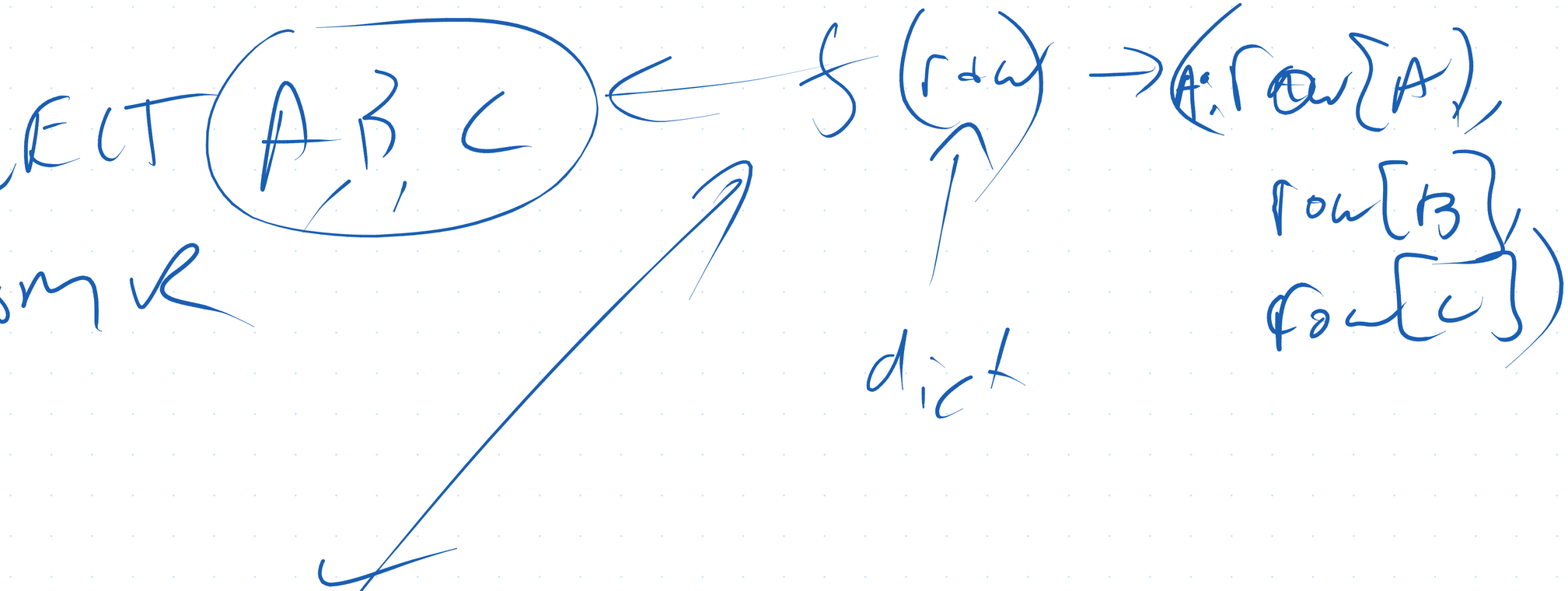
↳ <1 2 3>

<4 5 6>

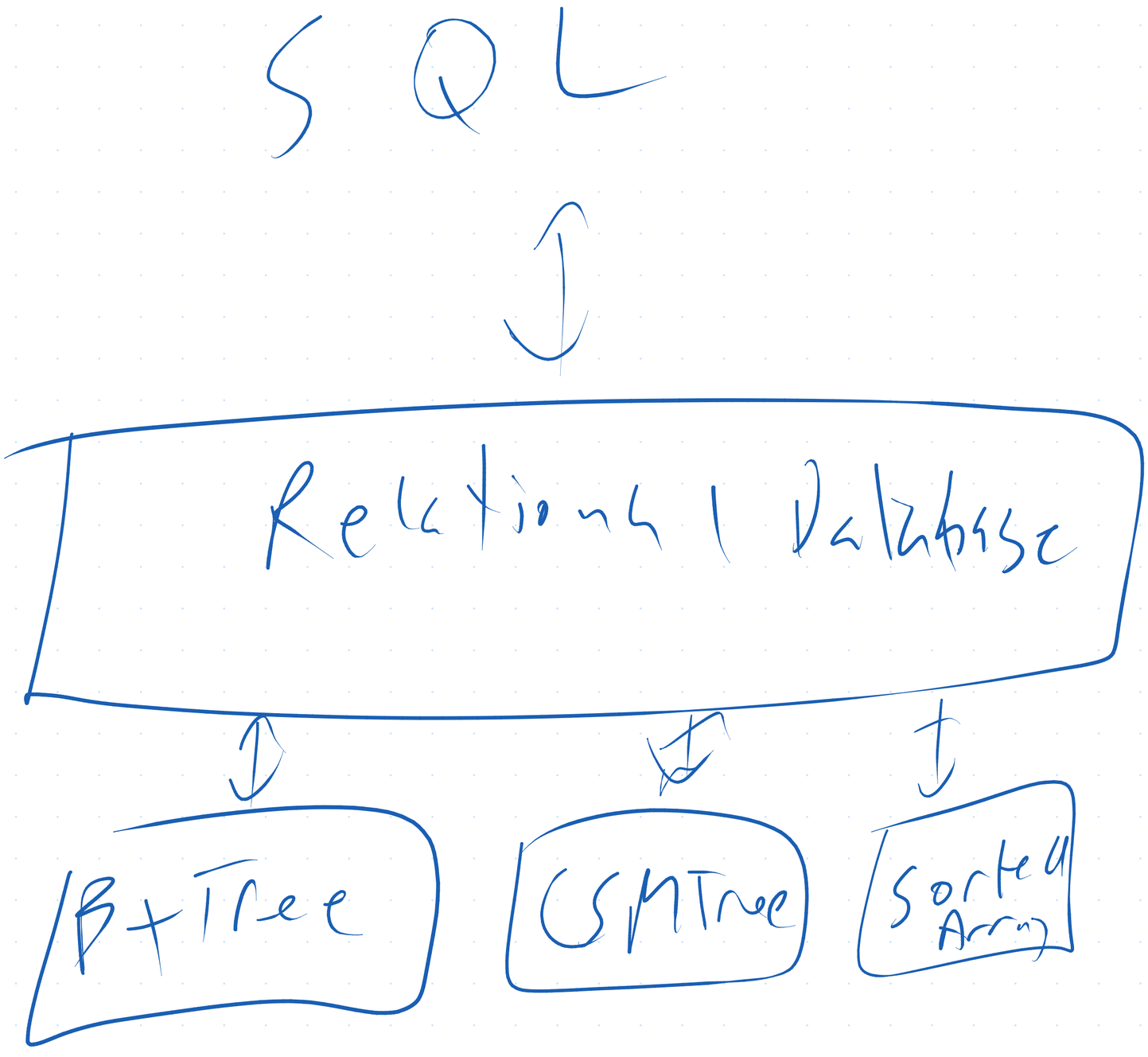
<7 8 9>

↳ Array<Row>

SELECT A, B, C
FROM R



[(foo.?) for foo in bar]


SQL
Relational Database
B+Tree
CSPTree
Sorted Array

```
struct
TransformIter {
    f: Fn(row) -> row,
    source: Iterator(Rows)
}
```

```
impl Iterator for TransformIter
{
    fn next() -> ro
        f(source.next())
}
```

get
put
delete
iterate
range

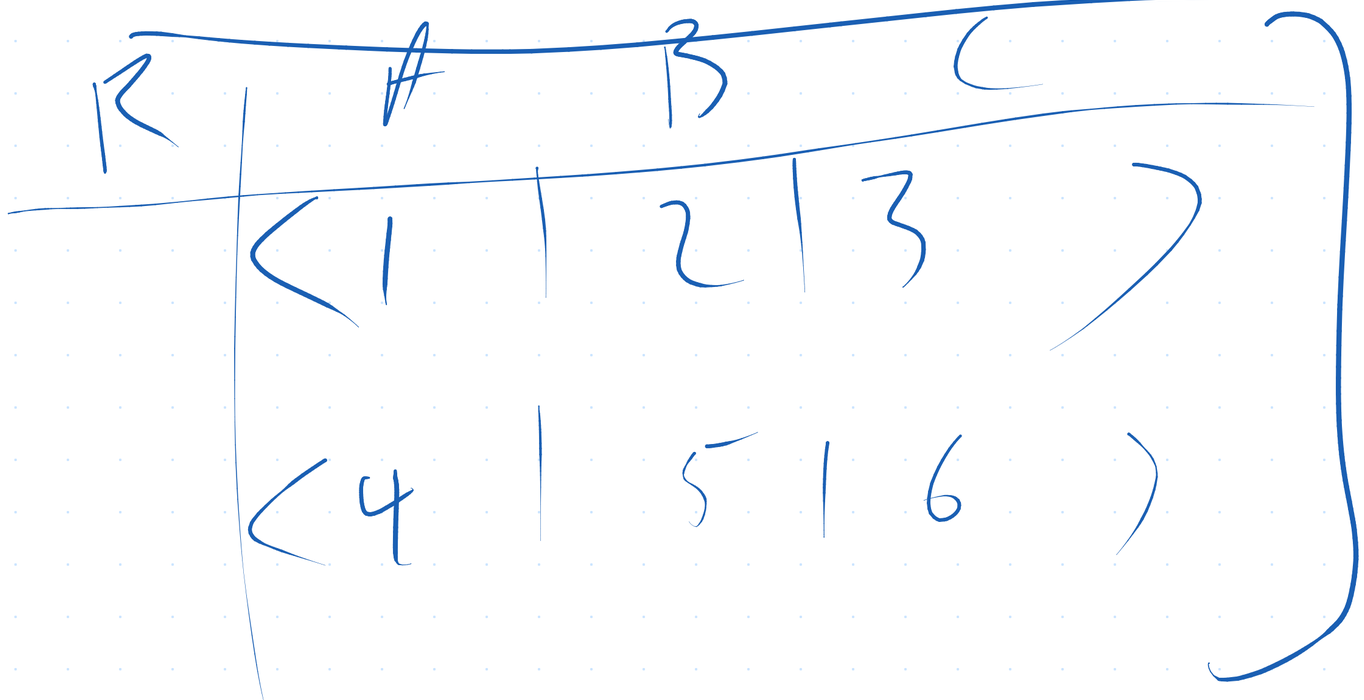
BTree

(K, V)

anything

anything
ordered

Key



Storage Flow

- BTree
- HashTable
- LSMTree
- Array (sorted, unique)

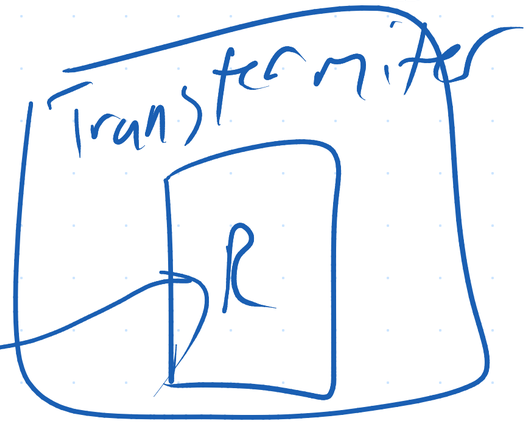
any combo of options

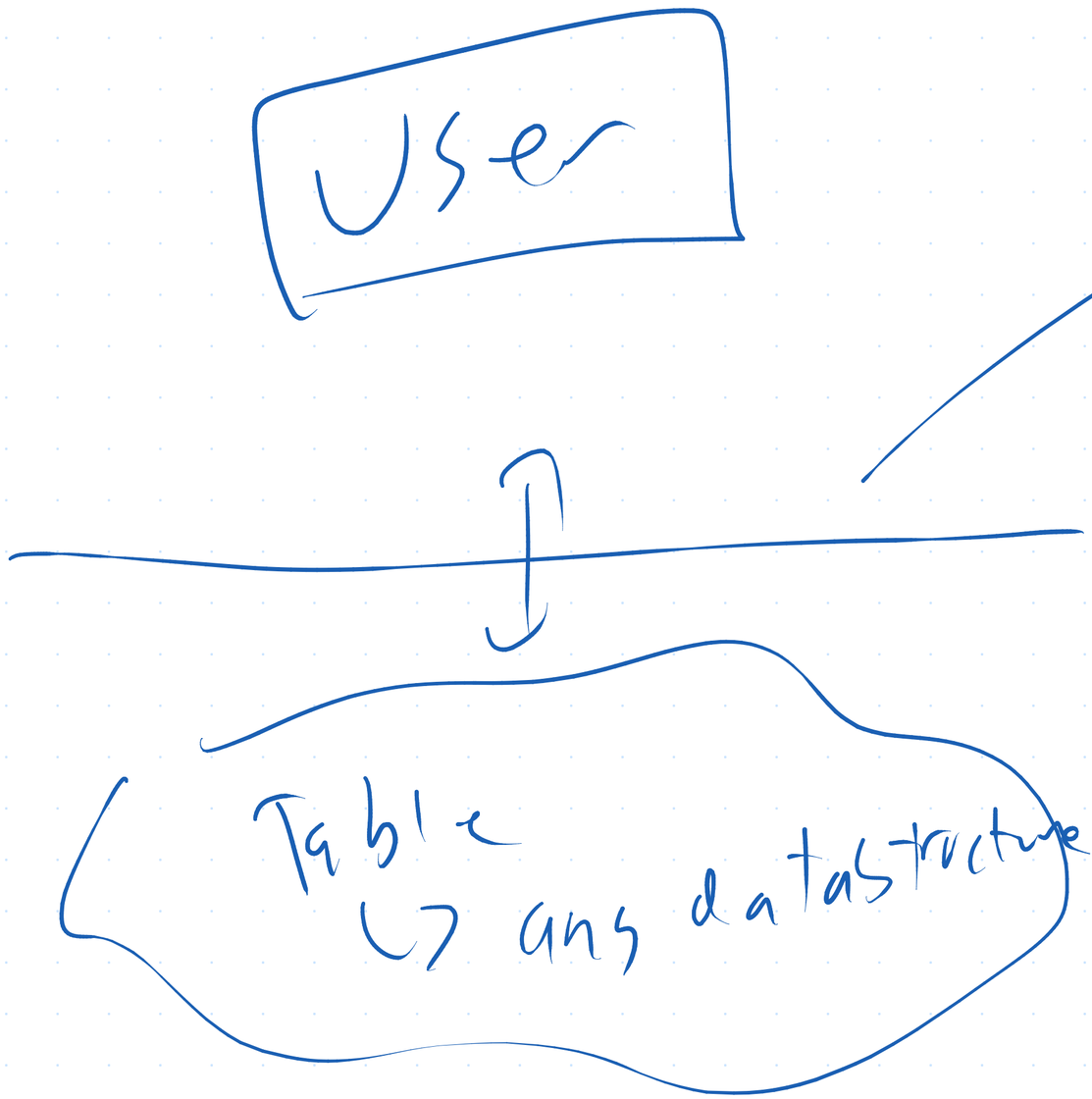
SELECT ABC

FROM R

WHERE $2.100

f(row) -> bool





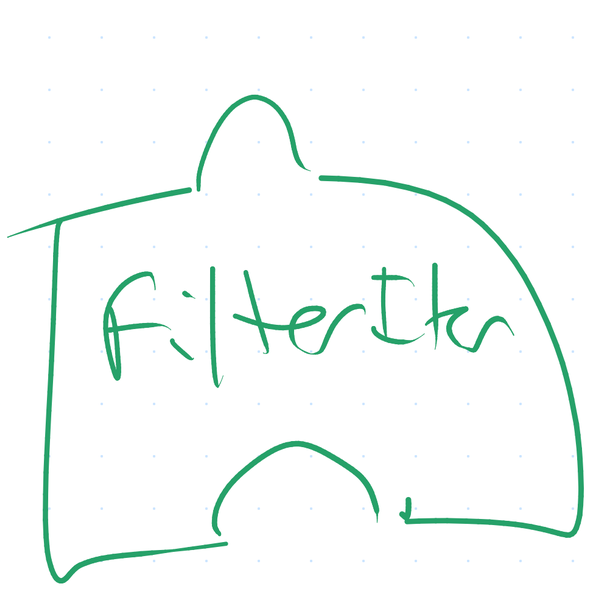
SELECT a
FROM b
WHERE c

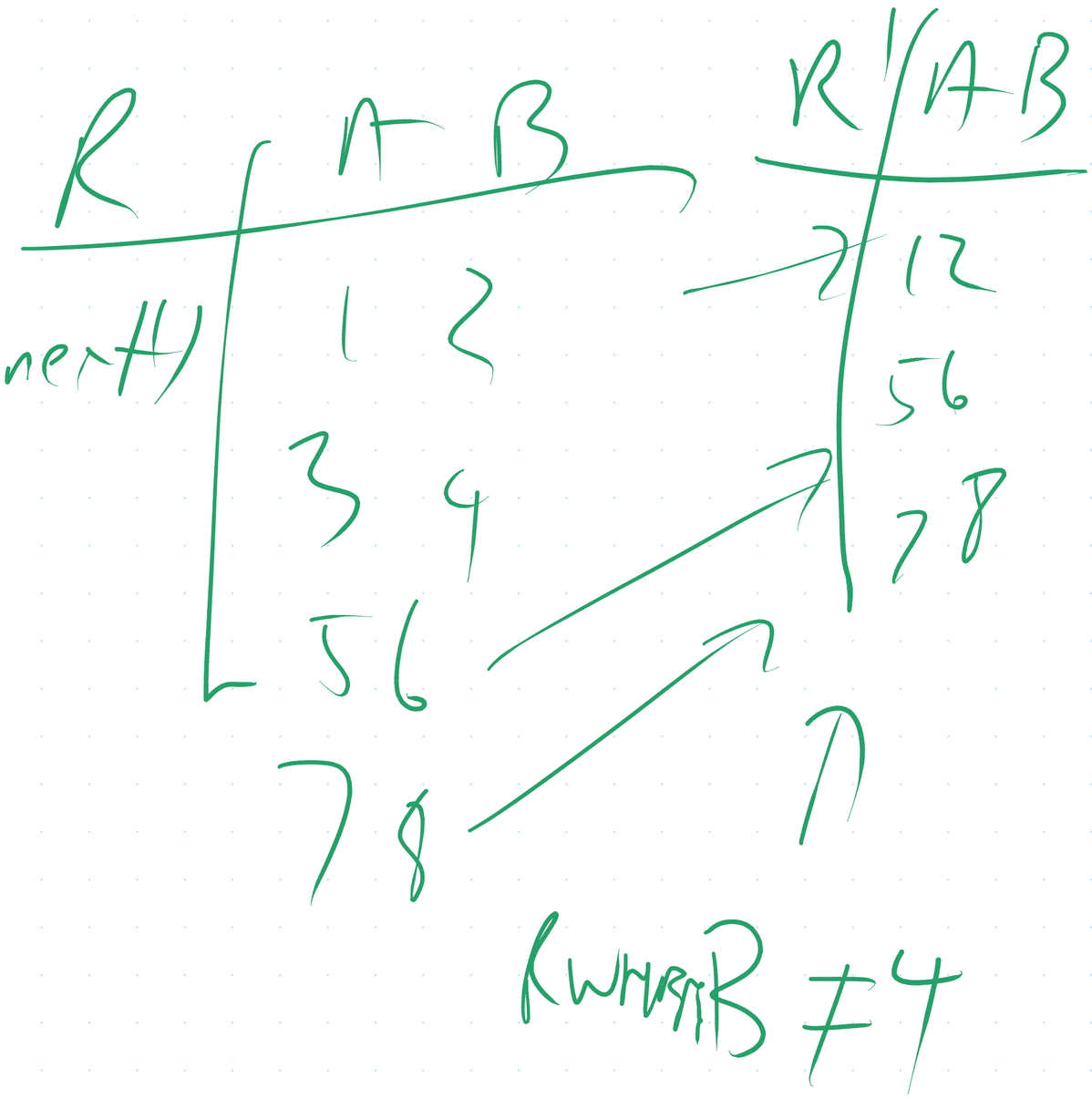
a → transformrows*

b → source data*

c → filter

```
fn next()
  while some(val) = source.next()
    if itest(val)
      return val
```





SELECT A, B, C
FROM R

→ Array(Row)

- create iterator for R
- call collect() → Vec<Row>

↓ iterator: Iterator(Row)

- B+Tree
- LSM Tree
- Array

→ Array of (Row)

TableIter

Filter

Transform

Transform

Filter

skip

SELECT - cols

FROM - R

WHERE

expr

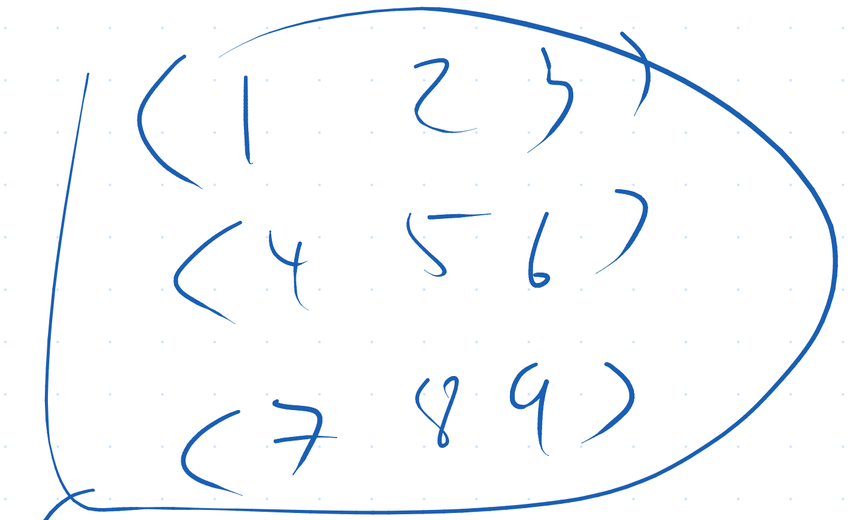
nowhere

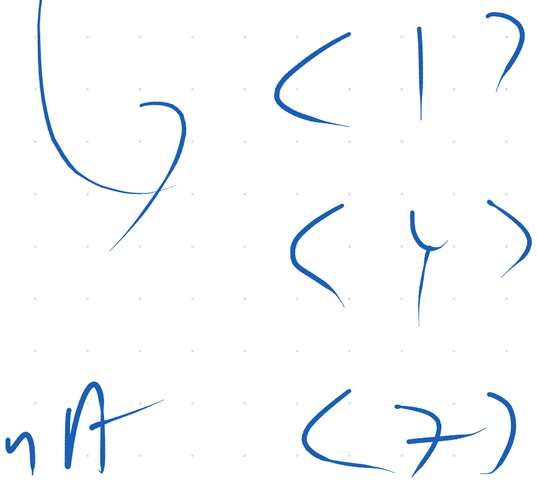
R

SELECT A FROM R

f(row) → row

one field: value from column A

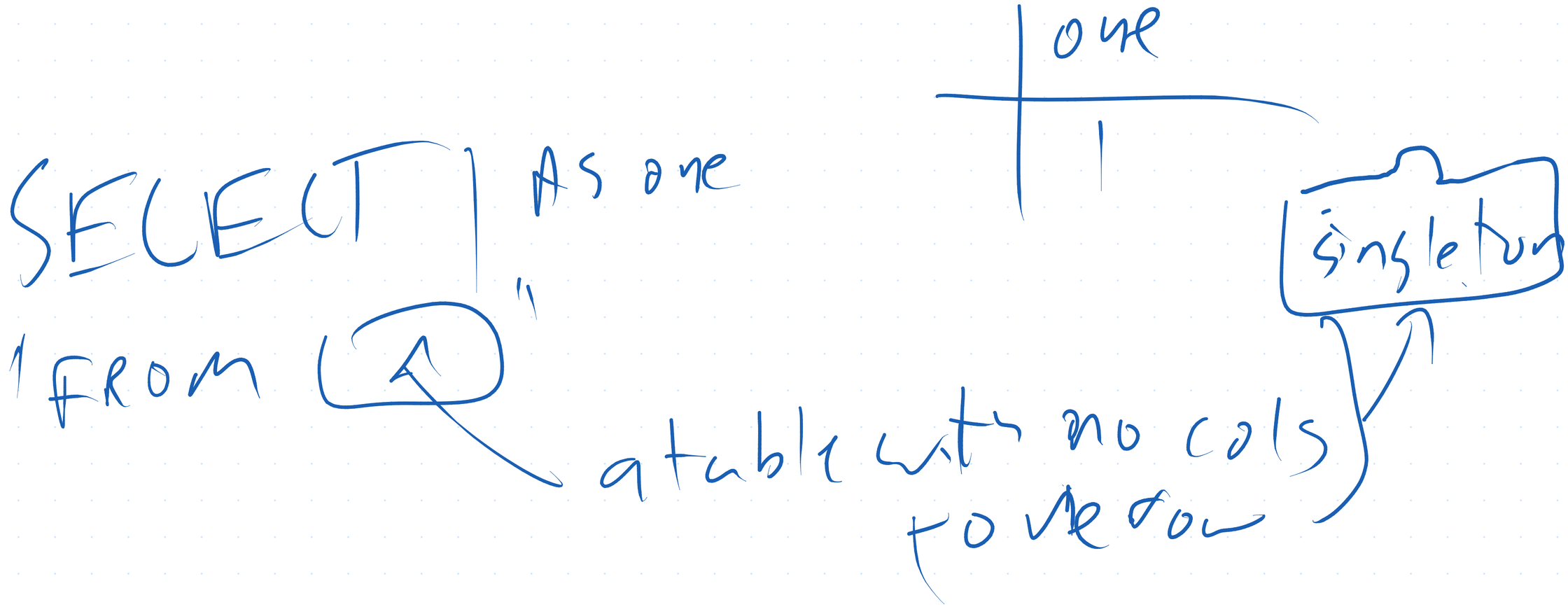




f((1,2,3)) → (1)

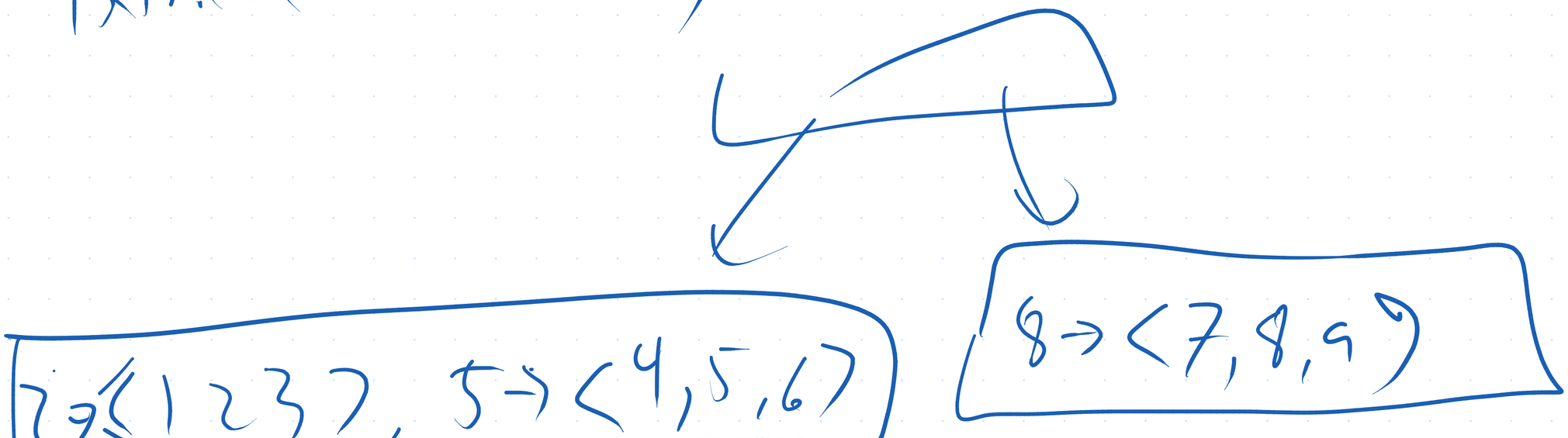
f((4,5,6)) → (4)

f((7,8,9) → (7)


SELECT
1
FROM
AS one
"
a table with no cols
to reason
one
1
singleton

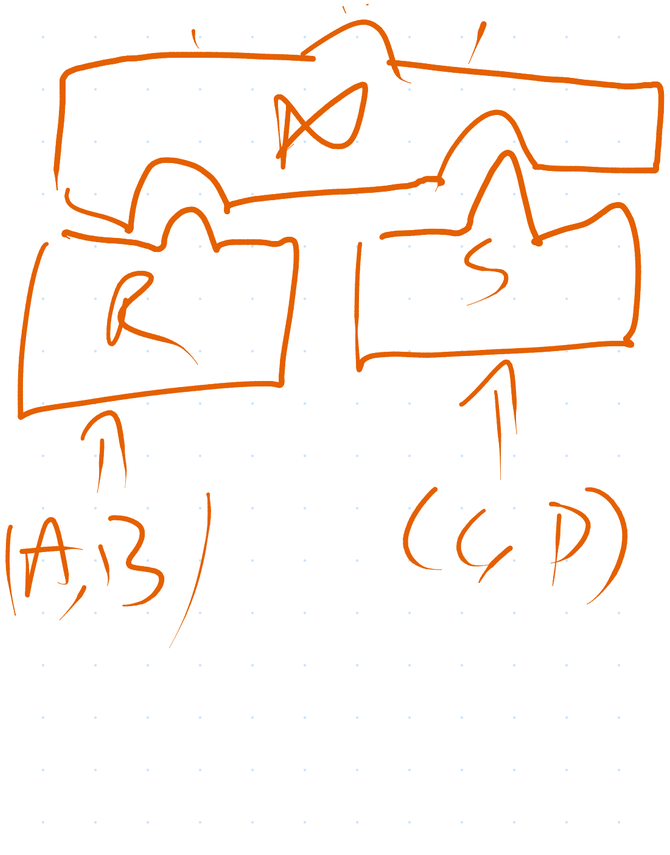
Option 1

Btree w R as key



Option 2 sorted array (on B)

(2,3),(4,5,6),(7,8,9)



SELECT *

FROM R JOIN S on

B = C

B C

Option1

Table

iterator

iteratorOverCols

iteratorWhere

iteratorWhereOverCols

iteratorAggregate

iteratorJoin

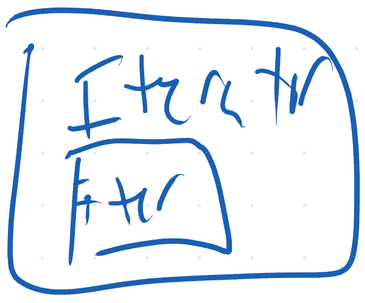
SELECT * FROM R

SELECT A, B FROM R

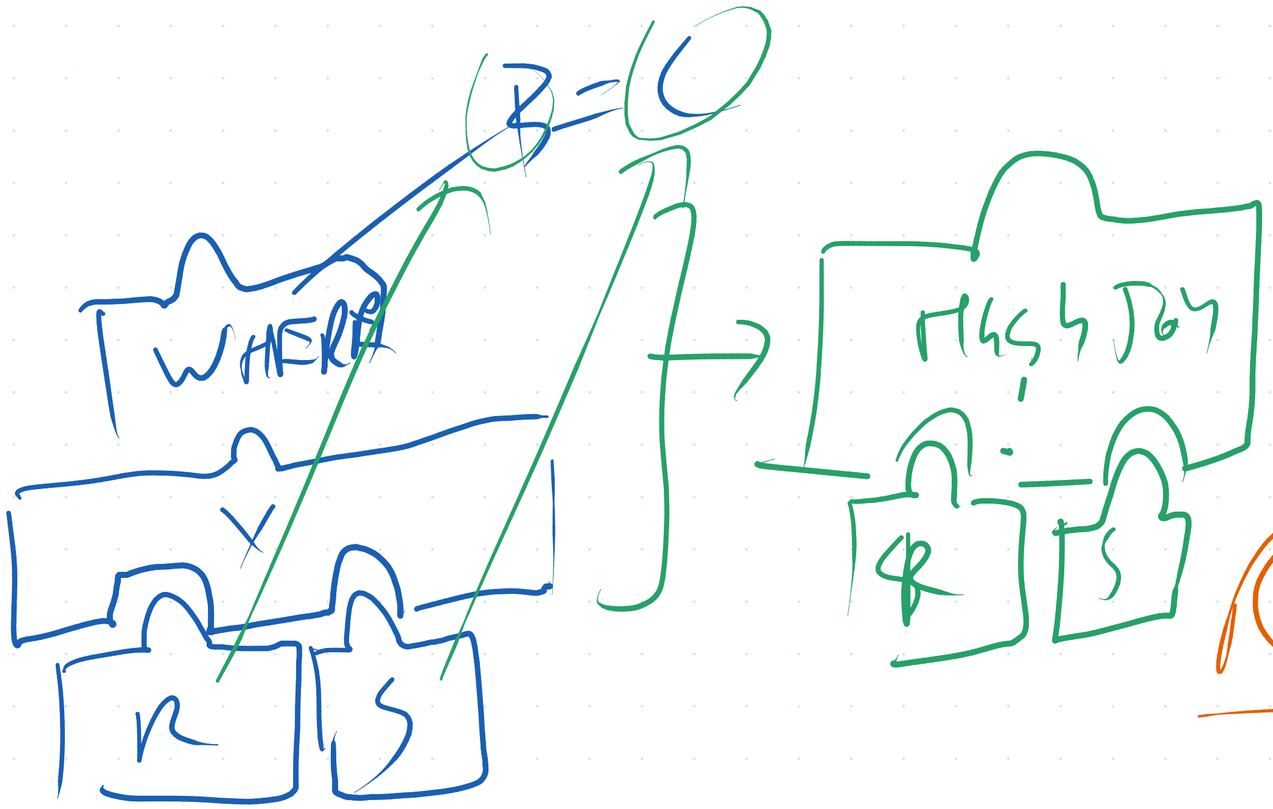
SELECT * FROM R WHERE

SELECT A, B FROM R WHERE

Option2



R JOIN S ON B=C



R	A	B
	1	2
	3	4

S	C	D
	2	7
	2	8
	5	9

R(x)S

A	B	C	D
1	2	2	7
1	2	2	8
1	2	x	5	9
3	4	x	2	7
3	4	x	2	8
3	4	x	5	9

SELECT (f) ← Apply f to every Row

```
struct TransformIterator {
    f: Fn(Row) -> row
    input: Iterator<Row>
}

fn next(self) -> Row
    self.f(self.input.next())
```

Must be =

must be from Diff tables

BinOp(
  Symbol('B')
  Equals
  Symbol('C')
)

Expression

TransformIter(<A>

R

R

↓ <1,2,3>

Transform

↓ <1>

collect

SELECT * FROM R → iterator over R

↓

SELECT A → TransformIterator