Be Inclusive: Welcome Non-key Columns in B-Tree Indexes

@MarkusWinand • @SQLPerfTips • @ModernSQL
Safe Harbour Statement ...
Instead of a **Safe Harbour Statement** ...
Instead of a Safe Harbour Statement ...

Take this Safe the Planet Statement:
I’m traveling a lot for business and always aim for the most environmental friendly way to do so.

If I cannot avoid flying, I offset its climate impact at a transparent climate protection company. (currently atmosfair.de)

Picture: https://upload.wikimedia.org/wikipedia/commons/2/2c/North_America_from_low_orbiting_satellite_Suomi_NPP.jpg
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
**B-tree Index:** A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...
B-tree Index: A Doubly-Linked List...

Logically ordered by KEY

Table (Heap)
B-tree Index: ... and a Tree
B-tree Index: ... and a Tree
B-tree Index: ... and a Tree
B-tree Index: ... and a Tree
B-tree Index: ... and a Tree
B-tree Index: ... and a Tree
CREATE INDEX ...
ON ...
( k )

B-tree Index: How it is used
B-tree Index: How it is used

CREATE INDEX ...
ON ... ( k )

SELECT data 
FROM ... 
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data 
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
( k )

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ... ON ... ( k )

SELECT data FROM ... WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ...
ON ...
(k)

SELECT data
FROM ...
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX ... 
ON ... 
( k )

SELECT data 
FROM ... 
WHERE k = $1
B-tree Index: How it is used

CREATE INDEX...
ON ...
( k )

1. Use tree
2. Use doubly linked list
3. Access Table
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
(k)

1. Use tree
2. Use doubly linked list
3. Access Table
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
( k, data )

1. Use tree
2. Use doubly linked list
3. Access Table
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
( k, data )

1. Use tree
2. Use doubly linked list
3. Access Table

No Heap Access
(if visibility map is clear)
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
(k, data)

1. Use tree
2. Use doubly linked list
3. Access Table

No Heap Access
(if visibility map is clear)
CREATE INDEX ... 
ON ... 
( k, data )

Visibility information is only stored in table

1. Use tree
2. Use doubly linked list
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ... 
ON ... 
( k, data )

1. Use tree
2. Use doubly linked list

Block-level info is in the Visibility Map (VM) (32k times smaller)

Visibility information is only stored in table

VM
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
( k, data )

1. Use tree
2. Use doubly linked list
3. Check visibility in VM

Visibility information is only stored in table

Block-level info is in the Visibility Map (VM)
(32k times smaller)
CREATE INDEX ...  
ON ...  
( k, data )

1. Use tree
2. Use doubly linked list
3. Check visibility in VM
4. Check visibility in Table

Block-level info is in the Visibility Map (VM)
(32k times smaller)

Visibility information is only stored in table

Table (Heap)

VM
B-tree Index: Index Only Scan (since 9.2)

CREATE INDEX ...
ON ...
( k, data )

1. Use tree
2. Use doubly linked list
3. Check visibility in VM
4. Check visibility in Table

Visibility information is only stored in table
Block-level info is in the Visibility Map (VM)
(32k times smaller)

Index Only Scan using ...
Heap Fetches: 1
B-tree Index: **INCLUDE** (since 11)
B-tree Index: **INCLUDE** (since 11)

```sql
CREATE INDEX ...
ON ...
  ( k )
INCLUDE (data)
```
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ...  
ON ...  
( k )  
**INCLUDE** (data)
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ...
ON ... 
( k )
INCLUDE (data)
B-tree Index: **INCLUDE** (since 11)

```
CREATE INDEX ... 
ON ...
(k)
INCLUDE (data)
```
B-tree Index: INCLUDE (since 11)

CREATE INDEX ... ON ... ( k ) INCLUDE (data)
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ... ON ...

( k )

**INCLUDE** (data)
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ...
  ON ...
  ( k )
**INCLUDE** (data)

SELECT data
  FROM ...
  WHERE k = $1
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ... 
ON ... 
( k ) 
INCLUDE (data)

SELECT data 
FROM ... 
WHERE k = $1

Table 
(Heap)
B-tree Index: INCLUDE (since 11)

CREATE INDEX ...
ON ...
( k )
INCLUDE (data)

SELECT data
FROM ...
WHERE k = $1
B-tree Index: INCLUDE (since 11)

CREATE INDEX ...
ON ...
( k )
INCLUDE (data)

SELECT data
FROM ...
WHERE k = $1

Table (Heap)
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ... 
ON ... 
( k )
INCLUDE (data)

SELECT data FROM ... 
WHERE k = $1

Table (Heap)
INCLUDE — Pro and Con vs. Key-Columns
Advantages:
Advantages:

- Shallower: < ~40%
Advantages:

- Shallower: < ~40%
- Slightly smaller: < ~3%
Advantages:
¬ Shallower: < ~40%
¬ Slightly smaller: < ~3%

Disadvantages:
¬ …
INCLUDE — Disadvantages: WHERE

SELECT * FROM ... WHERE k = $1 AND data = $2
INCLUDE — Disadvantages: WHERE

SELECT * FROM ... WHERE k = $1 AND data = $2

CREATE INDEX ...
ON ...
( k, data )

QUERY PLAN

Index Scan using ... on ... (actual rows=1)
Index Cond: ((key = $1) AND (data = $2))
Buffers: shared hit=5
INCLUDE — Disadvantages: WHERE

```
SELECT * FROM ... WHERE k = $1 AND data = $2
```

**CREATE INDEX ...**

**ON ...**

```
(k, data)
```

**QUERY PLAN**

Index Scan using ... on ... (actual rows=1)
Index Cond: (key = $1) AND (data = $2)
Buffers: shared hit=5

**CREATE INDEX ...**

**ON ... ( k )**

**INCLUDE ( data )**

**QUERY PLAN**

Bitmap Heap Scan on ... (actual rows=1)
Recheck Cond: (key = 123)
Filter: (data = $2)
Rows Removed by Filter: 9999
**Heap Blocks: exact=10000**
Buffers: shared hit=2186 read=7867
-> Bitmap Index Scan on ... (actual rows=10000)
   Index Cond: (key = 123)
   Buffers: shared read=53
**INCLUDE — Disadvantages: WHERE**

**SELECT * FROM ... WHERE k = $1 AND data = $2**

**CREATE INDEX ...**
ON ...
(k, data)

**QUERY PLAN**

Index Scan using ... on ... (actual rows=1)
Index Cond: ((key = $1) AND (data = $2))
Buffers: shared hit=5

**CREATE INDEX ...**
ON ... ( k )
INCLUDE ( data )

**QUERY PLAN**

Bitmap Heap Scan on ... (actual rows=1)
Recheck Cond: (key = 123)
Filter: (data = $2)
Rows Removed by Filter: 9999
Heap Blocks: exact=10000
Buffers: shared hit=2186 read=7867

-> Bitmap Index Scan on ... (actual rows=10000)
Index Cond: (key = 123)
Buffers: shared read=53

*Doesn't Filter on INCLUDE column*
B-tree Index: INCLUDE (since 11)

CREATE INDEX ...
ON ...
(k)
INCLUDE (data)

SELECT *
FROM ...
WHERE k = $1
AND data = $2
B-tree Index: **INCLUDE** (since 11)

CREATE INDEX ... ON ... ( k ) INCLUDE (data)

SELECT * FROM ... WHERE k = $1 AND data = $2

Table (Heap)
B-tree Index: INCLUDE (since 11)

CREATE INDEX ... 
ON ... 
( k ) 
INCLUDE (data)

SELECT * 
FROM ... 
WHERE k = $1 
AND data = $2

Table
(Heap)
B-tree Index: INCLUDE (since 11)

CREATE INDEX ... OF ... ( k ) INCLUDE (data)

SELECT * FROM ... WHERE k = $1 AND data = $2
**B-tree Index: INCLUDE (since 11)**

**CREATE INDEX ...**
**ON ...**
( k )
**INCLUDE (data)**

**SELECT * FROM ...**
WHERE k = $1
AND data = $2

---

**Table (Heap)**
### B-tree Index: **INCLUDE** (since 11)

#### CREATE INDEX ...

- **ON** ...
- *(k)*
- **INCLUDE** *(data)*

**SELECT** *

FROM ...

WHERE k = $1

AND data = $2

---

**Table access not reduced**

**Ignored (unknown if visible)**
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters
   (based on the operator class)
4. Check visibility
   (in VM and/or in table)
5. Apply remaining filters
B-tree Index: **INCLUDE** (since 11)

**Operations**

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters (based on the operator class)
4. Check visibility (in VM and/or in table)
5. Apply remaining filters
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters
   (based on the operator class)
4. Check visibility
   (in VM and/or in table)
5. Apply remaining filters

**Operations**

**Safe**
(in b-tree key: <=, <, =, >, >=)
### B-tree Index: INCLUDE (since 11)

<table>
<thead>
<tr>
<th></th>
<th>Safe</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>(in b-tree key: (\leq, &lt;, =, &gt;, \geq))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters
   (based on the operator class)
4. Check visibility
   (in VM and/or in table)
5. Apply remaining filters
### B-tree Index: INCLUDE (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters (based on the operator class)
4. Check visibility (in VM and/or in table)
5. Apply remaining filters

<table>
<thead>
<tr>
<th>Columns</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
<td>(in b-tree key: &lt;=, &lt;, =, &gt;, &gt;=)</td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters  
   (based on the operator class)
4. Check visibility  
   (in VM and/or in table)
5. Apply remaining filters

<table>
<thead>
<tr>
<th>Operations</th>
<th>Safe (in b-tree key: &lt;=, &lt;, =, &gt;, &gt;=)</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>Key</td>
<td></td>
</tr>
</tbody>
</table>
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters  
   (based on the operator class)
4. Check visibility  
   (in VM and/or in table)
5. Apply remaining filters

<table>
<thead>
<tr>
<th>Columns</th>
<th>Operations</th>
</tr>
</thead>
</table>
| Key     | Safe  
(in b-tree key: <=, <, =, >, >=) | Other |
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters  
   (based on the operator class)
4. Check visibility  
   (in VM and/or in table)
5. Apply remaining filters

<table>
<thead>
<tr>
<th>Columns</th>
<th>Key</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safe</td>
<td>✓</td>
<td>(in b-tree key: (\leq, &lt;, =, &gt;, \geq))</td>
</tr>
<tr>
<td>Other</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

Safe (in b-tree key: \(\leq, <, =, >, \geq\))
1. Use tree
2. Use doubly linked list
3. Apply “safe” filters (based on the operator class)
4. Check visibility (in VM and/or in table)
5. Apply remaining filters

B-tree Index: **INCLUDE** (since 11)
B-tree Index: **INCLUDE** (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters (based on the operator class)
4. Check visibility (in VM and/or in table)
5. Apply remaining filters

<table>
<thead>
<tr>
<th>Operations (in b-tree key: &lt;=, &lt;, =, &gt;, &gt;=)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Safe</strong></td>
</tr>
<tr>
<td><strong>Other</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Columns</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
**B-tree Index:** INCLUDE (since 11)

1. Use tree
2. Use doubly linked list
3. Apply “safe” filters (based on the operator class)
4. Check visibility (in VM and/or in table)
5. Apply remaining filters

### Columns

<table>
<thead>
<tr>
<th></th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCLUDE</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Operations

<table>
<thead>
<tr>
<th>Safe (in b-tree key: &lt;=, &lt;, =, &gt;, &gt;=)</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Not applicable: INCLUDE columns have no op class
SELECT data FROM ... WHERE k = $1 AND data = $2
INCLUDE — Disadvantages: WHERE

SELECT data FROM ... WHERE k = $1 AND data = $2
INCLUDE — Disadvantages: WHERE

SELECT data FROM ... WHERE k = $1 AND data = $2

Safe in B-Tree Key
INCLUDE — Disadvantages: WHERE

SELECT data FROM ... WHERE k = $1 AND data = $2

CREATE INDEX ...
ON ...
( k, data )

QUERY PLAN

Index Only Scan using ... on ... (actual rows=1)
Index Cond: (key = $1) AND (data =$2)
Heap Fetches: 0
Buffers: shared hit=5

Safe in B-Tree Key
INCLUDE — Disadvantages: WHERE

SELECT data FROM ... WHERE k = $1 AND data = $2

CREATE INDEX ...
ON ...
( k, data )

QUERY PLAN
Index Only Scan using ... on ... (actual rows=1)
Index Cond: (key = $1) AND (data =$2)
Heap Fetches: 0
Buffers: shared hit=5

CREATE INDEX ...
ON ... ( k )
INCLUDE ( data )

QUERY PLAN
Index Only Scan using ... on ... (actual rows=1)
Index Cond: (key = $1)
Filter: (data = $2)
Rows Removed by Filter: 9999
Heap Fetches: 0
Buffers: shared hit=56
Advantages:

› Shallower: < ~40%

› Slightly smaller: < ~3%

Disadvantages:
INCLUDE — Pro and Con vs. Key-Columns

Advantages:
- Shallower: < ~40%
- Slightly smaller: < ~3%

Disadvantages:
- No safe WHERE conditions
  (visibility always checked first)
INCLUDE — Disadvantages: ORDER BY

SELECT * FROM ... WHERE k = $1 ORDER BY data LIMIT 1
CREATE INDEX ...  
ON ...  
( k, data )  

SELECT * FROM ... WHERE k = $1 ORDER BY data LIMIT 1
INCLUDE — Disadvantages: ORDER BY

SELECT * FROM ... WHERE k = $1 ORDER BY data LIMIT 1

CREATE INDEX ...
ON ...
  ( k, data )

QUERY PLAN

Limit (actual rows=1)
Buffers: shared hit=5
-> Index Scan using ... (actual rows=1)
  Index Cond: (key = 123)
  Heap Fetches: 0
  Buffers: shared hit=5

No Sort
INCLUDE — Disadvantages: ORDER BY

SELECT * FROM ... WHERE k = $1 ORDER BY data LIMIT 1

CREATE INDEX ...
ON ...
(k, data)

No Sort

CREATE INDEX ...
ON ...
(k)
INCLUDE (data)

Limit (actual rows=1)
Buffers: shared hit=5
-> Sort (actual rows=1)
  -> Bitmap Heap Scan on...(actual rows=10000)
    Recheck Cond: (key = 123)
    Heap Blocks: exact=10000
    Buffers: shared hit=10053
  -> Bitmap Index Scan on...(act rows=10000)
    Index Cond: (key = 123)
    Buffers: shared hit=53
Advantages:

- Shallower: < ~40%
- Slightly smaller: < ~3%

Disadvantages:

- Doesn’t help WHERE (except Index Only Scan)
INCLUDE — Pro and Con vs. Key-Columns

**Advantages:**
- Shallower: < ~40%
- Slightly smaller: < ~3%

**Disadvantages:**
- Doesn’t help WHERE (except Index Only Scan)
- Cannot replace sorting (ORDER BY)
INCLUDE — Differences: Constraints
ALTER TABLE ... 
ADD PRIMARY KEY|UNIQUE
(key, data)
<table>
<thead>
<tr>
<th>ALTER TABLE ...</th>
<th>INSERT INTO ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD PRIMARY KEY</td>
<td>VALUES (1, 1)</td>
</tr>
<tr>
<td></td>
<td>, (1, 2)</td>
</tr>
<tr>
<td>UNIQUE</td>
<td></td>
</tr>
</tbody>
</table>
**INCLUDE — Differences: Constraints**

<table>
<thead>
<tr>
<th>ALTER TABLE ...</th>
<th>INSERT INTO ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD PRIMARY KEY</td>
<td>VALUES (1, 1)</td>
</tr>
</tbody>
</table>
| UNIQUE (key, data) | , (1, 2)✓

ALTER TABLE ...  
ADD PRIMARY KEY|UNIQUE (key)
INCLUDE(data)

✓
# INCLUDE — Differences: Constraints

<table>
<thead>
<tr>
<th>ALTER TABLE ...</th>
<th>INSERT INTO ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD PRIMARY KEY</td>
<td>UNIQUE (key, data)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALTER TABLE ...</th>
<th>INSERT INTO ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD PRIMARY KEY</td>
<td>UNIQUE (key) INCLUDE(data)</td>
</tr>
</tbody>
</table>
INCLUDE — Pro and Con vs. Key-Columns

Advantages:

‣ Shallower: < ~40%

‣ Slightly smaller: < ~3%

Disadvantages:

‣ Doesn’t help WHERE
  (except Index Only Scan)

‣ Cannot replace sorting
  (ORDER BY)

Differences:

PRIMARY KEY / UNIQUE don’t take
INCLUDE columns into account
INCLUDE and the Three Powers Of B-tree Indexes

https://use-the-index-luke.com/
INCLUDE and the Three Powers Of B-tree Indexes

- Finding data quickly

https://use-the-index-luke.com/
**INCLUDE** and the Three Powers Of B-tree Indexes

Finding data quickly

https://use-the-index-luke.com/
INCLUDE and the Three Powers Of B-tree Indexes

- Finding data quickly
- Clustering data
  (Index Only Scan)

https://use-the-index-luke.com/
INCLUDE and the Three Powers Of B-tree Indexes

- Finding data quickly
- Clustering data
  (Index Only Scan)

https://use-the-index-luke.com/
INCLUDE and the Three Powers Of B-tree Indexes

- Finding data quickly
- Clustering data *(Index Only Scan)*
- Sorting data

https://use-the-index-luke.com/
INCLUDE and the Three Powers Of B-tree Indexes

✗ Finding data quickly
 ✓ Clustering data (Index Only Scan)
✗ Sorting data

https://use-the-index-luke.com/
postgres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)
postgres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)

SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1
postgres=# \d ...  
... [ columns skipped ] ... 
Indexes:  
"..." btree (key, data) 

Three options:

```
SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1
```
postgres=# \d ... ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)

Three options:
- Add TS as last column
  ➔ Not useful for ORDER BY/LIMIT

SELECT data
  FROM ...
  WHERE key = $1
  ORDER BY ts DESC
  LIMIT 1
PostgreSQL=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)

Three options:
- Add TS as last column ➔ Not useful for ORDER BY/LIMIT
- Add TS after KEY ➔ Might break other queries

```
SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1
```
postgres=# \d ...  
... [ columns skipped ] ...  
Indexes:  
"..." btree (key, data)  

SELECT data  
FROM ...  
WHERE key = $1  
ORDER BY ts DESC  
LIMIT 1  

Three options:  
› Add TS as last column  
  ⇒ Not useful for ORDER BY/LIMIT  
› Add TS after KEY  
  ⇒ Might break other queries  

WHERE key = $1  
AND data = $2
postgres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)

SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1

Three options:
- Add TS as last column
  - Not useful for ORDER BY/LIMIT
- Add TS after KEY
  - Might break other queries

WHERE key = $1 AND data = $2
WHERE key = $1
ORDER BY data
LIMIT 1
postres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key, data)

SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1

Three options:
- Add TS as last column
  ⇒ Not useful for ORDER BY/LIMIT
- Add TS after KEY
  ⇒ Might break other queries
- Create new index (key, ts, data)
  ⇒ Adds considerable overhead
INCLUDE — Advantage: Documentation

postgres=# \d …
... [ columns skipped ] ...
Indexes:
"..." btree (key) INCLUDE(data)

SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1

Three options:
› Add TS as last column
  ⇒ Not useful for ORDER BY/LIMIT
› Add TS after KEY
  ⇒ Might break other queries
› Create new index (key, ts, data)
  ⇒ Adds considerable overhead
postgres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key) INCLUDE(data)

SELECT data
FROM ...
WHERE key = $1
ORDER BY ts DESC
LIMIT 1

Three options:
- Add TS as last column
  ⇒ Not useful for ORDER BY/LIMIT
- Add TS after KEY
  ⇒ Might break other queries
- Create new index (key, ts, data)
  ⇒ Adds considerable overhead
postgres=# \d ...
... [ columns skipped ] ...
Indexes:
"..." btree (key) INCLUDE(data)

Three options:
- Add TS as last column
  → Not useful for ORDER BY/LIMIT
- Add TS after KEY
  → Might break other queries
- Create new index (key, ts, data)
  → Adds considerable overhead
INCLUDE — Pro and Con vs. Key-Columns

Advantages:
- Shallower: < ~40%
- Slightly smaller: < ~3%

Disadvantages:
- Doesn’t help WHERE (except Index Only Scan)
- Cannot replace sorting (ORDER BY)

Differences:
PRIMARY KEY / UNIQUE don’t take INCLUDE columns into account
Advantages:

‣ Shallower: < ~40%
‣ Slightly smaller: < ~3%
‣ Documents its purpose

Disadvantages:

‣ Doesn’t help WHERE (except Index Only Scan)
‣ Cannot replace sorting (ORDER BY)

Differences:

PRIMARY KEY / UNIQUE don’t take INCLUDE columns into account
Concluding on **INCLUDE**

**INCLUDE** columns are **ignored** except by **Index Only Scan**
Concluding on **INCLUDE**

**INCLUDE** columns are ignored except by Index Only Scan

- Don’t move key columns to **INCLUDE** unless you are *really* sure
Concluding on **INCLUDE**

**INCLUDE** columns are **ignored** except by **Index Only Scan**

- Don’t move key columns to **INCLUDE** unless you are *really* sure
- If you add columns to enable an **Index Only Scan**, put them into **INCLUDE**
  - Also for PK/Unique!
Concluding on **INCLUDE**

**INCLUDE** columns are **ignored** except by **Index Only Scan**

- Don’t move key columns to **INCLUDE** unless you are *really* sure

- If you add columns to enable an **Index Only Scan**, put them into **INCLUDE**
  - Also for PK/Unique!

---

Training in Vienna: March (German), September (English) 2020.
Inhouse: whenever you want.

- use-the-index-luke.com
- winand.at
- modern-sql.com
Be Inclusive:
Welcome Non-key Columns in B-Tree Indexes
https://2019.pgconf.eu/f
https://use-the-index-luke.com/blog/2019-04/include-columns-in-btree-indexes

Training in Vienna: March (German), September (English) 2020.
Inhouse: whenever you want.