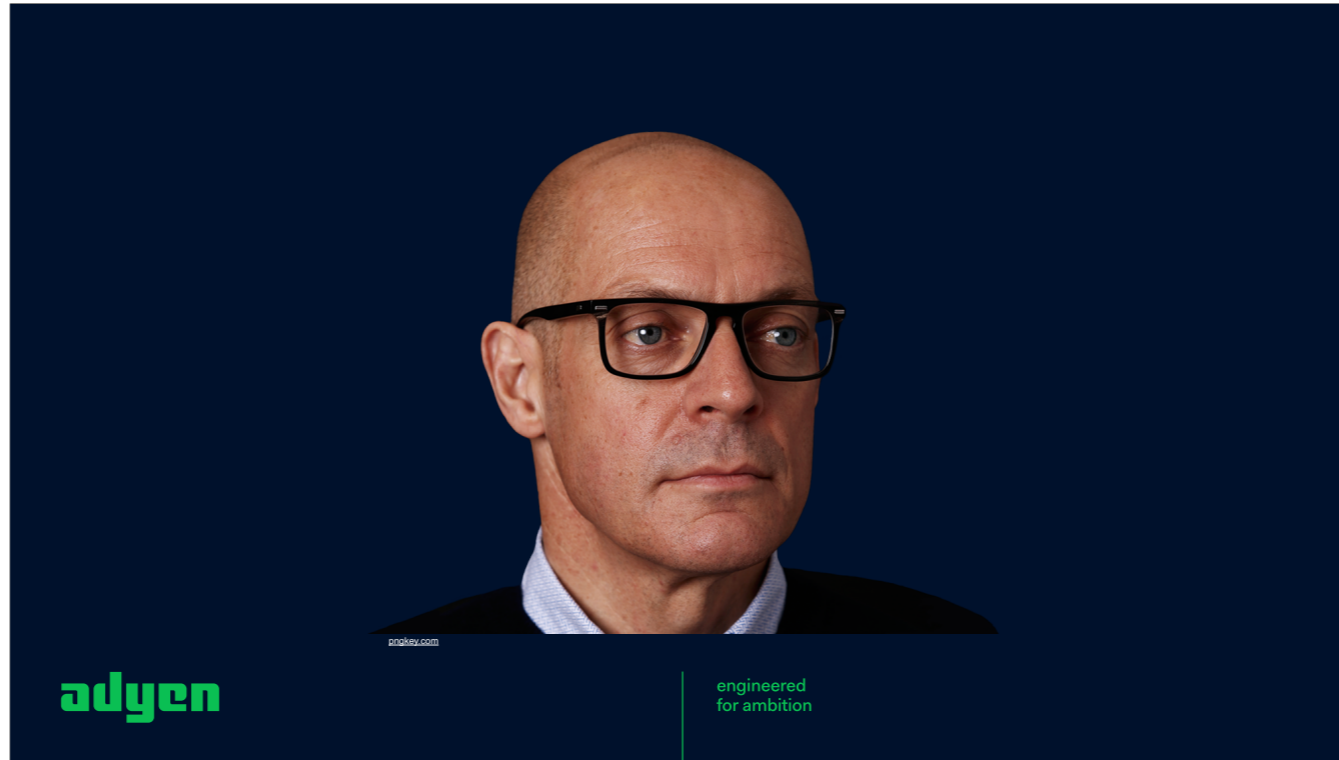


# Adding **small gains** for maximum results

**adyen**

engineered  
for ambition



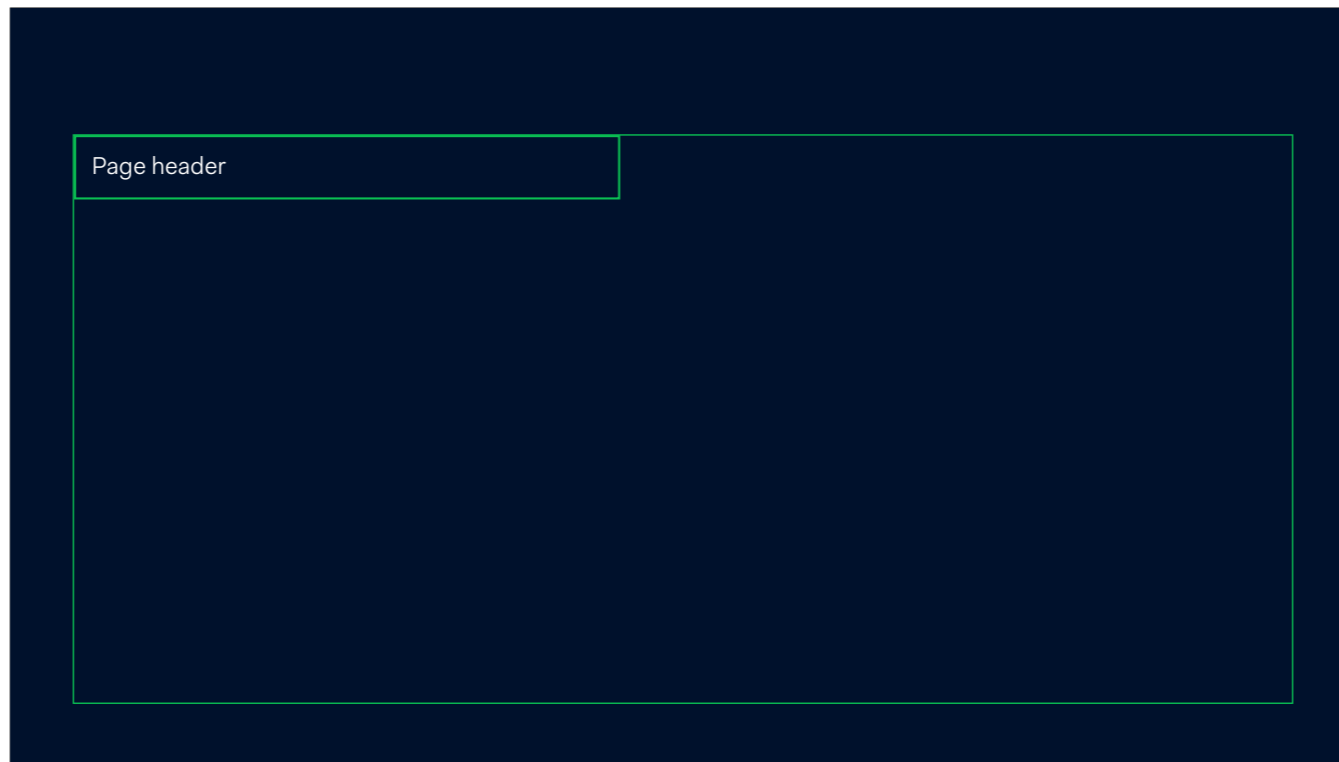
Sir David Brailsford, Team Principal, Team Sky



[https://ichef.bbci.co.uk/news/624/mcs/media/images/80345000/jpg/\\_80345073\\_daw@brailford.jpg](https://ichef.bbci.co.uk/news/624/mcs/media/images/80345000/jpg/_80345073_daw@brailford.jpg)

adyen

engineered  
for ambition



- Establish row sizing: `repeat('x', 700)` produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),
- aligned to 8 bytes. With a 4-byte line pointer, each row consumes 748 bytes.
- An 8192-byte page holds at most 10 such rows:
- page header: 24 bytes
- 10 line pointers: 40 bytes → `pd_lower = 64`
- $10 \times 744$  bytes of tuple data → `pd_upper = 8192 - 7440 = 752`
- free space:  $752 - 64 = 688$  bytes (not enough for an 11th row)

```
CREATE TABLE onepage (id bigint, t text, d date);
ALTER TABLE onepage SET (autovacuum_enabled = false);
```

Page header

- Establish row sizing: repeat('x', 700) produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),
- aligned to 8 bytes. With a 4-byte line pointer, each row consumes 748 bytes.
- An 8192-byte page holds at most 10 such rows:
- page header: 24 bytes
- 10 line pointers: 40 bytes → pd\_lower = 64
- 10 × 744 bytes of tuple data → pd\_upper = 8192 - 7440 = 752
- free space: 752 - 64 = 688 bytes (not enough for an 11th row)

**MVCC**

Page header

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

(0,1) xmin: 770, xmax: 0

- Establish row sizing: repeat('x', 700) produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-01');
```

Page header

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 0

```
INSERT INTO onepage VALUES (3, repeat('x', 700), '2024-01-01');
```

Page header

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 0

```
UPDATE onepage SET d = '2024-06-01' WHERE id = 1;
```

Page header

(0,4) xmin: 774, xmax: 0

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 774

```
BEGIN; UPDATE onepage SET d = '2024-06-01' WHERE id = 2; ROLLBACK;
```

Page header

(0,5) xmin: 775, xmax: 0

(0,4) xmin: 774, xmax: 0

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 775

(0,1) xmin: 770, xmax: 774

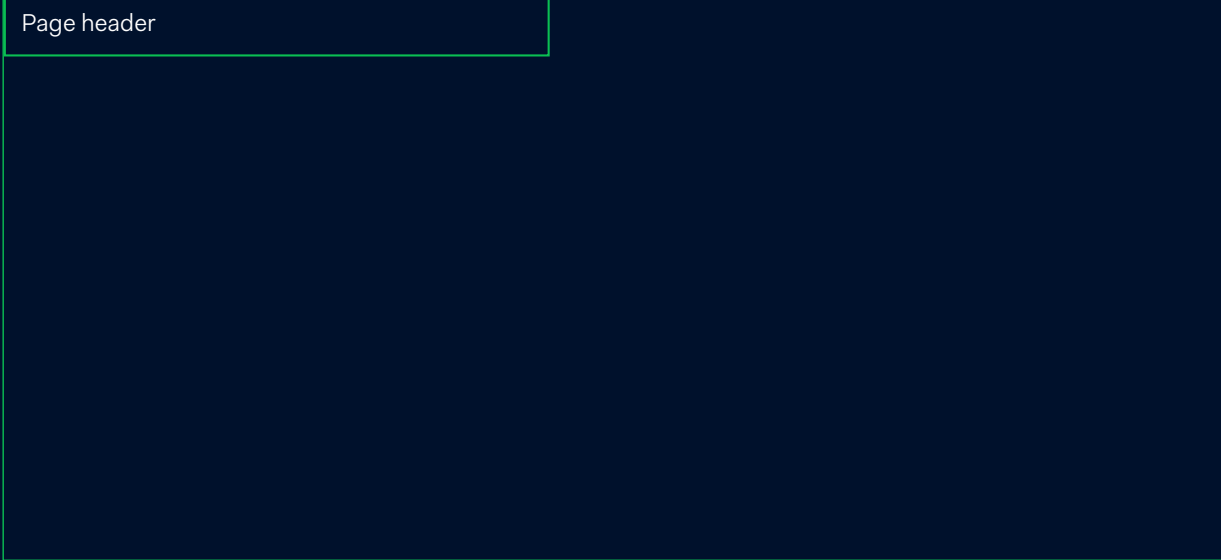
Update and abort leaves a ghost tuple. xmax of the original row is also marked as being aborted.

**Why** does this matter?

**Hint bits**

```
TRUNCATE onepage;
```

Page header



Checking whether `xmin` or `xmax` is committed requires reading the transaction status from the commit log (CLOG/pg\_xact). This is a disk I/O per tuple on every visibility check — very expensive. PostgreSQL caches the result directly in the tuple's `t\_infomask` field as **hint bits**, set lazily on the first read after commit.

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

xmin: 776, xmax: 0, hint: 802

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 776, xmax: 0, hint: 802

802 translates to varwidth + xmax invalid

The transaction is not listed as being committed

```
SELECT id, d FROM onepage;
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 776, xmax: 0, hint: 902

Now the hint bit is set as being committed and other transactions will rely on this information from now on.

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-02');  
DELETE FROM onepage WHERE id = 1; SELECT id FROM onepage;
```

Page header

xmin: 777, xmax: 0, hint: 902

xmin: 776, xmax: 778, hint: 502

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-02');
DELETE FROM onepage WHERE id = 1; SELECT id FROM onepage;
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 777, xmax: 0, hint: 902

xmin: 776, xmax: 778, hint: 502

Row1 has xmin committed, xmax committed, var width

Row2 has xmin committed, xmax invalid

**Why** does this matter?

# Single Page Cleanup

```
TRUNCATE onepage;  
CREATE INDEX ON onepage(id);
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 5) g;
```





Lets take a look at the page header. It lists where the data in the page can go. We have  $4472 - 44 = 4.428$  bytes left on this page

```
UPDATE onepage SET d = '2025-06-01' WHERE id IN (1, 2, 3);
```



prune\_txid is now set to 791, the oldest transaction id that needs information from this page.

The old versions of the updated rows still live within this page

```
SELECT COUNT(*) FROM onepage;
```



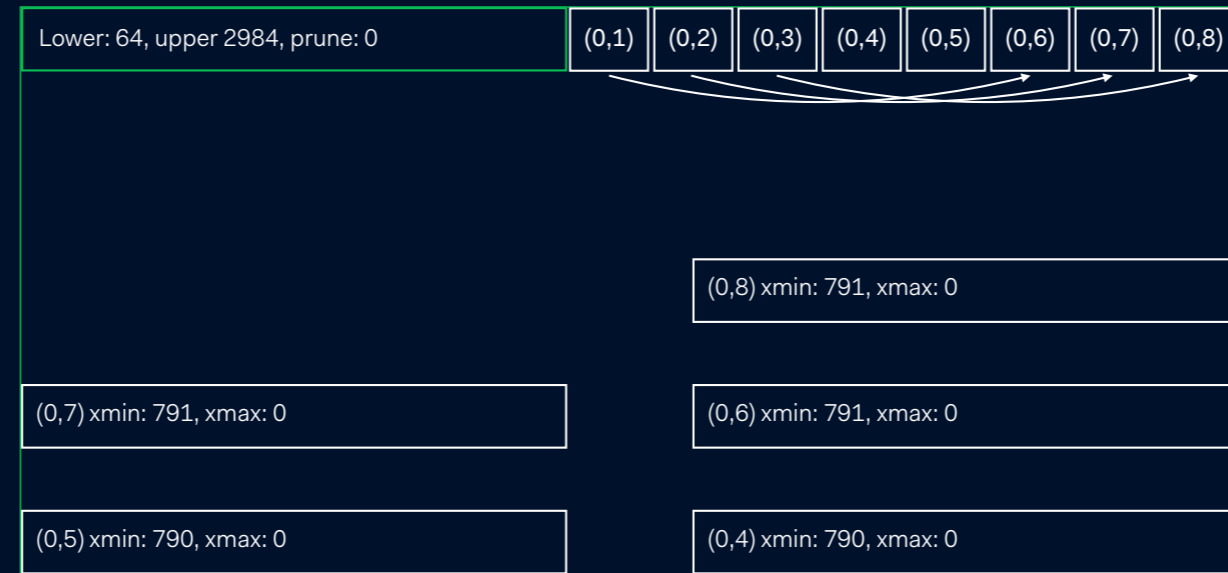
prune\_txid is now back to 0, no pruning possible

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are not redirects to the live tuples.

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are not redirects to the live tuples.

**Why** does this matter?

**Where to insert a row?**

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are now redirects to the live tuples.

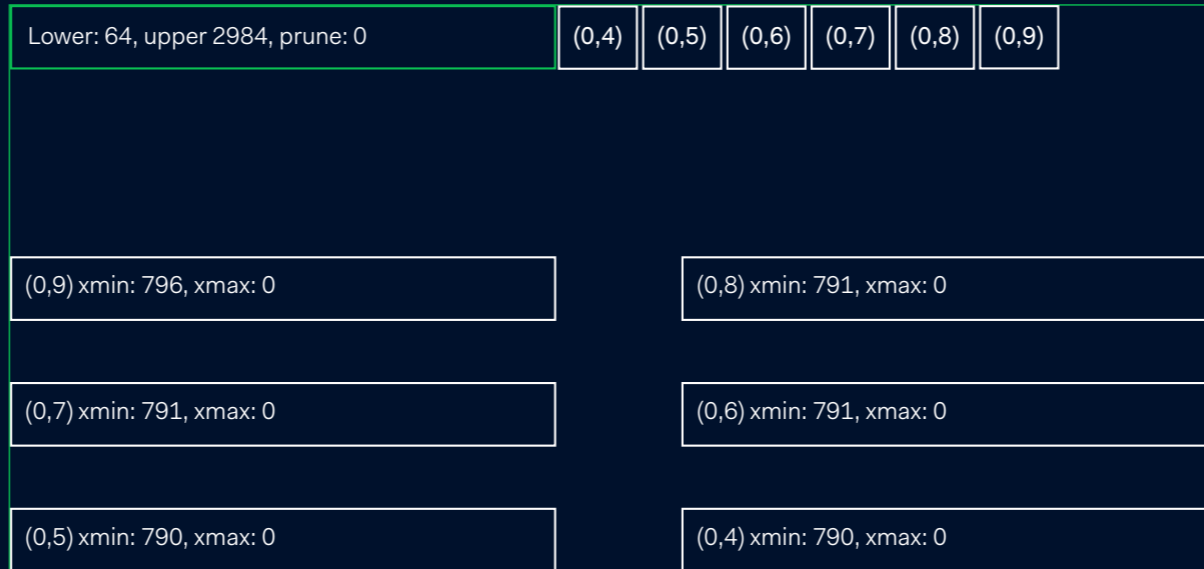
```
INSERT INTO onepage VALUES (200, repeat('z', 700), '2026-01-01');
```



Where to insert the next incoming row —> didn't arrive at this page.

Logic for where to insert is based on the free space map. The free space map is usually not updated on Single Page Cleanup.

```
VACUUM onepage;  
INSERT INTO onepage VALUES (300, repeat('z', 700), '2026-06-01');
```



Where to insert the next incoming row —> didn't arrive at this page.

Logic for where to insert is based on the free space map. The free space map is usually not updated on Single Page Cleanup.

TODO: when does pruning update FSM

get confirmation on lp cleanup

Update lower and upper and prune values.

**Why** does this matter?

**Fillfactor**

```
TRUNCATE onepage; DROP INDEX onepage_id_idx;  
ALTER TABLE onepage SET (fillfactor = 75);
```

Page header

With fillfactor=75, PostgreSQL reserves 25% of each page for updates. Threshold calculation:

Available space: 8192 bytes

Reserved:  $8192 \times 25\% = 2048$  bytes minimum free space required after each INSERT

A new row needs: 748 bytes (4 LP + 744 tuple)

After 7 rows:  $\text{free} = 8168 - (7 \times 748) = 2932$  bytes  $\rightarrow$  check:  $2932 - 748 = 2184 \geq 2048 \rightarrow$  row 8 fits

After 8 rows:  $\text{free} = 8168 - (8 \times 748) = 2184$  bytes  $\rightarrow$  check:  $2184 - 748 = 1436 < 2048 \rightarrow$  row 9 does NOT fit on page 0

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 9) g;
```

Lower: 56, upper 2240, prune: 0

(0,8) xmin: 801, xmax: 0

(0,7) xmin: 801, xmax: 0

(0,6) xmin: 801, xmax: 0

(0,5) xmin: 801, xmax: 0

(0,4) xmin: 801, xmax: 0

(0,3) xmin: 801, xmax: 0

(0,2) xmin: 801, xmax: 0

(0,1) xmin: 801, xmax: 0

With fillfactor=75, PostgreSQL reserves 25% of each page for updates. Threshold calculation:

Available space: 8192 bytes

Reserved:  $8192 \times 25\% = 2048$  bytes minimum free space required after each INSERT

A new row needs: 748 bytes (4 LP + 744 tuple)

After 7 rows:  $\text{free} = 8168 - (7 \times 748) = 2932$  bytes  $\rightarrow$  check:  $2932 - 748 = 2184 \geq 2048 \rightarrow$  row 8 fits

After 8 rows:  $\text{free} = 8168 - (8 \times 748) = 2184$  bytes  $\rightarrow$  check:  $2184 - 748 = 1436 < 2048 \rightarrow$  row 9 does NOT fit on page 0

```
CREATE INDEX onepage_id_idx ON onepage(id);
UPDATE onepage SET d = '2025-06-01' WHERE id IN (1, 2, 3);
```

Lower: 64, upper 752, prune: 803, flags: 2

(0,10) xmin: 803, xmax: 0

(0,9) xmin: 803, xmax: 0

(0,8) xmin: 801, xmax: 0

(0,7) xmin: 801, xmax: 0

(0,6) xmin: 801, xmax: 0

(0,5) xmin: 801, xmax: 0

(0,4) xmin: 801, xmax: 0

(0,3) xmin: 801, xmax: 803

(0,2) xmin: 801, xmax: 803

(0,1) xmin: 801, xmax: 803

Flags 2: page full

The third update went to page 1, didn't fit this page anymore

Free space = 752 - 64 = 688; doesn't fit a 744 wide row.

This page only contains two new rows, the third row went to page 1.

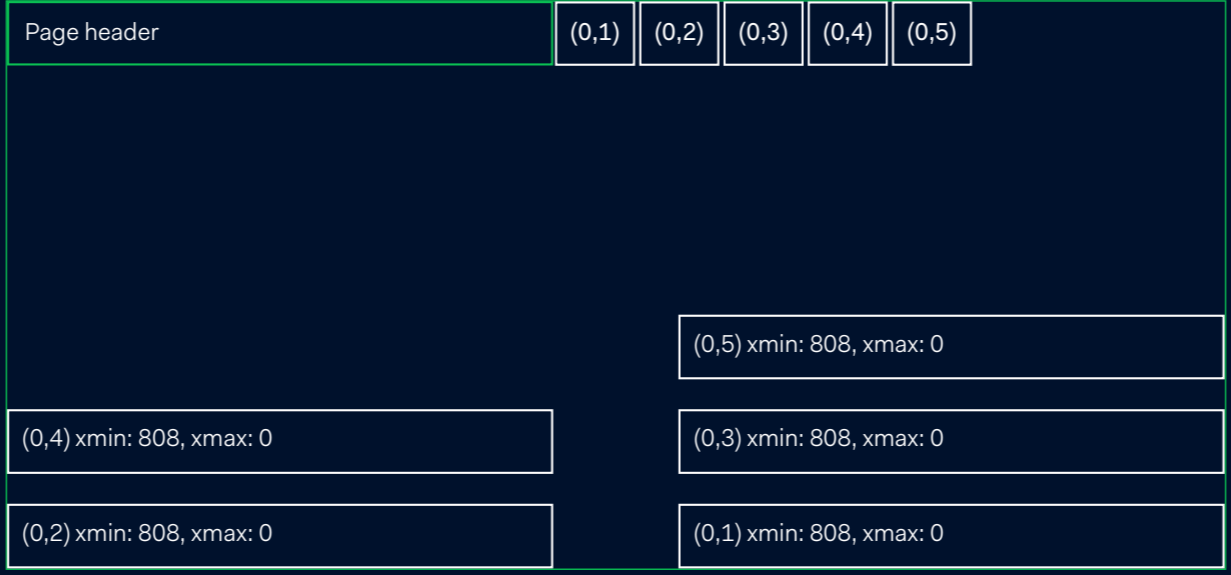
**Why** does this matter?

**HOT Updates**

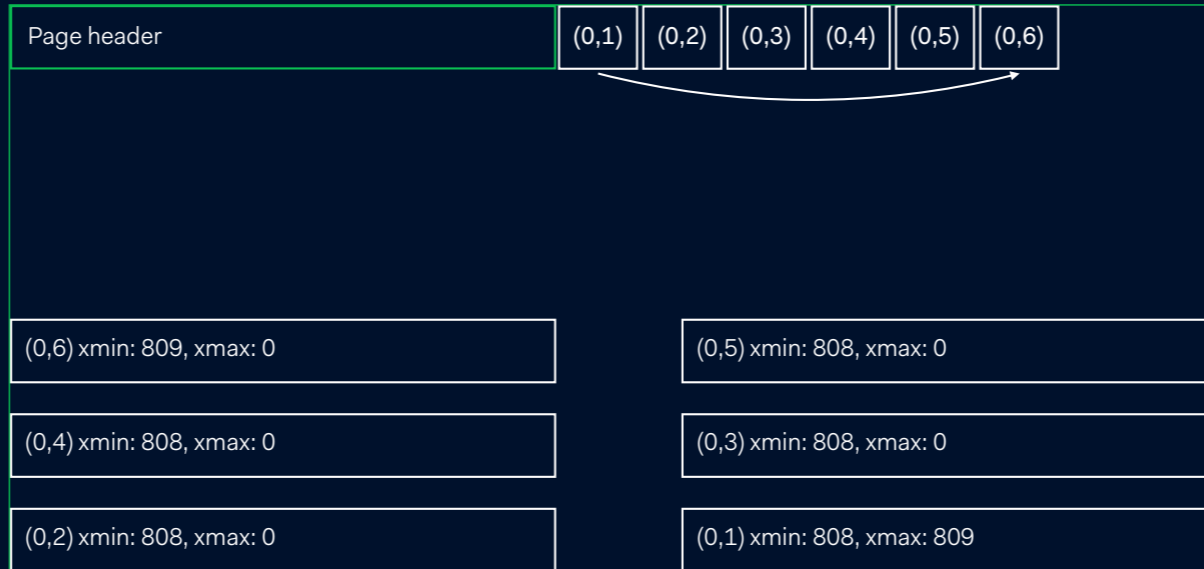
```
TRUNCATE onepage;  
-- Fillfactor and index remain
```

Page header

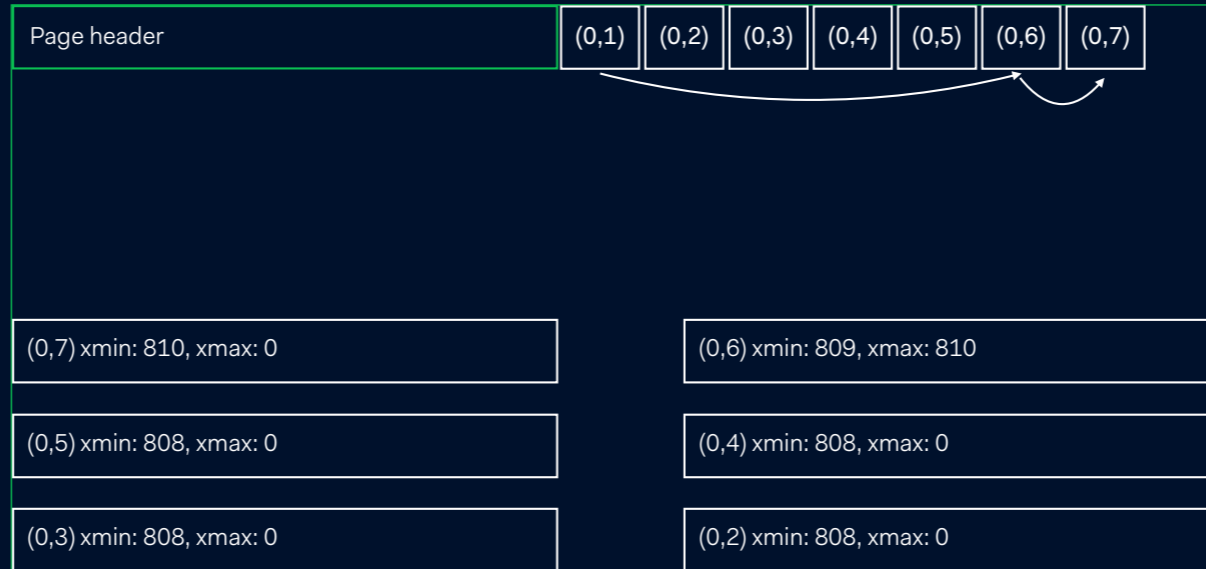
```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 5) g;
```

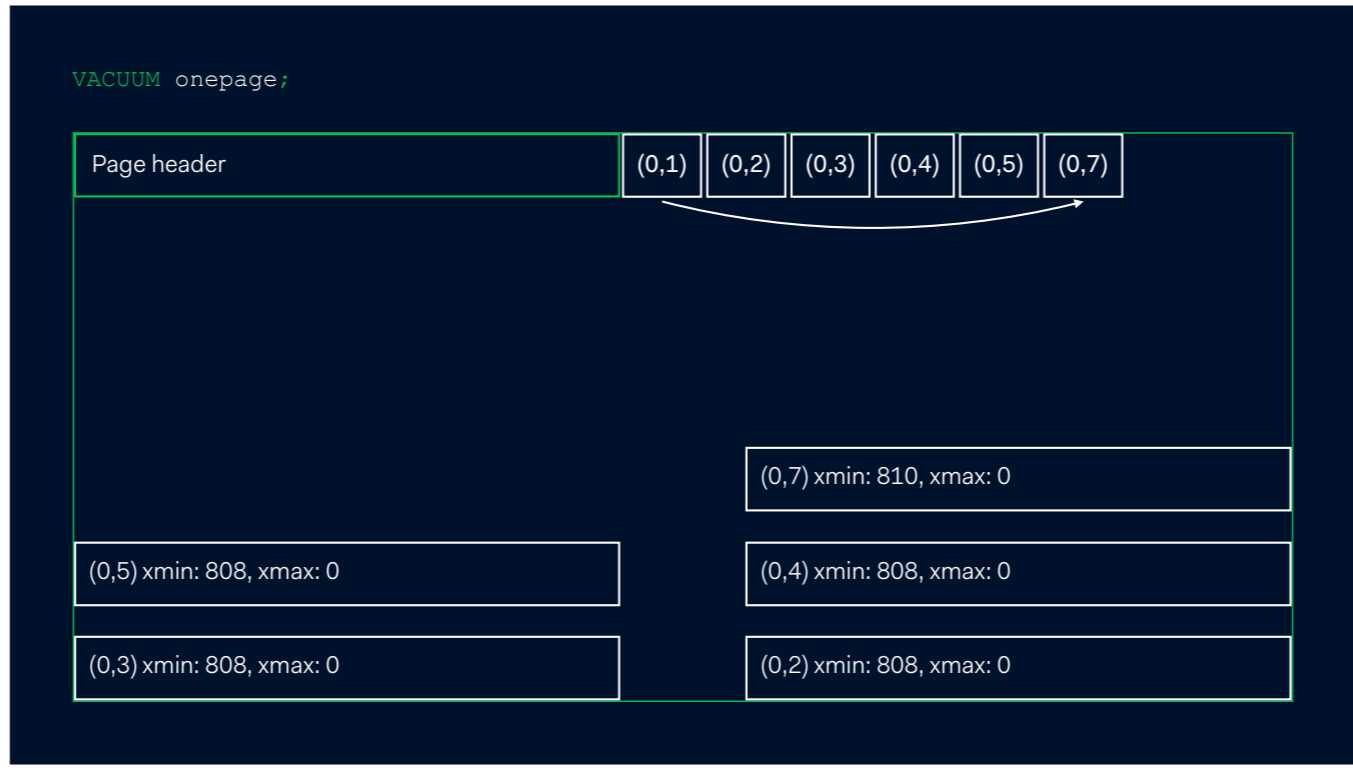


```
UPDATE onepage SET d = '2024-06-01' WHERE id = 1;
```



```
UPDATE onepage SET d = '2024-12-01' WHERE id = 1;
```





No index changes on the updates, no requirement to vacuum the index!

Number of pages changed for a single row with 10 indexes with and without HOT update

Impact of the fillfactor

**Why** does this matter?

**Dead rows vs bloat**

```
TRUNCATE onepage; DROP INDEX onepage_id_idx;  
ALTER TABLE onepage SET (fillfactor = 100);
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date FROM
generate_series(1,10) g;
```

Page header

(0,10) xmin: 825, xmax: 0

(0,9) xmin: 825, xmax: 0

(0,8) xmin: 825, xmax: 0

(0,7) xmin: 825, xmax: 0

(0,6) xmin: 825, xmax: 0

(0,5) xmin: 825, xmax: 0

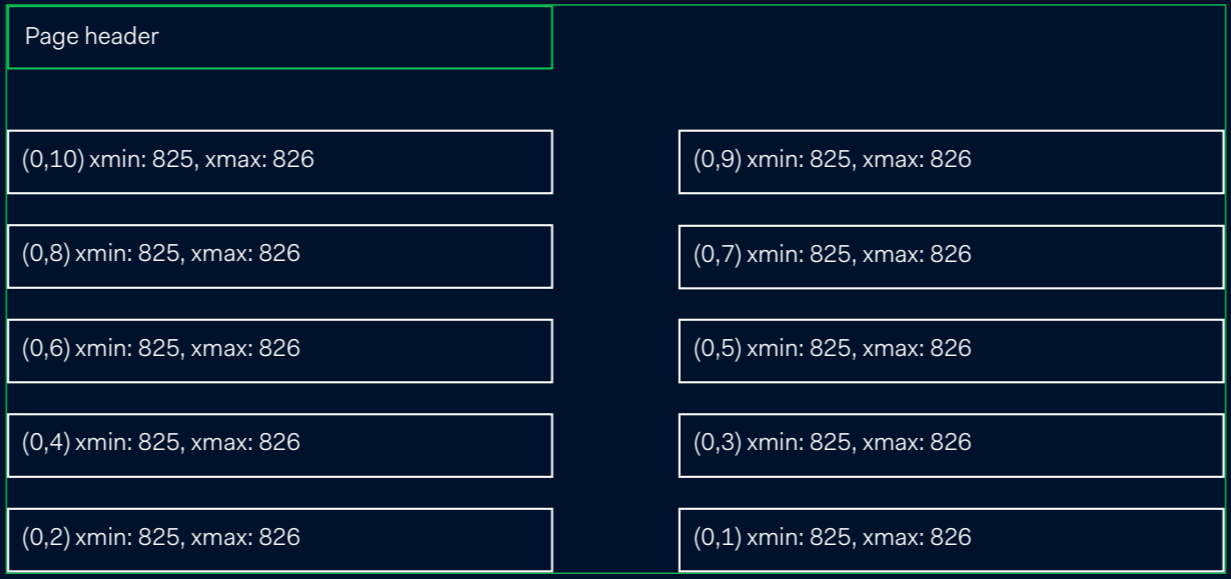
(0,4) xmin: 825, xmax: 0

(0,3) xmin: 825, xmax: 0

(0,2) xmin: 825, xmax: 0

(0,1) xmin: 825, xmax: 0

```
UPDATE onepage SET d = '2025-01-01';  
UPDATE onepage SET d = '2026-01-01';
```



Now the page has 10 dead rows, a row where xmax has been set;

```
SELECT * FROM onepage;
```

Page header

Now the page has 10 dead rows, a row where xmax has been set;

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables
WHERE relname = 'onepage';
n_live_tup | n_dead_tup
```

```
-----+-----
          10 |          20
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));
pg_size_pretty
```

```
-----
24 kB
```

Now the page has 10 dead rows, a row where xmax has been set;

```
VACUUM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
10 | 0
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
24 kB
```

Now vacuum brings the table size back to 1 page! No need to run vacuum full.

```
UPDATE onepage SET d = '2026-04-21';
```

Page header

(0,10) xmin: 827, xmax: 0

(0,8) xmin: 827, xmax: 0

(0,6) xmin: 827, xmax: 0

(0,4) xmin: 827, xmax: 0

(0,2) xmin: 827, xmax: 0

(0,9) xmin: 827, xmax: 0

(0,7) xmin: 827, xmax: 0

(0,5) xmin: 827, xmax: 0

(0,3) xmin: 827, xmax: 0

(0,1) xmin: 827, xmax: 0

The next update moves all rows back to my first page

```
VACUUM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
          10 |          0
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
8192 bytes
```

Now vacuum brings the table size back to 1 page! No need to run vacuum full.

```
TRUNCATE onepage;
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date FROM
generate_series(1,10) g;
```

Page header

(0,10) xmin: 833, xmax: 0

(0,9) xmin: 833, xmax: 0

(0,8) xmin: 833, xmax: 0

(0,7) xmin: 833, xmax: 0

(0,6) xmin: 833, xmax: 0

(0,5) xmin: 833, xmax: 0

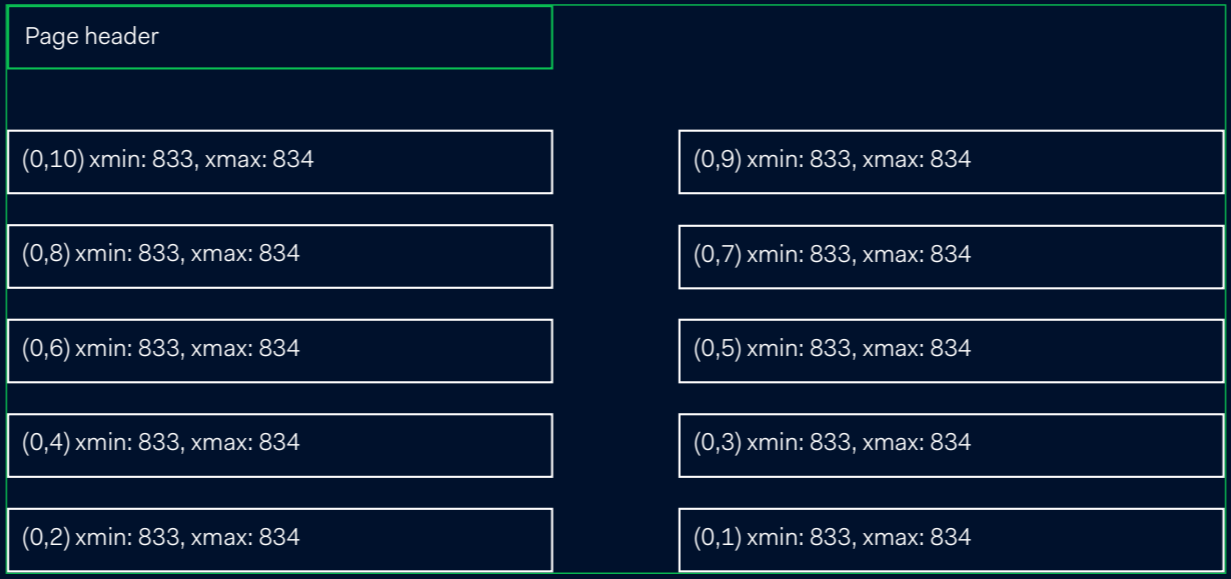
(0,4) xmin: 833, xmax: 0

(0,3) xmin: 833, xmax: 0

(0,2) xmin: 833, xmax: 0

(0,1) xmin: 833, xmax: 0

```
UPDATE onepage SET d = '2025-01-01';  
UPDATE onepage SET d = '2026-01-01';
```



Now the page has 10 dead rows, a row where xmax has been set;

```
DELETE FROM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
0 | 30
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
24 kB
```

Now the page has 10 dead rows, a row where xmax has been set;

**Why** does this matter?

# Adding **small gains** for maximum results

**adyen**

engineered  
for ambition

Operational hazards of  
managing PostgreSQL  
DBs over 100TB

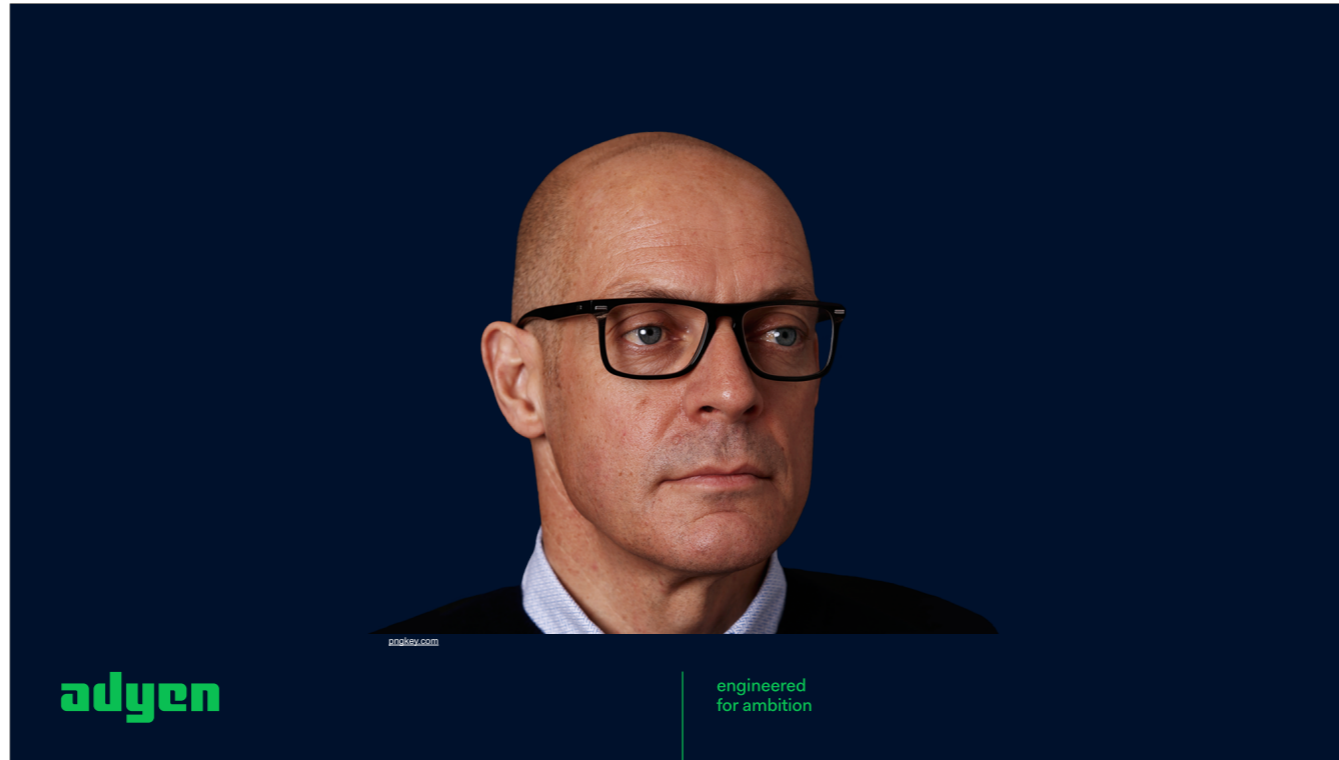


adys

# Adding **small gains** for maximum results

**adyen**

engineered  
for ambition

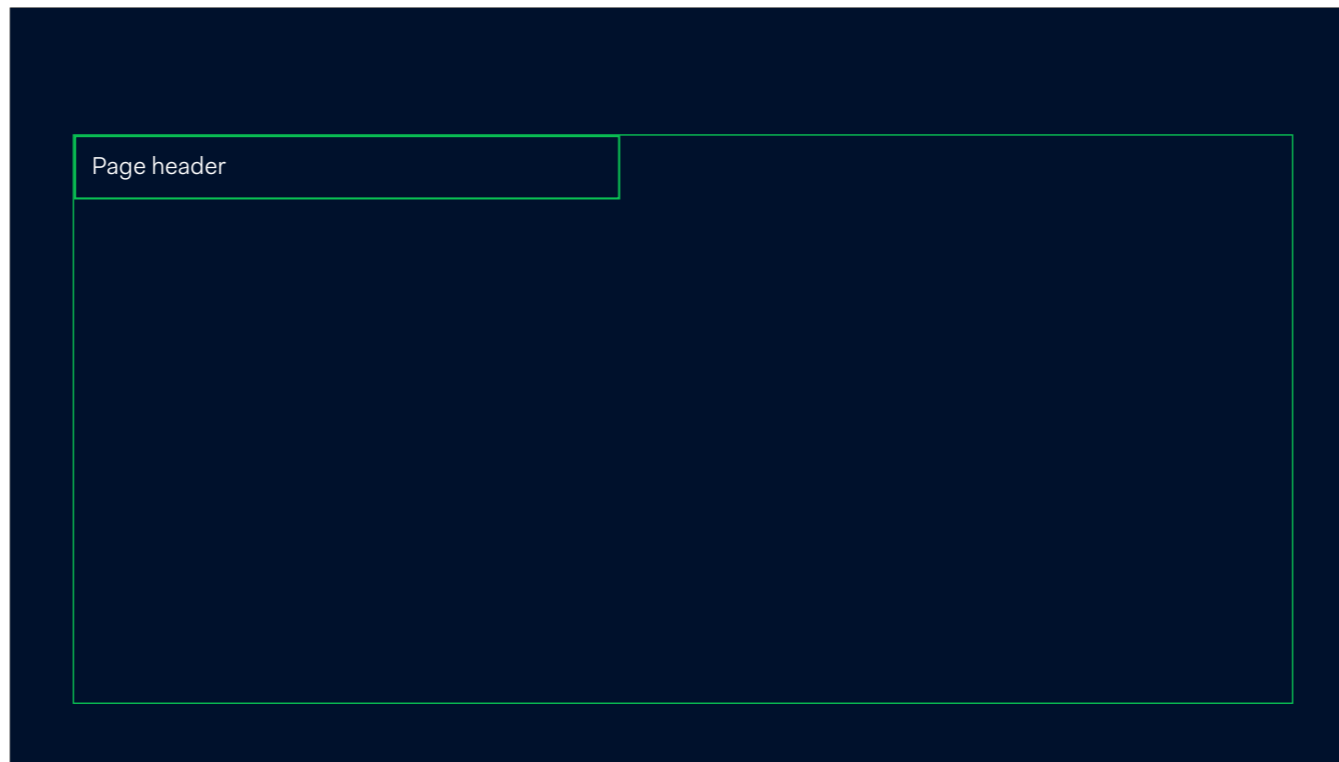


Sir David Brailsford, Team Principal, Team Sky



adyen

engineered  
for ambition



- Establish row sizing: `repeat('x', 700)` produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),
- aligned to 8 bytes. With a 4-byte line pointer, each row consumes 748 bytes.
- An 8192-byte page holds at most 10 such rows:
- page header: 24 bytes
- 10 line pointers: 40 bytes → `pd_lower = 64`
- $10 \times 744$  bytes of tuple data → `pd_upper = 8192 - 7440 = 752`
- free space:  $752 - 64 = 688$  bytes (not enough for an 11th row)

```
CREATE TABLE onepage (id bigint, t text, d date);
ALTER TABLE onepage SET (autovacuum_enabled = false);
```

Page header

- Establish row sizing: repeat('x', 700) produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),
- aligned to 8 bytes. With a 4-byte line pointer, each row consumes 748 bytes.
- An 8192-byte page holds at most 10 such rows:
- page header: 24 bytes
- 10 line pointers: 40 bytes → pd\_lower = 64
- 10 × 744 bytes of tuple data → pd\_upper = 8192 - 7440 = 752
- free space: 752 - 64 = 688 bytes (not enough for an 11th row)

**MVCC**

Page header

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

(0,1) xmin: 770, xmax: 0

- Establish row sizing: repeat('x', 700) produces a 704-byte varlena column.
- Each full tuple is 744 bytes (24 header + 8 id + 704 text + 4 date + 4 alignment padding),

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-01');
```

Page header

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 0

```
INSERT INTO onepage VALUES (3, repeat('x', 700), '2024-01-01');
```

Page header

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 0

```
UPDATE onepage SET d = '2024-06-01' WHERE id = 1;
```

Page header

(0,4) xmin: 774, xmax: 0

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 0

(0,1) xmin: 770, xmax: 774

```
BEGIN; UPDATE onepage SET d = '2024-06-01' WHERE id = 2; ROLLBACK;
```

Page header

(0,5) xmin: 775, xmax: 0

(0,4) xmin: 774, xmax: 0

(0,3) xmin: 772, xmax: 0

(0,2) xmin: 771, xmax: 775

(0,1) xmin: 770, xmax: 774

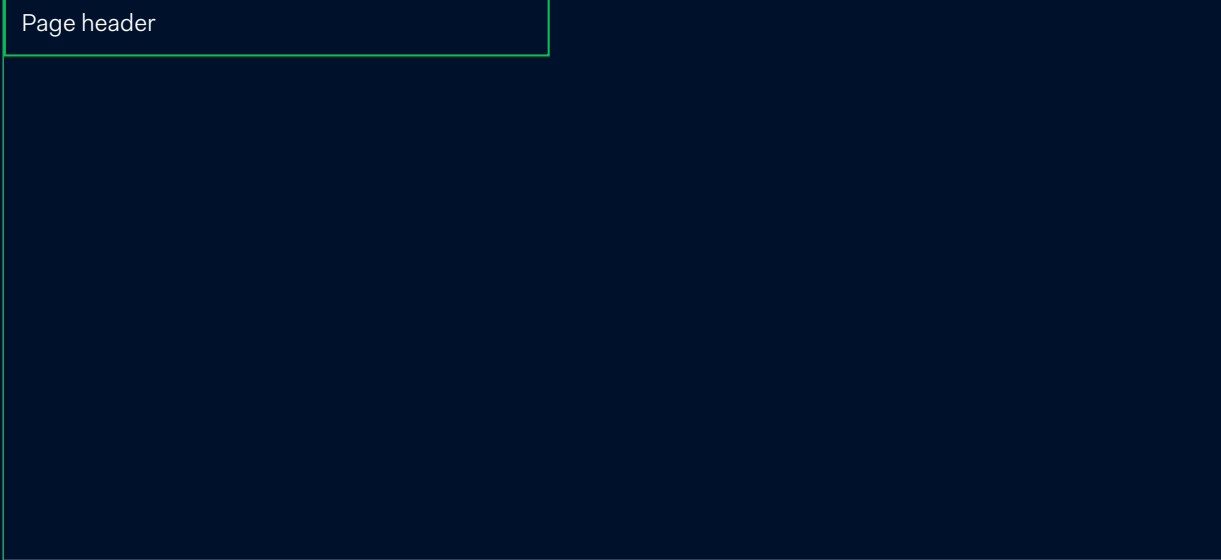
Update and abort leaves a ghost tuple. xmax of the original row is also marked as being aborted.

**Why** does this matter?

**Hint bits**

```
TRUNCATE onepage;
```

Page header



Checking whether `xmin` or `xmax` is committed requires reading the transaction status from the commit log (CLOG/pg\_xact). This is a disk I/O per tuple on every visibility check — very expensive. PostgreSQL caches the result directly in the tuple's `t\_infomask` field as **hint bits**, set lazily on the first read after commit.

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

xmin: 776, xmax: 0, hint: 802

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

```
INSERT INTO onepage VALUES (1, repeat('x', 700), '2024-01-01');
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 776, xmax: 0, hint: 802

802 translates to varwidth + xmax invalid

The transaction is not listed as being committed

```
SELECT id, d FROM onepage;
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 776, xmax: 0, hint: 902

Now the hint bit is set as being committed and other transactions will rely on this information from now on.

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-02');  
DELETE FROM onepage WHERE id = 1; SELECT id FROM onepage;
```

Page header

xmin: 777, xmax: 0, hint: 902

xmin: 776, xmax: 778, hint: 502

```
INSERT INTO onepage VALUES (2, repeat('x', 700), '2024-01-02');
DELETE FROM onepage WHERE id = 1; SELECT id FROM onepage;
```

Page header

0x0002	HEAP_HASVARWIDTH
0x0100	HEAP_XMIN_COMMITTED
0x0200	HEAP_XMIN_INVALID
0x0400	HEAP_XMAX_COMMITTED
0x0800	HEAP_XMAX_INVALID
0x1000	HEAP_UPDATED

xmin: 777, xmax: 0, hint: 902

xmin: 776, xmax: 778, hint: 502

Row1 has xmin committed, xmax committed, var width

Row2 has xmin committed, xmax invalid

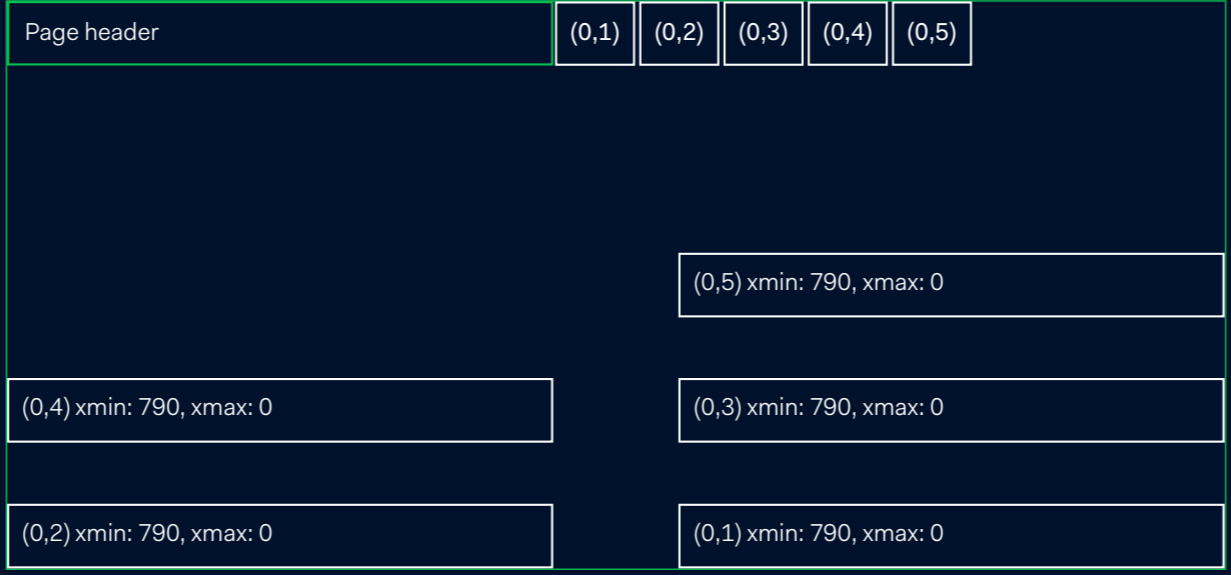
**Why** does this matter?

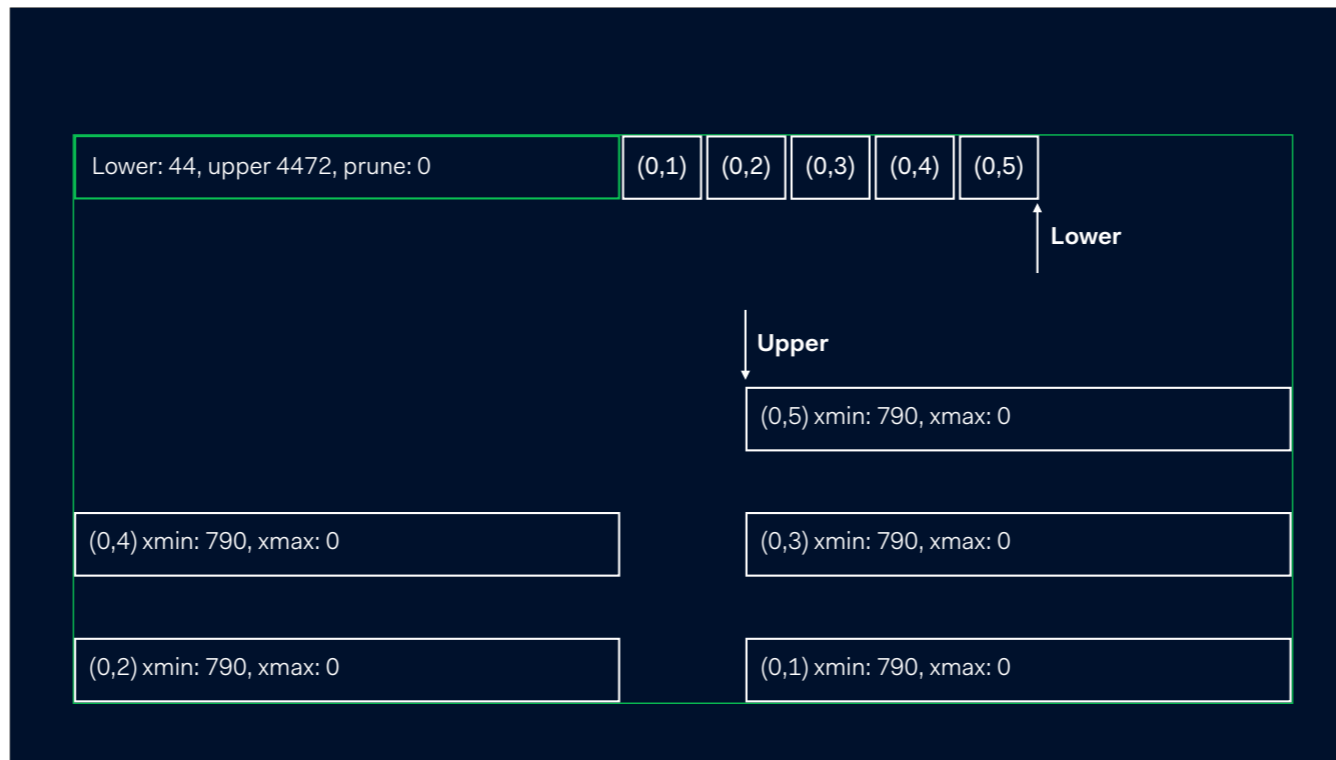
# Single Page Cleanup

```
TRUNCATE onepage;  
CREATE INDEX ON onepage(id);
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 5) g;
```





Lets take a look at the page header. It lists where the data in the page can go. We have  $4472 - 44 = 4.428$  bytes left on this page

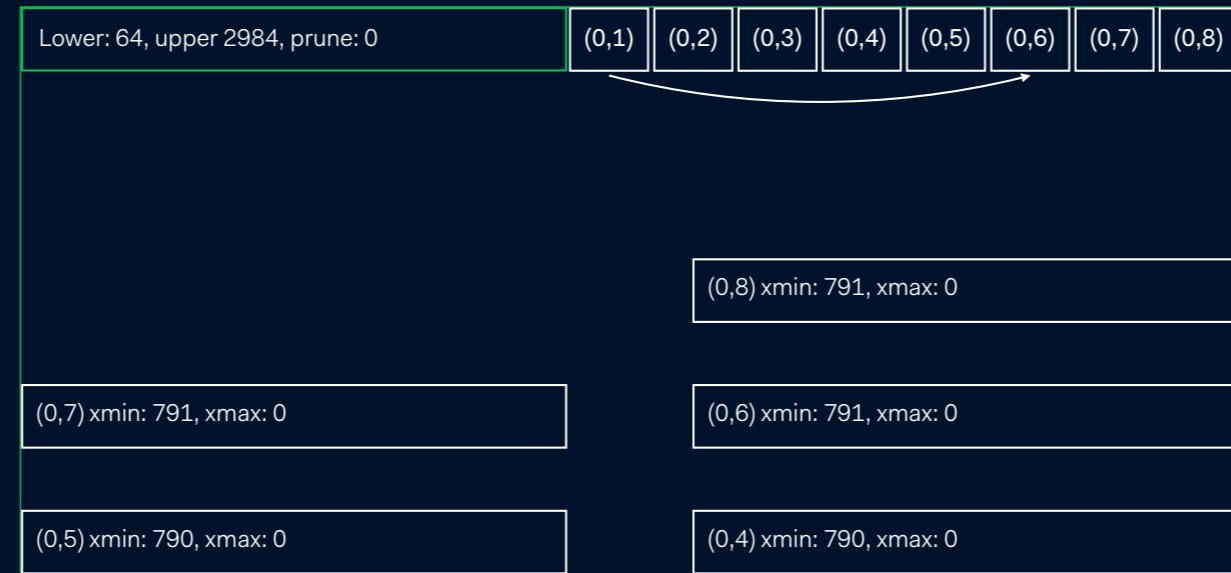
```
UPDATE onepage SET d = '2025-06-01' WHERE id IN (1, 2, 3);
```



prune\_txid is now set to 791, the oldest transaction id that needs information from this page.

The old versions of the updated rows still live within this page

```
SELECT COUNT(*) FROM onepage;
```



prune\_txid is now back to 0, no pruning possible

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are not redirects to the live tuples.

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are not redirects to the live tuples.

**Why** does this matter?

**Where to insert a row?**

```
SELECT COUNT(*) FROM onepage;
```



Tuple data has been removed, but line pointers remain and are now redirects to the live tuples.

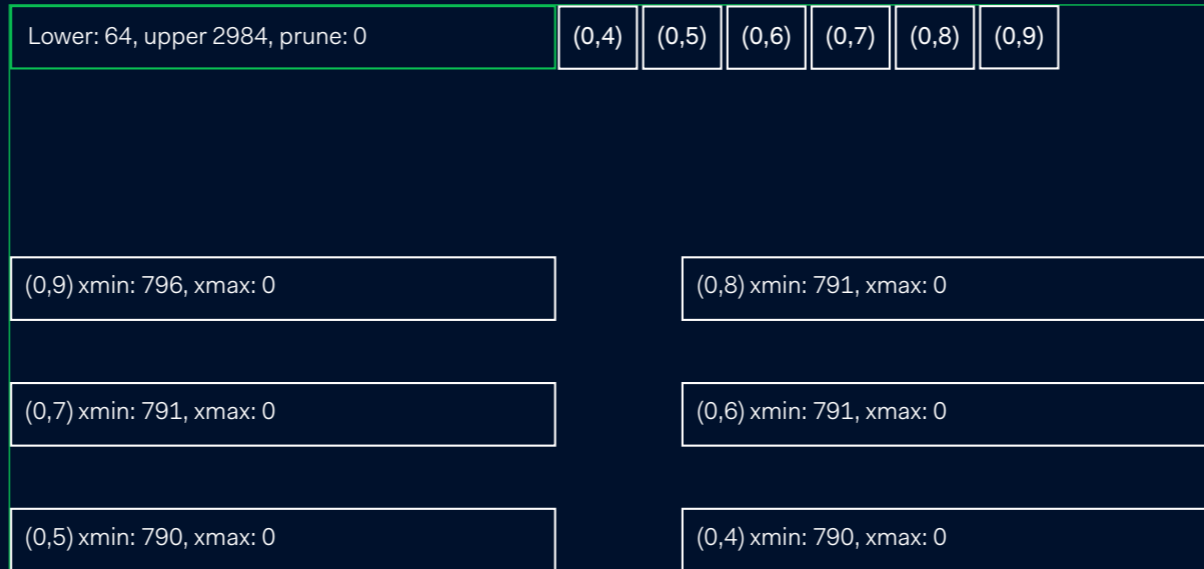
```
INSERT INTO onepage VALUES (200, repeat('z', 700), '2026-01-01');
```



Where to insert the next incoming row —> didn't arrive at this page.

Logic for where to insert is based on the free space map. The free space map is usually not updated on Single Page Cleanup.

```
VACUUM onepage;  
INSERT INTO onepage VALUES (300, repeat('z', 700), '2026-06-01');
```



Where to insert the next incoming row —> didn't arrive at this page.

Logic for where to insert is based on the free space map. The free space map is usually not updated on Single Page Cleanup.

TODO: when does pruning update FSM  
get confirmation on lp cleanup  
Update lower and upper and prune values.

**Why** does this matter?

**Fillfactor**

```
TRUNCATE onepage; DROP INDEX onepage_id_idx;  
ALTER TABLE onepage SET (fillfactor = 75);
```

Page header

With fillfactor=75, PostgreSQL reserves 25% of each page for updates. Threshold calculation:

Available space: 8192 bytes

Reserved:  $8192 \times 25\% = 2048$  bytes minimum free space required after each INSERT

A new row needs: 748 bytes (4 LP + 744 tuple)

After 7 rows:  $\text{free} = 8168 - (7 \times 748) = 2932$  bytes  $\rightarrow$  check:  $2932 - 748 = 2184 \geq 2048 \rightarrow$  row 8 fits

After 8 rows:  $\text{free} = 8168 - (8 \times 748) = 2184$  bytes  $\rightarrow$  check:  $2184 - 748 = 1436 < 2048 \rightarrow$  row 9 does NOT fit on page 0

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 9) g;
```

Lower: 56, upper 2240, prune: 0

(0,8) xmin: 801, xmax: 0

(0,7) xmin: 801, xmax: 0

(0,6) xmin: 801, xmax: 0

(0,5) xmin: 801, xmax: 0

(0,4) xmin: 801, xmax: 0

(0,3) xmin: 801, xmax: 0

(0,2) xmin: 801, xmax: 0

(0,1) xmin: 801, xmax: 0

With fillfactor=75, PostgreSQL reserves 25% of each page for updates. Threshold calculation:

Available space: 8192 bytes

Reserved:  $8192 \times 25\% = 2048$  bytes minimum free space required after each INSERT

A new row needs: 748 bytes (4 LP + 744 tuple)

After 7 rows:  $\text{free} = 8168 - (7 \times 748) = 2932$  bytes  $\rightarrow$  check:  $2932 - 748 = 2184 \geq 2048 \rightarrow$  row 8 fits

After 8 rows:  $\text{free} = 8168 - (8 \times 748) = 2184$  bytes  $\rightarrow$  check:  $2184 - 748 = 1436 < 2048 \rightarrow$  row 9 does NOT fit on page 0

```
CREATE INDEX onepage_id_idx ON onepage(id);
UPDATE onepage SET d = '2025-06-01' WHERE id IN (1, 2, 3);
```

Lower: 64, upper 752, prune: 803, flags: 2	
(0,10) xmin: 803, xmax: 0	(0,9) xmin: 803, xmax: 0
(0,8) xmin: 801, xmax: 0	(0,7) xmin: 801, xmax: 0
(0,6) xmin: 801, xmax: 0	(0,5) xmin: 801, xmax: 0
(0,4) xmin: 801, xmax: 0	(0,3) xmin: 801, xmax: 803
(0,2) xmin: 801, xmax: 803	(0,1) xmin: 801, xmax: 803

Flags 2: page full

The third update went to page 1, didn't fit this page anymore

Free space = 752 - 64 = 688; doesn't fit a 744 wide row.

This page only contains two new rows, the third row went to page 1.

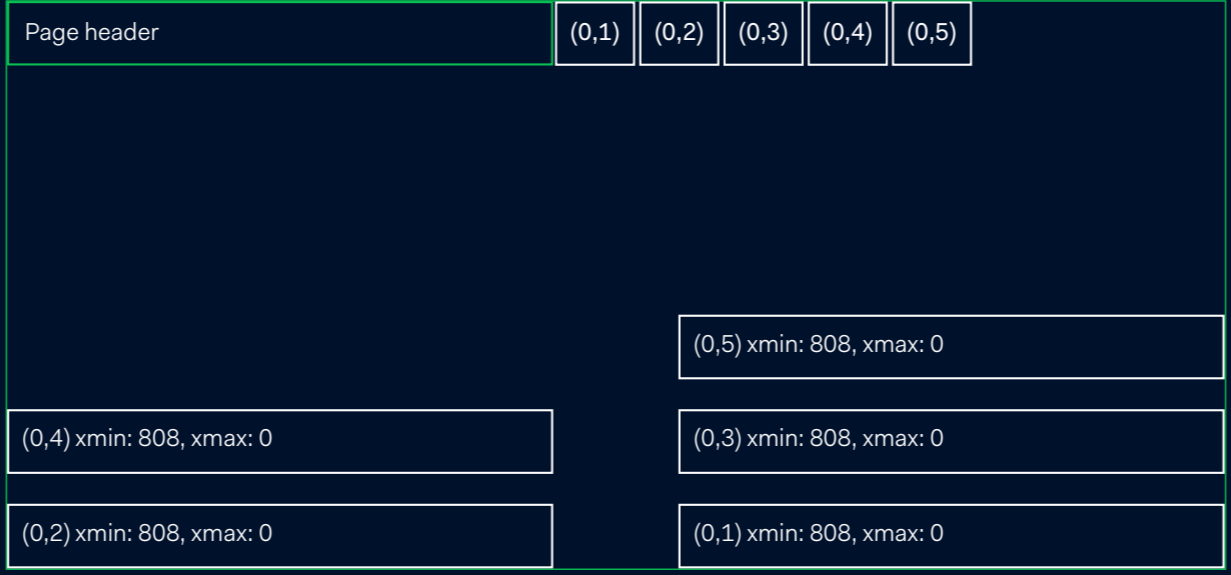
**Why** does this matter?

**HOT Updates**

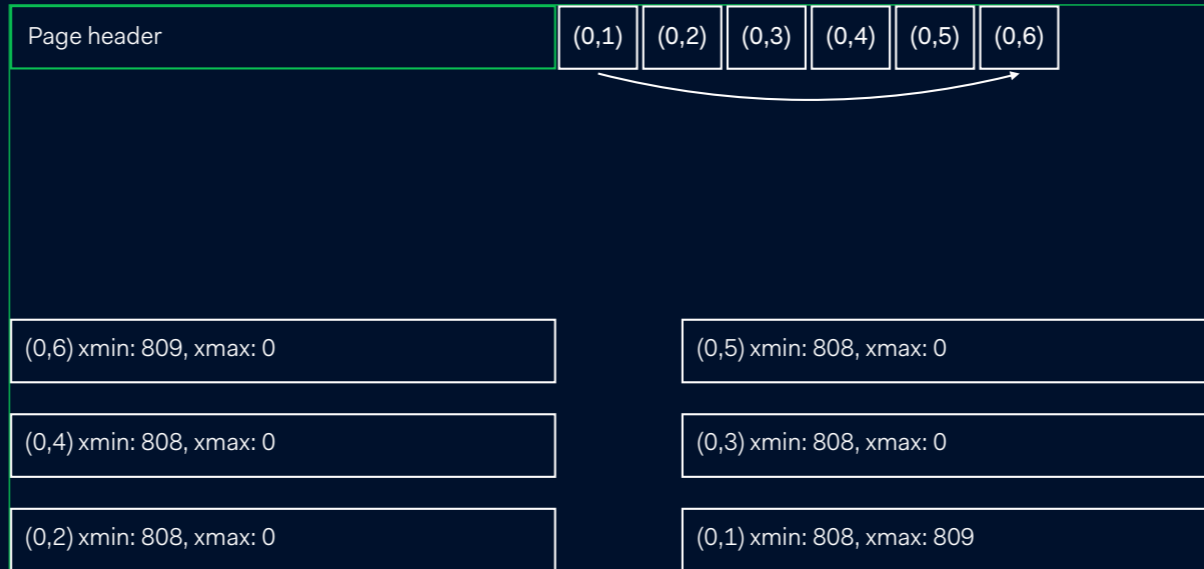
```
TRUNCATE onepage;  
-- Fillfactor and index remain
```

Page header

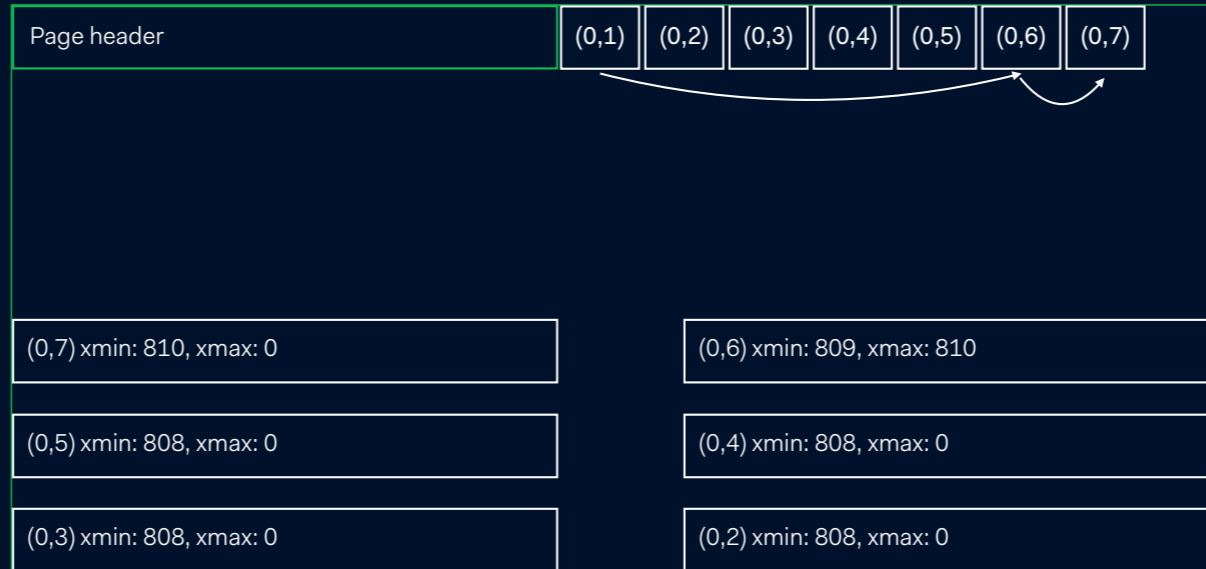
```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date
FROM generate_series(1, 5) g;
```

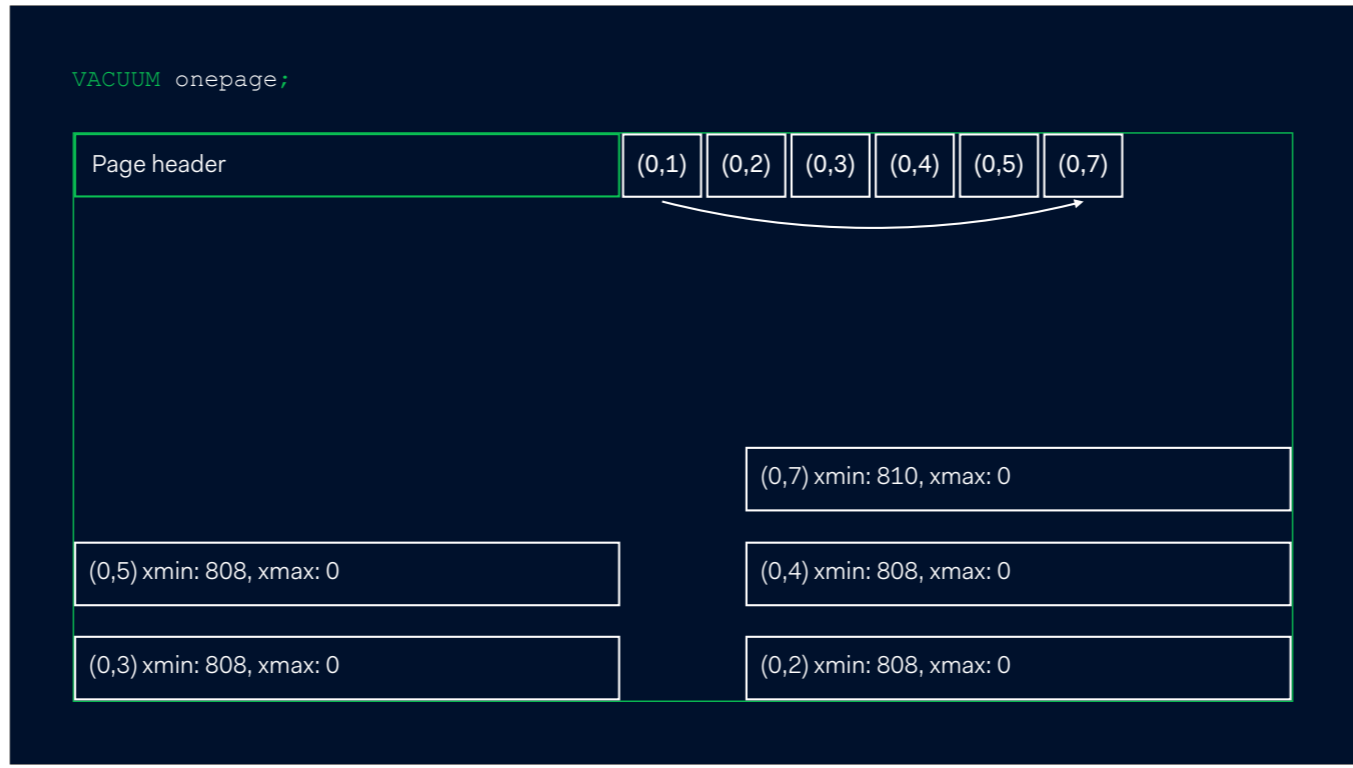


```
UPDATE onepage SET d = '2024-06-01' WHERE id = 1;
```



```
UPDATE onepage SET d = '2024-12-01' WHERE id = 1;
```





No index changes on the updates, no requirement to vacuum the index!

Number of pages changed for a single row with 10 indexes with and without HOT update

Impact of the fillfactor

**Why** does this matter?

**Dead rows vs bloat**

```
TRUNCATE onepage; DROP INDEX onepage_id_idx;  
ALTER TABLE onepage SET (fillfactor = 100);
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date FROM
generate_series(1,10) g;
```

Page header

(0,10) xmin: 825, xmax: 0

(0,9) xmin: 825, xmax: 0

(0,8) xmin: 825, xmax: 0

(0,7) xmin: 825, xmax: 0

(0,6) xmin: 825, xmax: 0

(0,5) xmin: 825, xmax: 0

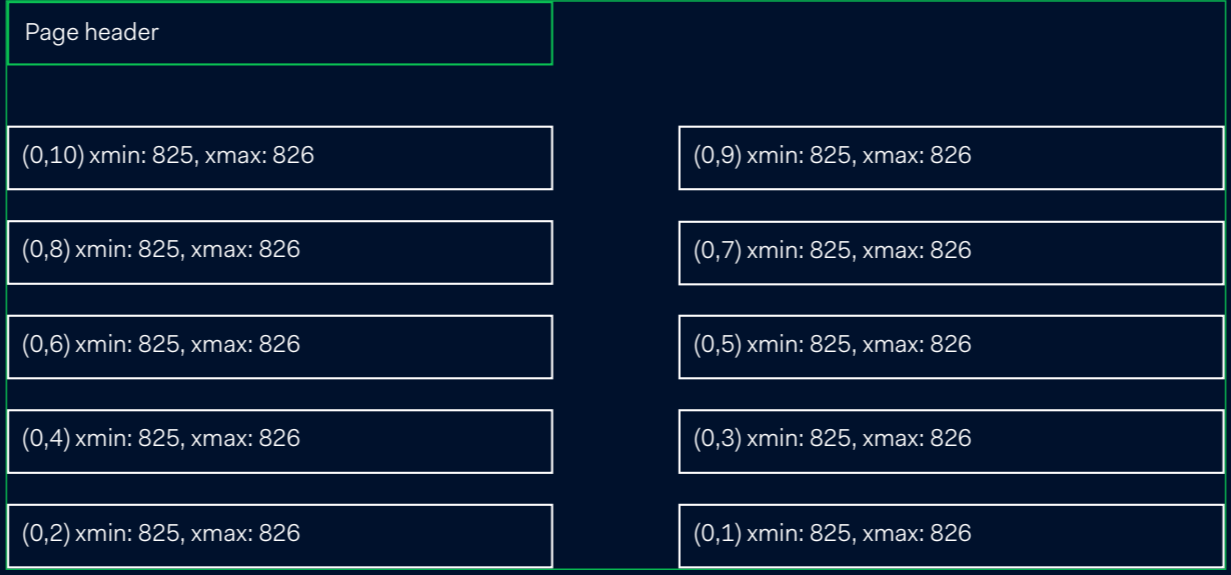
(0,4) xmin: 825, xmax: 0

(0,3) xmin: 825, xmax: 0

(0,2) xmin: 825, xmax: 0

(0,1) xmin: 825, xmax: 0

```
UPDATE onepage SET d = '2025-01-01';  
UPDATE onepage SET d = '2026-01-01';
```



Now the page has 10 dead rows, a row where xmax has been set;

```
SELECT * FROM onepage;
```

Page header

Now the page has 10 dead rows, a row where xmax has been set;

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables
WHERE relname = 'onepage';
n_live_tup | n_dead_tup
```

```
-----+-----
          10 |          20
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));
pg_size_pretty
```

```
-----
24 kB
```

Now the page has 10 dead rows, a row where xmax has been set;

```
VACUUM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
10 | 0
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
24 kB
```

Now vacuum brings the table size back to 1 page! No need to run vacuum full.

```
UPDATE onepage SET d = '2026-04-21';
```

Page header	
(0,10) xmin: 827, xmax: 0	(0,9) xmin: 827, xmax: 0
(0,8) xmin: 827, xmax: 0	(0,7) xmin: 827, xmax: 0
(0,6) xmin: 827, xmax: 0	(0,5) xmin: 827, xmax: 0
(0,4) xmin: 827, xmax: 0	(0,3) xmin: 827, xmax: 0
(0,2) xmin: 827, xmax: 0	(0,1) xmin: 827, xmax: 0

The next update moves all rows back to my first page

```
VACUUM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
          10 |          0
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
8192 bytes
```

Now vacuum brings the table size back to 1 page! No need to run vacuum full.

```
TRUNCATE onepage;
```

Page header

```
INSERT INTO onepage SELECT g, repeat('x', 700), '2024-01-01'::date FROM
generate_series(1,10) g;
```

Page header

(0,10) xmin: 833, xmax: 0

(0,9) xmin: 833, xmax: 0

(0,8) xmin: 833, xmax: 0

(0,7) xmin: 833, xmax: 0

(0,6) xmin: 833, xmax: 0

(0,5) xmin: 833, xmax: 0

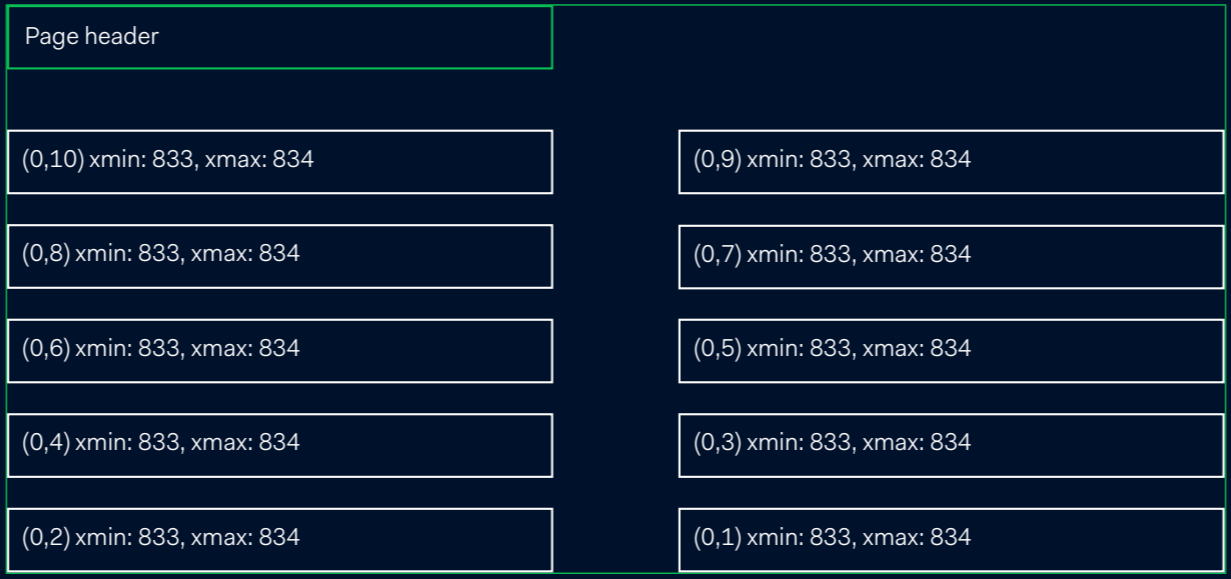
(0,4) xmin: 833, xmax: 0

(0,3) xmin: 833, xmax: 0

(0,2) xmin: 833, xmax: 0

(0,1) xmin: 833, xmax: 0

```
UPDATE onepage SET d = '2025-01-01';  
UPDATE onepage SET d = '2026-01-01';
```



Now the page has 10 dead rows, a row where xmax has been set;

```
DELETE FROM onepage;
```

Page header

```
SELECT n_live_tup, n_dead_tup FROM pg_stat_user_tables  
WHERE relname = 'onepage';  
n_live_tup | n_dead_tup
```

```
-----+-----  
0 | 30
```

```
SELECT pg_size_pretty(pg_relation_size('onepage'));  
pg_size_pretty
```

```
-----  
24 kB
```

Now the page has 10 dead rows, a row where xmax has been set;

**Why** does this matter?

# Adding **small gains** for maximum results

**adyen**

engineered  
for ambition

Operational hazards of  
managing PostgreSQL  
DBs over 100TB



adye

**SAVE  
THE DATE  
10.09.26**



**PGDAY  
LOWLANDS  
UTRECHT**