



B-tree - explore the heart of PostgreSQL

Anastasia Lubennikova

- PostgreSQL indexes overview
- Inspect internals of B-tree index
- Present B-tree features
 - Covering indexes
 - Deduplication

About me

- PostgreSQL contributor since 2015
 - Index-only scan for GiST
 - Microvacuum for GiST
 - B-tree INCLUDE clause
 - B-tree. Compression of duplicates
 - pg_probackup



a.lubennikova@postgrespro.ru

Indexes in PostgreSQL

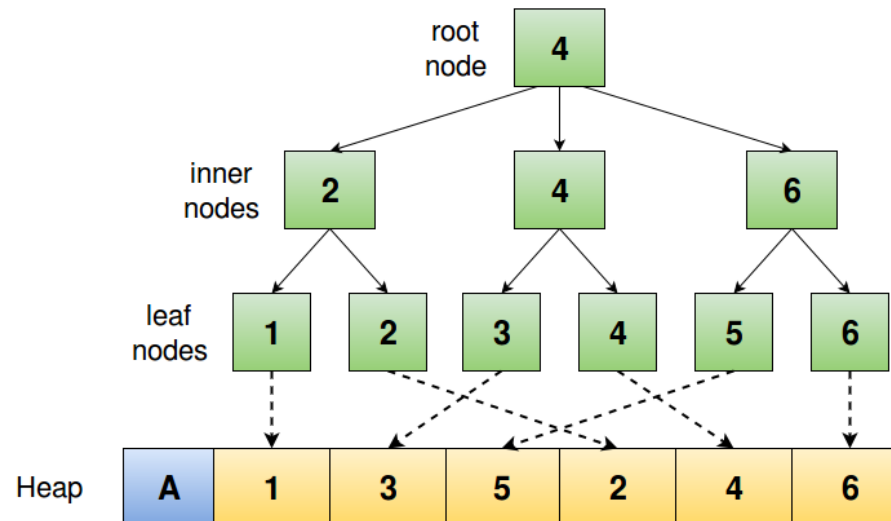
- All indexes are secondary
 - TID (tuple identifier) points to heap entry
- Index files are divided into standard-size pages
 - page layout is defined by Access Method
- Data type support is provided by opclass

Indexes in PostgreSQL (2)

- There is no visibility information in index
 - index-only scan optimization
- Update = insert
 - HOT-update optimization
- VACUUM removes dead tuples
 - microvacuum optimization

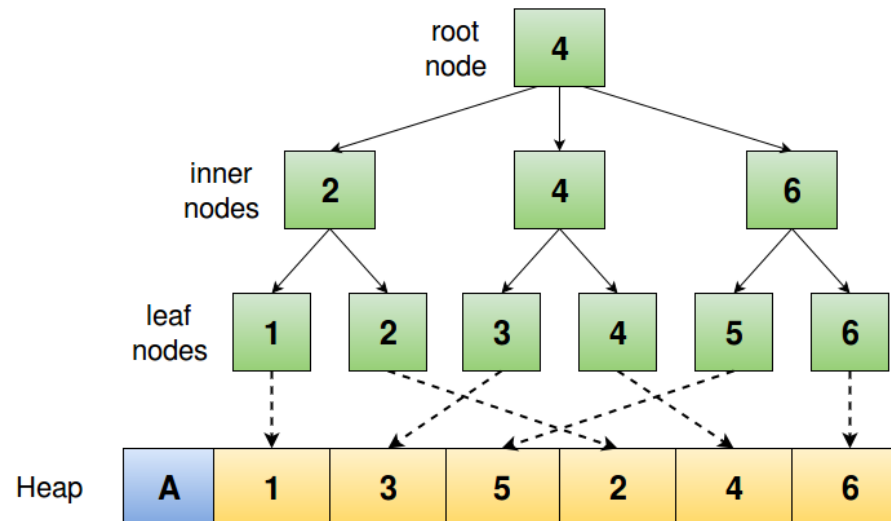
B+ tree

- Tree is balanced
- Root node and inner nodes contain keys and pointers to lower level nodes
- Leaf nodes contain keys and pointers to the heap
- All keys are sorted inside the node



B-tree in PostgreSQL

- Default index type
- Only B-tree indexes can be declared UNIQUE or PRIMARY KEY
- <, <=, =, >=, >, LIKE
- ORDER BY
- JOIN



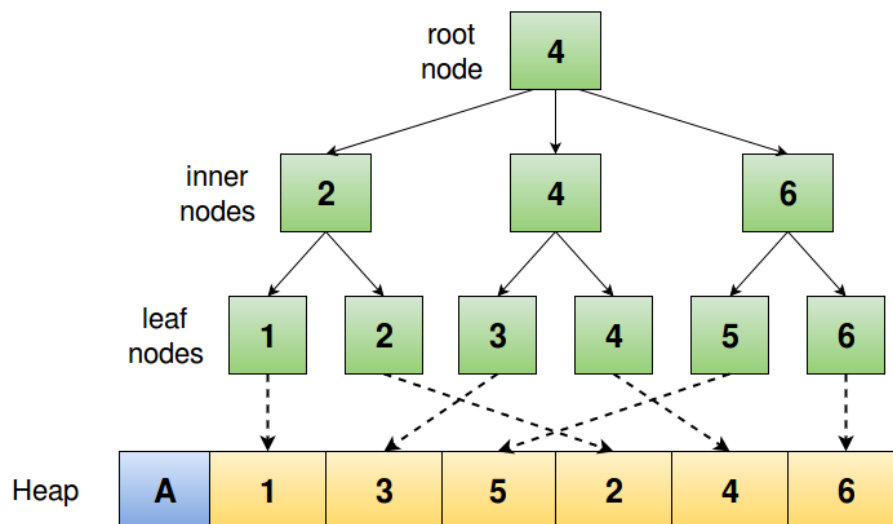
System and TOAST indexes

```
SELECT count(*) FROM pg_indexes
WHERE schemaname='pg_catalog';
```

```
count
```

```
-----
```

```
116
```



- ✓ Right-link pointer to the page's right sibling
- ✓ "High key" - upper bound on the keys that are allowed on that page
- ✓ Assume that we can fit at least three items per page

<https://www.csd.uoc.gr/~hy460/pdf/p650-lehman.pdf>

Lehman & Yao (2)

- × Assume fixed size keys
- × Assume that in-memory copies of tree pages are unshared
- × Assume that all btree keys are unique

<https://www.csd.uoc.gr/~hy460/pdf/p650-lehman.pdf>

- Left-link pointer to the page's left sibling
 - For backward scan purposes
- Variable-size keys
 - Require special handling on page splits
- Pages are shared
 - Page-level read locking is required

<https://github.com/postgres/postgres/blob/master/src/backend/access/nbtree/README>

PostgreSQL B-tree

- Nonunique keys
 - Use TID to check tuples equality
 - «Heap TID as a tiebreaker» optimization in Postgres v12
 - «Split after new tuple» optimization in Postgres v12

<https://www.cvbertec-postgresql.com/en/b-tree-index-improvements-in-postgresql-v12/>

<https://www.pgcon.org/2019/schedule/events/1282.en.html>

- Optimized index build
 - Parallel CREATE INDEX in Postgres v11
- amcheck extension for index verification

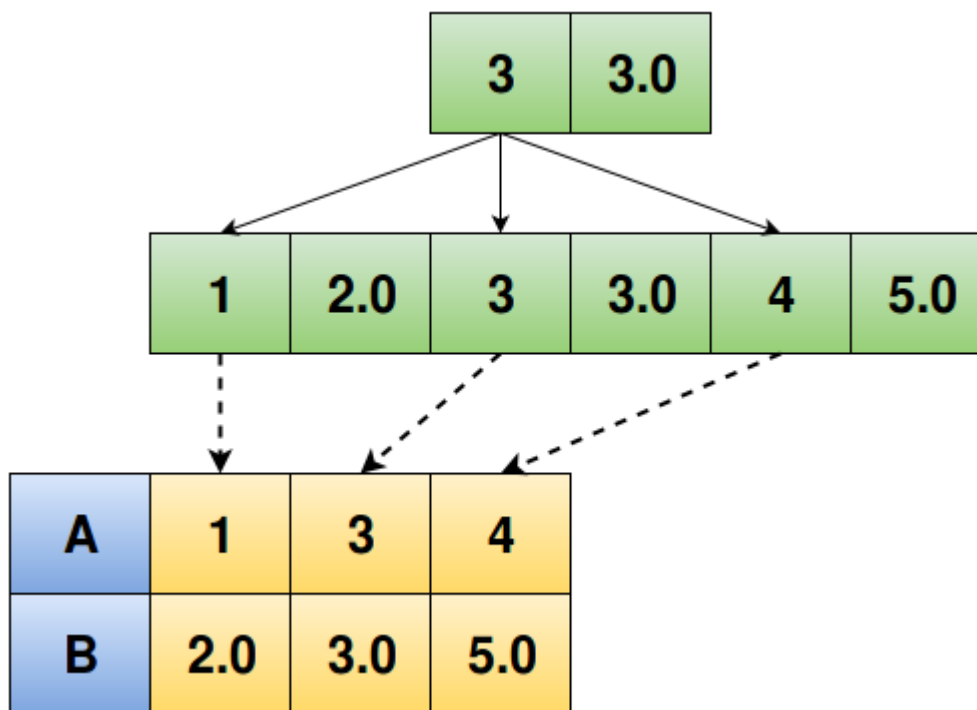
<https://www.cvbortec-postgresql.com/en/postgresql-parallel-create-index-for-better-performance/>

COVERING INDEXES

Covering index

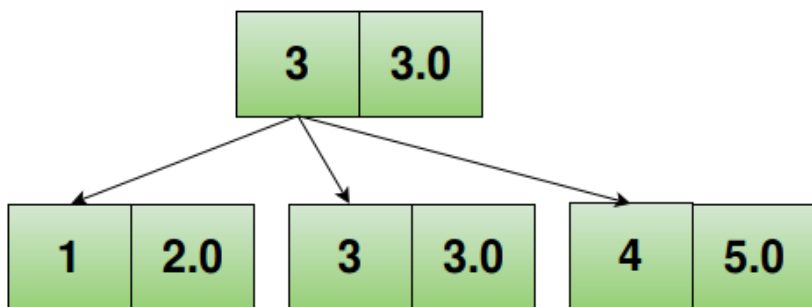
```
CREATE INDEX ON tbl (a, b);
```

```
SELECT a, b from tbl WHERE a > 3;
```

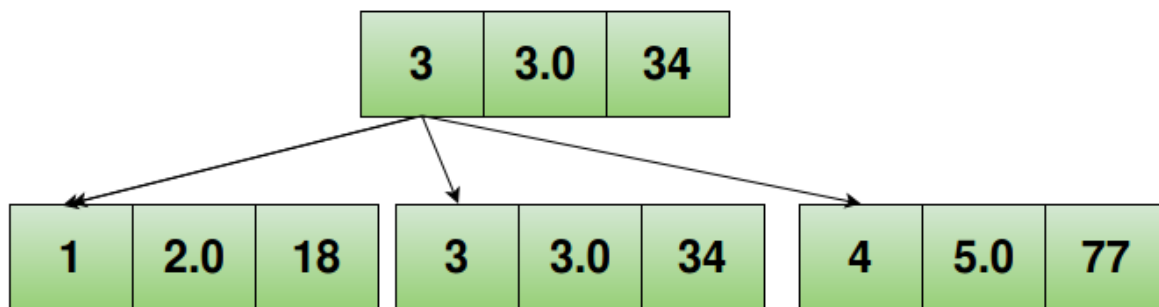


Covering indexes

Unique index on (c1, c2)



Covering index on (c1, c2 c3)

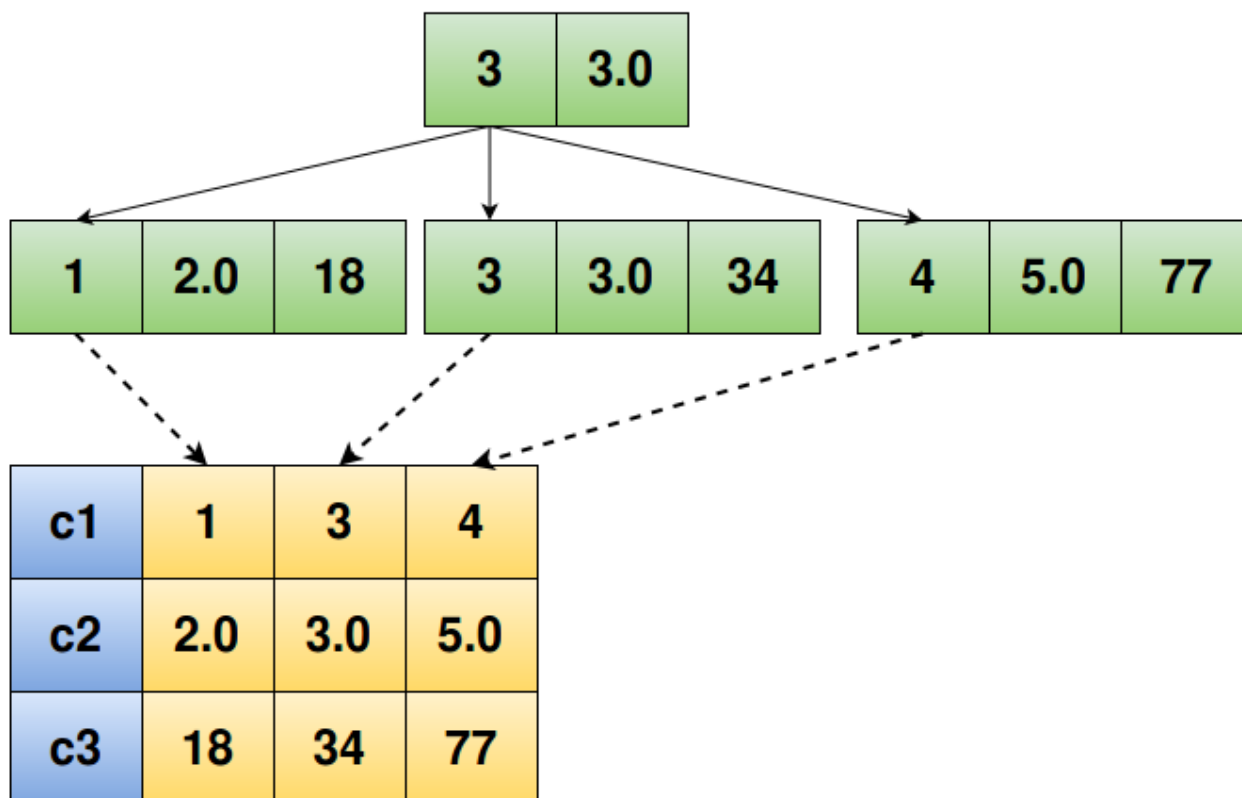


c1	1	3	4
c2	2.0	3.0	5.0
c3	18	34	77

Index with included columns

```
CREATE INDEX ON tbl (c1, c2) INCLUDE (c3);
```

```
SELECT c1, c2, c3 from tbl WHERE c1 > 3;
```



Index with INCLUDED columns

```
CREATE UNIQUE INDEX ON tbl (a) INCLUDE (b);
```

```
CREATE INDEX ON tbl (a) INCLUDE (b);
```

```
CREATE TABLE tbl (c1 int, c2 int, c3 int, c4 box);
```

```
CREATE UNIQUE INDEX tbl_idx_unique ON tbl  
using btree(c1, c2) INCLUDE(c3,c4);
```

```
ALTER TABLE tbl add UNIQUE USING INDEX  
tbl_idx_unique;
```

```
CREATE TABLE tbl (c1 int, c2 int, c3 int, c4 box);
```

```
ALTER TABLE tbl add UNIQUE(c1,c2) INCLUDE(c3,c4);
```

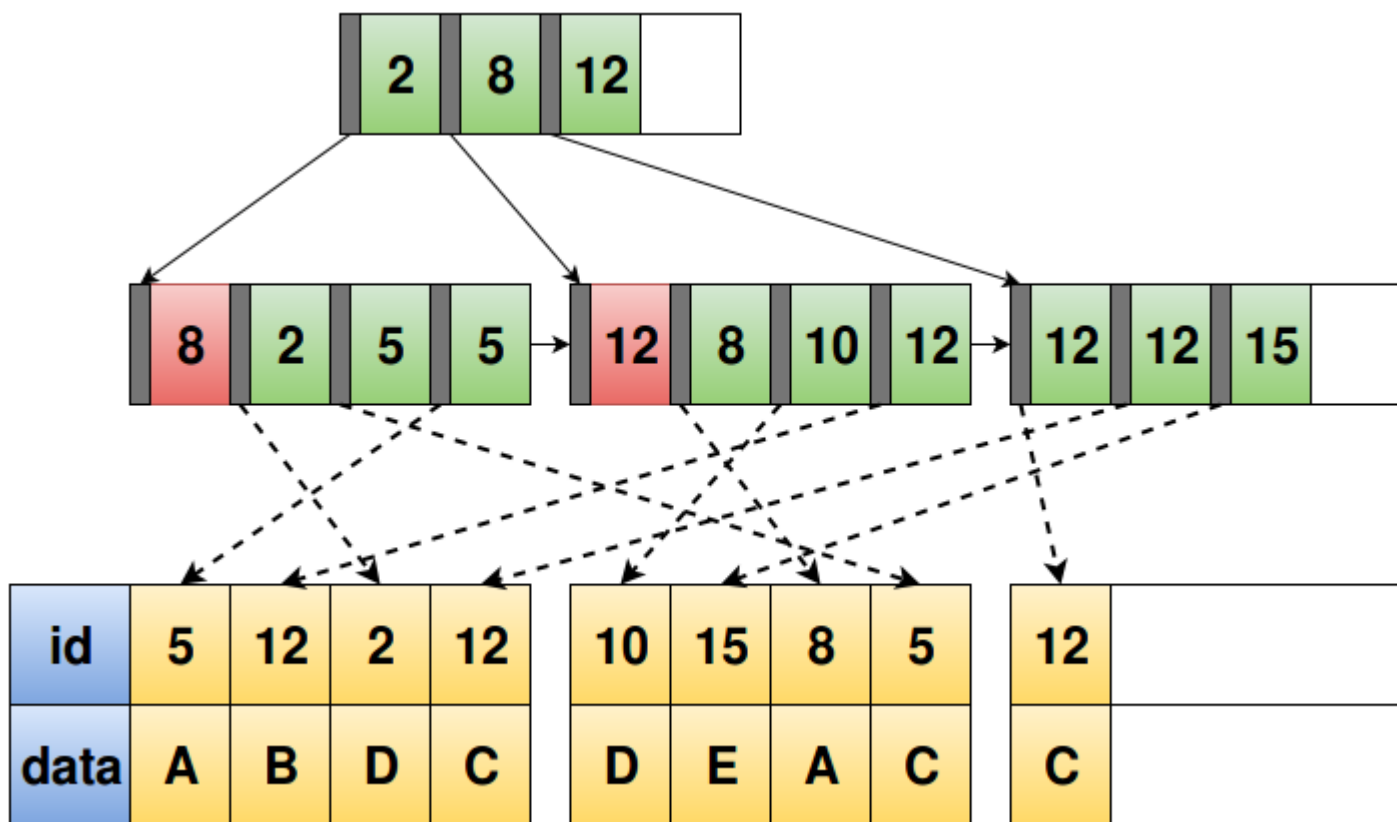
```
CREATE TABLE tbl(c1 int,c2 int, c3 int, c4 box,  
UNIQUE(c1,c2) INCLUDE(c3,c4) );
```

Index with INCLUDEd columns

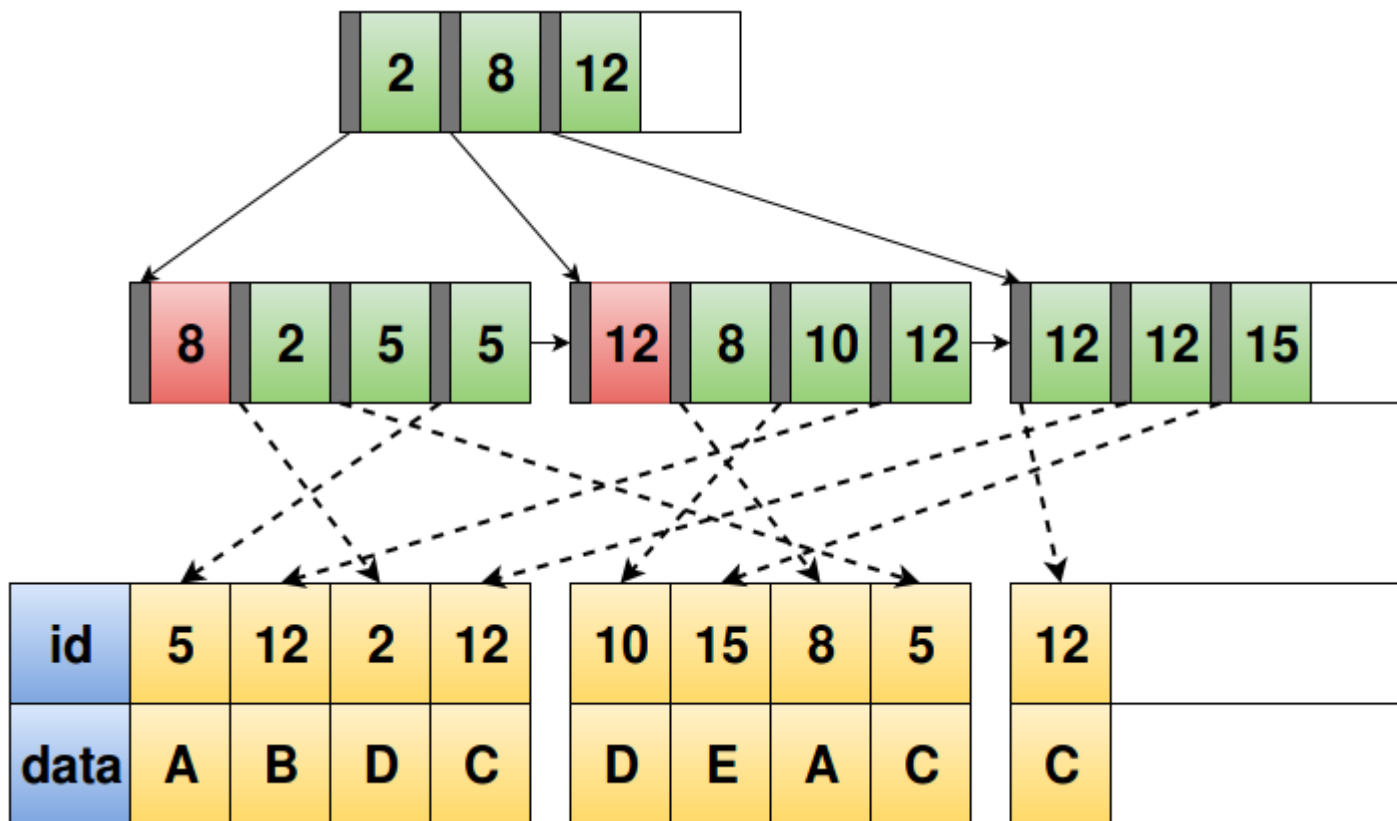
- Allows to combine constraint and covering index to decrease maintenance overhead
 - Size is smaller
 - Inserts are faster
- Allows to add into index data with no suitable opclass
 - + You can store any datatype as a payload
 - - Index ordering does not apply to included columns.

HOT-updates

- Heap-Only Tuples (HOT) accelerate space reuse for most UPDATES and DELETES

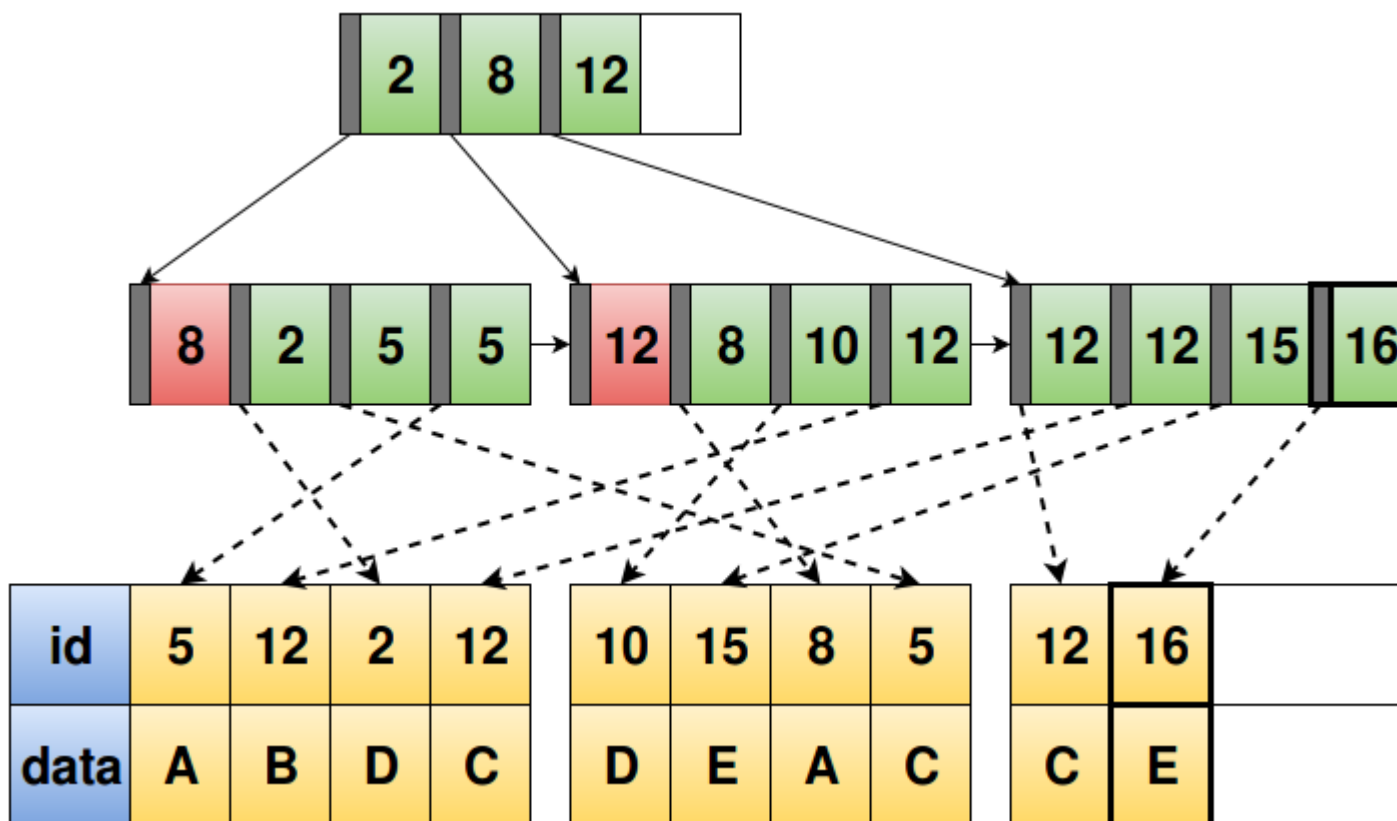


HOT-updates



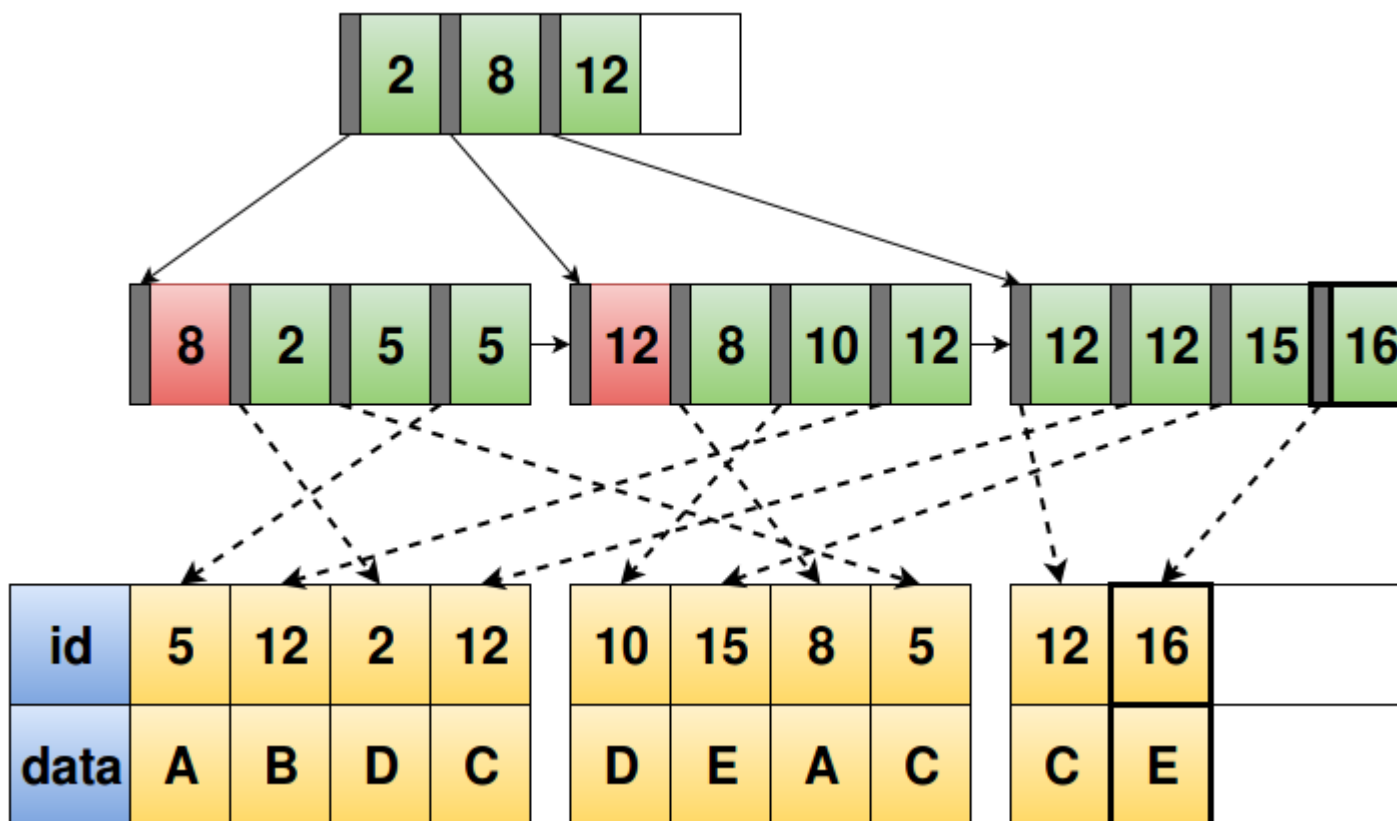
```
UPDATE tbl SET id = 16 WHERE id = 15;
```

HOT-updates



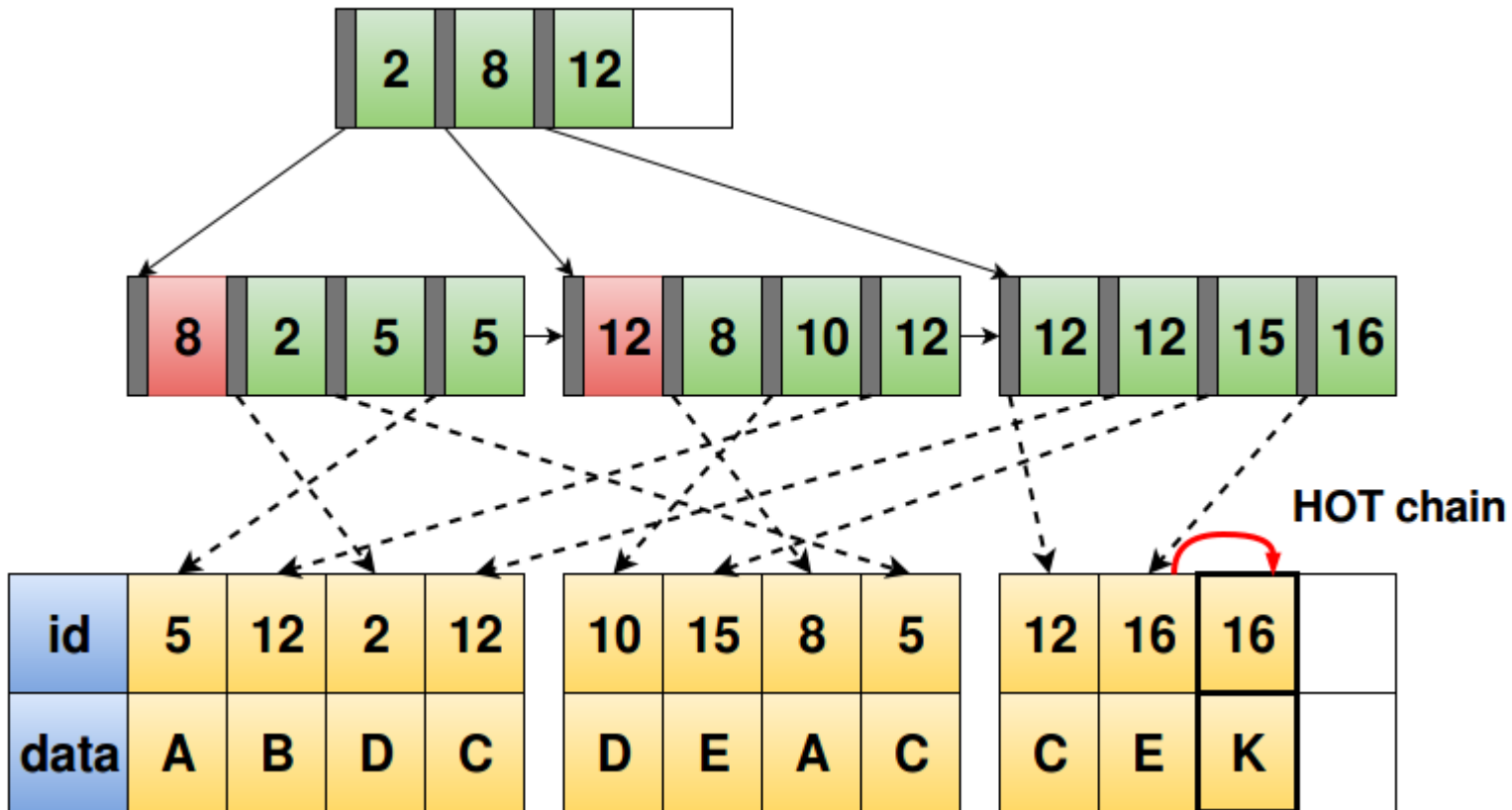
```
UPDATE tbl SET id = 16 WHERE id = 15;
```

HOT-updates



UPDATE tbl SET **data** = **K** WHERE id = 16;

HOT-updates



```
UPDATE tbl SET data = K WHERE id = 16;
```

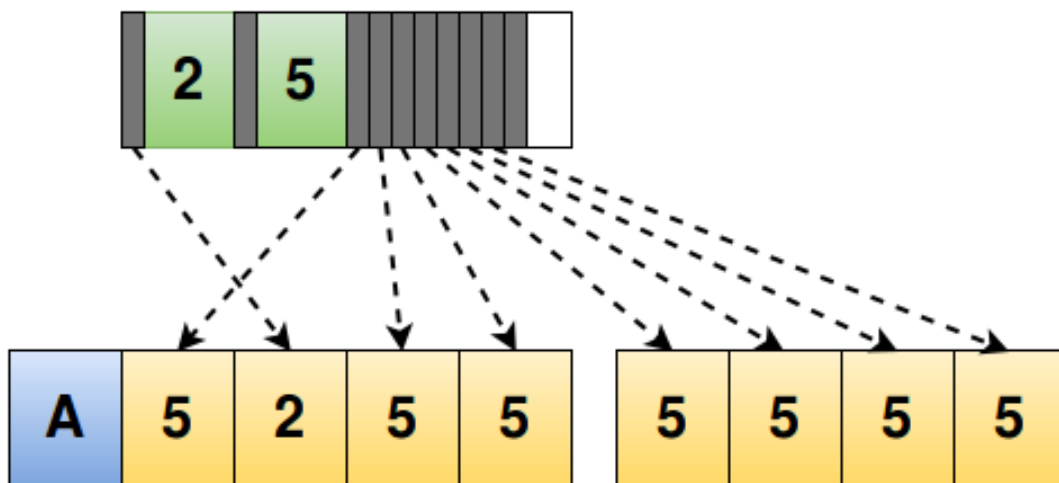
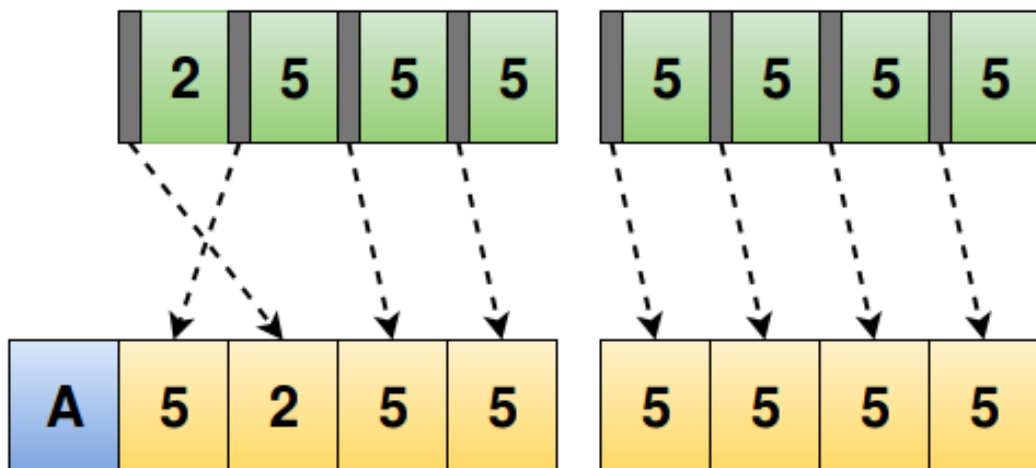
Index with INCLUDED columns

- Covering multicolumn index
 - + Enables index-only scan for READ queries
 - – Disables HOT updates for WRITE queries

DEDUPLICATION



Effective storage of duplicates



Effective storage of duplicates

- Major feature of PostgreSQL 13
- New b-tree parameter: `deduplicate_items`
 - Enabled by default for all user indexes
- Opclass restrictions
 - Does not support numeric, float and container types
- Deduplication overhead is amortized across insertions
 - Only 2% overhead on append-only benchmark

<https://aithub.com/postares/postares/commit/0d861bbb702f8aa05c2a4e3f1650e7e8df8c8c27>

Deduplication performance

- Transparently makes indexes 2.5X — 4X smaller
- For TPCCH benchmark saves 40% of space
 - 5921 MB → 3576 MB
 - Almost no slowdown on index creation

Deduplication TPCB benchmark

indexname	regular_size	deduplication
idx_customer_nationkey	32 MB	10184 kB
idx_lineitem_orderkey	1286 MB	742 MB
idx_lineitem_part_supp	1807 MB	650 MB
idx_lineitem_shipdate	1807 MB	1807 MB
idx_nation_regionkey	16 kB	16 kB
idx_orders_custkey	322 MB	120 MB
idx_orders_orderdate	322 MB	100 MB
idx_partsupp_partkey	172 MB	94 MB
idx_partsupp_suppkey	172 MB	52 MB
idx_supplier_nation_key	2224 kB	704 kB

Microvacuum

- Remove dead index entries before B-Tree page split
- Also known as on-the-fly deletion
- Mark tuples as dead when reading table.
- Then remove them while performing insertion before the page split.

- Allows to delay splits caused by MVCC copies
- In synergy with microvacuum helps to avoid index bloat
- Benchmark with pgbench
 - Old index growth: 10.5 GiB → 19.4 GiB
 - New index growth: 10.5 GiB → 12.7 GiB

What is next?

- KNN for B-tree
 - <https://commitfest.postgresql.org/27/1804/>
- Index Skip Scan
 - <https://commitfest.postgresql.org/27/1741/>
- Index deletion optimizations
 - <https://commitfest.postgresql.org/20/1802/>
- Index-Organized-Tables
- Indexes on partitioned tables
 - Global indexes
 - Global Partitioned Indexes

Thank you! Questions?

