PostgreSQL Indexes

Overview

Boost read performance:

Efficient data structures that speed up data retrieval.

Reduce disk I/O:

By scanning less data, queries become faster.

Trade-off: Slower writes

Indexes need to be updated on INSERT, UPDATE, DELETE operations, which can impact write performance.

Index Types

B-TREE

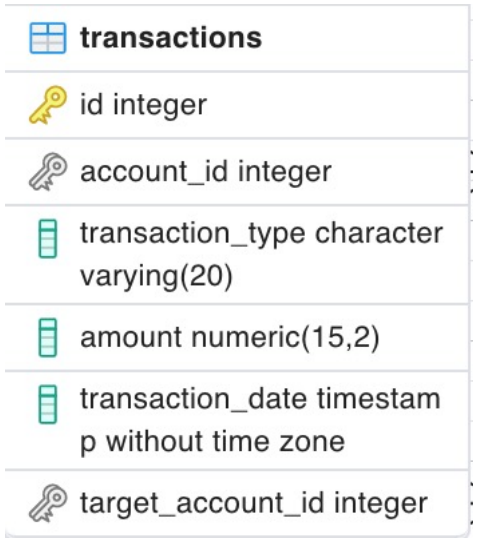
(default)

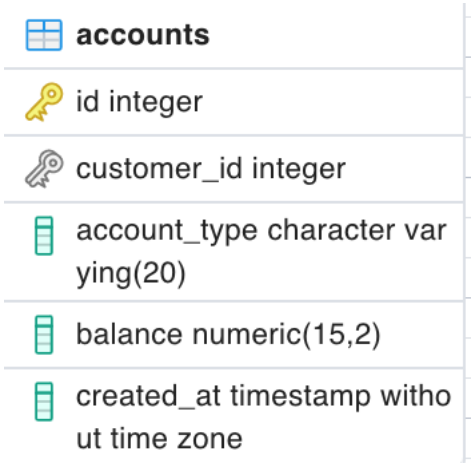
GIN

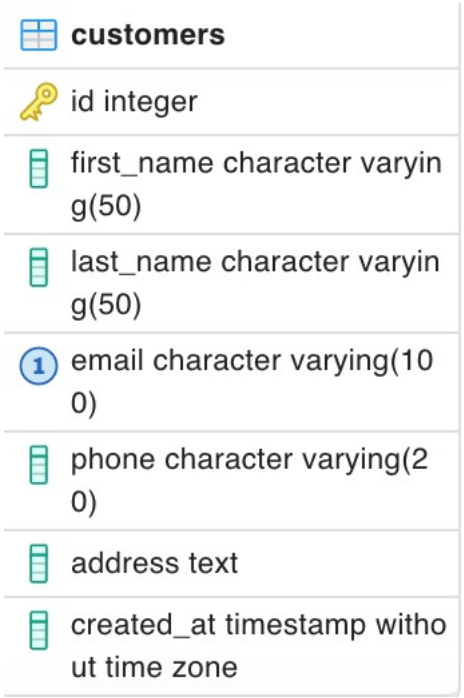
HASH

GiST

Data Model


transactions
id integer
account_id integer
transaction_type character varying(20)
amount numeric(15,2)
transaction_date timestamp without time zone
target_account_id integer


accounts
id integer
customer_id integer
account_type character varying(20)
balance numeric(15,2)
created_at timestamp without time zone

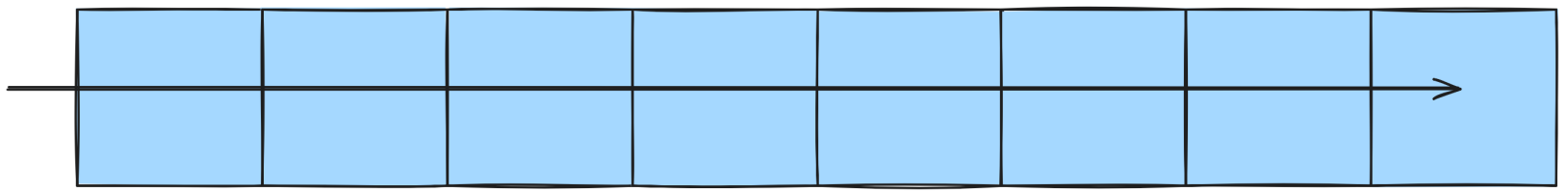

customers
id integer
first_name character varying(50)
last_name character varying(50)
email character varying(100)
phone character varying(20)
address text
created_at timestamp without time zone

Query without index

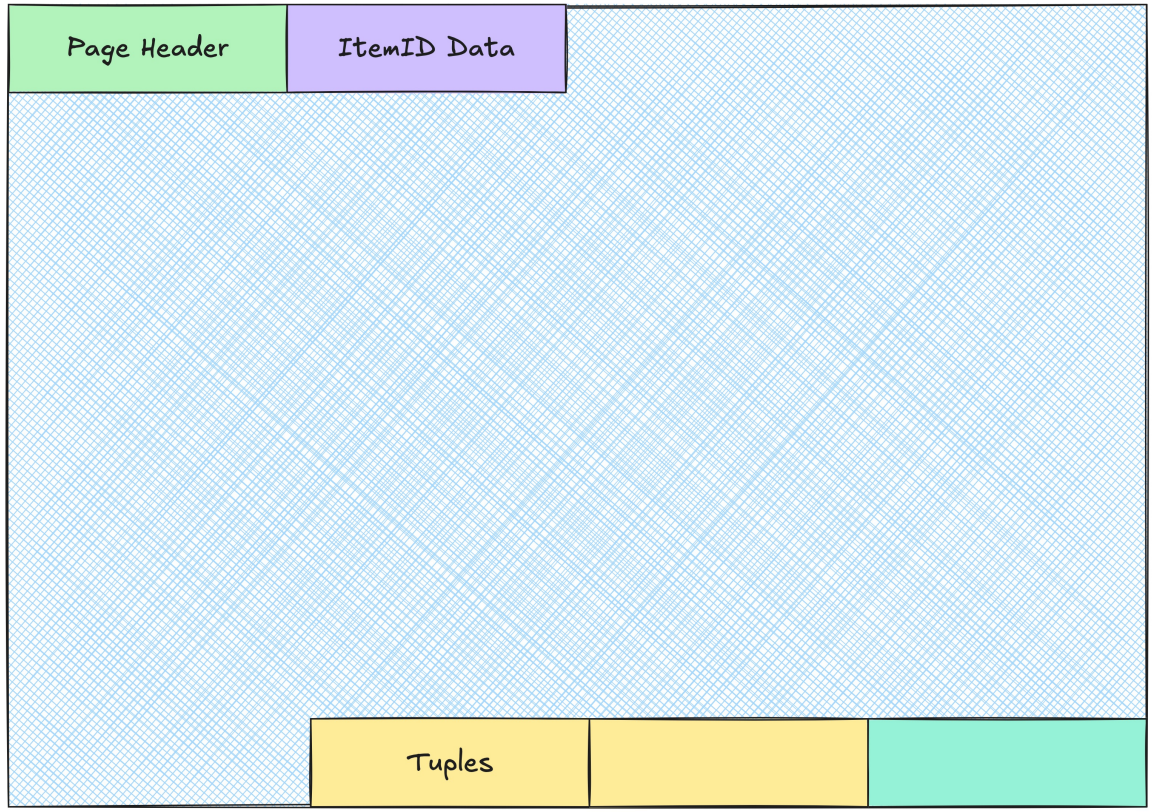
```
EXPLAIN ANALYZE SELECT * FROM transactions WHERE amount > 500;
```

QUERY PLAN text
Gather (cost=1000.00..13785.33 rows=100 width=34) (actual time=62.354..66.524 rows=0 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on transactions (cost=0.00..12775.33 rows=42 width=34) (actual time=58.934..58.935 rows=0 loop.
Filter: (amount > '500'::numeric)
Rows Removed by Filter: 333333
Planning Time: 0.074 ms
Execution Time: 66.645 ms

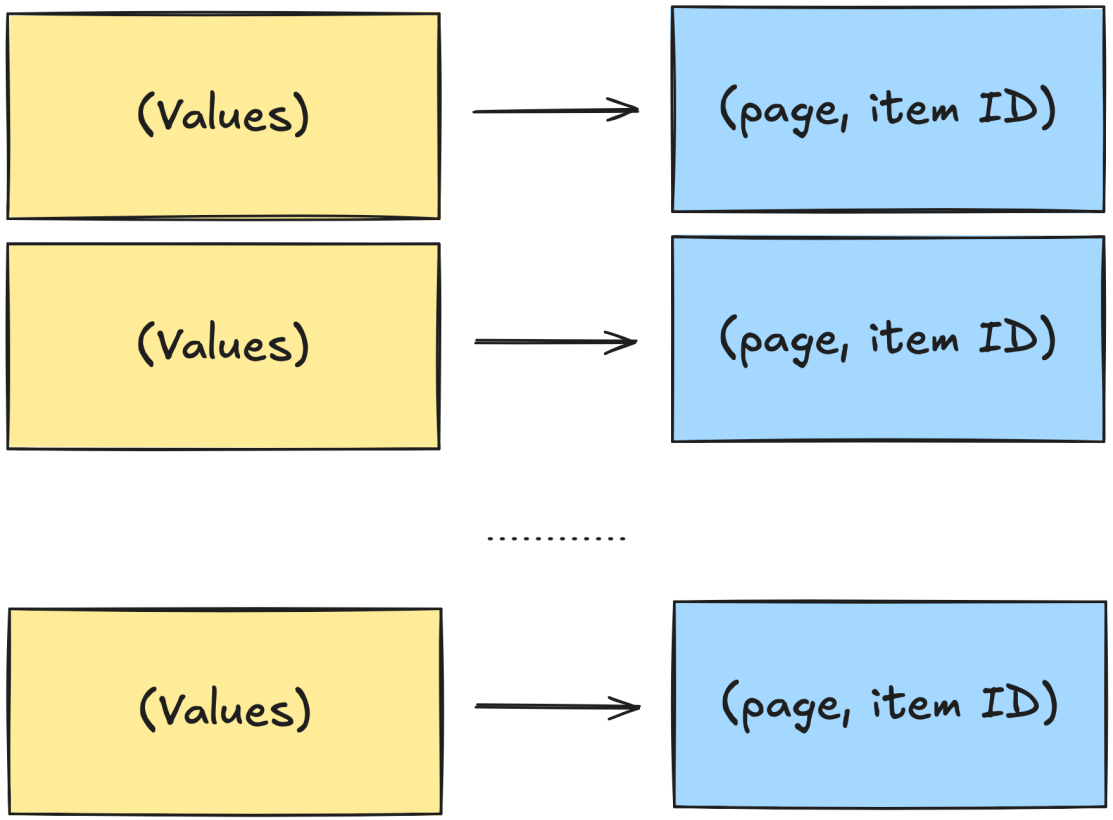
Full Table Scan



How Postgres stores data

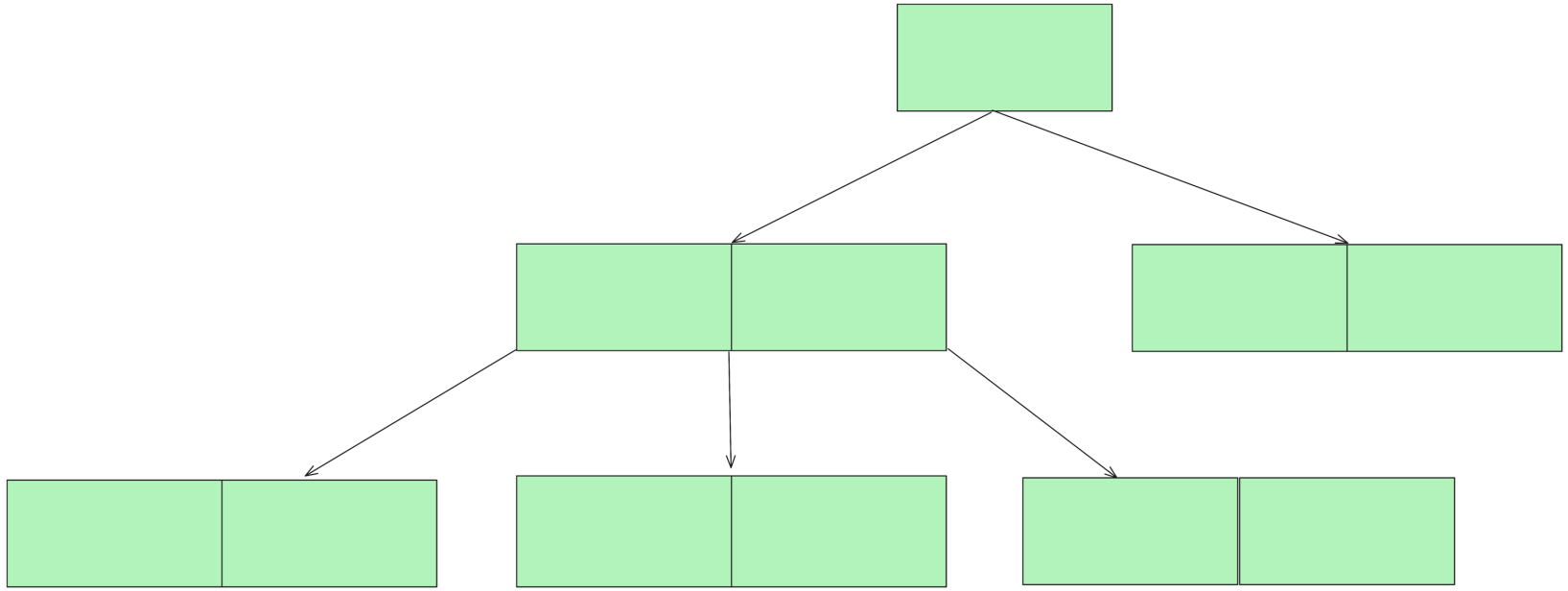


What is an index

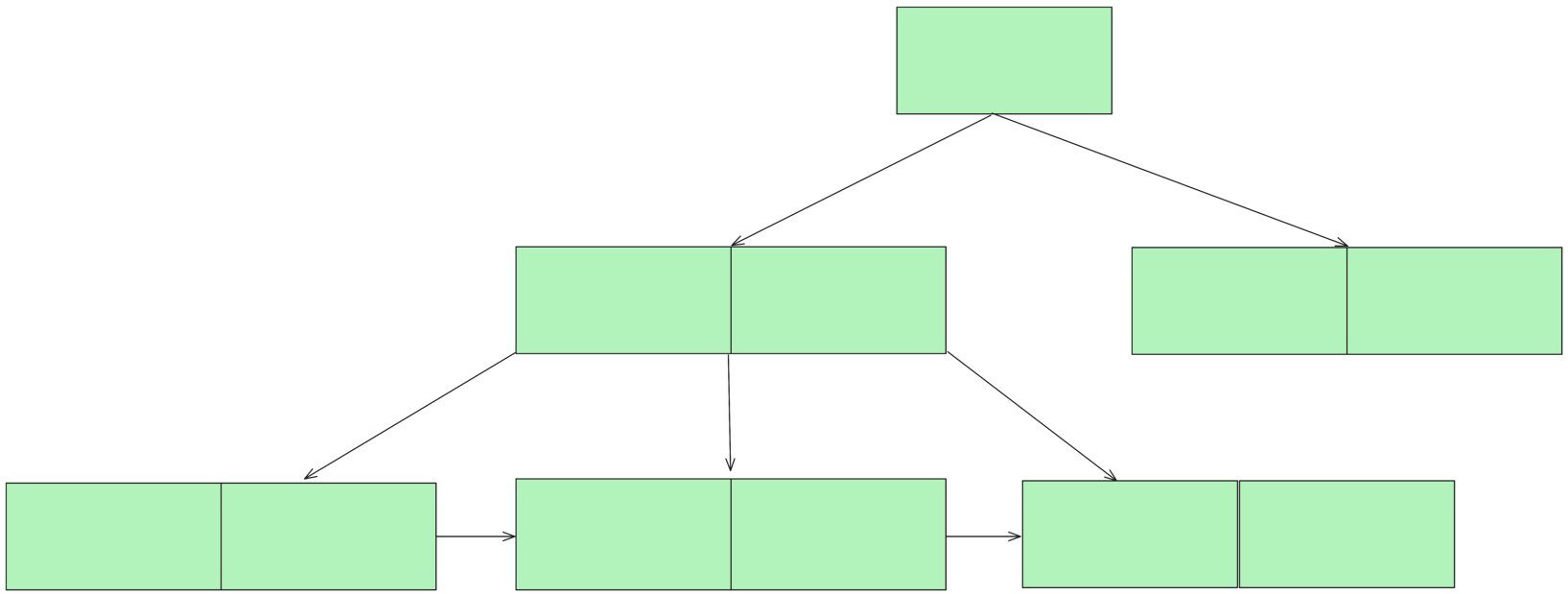


B-TREE

Self-balancing search tree



B+TREE



Query with index

```sql
SELECT * FROM transactions WHERE amount > 500;
```

```sql
CREATE INDEX transaction_amount_idx ON transactions(amount);
```

QUERY PLAN text
Index Scan using transaction_amount_idx on transactions (cost=0.42..8.44 rows=1 width=34) (actual time=0.005..0.005 rows=0 loop..
Index Cond: (amount > '500'::numeric)
Planning Time: 0.320 ms
Execution Time: 0.017 ms

was 66.645 ms

Composite index

```sql
EXPLAIN ANALYZE SELECT amount, transaction_date
    FROM transactions
    WHERE amount > 500 AND transaction_date > NOW() - INTERVAL '10 days';
```

QUERY PLAN text
Gather (cost=1000.00..16903.23 rows=29 width=14) (actual time=80.116..85.207 rows=0 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on transactions (cost=0.00..15900.33 rows=12 width=14) (actual time=76.336..76.336 rows=0
Filter: ((amount > '500'::numeric) AND (transaction_date > (now() - '10 days'::interval)))
Rows Removed by Filter: 333333
Planning Time: 0.166 ms
Execution Time: 85.225 ms

Composite index

```
EXPLAIN ANALYZE SELECT amount, transaction_date
    FROM transactions
    WHERE amount > 500 AND transaction_date > NOW() - INTERVAL '10 days';
```

```
CREATE INDEX transaction_amount_and_date_idx on transactions(amount, transaction_date);
```

QUERY PLAN text
Index Only Scan using transaction_amount_and_date_idx on transactions (cost=0.43..4.45 rows=1 width=14) (actual time=0.007..0.008 rows=0
Index Cond: ((amount > '500'::numeric) AND (transaction_date > (now() - '10 days'::interval)))
Heap Fetches: 0
Planning Time: 0.486 ms
Execution Time: 0.021 ms

was 85.225 ms

Composite index

```
EXPLAIN ANALYZE SELECT amount, transaction_date
    FROM transactions
    WHERE transaction_date > NOW() - INTERVAL '10 days';
```

```
CREATE INDEX transaction_amount_and_date_idx on transactions(amount, transaction_date);
```

QUERY PLAN text
Seq Scan on transactions (cost=0.00..25067.00 rows=290884 width=14) (actual time=0.013..381.238 rows=290714
Filter: (transaction_date > (now() - '10 days'::interval))
Rows Removed by Filter: 709286
Planning Time: 0.062 ms
Execution Time: 396.937 ms

Partial Index

```
CREATE INDEX trasaction_status_idx on trasactions(amount)
where trasaction_status = 'in-progress';
```

”in-progress” – 10%

Reduce the size of the index

Covering Index

```sql
SELECT account_id, transaction_type
FROM transactions WHERE amount > 150;
```

```sql
CREATE INDEX transaction_amount_idx ON transactions(amount);
```

```
Bitmap Heap Scan on transactions  (cost=2447.47..11645.18 rows=130457 width=12) (actual time=29.154..74.158 rows=134064 loops=1)
  Recheck Cond: (amount > '150'::numeric)
  Heap Blocks: exact=7567
  ->  Bitmap Index Scan on transaction_amount_idx  (cost=0.00..2414.85 rows=130457 width=0) (actual time=27.532..27.533 rows=134064 loop…
        Index Cond: (amount > '150'::numeric)
Planning Time: 0.219 ms
Execution Time: 81.909 ms
```

Covering Index

```sql
SELECT account_id, transaction_type
FROM transactions WHERE amount > 150;
```

```sql
CREATE INDEX transaction_amount_covering_idx ON transactions(amount)
INCLUDE (account_id, transaction_type);
```

```
Index Only Scan using transaction_amount_covering_idx on transactions  (cost=0.42..4871.42 rows=130457 width=12)
  Index Cond: (amount > '150'::numeric)
  Heap Fetches: 0
Planning Time: 0.085 ms
Execution Time: 38.894 ms
```

Index on Join Column

```sql
SELECT c.first_name, c.last_name, a.created_at FROM customers c
    JOIN accounts a ON a.customer_id = c.id
    WHERE c.id = 42;
```

QUERY PLAN text
Nested Loop (cost=0.29..670.98 rows=1 width=21) (actual time=0.025..2.434 rows=1 loops=1)
-> Index Scan using customers_pkey on customers c (cost=0.29..8.31 rows=1 width=17) (actual time=0.012..0.014 rows=1
Index Cond: (id = 42)
-> Seq Scan on accounts a (cost=0.00..662.66 rows=1 width=12) (actual time=0.011..2.417 rows=1 loops=1)
Filter: (customer_id = 42)
Rows Removed by Filter: 33332
Planning Time: 0.220 ms
Execution Time: 2.456 ms

CODEUS_

Index on Join Column

```sql
SELECT c.first_name, c.last_name, a.created_at FROM customers c
    JOIN accounts a ON a.customer_id = c.id
    WHERE c.id = 42;
```

```sql
CREATE INDEX accounts_customer_id_idx ON accounts(customer_id);
```

QUERY PLAN text
Nested Loop (cost=0.58..16.62 rows=1 width=21) (actual time=0.034..0.035 rows=1 loops=1)
-> Index Scan using customers_pkey on customers c (cost=0.29..8.31 rows=1 width=17) (actual time=0.009..0.009 rows=1 loops=1)
Index Cond: (id = 42)
-> Index Scan using accounts_customer_id_idx on accounts a (cost=0.29..8.31 rows=1 width=12) (actual time=0.022..0.022 rows=1
Index Cond: (customer_id = 42)
Planning Time: 1.463 ms
Execution Time: 0.054 ms

Function Based Index

```sql
CREATE INDEX idx_customers_lower_first_name ON customers(email);
```

```sql
SELECT first_name FROM customers WHERE lower(email) = 'test@example.com';
```

QUERY PLAN text
Seq Scan on customers (cost=0.00..1052.99 rows=167 width=6) (actual time=38.595..38.596 rows=0 l
Filter: (lower((email)::text) = 'test@example.com'::text)
Rows Removed by Filter: 33333
Planning Time: 0.338 ms
Execution Time: 38.625 ms

```sql
CREATE INDEX idx_customers_lower_first_name ON customers(lower(email));
```

Query With Casting

```sql
SELECT * FROM customers WHERE CAST(created_at AS DATE) = '2025-12-13';
```

```sql
CREATE INDEX idx_customers_date ON customers(created_at);
```

Seq Scan on customers (cost=0.00..1052.99 rows=167 width=101) (actual time=4.758..4.758 rows=0 loops…
Filter: ((created_at)::date = '2025-12-13'::date)
Rows Removed by Filter: 33333
Planning Time: 0.051 ms
Execution Time: 4.772 ms

```sql
CREATE INDEX idx_customers_date ON customers((CAST(created_at AS DATE)));
```

Query With Like

```sql
SELECT * FROM customers WHERE first_name LIKE 'sa%';
```

```sql
CREATE INDEX idx_customers_first_name ON customers(first_name);
```

Seq Scan on customers (cost=0.00..969.66 rows=1 width=101) (actual time=2.984..2.984
Filter: ((first_name)::text ~~ 'sa%'::text)
Rows Removed by Filter: 33333
Planning Time: 1.319 ms
Execution Time: 2.996 ms

```sql
CREATE INDEX idx_customers_first_name ON customers(first_name text_pattern_ops);
```

Low Selectivity

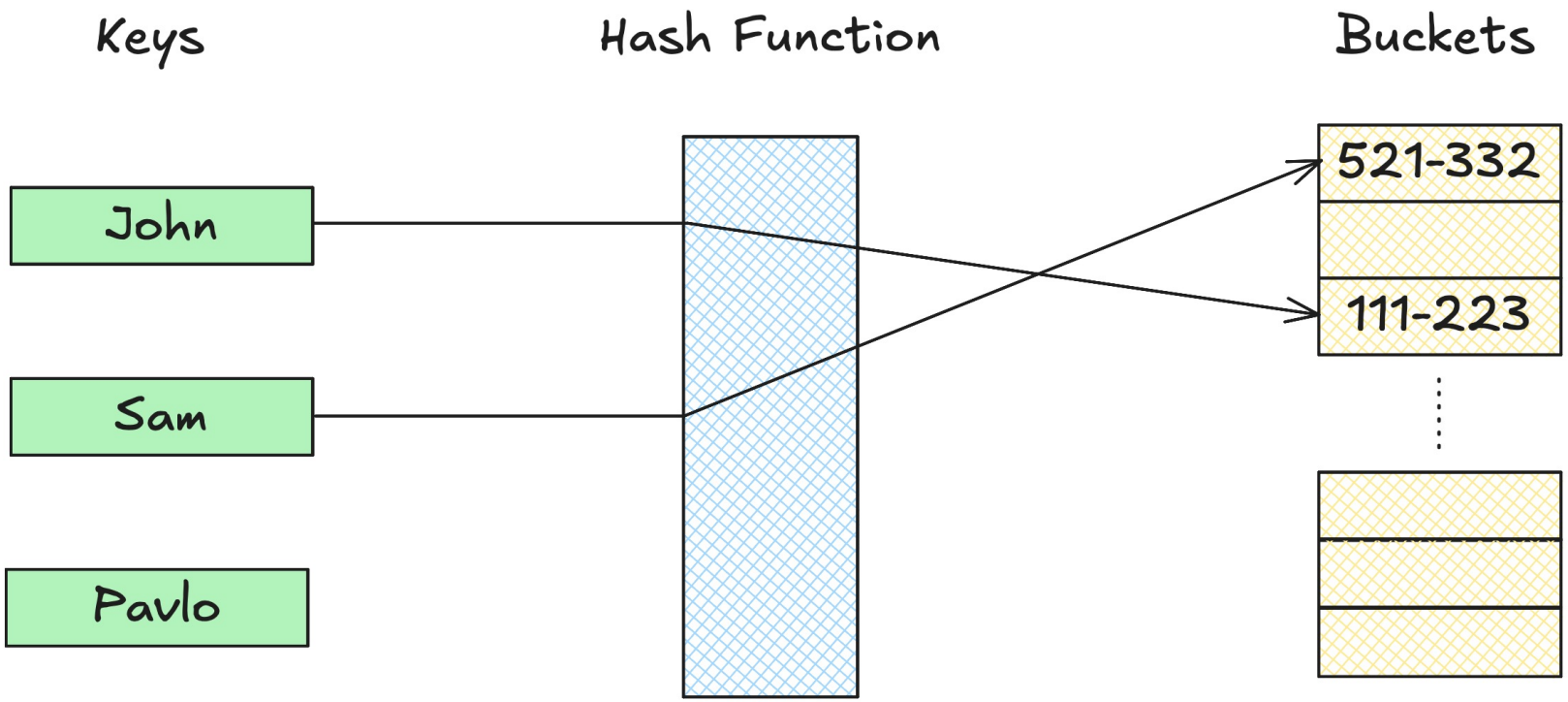
```sql
SELECT * FROM transactions
        WHERE transaction_type = 'deposit';
```

The transaction type 'deposit' has a frequency of 80% among all transactions.

```sql
CREATE INDEX idx_transactions_type ON transactions(transaction_type);
```

Seq Scan on transactions (cost=0.00..20067.00 rows=798667 width=34) (actual time=0.007..118.292 rows=799830
Filter: ((transaction_type)::text = 'deposit'::text)
Rows Removed by Filter: 200170
Planning Time: 0.227 ms
Execution Time: 148.834 ms

Hash Index



Hash Index

```sql
SELECT first_name FROM customers WHERE first_name = 'David';
```

```sql
CREATE INDEX idx_customers_first_name_hash ON customers USING hash (first_name);
```

Index Scan on idx_customers_first_name_hash

IMPORTANT: Hash index works **only** with the = operator.

Thank you

- Author: Dmytro Halchenko
- My LinkedIn: https://www.linkedin.com/in/dmytro-halchenko-0307751b6/
- Date: March 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn