PostgreSQL Essentials: Index Structures, Cursors, CRUD Operations, and Query Analysis

Ali Emre Pamuk

March 2025

Contents

1 Introduction

PostgreSQL is a powerful, open-source relational database system. Understanding core features like indexing, cursors, data operations, and performance analysis tools is essential for designing efficient and scalable applications.

2 Index Structures in PostgreSQL

Indexes help databases retrieve data faster. They work similarly to an index in a book. PostgreSQL supports several index types tailored to specific use cases.

2.1 B-Tree Index

Best for: Equality and range queries (e.g., =, ¡, ¿, BETWEEN).

```
CREATE INDEX idx_users_email ON users(email);
SELECT * FROM users WHERE email = ’ali@example.com’;
```

2.2 Hash Index

Best for: Fast equality comparisons.

```
CREATE INDEX idx_users_hash_email ON users USING hash(email);
```

2.3 GIN Index

Best for: Arrays, JSONB, and full-text search.

```
CREATE INDEX idx_posts_tags ON posts USING gin(tags);
SELECT * FROM posts WHERE tags @> ARRAY[’urgent’];
```

2.4 GiST Index

Best for: Geospatial data and range types.

```
CREATE INDEX idx_locations_geom ON locations USING gist(geom);
```

2.5 BRIN Index

Best for: Very large tables with naturally ordered data.

```
CREATE INDEX idx_logs_created_at ON logs USING brin(created_at);
```

2.6 Expression and Partial Indexes

```
CREATE INDEX idx_lower_email ON users(LOWER(email));
CREATE INDEX idx_active_users ON users(last_login) WHERE is_active = true;
```

3 Cursors in PostgreSQL

Cursors enable processing query results one row at a time. Useful for large result sets and procedural logic.

3.1 Cursor Lifecycle

1. Declare the cursor
2. Open the cursor
3. Fetch data
4. Close the cursor

3.2 Example: Looping Through Orders

```
DO $$
DECLARE
  rec RECORD;
  cur CURSOR FOR SELECT id, total FROM orders WHERE status = 'pending';
BEGIN
  OPEN cur;
  LOOP
    FETCH cur INTO rec;
    EXIT WHEN NOT FOUND;
    RAISE NOTICE 'Order␣ID:␣%,␣Total:␣%', rec.id, rec.total;
  END LOOP;
  CLOSE cur;
END $$;
```

4 CRUD Operations

CRUD represents the four basic operations on database data: Create, Read, Update, and Delete.

4.1 Create

```
INSERT INTO users (name, email, age) VALUES ('Alice', 'alice@example.com',
    30);
```

4.2 Read

```
SELECT * FROM users;
SELECT * FROM users WHERE age > 25;
```

4.3 Update

```
UPDATE users SET age = 31 WHERE email = ’alice@example.com’;
```

4.4 Delete

```
DELETE FROM users WHERE email = ’alice@example.com’;
```

5 EXPLAIN: Query Execution Plans

`EXPLAIN` helps understand how PostgreSQL plans to execute a query. Combine with `ANALYZE` to see actual performance.

5.1 Basic Usage

```
EXPLAIN SELECT * FROM users WHERE email = ’alice@example.com’;
```

Sample Output:

```
Index Scan using idx_users_email on users  (cost=0.29..8.30 rows=1 width
   =64)
  Index Cond: (email = ’alice@example.com’)
```

5.2 With ANALYZE

```
EXPLAIN ANALYZE SELECT * FROM users WHERE email = ’alice@example.com’;
```

Sample Output:

```
Index Scan using idx_users_email on users  (cost=0.29..8.30 rows=1 width
   =64)
  Index Cond: (email = ’alice@example.com’)
  (actual time=0.025..0.026 rows=1 loops=1)
Planning Time: 0.098 ms
Execution Time: 0.031 ms
```

5.3 Sample Output Explanation

- **Seq Scan:** Scans the whole table.
- **Index Scan:** Uses an index to access rows.
- **Cost:** Estimated cost of query execution.
- **Rows:** Estimated number of rows returned.
- **Actual time:** Real timing information (with ANALYZE).

6 Full Example: Orders System

Combining all concepts into a practical example.

6.1 Schema and Indexes

```
CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL,
  status TEXT,
  total NUMERIC,
  tags TEXT[],
  created_at TIMESTAMP DEFAULT now()
);
CREATE INDEX idx_orders_status ON orders(status);
CREATE INDEX idx_orders_tags ON orders USING gin(tags);
```

6.2 Insert Data

```
INSERT INTO orders (user_id, status, total, tags)
VALUES
(1, ’pending’, 200.00, ARRAY[’express’]),
(2, ’completed’, 500.00, ARRAY[’gift’]),
(3, ’pending’, 800.00, ARRAY[’priority’]);
```

6.3 Cursor Processing

```
DO $$
DECLARE
  rec RECORD;
  cur CURSOR FOR SELECT id, total FROM orders WHERE status = ’pending’;
BEGIN
  OPEN cur;
  LOOP
    FETCH cur INTO rec;
    EXIT WHEN NOT FOUND;
    IF rec.total > 500 THEN
      UPDATE orders SET status = 'approved' WHERE id = rec.id;
    END IF;
  END LOOP;
  CLOSE cur;
END $$;
```

6.4 Query Analysis

```
EXPLAIN ANALYZE SELECT * FROM orders WHERE status = 'approved';
```

6.5 Result Interpretation

If the index is used, the plan should show:

```
Index Scan using idx_orders_status on orders  (cost=0.29..8.30 rows=1
   width=64)
  Index Cond: (status = 'approved')
  (actual time=0.030..0.032 rows=1 loops=1)
Planning Time: 0.110 ms
Execution Time: 0.035 ms
```

Otherwise, a sequential scan will appear, indicating a possible optimization opportunity.

Conclusion

This document provided a hands-on, detailed look into core PostgreSQL features. Indexes, cursors, CRUD operations, and query analysis tools are foundational to efficient database design. The full example tied together all concepts for practical understanding.