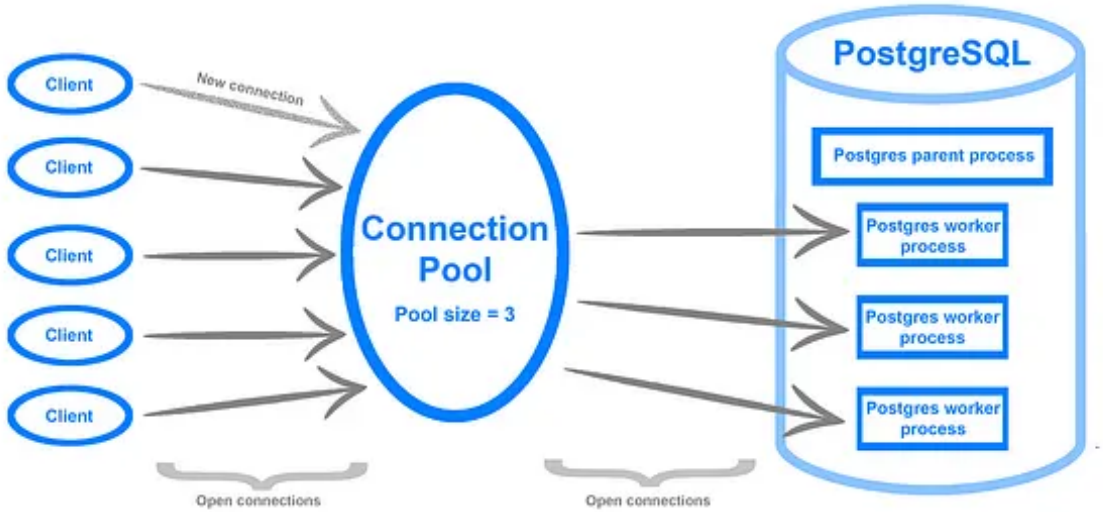
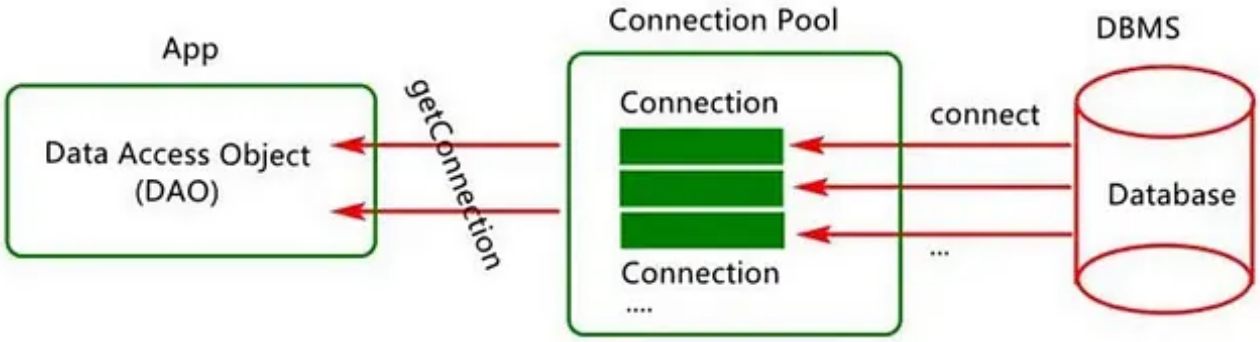


With Connection Pool



Database Connection Pool

Sujoy Nath Follow 4 min read · Oct 1, 2023

 314  3   

A database connection pool is a cache of database connections maintained so that the connections can be reused when needed. It is a common optimization technique used in applications that interact with databases to improve performance and manage resource utilization. Instead of opening and closing a new database connection for every database operation, a connection pool keeps a set of connections open and ready for use.

Here's how a database connection pool works:

- **Initialization:** When an application starts, a fixed number of database connections are created and added to the pool.

- **Connection Request**: When an application needs to perform a database operation, it requests a connection from the pool.
- **Usage**: The application uses the borrowed connection to perform the database operation.
- **Return**: After the operation is complete, the connection is returned to the pool instead of being closed. It is marked as available for reuse.
- **Reuse**: The next time the application needs a database connection, it can request one from the pool. If a connection is available, it is reused; otherwise, the application waits until a connection becomes available.

Here are some of the key advantages:

- **Improved Performance**: Connection pooling reduces the overhead associated with opening and closing database connections for every database operation. Reusing existing connections is faster than establishing new ones, leading to improved query response times and overall application performance.
- **Resource Efficiency**: Database connections can be a limited and valuable resource. Connection pooling ensures that connections are used

efficiently by minimizing the number of idle and unused connections. This allows more users to access the database simultaneously without overloading it.

- **Connection Reuse:** Reusing existing connections means that the database doesn't have to create a new session for every user or request. This reduces the load on the database server and can lead to significant performance improvements, especially in high-traffic applications.
- **Connection Management**: Connection pools handle the management of connections, including acquiring, releasing, and maintaining them. Developers don't have to worry about the intricacies of connection management, which simplifies application code and reduces the risk of resource leaks.
- **Concurrency Control**: Connection pools often include mechanisms for controlling the number of simultaneous connections to the database. This helps prevent resource contention and database bottlenecks by limiting the number of concurrent queries.
- **Fault Tolerance:** Many connection pool implementations include features for monitoring and managing database connections. In the event of a connection failure, some connection pools can automatically

reconnect or replace failed connections, enhancing application resilience.

- **Customization**: Connection pooling libraries typically offer configuration options that allow developers to fine-tune pool behavior to match application requirements. This includes setting the maximum pool size, timeout values, and other parameters.
- **Connection Recycling**: Database connection pools can recycle and refresh connections periodically to prevent issues like stale connections. This ensures that connections are always in a healthy state.
- **Reduced Overhead:** Opening and closing database connections can introduce overhead due to authentication, network setup, and connection teardown. Connection pooling reduces this overhead by keeping connections open and ready for use.
- **Compatibility**: Connection pooling is a well-established practice and is supported by most database systems and programming languages. It is compatible with a wide range of database drivers and libraries.
- **Scalability**: Connection pooling is essential for scaling applications, as it allows you to efficiently manage database connections across multiple

application instances or servers.

Here's an example of setting up a PostgreSQL database connection pool using Node.js with the pg-pool library:

```
const { Pool } = require('pg');

// Configure the database connection pool
const pool = new Pool({
  user: 'your_username',
  host: 'your_database_host',
  database: 'your_database_name',
  password: 'your_password',
  port: 5432, // PostgreSQL default port
  max: 10, // Maximum number of connections in the pool
  idleTimeoutMillis: 30000, // How long a connection is allowed to be idle (in m
  connectionTimeoutMillis: 2000, // How long to wait for a connection to become
});

// Example database query using the connection pool
pool.query('SELECT * FROM your_table_name', (err, result) => {
  if (err) {
    console.error('Error executing query', err);
  } else {
    console.log('Query result:', result.rows);
  }

  // Release the connection back to the pool
  pool.end(); // Note: In a production application, you would typically release
```

```
});
```

In this example, a PostgreSQL connection pool is created with a maximum of 10 connections. When a query is executed, it borrows a connection from the pool, performs the query, and then releases the connection back to the pool for reuse. The pool takes care of managing and reusing connections, which helps improve the efficiency and performance of database interactions in

Database | Database Design | Database Connection | Postgres

Written by Sujoy Nath

52 followers · 3 following

Full Stack Engineer experienced in Node.js, React, TypeScript, and blockchain. Write about Crypto currencies, Backend Technologies and Databases

Gaurav Goel
The Medium Blog
Reshma Bidikar

Find writers and publications to follow.

See suggestions

Responses (3)

Ishu

What are your thoughts?

Jasurbek Kalandarov
Sep 25

What if there are more than 10 requests at the same time? Will the 11th request waits for the connection to be available? or Does PG creates another pool?

1 reply Reply

Shreyansh Tiwari
Apr 10

Great article

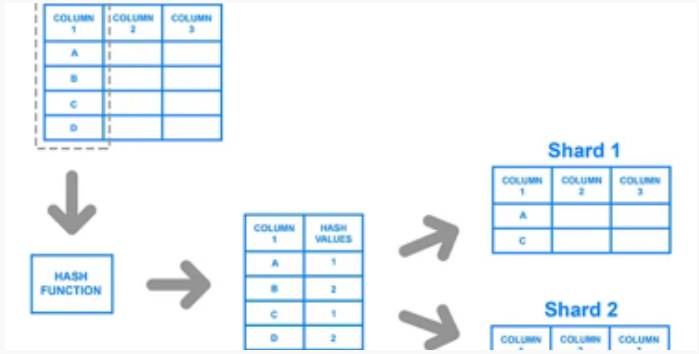
Reply

Swasti Prakash Bhanja
Mar 4

well written

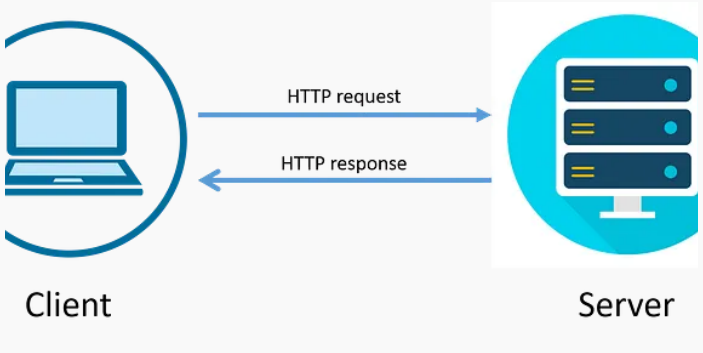
Reply

More from Sujoy Nath



Sujoy Nath

Database Sharding



Sujoy Nath

Request Response Model, Usages,

Database sharding, also known as data sharding or distributed database sharding, i...

Oct 1, 2023 63 1

The Request-Response model is a fundamental architectural pattern in backen...

Oct 3, 2023 3

Sujoy Nath

Request Polling With Examples

Request polling, often referred to simply as polling, is a client-server communication...

Oct 7, 2023 2

Sujoy Nath

Database Indexing and the Various Types of Scanning

Indexing in a database is a data structure that enhances the speed of data retrieval...

Oct 3, 2023 1

See all from Sujoy Nath

Recommended from Medium

Noah Byteforge

Kafka + Postgres in 2025: Outbox, Idempotency, and Exactly-Once...

A production-first guide to Kafka + Postgres in 2025. Learn how Transactional Outbox,...

Oct 19 30

In Javarevisited by Firas Ahmed

Why Use Java Enums instead of static final Constants?

Enums are one of the key features in Java, used extensively in programs. They represe...

Oct 2 263 7

The Latency Gambler

I Interviewed 20+ Engineers. Here's

Java Interview

Why Senior Developers Use

I Interviewed 20+ Engineers. Here's Why Most Can't Code

Over the past year as a Senior Software Engineer at a B2B SaaS company, I've...

Sep 9 2.9K 95

Why Senior Developers Use @EventListener Instead of Direct...

When you start with Spring Boot, it's natural to structure your app by calling methods...

Oct 18 52 4

nemo

Why I Stopped Writing Retry Logic Forever—and How Temporal Fixe...

I still remember the night a simple payment flow broke in production. We had everything...

Oct 17

Mathildaduku

How to Design Idempotent APIs Safely: What to Cache and What t...

Building resilient APIs takes more than adding an Idempotency-Key. Caching the...

Jun 17 31 2

See more recommendations