Deep Dive into Data Storage in Databases: The InnoDB Engine

13 min read · Apr 1, 2024

Sameer Soin Follow

Listen Share More

Introduction

The way databases store data on disk is both intricate and fascinating. It combines principles of computer science, data management, and software engineering. In this deep dive, we'll expand our focus on InnoDB, a popular storage engine for MySQL, and explore its sophisticated mechanisms for data storage, including data file structures, write-ahead logging, buffer pool management, and MVCC.

Focusing solely on the actual data storage mechanism on the disk in the InnoDB storage engine, we'll delve into how data is physically organized and managed on the storage media. This aspect is crucial for understanding the efficiency and performance characteristics of InnoDB.

Data Storage in InnoDB

1. Tablespaces

- **Role:** Tablespaces in InnoDB serve as the top-level structure for data storage. They are logical units that contain the actual physical files where data is stored.
- **Types:** InnoDB utilizes a system tablespace for the entire database and can also use file-per-table tablespaces for individual tables.

Understanding Tablespaces

- InnoDB stores data within structures called tablespaces. A tablespace in InnoDB can be a single file (file-per-table tablespaces) or comprise multiple files (system
  files are not simple text files but are binary files specially
  ient data access.

  highest-level organizational unit in many database systems.
  container that groups related database objects, such as tables and indexes. You can think of it as a conceptual “bookshelf” that holds various “books” (data files) related to each other.

2. Data Files

- **Physical Storage:** The actual data in InnoDB tablespaces is stored in one or more data files. These files are binary and reside on the disk.
- **File Structure:** These data files are composed of a series of pages or blocks, which are the fundamental units of storage in InnoDB.

Data Files

- Within a tablespace, data is physically stored in one or more **data files**. These files are the actual binary files on the disk. In InnoDB, you can have different setups for these data files:
- **System Tablespace**: A shared tablespace for multiple tables, typically consisting of one or more files.
- **File-Per-Table Tablespaces:** Separate, independent files for each table.
- **General Tablespaces:** A single file that can hold multiple tables.
- Each data file in a tablespace corresponds to physical storage on the disk but is managed logically by the database system.

The Anatomy of an InnoDB Data File

- Each data file consists of pages, typically 16KB in size. The pages are the basic units of data storage in InnoDB and are grouped into extents (collections of contiguous pages). These pages store various types of data:
- **Data Pages:** Contain the actual table data.
- **Index Pages:** Store B-tree structures for indexes.

- **Undo Pages**: Used for storing data necessary to roll back transactions.

System Pages: Include information about the tablespace itself.

ze of a page in InnoDB is 16KB, though this can be

- **Structure**: Each page can store one or several rows, depending on the size of the rows. A page is organized into a page header, data area, and page trailer.
- **Types of Pages**: InnoDB uses different types of pages for different purposes, such as data pages (for storing table data), undo log pages (for transaction rollback information), and index pages (for indexes).

Pages

- A data file is further divided into **pages**. A page is the basic unit of data storage in InnoDB and is a fixed-size block. The default page size in InnoDB is 16KB, but it can be configured. These pages are where the actual rows of your tables or index entries are stored.
- When the database system reads or writes data, it does so one page at a time.
- Each page can contain multiple rows, part of a row (in the case of large row sizes), or other information such as index entries.

Blocks

- The term **block** generally refers to the physical unit of storage at the OS or disk level. In the context of database systems like InnoDB, the term “page” and “block” are often used interchangeably, both referring to that fixed-size unit of data (16KB by default in InnoDB). However, in some contexts, “block” might refer to a lower-level storage unit, depending on the context and the specific storage system.
- The relationship between tablespaces, data files, pages, and blocks in a database environment, particularly in systems like InnoDB of MySQL, forms a hierarchy that defines how data is physically stored on disk. Understanding this hierarch is crucial for comprehending the internal workings of a database management system. Let’s break down these components and their interrelationships:

Relationship and Interaction

[illegible]aces > Data Files > Pages (Blocks)

[illegible]el serves a purpose in the organization and management of

[illegible]al Management: While tablespaces and data files are logical constructs for managing storage, pages represent the physical allocation of this storage.

- **I/O Operations:** Data is read and written at the page level, making these operations efficient. The database engine reads and writes entire pages, even if only a single row is needed or modified.

In summary, tablespaces are logical containers for data files, data files are physical files on disk composed of pages (or blocks), and pages are the fundamental units where the actual data resides. This hierarchy helps in efficient data storage, retrieval, and management, playing a critical role in the performance and organization of the database.

4. Clustered Index (Primary Key Index)

- **Data Organization**: InnoDB organizes table data based on the clustered index, typically the primary key. This means the actual data rows are stored in the order of the primary key.

- **Benefits:** This organization allows faster access to rows when queries are based on the primary key. It also helps in range scans and sorted results based on the primary key.

5. Secondary Indexes

- **Reference to Clustered Index**: Secondary indexes in InnoDB store the primary key value for each row, rather than a direct pointer to the physical location of the row. This design choice has implications for how secondary index lookups are performed.

6. Write-Ahead Logging (WAL)

- **Principle**: WAL is a fundamental concept in database systems used for ensurin data integrity and durability. Before any changes are made to the actual data on disk, these changes are first logged in a special log file.

- **Purpose:** This approach ensures that in the event of a crash or failure, the database can recover by replaying these logs, thus maintaining the ACID
atabase.

changes are made to the actual data pages, they are recorded
rite-ahead logging approach ensures data integrity and aids

7. Data and Index Page Management

- **Inserts, Updates, and Deletes:** When data is inserted, updated, or deleted, InnoDB modifies the pages in the buffer pool. These changes are then flushed to disk in a controlled manner, optimizing I/O operations.
- **Page Splitting:** When a page becomes full (due to inserts), it is split, and the data may be redistributed, maintaining the B-tree structure of indexes.

8. File Extensions and Auto-Extension

- **Data File Management:** InnoDB data files can be set to extend automatically when they become full. This auto-extension feature helps in managing disk space efficiently, ensuring that the database can grow without manual intervention.

9. Data Compression

- **Reducing Disk Space Usage:** Tables and indexes can be compressed to save disk space, which is particularly beneficial for I/O-bound workloads and large datasets.

10. Segment and Extent Management

- **Segments:** Within a tablespace, InnoDB manages storage in segments (for different types of data, like data or indexes).
- **Extents:** Segments are divided into extents, which are groups of contiguous pages. Extents improve the efficiency of space management.

Conclusion

In summary, the data storage mechanism in InnoDB is a sophisticated blend of logical structures (like tablespaces and indexes) and physical storage components (like data files and pages). This blend ensures data integrity, optimizes performan and provides efficient space management on the disk. The careful management of

pages, along with the use of clustered and secondary indexes, allows InnoDB to deliver high-performance data storage and retrieval capabilities.

ınd Buffer Pool

› the intricate details of logging mechanisms within database (DBMS), focusing on how these mechanisms contribute to ecovery, and the overall management of transactions. The narrative underscores the importance of various types of log files, including Write-Ahead Logging (WAL), redo logs, and undo logs, and their pivotal roles in ensuring the consistency and reliability of database operations. Here's a more detailed summary of the key technical points:

Core Concepts of Database Logging

- **Critical Nature of Logs:** Logs are essential for the durability of databases, enabling systems to recover from crashes by replaying or undoing transactions. The discussion highlights three primary types of logs: WAL, redo logs, and undo logs.

Write-Ahead Logging (WAL)

- **Principle:** WAL is a fundamental concept in database systems used for ensuring data integrity and durability. Before any changes are made to the actual data on disk, these changes are first logged in a special log file.
- **Purpose:** This approach ensures that in the event of a crash or failure, the database can recover by replaying these logs, thus maintaining the ACID properties of the database.

How WAL Works in InnoDB

- **Change in Memory:** When a transaction modifies data, these changes are initially made in the buffer pool, which is in memory.
- **Logging to Redo Logs:** Concurrently, details of these changes are written to the redo logs. This logging happens before the changes are written to disk.
- **Data File Update:** Eventually, the modified data from the buffer pool is flushed to the actual data files on disk, but this can happen later, independently of the transaction commit.

- **Recovery Process:** If there's a crash, InnoDB uses the redo logs to redo transactions and bring the database to a consistent state.

ı InnoDB

ɔgs)

ʌL: Redo logs are the key component of the WAL system in InnoDB. They record all changes to the database as they are made in memory but before they are written to the actual data files.

- **Data Recorded:** This includes all modifications to data, like INSERT, UPDATE, and DELETE operations.
- **Recovery Use:** In case of a system crash, redo logs are used to "redo" or replay these changes to bring the database to a consistent state.

Undo Logs

- **Role:** Undo logs are used to record information to reverse a transaction. This is necessary for transaction management and to maintain isolation levels in the database.
- **Data Recorded:** They store the old values of records that are modified by transactions.
- **Use in Rollback and MVCC:** Undo logs are critical for rolling back transactions when necessary and for providing the versioning mechanism used in Multi-Version Concurrency Control (MVCC).

Binary Logs (Binlogs)

- **Role:** Binary logs are not part of the WAL system but are crucial for replication and data recovery procedures in MySQL.
- **Data Recorded:** They record all changes to the database, but unlike redo logs, they do so in a format suitable for replication to other databases.
- **Use in Replication and Point-in-Time Recovery:** Binlogs are used for setting up replication slaves and for performing point-in-time recoveries.

General Query and Slow Query Logs

- **Role:** These logs are used for monitoring and troubleshooting.

Data Recorded: The general query log records all queries that the server

ow query log records queries that take longer than a specified
execute.

ıl for analyzing database usage patterns and identifying
inefficient queries.

Data change vs schema change logging

In the InnoDB storage engine of MySQL, redo logs primarily capture changes to the database at the data level, such as insertions, updates, and deletions made to the rows of a table. Their main purpose is to ensure data durability and to aid in the recovery process in case of a crash or failure.

When it comes to database schema changes, such as creating, altering, or dropping tables or indexes, the handling is somewhat different:

1. **Direct Writing to Disk:** Schema changes in InnoDB are typically written directly to the data dictionary on disk. The data dictionary is a system table where metadata about database objects is stored.
2. **Redo Logging of Physical Structure Changes:** If a schema change involves modification of the physical storage of a table (for example, adding a column), then the changes to the physical structure of the table might be logged in the redo logs. This is because these operations affect the actual data pages.
3. **Not Captured for Replication in Redo Logs:** Redo logs in InnoDB are not intended for capturing DDL (Data Definition Language) statements for replication purposes. This is the role of the binary log in MySQL. The binary log records all DDL changes and is crucial for MySQL replication, as it allows slave databases to replicate the schema changes executed on the master.
4. **Atomicity of Schema Changes:** InnoDB strives to make schema changes atomic, meaning they either fully complete or fully revert. This approach reduces the complexity of redo logging for schema changes since the system does not need to record intermediate states.
5. **Flush and Sync Operations:** During schema changes, InnoDB might perform flush and sync operations to ensure that data files are consistent and to update the data dictionary.

6. **Crash Safety and Atomic DDL**: InnoDB's DDL operations are designed to be crash-safe. This means that if a crash occurs during a schema change, the ...er be fully completed or fully rolled back when the database ...aving the database in an inconsistent state.

...do logs in InnoDB are primarily used for logging data-level changes to ensure transaction durability, they can also be involved in logging physical structure changes resulting from schema modifications. However, the primary mechanism for tracking and replicating schema changes in MySQL is the binary log, not the redo log.

Transaction Management and Durability

- **Committing Transactions**: The act of committing a transaction signals the intention to make all changes made during the transaction permanent. The discussion emphasizes the role of the commit operation in ensuring that changes persist across sessions and survive system restarts, highlighting the significance of WAL in this process.

- **Checkpointing**: This process involves periodically flushing dirty pages from memory to disk, thereby managing the size of the WAL and ensuring efficient use of disk space. Checkpointing is described as an I/O-intensive operation that can impact performance due to increased disk activity.

Advanced Logging Mechanisms

- **Fsync in WAL**: The fsync option allows databases to ensure that changes logged in the WAL are directly written to disk, bypassing the operating system's cache. This feature is critical for maintaining data durability, as it guarantees that log entries are safely stored on disk before a commit is acknowledged.

- **Undo Log Utilization**: Undo logs not only facilitate transaction rollback but also play a crucial role in MVCC by storing the pre-transaction state of data. This allows concurrent transactions to access consistent data snapshots, even if other transactions are making changes.

Database Logs and System Architecture

- **Interplay with the OS**: The transcript touches upon how databases interact with the underlying operating system, particularly regarding how writes to the disk are managed. The use of fsync to control the direct writing of logs to disk,

bypassing the OS cache, is highlighted as a key consideration for database administrators.

ınce Considerations: The discussion acknowledges the especially checkpointing, on database performance. It cing the size of the WAL and the frequency of checkpointing erformance degradation during high-intensity workloads.

In summary, InnoDB uses a combination of different log files for various purposes: redo logs for write-ahead logging, undo logs for transaction rollback and MVCC, and binary logs for replication and recovery. Write-ahead logging specifically involves writing changes to redo logs before they are committed to data files, ensuring data integrity and enabling recovery in case of failures. Understanding the distinct roles of these log files is crucial for comprehensively grasping how InnoDB manages data operations and maintains consistency and durability.

Buffer Pool Management in InnoDB

- **Memory Area:** The buffer pool is an in-memory area where InnoDB caches data pages, index pages, and other structures needed for database operations.
- **Purpose:** Its primary goal is to keep frequently accessed data in memory, thus avoiding costly disk reads and writes.

How Buffer Pool Works

- **Data and Index Caching:** When a query accesses a data page, InnoDB first checks if the page is in the buffer pool. If not, the page is read from disk into the pool. Subsequent accesses to this page don't require disk I/O, significantly improving performance.
- **Page Management:** InnoDB uses algorithms like the Least Recently Used (LRU) algorithm to manage pages in the buffer pool. Frequently accessed pages are kept in memory, while less frequently accessed pages may be evicted to make room for new pages.
- **Dirty Pages:** Pages that have been modified in memory but not yet written back to disk are marked as "dirty." InnoDB periodically flushes these dirty pages to

disk to ensure data durability and to free up space in the buffer pool for new pages.

ırrency Control (MVCC) in InnoDB

y control method used by InnoDB to achieve high transaction tency without locking rows for reading.

What is MVCC?

- **Versioning Mechanism:** MVCC in InnoDB works by maintaining multiple versions of data rows in the database. Each transaction sees a snapshot of the database at a particular point in time, depending on its isolation level.
- **Concurrency and Isolation:** This allows multiple transactions to read and write to the database concurrently without waiting for other transactions to complete, thereby improving performance and ensuring isolation.

How MVCC Works in InnoDB

- **Read Views:** When a transaction reads data, InnoDB generates a "read view" that identifies which versions of rows are visible to the transaction. This ensures that the transaction sees a consistent snapshot of the database.
- **Row Versions:** InnoDB stores row versions in the undo logs. When a transaction modifies a row, InnoDB does not overwrite the existing data. Instead, it creates a new version of the row. The original version is kept to serve other transactions that need to see the database state before the change.
- **Visibility Check:** Before accessing a row, InnoDB performs a visibility check to determine if the transaction should see the current version of the row, a previous version, or if it should wait.
- **Purge Process:** Old versions of rows that are no longer needed by any transactions are periodically purged by InnoDB to reclaim space and maintain performance.

Interaction Between Buffer Pool Management and MVCC

- **Caching Versions:** The buffer pool caches the current and some older versions of data and index pages, which are used by MVCC to provide consistent views.
- **Efficiency:** By caching frequently accessed versions of rows, the buffer pool reduces the need for accessing the undo logs on disk, thus enhancing the

efficiency of MVCC.

ffer pool management and MVCC in InnoDB provides a
or high-performance, concurrent data access in MySQL
management reduces disk I/O by caching data in memory,
ansaction isolation and consistency by maintaining multiple
versions of data. Together, they enable InnoDB to support heavy workloads with minimal locking and high throughput.

Innodb Database Dbms Data Storage

Follow

Written by Sameer Soin

92 followers · 118 following

A Computer Science graduate exploring different fields in software engineering and machine learning.

Responses (1)

Ducan

What are your thoughts?

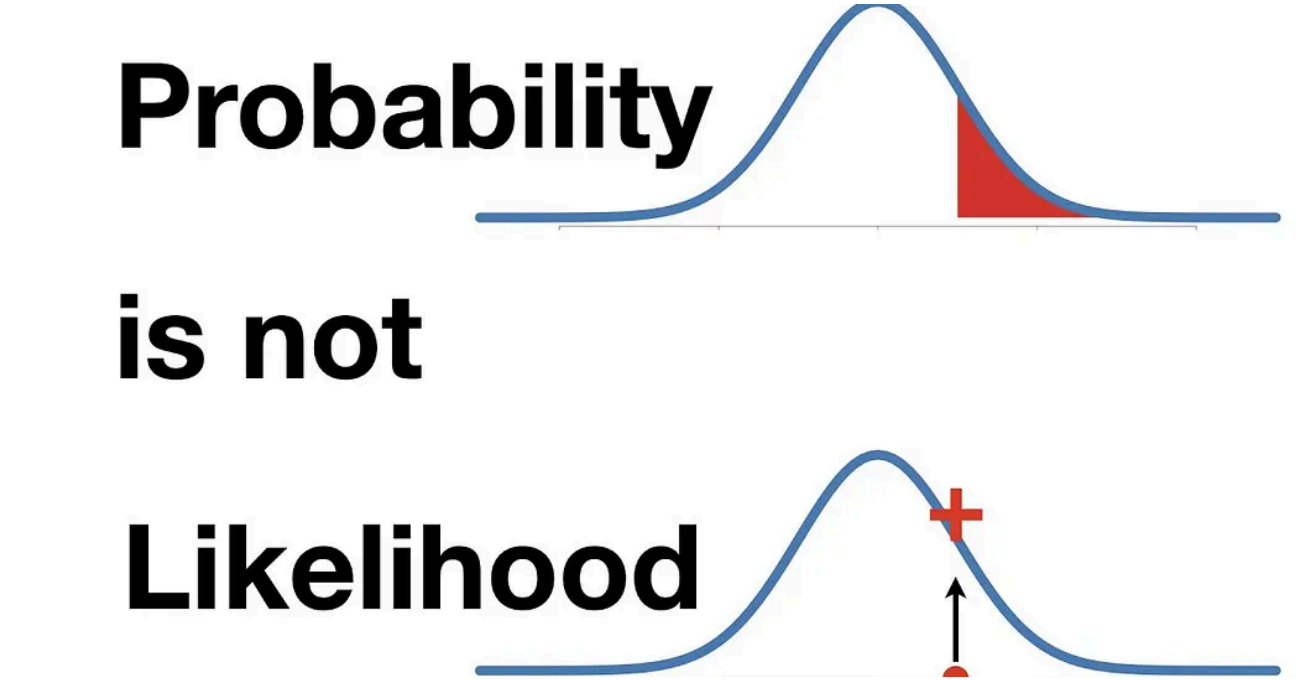
Data-Sleek
Aug 25, 2024

Hi Sameer, Well written article. How can i reach out to you?

1 1 reply Reply

All your favorite parts of Medium are now in one sidebar for easy access.

More from Sameer Soin

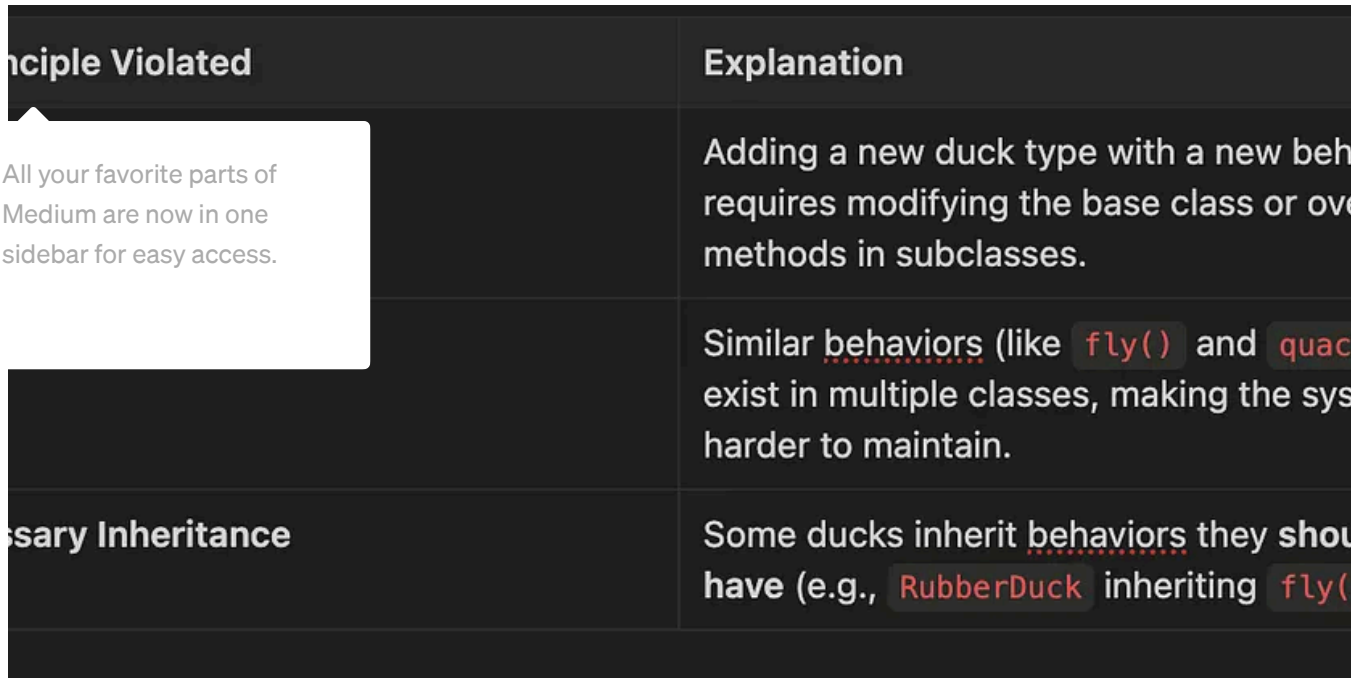


Sameer Soin

Probability vs Likelihood

When I first came across the term ‘Likelihood’ in the same sentence having the word ‘Probability’, I was very confused and could not make...

Sep 10, 2021 894 8



Sameer Soin

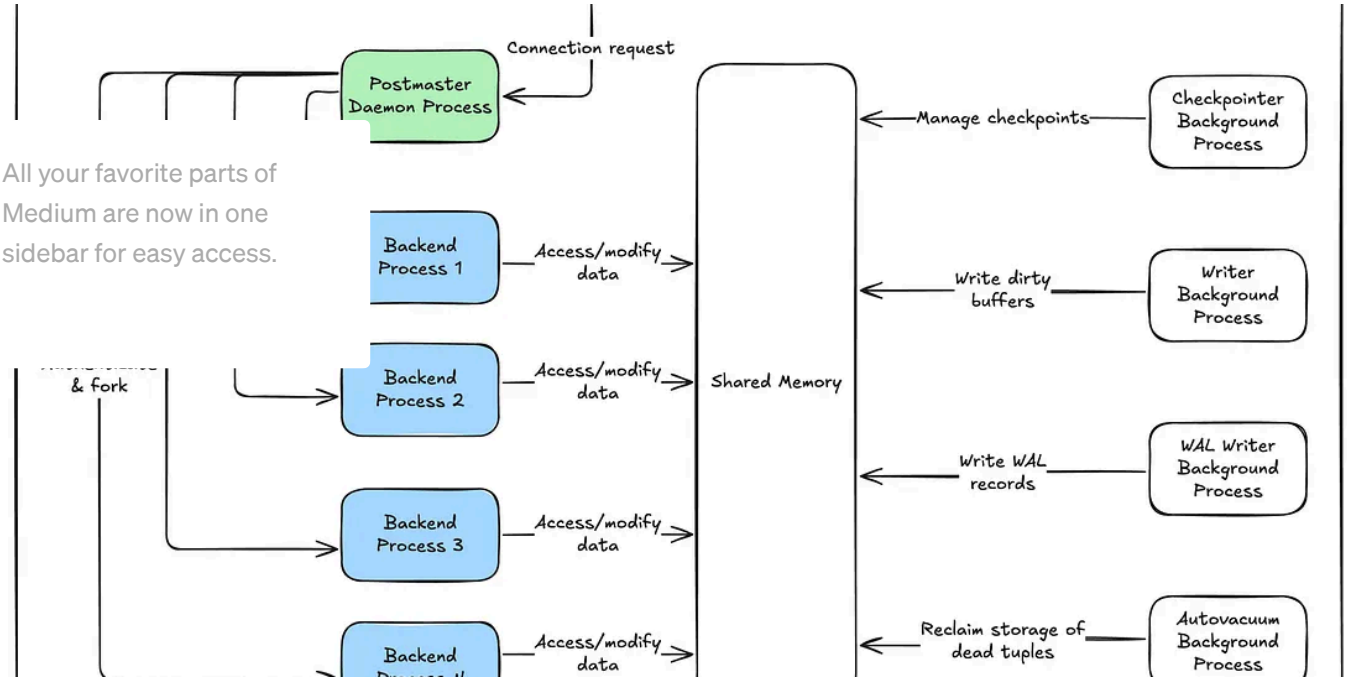
A Journey from the OOP First Principles to intuitively mastering the Strategy Design Pattern

Introduction

Feb 16

See all from Sameer Soin

Recommended from Medium

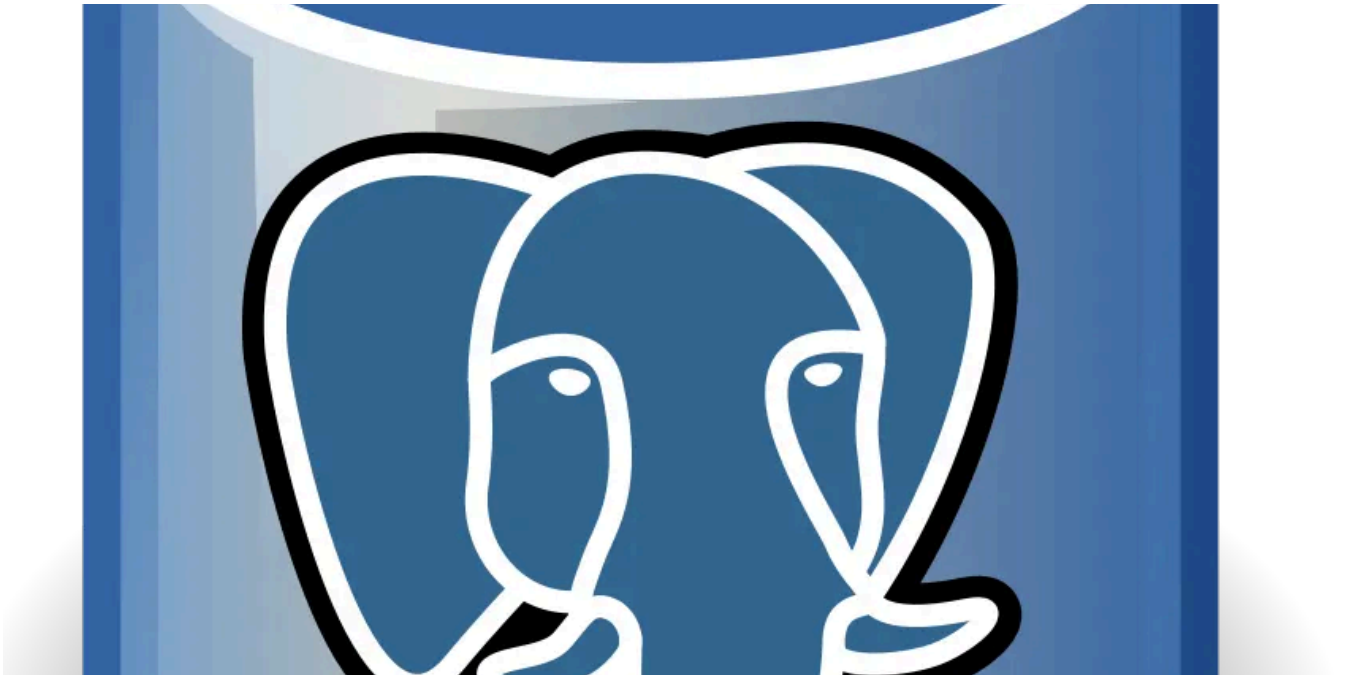


Kareem Mohllal

Database Connection Pooling with PgBouncer

Database connections are expensive, let's explore why and how PgBouncer can save your database from drowning in connections.

Aug 5 10



In Lyft Engineering by Jay Patel

Beyond Query Optimization: Aurora Postgres Connection Pooling with SQLAlchemy & RDSProxy

Written by Jay Patel and Creston Jamison

May 20 38

Ritik

I Replaced Redis with Go Maps and Got 10x Faster Performance

When Go 1.25's Swiss Tables met our caching nightmare, the results shocked everyone (including our Redis vendor)

5d ago 11 2

AkashSDas

Database Replication

Replication is the process of sharing data across multiple database instances to improve fault [illegible] read...

Lester Martin

building trino data pipelines (with sql or python)

When people think of Trino they naturally hone in the benefits of querying data with SQL in this “engine that runs at ludicrous speed”...

Aug 19

Rizqi Mulki

All your favorite parts of Medium are now in one sidebar for easy access.

loat: The Silent Killer of Performance

ame 3x faster by fixing the invisible problem that plagues 90% of eployments

See more recommendations