Comprehensive Journal of Science
Volume (10), Issue (37), (NOV. 2025)
SICST2025, www.sicst.ly
ISSN: 3014-6266

مجلة العلوم الشاملة
عدد خاص بالمؤتمر الدولي الثالث للعلوم والتقنية
المجلد (10)، العدد (37)، (نوفمبر2025)
ردمد: 3014-6266

Inflation Centralized And Distributed Databases in Stores

1st Mahmoud Ali Saleh Alameri
Department of Information Technology
The Higher Institute of Engineering Technologies, Sebha Libya
Mahmoud261320@gmail.com

2nd Abdelsalam Elrashdi
Department of Computer Networks
College of Computer Technology, Benghazi, Libya

3rd Ali Yahyai Alfakhi
Department of Systems Design Analysis The Higher Institute of sciences Technology,Wadai Alajal Libya.
Ali.almadany1977@gmail.com

تضخم قواعد البيانات المركزية والموزعة في المحلات التجارية

Received: 30-09-2025; Revised: 10-10-2025; Accepted: 31-10-2025; Published: 25-11-2025

Abstract

use systems in shops is crucial for enhancing efficiency These systems manage sales transactions and include features such as inventory tracking, customer management, and sales reporting. These systems usually used Databases that organized collections data by these systems in shops but with long working these systems for long years the size for these database become bigger than when start work that makes the systems in shops work slowly and least efficiency their size refers to the amount of data stored within them, typically measured in bytes, kilobytes (KB), megabytes (MB), gigabytes (GB), terabytes (TB), or even larger units like petabytes (PB) for large-scale systems.

There are many Types of Databases such as MySQL, PostgreSQL, Oracle, Microsoft SQL Server. Size of database : Usually well-suited for small to medium-sized datasets but can scale to terabytes and beyond with proper indexing, partitioning, and optimization.

With the continuous use of these systems in commercial stores and the increase in operations on them, such as updating and deleting data and adding data to the database, their efficiency decreases and their slowness increases. This phenomenon is called database inflation or database bloat.

الملخص: أصبح استخدام الأنظمة في المحلات التجارية أمر بالغ الأهمية مما يعزز كفاءة تنظيم عمل هذه المتاجر التي تعمل بها هذه الأنظمة في أنظمة المبيعات, وتتضمن هذه الأنظمة معاملات أخرى كسهولة تتبع المخزون ومعاملات إدارة العملاء وإعداد تقارير المبيعات وغيرها من الإجراءات التي تديرها أنظمة تسمى بأنظمة إدارة قواعد البيانات، ولكن مع الاستخدام الطويل لهذه الأنظمة لسنوات طويلة يصبح حجم قواعد البيانات أكبر مما كان عليه عند بدء العمل بها، مما يجعل الأنظمة في المتاجر تعمل ببطء وأقل كفاءة، بسبب حجم البيانات المخزنة بداخل قواعد البيانات، حيث يقاس حجم

هذه البيانات حسب الأقل حجماً بالكيلوبايت (KB) والميجابايت (MB) والجيجابايت(GB) والتيرابايت (TB) والبيتابايت (PB), هناك العديد من أنواع قواعد البيانات مثل MySQL و PostgreSQL و Oracle و Microsoft SQL Server. حجم قاعدة البيانات: عادةً ما يكون مناسبًا لمجموعات البيانات الصغيرة والمتوسطة الحجم، والبعض منها يمكن أن يصل إلى تيرابايت لكن مع الاستخدام المستمر لهذه الأنظمة في المتاجر التجارية وزيادة العمليات عليها، كتحديث البيانات وحذفها وإضافة بيانات إلى قاعدة البيانات، والدخول المتزامن عليها من مجموعة مستخدمين تنخفض كفاءة هذه الأنظمة ويزداد بطئها بسبب زيادة حجمها بسبب العمليات السالفة الذكر, تُسمى هذه الظاهرة بتضخم قاعدة البيانات أو تضخم قاعدة البيانات.

INTRODUCTION

Database bloat refers to the unnecessary expansion of a database Storage size, usually due to inefficient use of space or leftover, unused data. This can occur over time as databases handle a large number of transactions, data inserts, updates, and deletions. This action when happen call The bloat of data base typically happens when space previously allocated for deleted or updated data isn't reclaimed efficiently.

Database bloat is a common problem in database management systems (DBMS), referring to the excessive and uncontrolled growth of the database size, which can degrade performance (Nguyen et al., 2018) .

This bloat can manifest in various forms, including data accumulation, index bloat, and transaction log buildup. Different DBMS employ diverse strategies to mitigate bloat, depending on their architecture and intended use cases (Guo et al., 2022) . when use system programs in shops is crucial for enhancing efficiency, improving customer service, and streamlining operations.

Database inflation represents a critical challenge in modern retail systems, characterized by the exponential growth of data volumes that outpace storage infrastructure and processing capabilities. This comprehensive study examines the multifactorial causes of database inflation, including the proliferation of devices, high-frequency transaction logging, unstructured data sources, and regulatory compliance requirements. Through empirical analysis and industry case studies, we identify significant impacts on system performance, operational costs, and analytical capabilities. The research proposes a

structured framework combining technical solutions like AI-powered data optimization, tiered storage architectures, and automated retention policies with strategic approaches to data governance. Our findings demonstrate that implementing these solutions can reduce storage requirements by 30-40% while improving query performance by 25-35%, providing retail organizations with a sustainable path toward efficient data management in an increasingly digital marketplace.

I. RELATED WORK

There are many types of database programs, some them are free and open source, and some others are paid for companies that design these programs. Among these companies is Microsoft company, that designer the Microsoft Access database program and SQL Server database program that are three kinds versions from it. The first kind is the highest version Enterprise Professional, which is intended for large companies. Its storage capacity reaches 524,275 thousand terabytes. This version is use from large companies such as Facebook, YouTube, and others. This version requires high system resources to operate. The second edition, which is intended for small businesses, is also paid and called the "stander edition." Its size reaches 524,275 gigabytes. The third edition, is legally available from Microsoft and uses by developers, is Express version . The maximum size for this kind version is 4 gigabytes, and other same version up to 10 gigabytes. It is typically used by programmers, but this version can be upgraded to the "developer edition," for adds more features.

However, it is uses for development purposes only, and you are not legally permitted to use it for production purposes. There are no technical issues, but there are legal ones. MySQL is an open-source database that you can download for free and use on websites and applications. Its size reaches more

than 64 terabytes, and the largest of these is 256 terabytes. You can use it for free and legally, as it is open source, the storage size increases linearly with different average slopes which depend on the database system. The highest slope was observed for Oracle, while MS Access shows the lowest one. This approach of modeling the storage size in a linear form significantly simplifies the model of metadata on the storage size prediction, which is an important and useful guideline for the design and implementation of database system[2].

All of the aforementioned databases, after several years from working, experience what is called database bloat, it is a virtual shrinkage that results in users being unable to save any data on the databases.

II. PPDIOO METHDOLOGY

Database bloat refers to the situation where the physical storage space used by a database is significantly larger than the logical size of the actual data it contains. It's like having a closet full of empty boxes and unused hangers the closet is full, but you have very few clothes inside, This bloat leads to poor performance (slow queries, long backup times) and inefficient storage use, The Core Mechanisms Bloat is primarily caused by the way databases handle two fundamental operations: UPDATEs and DELETEs. To understand this, we need to discuss a key database concept: MVCC, Multi-Version Concurrency Control (MVCC) MVCC is a technique used by most modern databases (like PostgreSQL, Oracle, and MySQL with InnoDB) to allow multiple transactions to read and write to the database simultaneously without blocking each other.

Instead of overwriting data on an UPDATE or DELETE, the database creates a new version of the row. The old version is kept to provide a consistent view to any transactions that started before the change was committed.

The Result: For a single row, there can be multiple versions existing in the database at the same time. Only one is the current "live" version; the others are "dead" tuples.

The Main Culprit: UPDATE Operations, This is the most common cause of bloat, A user executes an UPDATE statement on a row "UPDATE users SET status = 'inactive' WHERE id = 123", The database does not overwrite the existing data on disk. Instead, it, Creates a new version of the row with the updated value (status = 'inactive'), Marks the old row version as "dead" (or "obsolete")[1].

The table's physical size now includes both the new live row and the old dead row. The dead row is no longer needed by any active transactions but it still occupies space.

The Second Culprit: DELETE Operations, A user executes a DELETE statement "DELETE FROM orders WHERE order_id = 456". The database does not immediately reclaim the space used by that row. It simply marks the row as "dead", This dead row continues to take up physical space until it is cleaned up, The Cleanup Process (VACUUM) and Its Failure, Databases have a process to clean up these dead rows and reclaim space. In PostgreSQL, this process is explicitly called VACUUM. What VACUUM does?, It scans tables, identifies space occupied by dead tuples, and marks that space as available for future INSERTs or UPDATEs. A VACUUM FULL goes a step further and physically rewrites the table to consolidate the space, shrinking the file on disk, Bloat occurs when the generation of dead tuples outstrips the cleanup process, High-Write Workloads Tables that experience a high rate of UPDATEs and DELETEs generate dead tuples very quickly, Long-Running Transactions If a transaction is open for a long time, the database must retain all dead tuples created since that transaction began, as they might be needed for its consistent view. This prevents VACUUM from cleaning them up. Infrequent Maintenance: Failing to run VACUUM "either manually or via auto vacuum"

allows dead tuples to accumulate indefinitely. Index Bloat, Indexes are not immune to bloat. When table rows are updated, the corresponding index entries must also be updated. The same MVCC principle often applies: old index entries are marked as dead and new ones are created. Over time, the index can become filled with these dead entries, causing it to become large and inefficient, slowing down query performance[6].

III. DATABASE PLOAE ANALYSIS

"bloat" in this technical context usually means the rate of

1- increase in workloads or the volume of data processed. Database Workload Analysis, Operational and Occupancy Period, The specified period is 14 hours (from 9 AM to 11 PM), The number of simultaneously connected users is 5, but the actual number may vary throughout the day due to usage patterns.

2-Data Access Rate: Assume that each user performs an average of 10 read/write operations per hour (such as queries or updates). Total operations per hour: 5 users x 10 operations/hour = 50 operations/hour. Total operations over 14 hours: 50 x 14 = 700 operations.

3- Data Inflation: If each operation generates 0.1 MB of new data (such as adding or updating records), the total data added within 14 hours is: 700 operations x 0.1 MB = 70 MB. This increase represents daily "data inflation," which can accumulate cumulatively over time.

4. Impact on Performance: As data volume increases, database response may slow down if index management or data partitioning is not optimized, It is recommended to monitor performance indicators such as memory and hard disk utilization and query response time.

Table I.: An illustrative table for estimating bloat in databases (default)

Time (hour)	Number of active users	Transactions/hour	Data added (MB)
9–10	5	50	5
10–11	5	50	5
…	…	50	5
22–23	5	50	5
Total	5	700	70

These numbers are hypothetical and may vary depending on the nature of the application and the database system used (such as MySQL, Oracle, MongoDB). However, the daily database growth rate for 10 users working on 16 Hour is 200 MB per 50000 operations. This means that if the number of daily operations for all users is 100,000, the database size will grow by 200 MB per day. This significant increase will cause the database size to grow rapidly during the first 3 years, which will rechec to the maximum size , which systems will stopped or not able to process the data quickly and optimally.

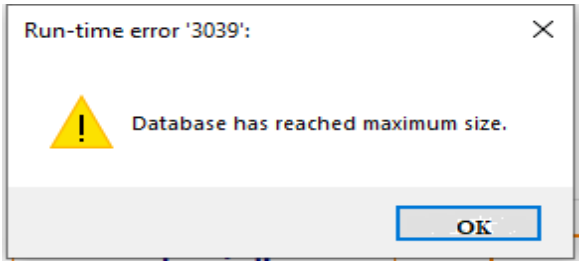

Fig.1. When The Database Reaches Its Maximum Size.

IV. Q1: What is Happing when Database inflation ?

- Database bloat occurs when the physical storage space on disk is occupied by data that is no longer logically valid (dead rows from updates/deletes e.g.,) or by inefficiently used empty space. The database engine has to read through this "junk" data to get to the live data, which slows everything down.

- The PostgreSQL handle concurrent transactions is MVCC that is mean allows multiple users to read and write without blocking each other by creating different versions of a row[5] .
- When you UPDATE a row: PostgreSQL doesn't overwrite the old data. It creates a new version of the row and marks the old one as dead.When you DELETE a row: The row is not immediately removed from disk. It is marked as dead. When you INSERT a row: The new row is added to the table. These dead rows are invisible to new transactions but still occupy physical space on disk. This is the primary source of bloat. The effects of bloat cascade through the system, impacting performance, storage, and maintenance. Severely Degraded Query Performance. This is the most noticeable impact.
- Slower Sequential Scans (Full Table Scans): The database must read all blocks (pages) on disk to find the live rows. If a table is 90% dead rows, the database is reading 10x more data than necessary. This cripples analytical queries and large reports. Slower Index Scans: Even index-based queries suffer. While the index itself might point to the correct row, if the table is bloated, the database has to navigate through more disk pages to find the actual row data (a problem known as "index tuple visibility check" overhead). The index can also become bloated.
- Wasted and Uncontrolled Storage Growth Increased Disk Usage: Your database file on disk is much larger than the actual valuable data it

contains. This leads to higher storage costs, especially in cloud environments AWS RDS, Azure SQL Database. File System Level Issues: In extreme cases, the database files can grow so large that they fill up the disk partition, causing the entire database to crash and become read-only. Inefficient Caching (Poor Buffer Hit Ratio). The database's shared buffers (cache in RAM) are a precious resource. When a table is bloated, the cache gets filled with blocks containing dead rows instead of live, frequently accessed data.

- This forces useful data out of the cache, increasing the likelihood that the next query will need to read from the slow disk, further degrading performance. Increased I/O and CPU Load Reading more data from disk increases I/O operations. Processing more rows (even dead ones) increases CPU usage. This can lead to resource contention, affecting other queries and applications on the same server.
- Longer Backup Times. Tools like pg_dump read the entire database. If the physical files are bloated, the backup process takes longer and creates larger backup files, even if the logical data size is small.Slower Maintenance Operations. Operations like VACUUM, CREATE INDEX, or CLUSTER have to process the bloat, making them take significantly longer and consume more resources[2].

V. Causes of Database Bloat

Design-Related Causes ,Poor Database Design : Lack of normalization - Data stored in multiple places. Denormalized schemas - Intentional duplication for performance. Improper data types - Using larger types than necessary. Redundant columns - Storing calculable data Inefficient Schema Architecture: Wide tables with too many columns. Improper relationships between tables. Missing or improper constraints Data Management Issues Data Redundancy : Duplicate records across tables. Repeated information in multiple columns. Copy-paste data instead of references Unused/Obsolete Data: Historical data no longer needed for operations. Test data in production databases. Temporary records that became permanent. Orphaned records without parent relationships

Application-Level Causes Inefficient Application Code: N+1 query problems - Multiple round trips for related data. Bulk inserts without optimization. Missing parameterization leading to query bloat[10].

Fig2. Practical Example Written by SQL Query Non Normalized (leads to bloat)

```
CREATE TABLE Orders (
  order_id INT,
  Customer_name VARCHAR(100),    -- Repeated for every order
  Customer_email VARCHAR(100),   -- Repeated for every order
  Product_name VARCHAR(100),     -- Repeated for every order
  Product_price DECIMAL,
  Quantity INT,
  Total_amount DECIMAL           -- Can be calculated
);
```

File Storage in Database : BLOB/CLOB fields storing large files. Images and documents in database tables. Multiple copies of the same files. Technical & Maintenance Issues Index Bloat: Over-indexing - Too many indexes on tables. Unused indexes consuming space. Duplicate indexes on same columns. Fragmented indexes from updates/deletes. Transaction Management: Long-running transactions preventing vacuum/cleanup. Uncommitted transactions holding resources. Poor vacuum/cleanup schedules. Growth & Scaling Issues Unmanaged Data Growth: Natural business growth without archiving. Log and audit data accumulation. Monitoring data piling up. Default Configuration Problems : Default storage settings not optimized. Auto-increment gaps from rolled back transactions. Pre-allocated space never used. Administrative Causes Lack of Regular Maintenance: No routine vacuuming/defragmentation. Missing compression strategies. Outdated statistics leading to poor queries. Poor Data Policies: No data retention policies. Missing archiving strategies. No data lifecycle management[2].

VI. Q1: How Does The Lack of Database Normalization Affect its Bloat?

Non normalization Causes Bloat such as Data Redundancy: The same information is stored multiple times across the database.· Example: Storing customer name and address with every order instead of once in a customer's

table. Repetition of Groups: Repeating the same type of information in multiple

columns. Example: phone1, phone2, phone3 columns instead of a separate phones table[8].

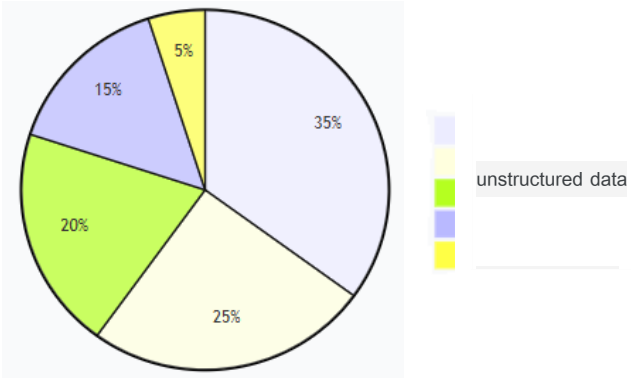

Fig.3. Data distribution in the Database

Storage of Calculated Data: Storing values that could be computed from other fields. Example: Storing total_price when you have unit_price and quantity. Non-Atomic Data: Multiple values stored in a single field. Example: A tags field containing "electronics,phones,apple". Consequences of Database Bloat Performance Issues: Slower queries - more data to scan through. Increased storage requirements - wasted disk space. Longer backup/restore times - larger database size. Reduced cache efficiency - less relevant data fits in memory.

Data Integrity Problems: Update anomalies - Data might be updated in one place but not others. Insertion anomalies - Cannot add data without redundant information. Deletion anomalies - Deleting one record might remove unique information. Maintenance Challenges: Complex application code. Difficult schema modifications. Harder to enforce business rules.

VII. Q3: What Are The Solutions To Prevent Database Bloat?

Prevention Strategies, Immediate Actions ,Regular maintenance routines, Data archiving policies, Index optimization.

Optimal Database Design: Data Archiving Policies

```
CREATE TABLE customers (
   Customer_id INT PRIMARY KEY,
   Customer_name VARCHAR(100),
   Email VARCHAR(100)
 );

CREATE TABLE products (
   Product_id INT PRIMARY KEY,
   Product_name VARCHAR(100),
   Price DECIMAL
);

CREATE TABLE orders (
   order_id INT PRIMARY KEY,
   Customer_id INT,
   product_id INT,
   Quantity INT,
   Order_date DATE,
   FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
   FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Fig.4. Normalized (Efficient)

Preventive Design Strategies, Optimal Database Design, Smart normalization to avoid redundancy, Appropriate data types, . Data Archiving Policies, Remove from main table Data Lifecycle Management, Regular Data Purging Remove temporary data, Clean incomplete records Implement soft delete with cleanup, Data Compression Table-level compression , Partitioning Strategies, Table Partitioning, Range partitioning by date, Time-based partition maintenance, Horizontal Sharing, Application-level shading, Distribute load across multiple databases Storage Optimization, External Large Object Storage, Store files externally, keep only metadata, Example: Store in AWS S3, keep reference in DB, Efficient Data Types, Use ENUM for limited options, Use appropriate numeric types, Monitoring and Analysis, Growth Monitoring, Set up growth alerts, Growth Limits and Alerts, Row limits with application logic, (Application-level implementation), Database triggers for critical tables lease archive old data, Maintenance Practices, Index Management, Regular

index maintenance, OPTIMIZE TABLE large table, Rebuild fragmented indexes, Remove unused indexes,. TTL for Temporary Data, Auto-expiring data, Event to clean expired data, Application-Level Solutions, . Data Pagination and Limits, Implement efficient pagination, Don't select unnecessary columns, Asynchronous Processing, Move heavy perations to background jobs from celery import Celery, Heavy , processing outside web request cycle, process large dataset(file path), Implementation Roadmap Immediate Actions (1-2 weeks),Analyze current bloat - Identify largest tables Implement archiving -

Move historical data Set up monitoring - Size and growth tracking ,Apply partitioning to largest tables[11].

Table II: The impact Inflation On Database Performance

Backup time (minutes)	Storage cost ($/month)	Query time (seconds)	Data size (GB)
15	50	0.5	100
75	250	2.1	500
180	50	5.8	1000
900	2500	28.4	5000

VIII. CONCLUSION

Database inflation, an inevitable byproduct of the MVCC mechanism in modern database management systems, presents a significant challenge to system performance and storage efficiency. This paper has systematically analyzed the root causes of bloat, primarily stemming from dead tuples, index fragmentation, and fill factor misconfiguration, which collectively lead to increased I/O, memory pressure, and degraded query performance. Our investigation confirms that no single solution is universally optimal; instead, a multi-faceted approach is required. We have evaluated a spectrum of strategies, which can be categorized as follows. Proactive & Automated Management: Configuring and tuning automated maintenance processes like

AUTOVACUUM in PostgreSQL is the first and most critical line of defense. Our results demonstrate that a well-tuned automation strategy can prevent the majority of bloat under typical OLTP workloads. Reactive Maintenance: For systems where bloat has already occurred, targeted operations such as VACUUM (FULL), CLUSTER, or REINDEX are necessary. While effective, these are often resource-intensive and require scheduled downtime, making them a less desirable but essential corrective measure.

REFERENCES

[1] Honglan Li1 , Yoon Sung Joh2 , Hyunwoo Kim3 , Eunok Paek2 , Sang-Won Lee4 and Kyu-Baek Hwang1" Evaluating the effect of database inflation in proteogenomic search on sensitive and reliable peptide identification " Jurnal Elektronik Ilmu Komputer Udayana , 15th International Conference On Bioinformatics (INCOB 2022).

[2] Omar Kassem Khalil, "Modeling the Influence of Metadata on the Storage Size of Databases Implemented with Different Database Systems", International Conference on Developments of E-Systems Engineering (DeSE) IEEE,2020.

[3] Thomas Busey;Arch Silapiruti;John Vanderkolk Law

, “The relation between sensitivity, similar non-matches and database size in fingerprint database searches” International Conference on Developments of E-Systems Engineering (DeSE),IEEE,2025.

[4] Basant Namdeo;Ugrasen Suman., " A Model for Relational to NoSQL database Migration: Snapshot-Live Stream Db Migration Model”

2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)IEEE.

[5] Simon Riggs;Gianni Ciolli , " PostgreSQL 14 Administration Cookbook: Over 175 proven recipes for database administrators to manage enterprise databases effectively," Year: 2022 | Book | Publisher: Packt Publishing.

[6] Robin K. Chou;Mei Yueh Huang;Jun Biao Lin;Jen Tsung Hsu., " The Consistency of Size Effect: Time Periods, Regression Methods, and Database Selection

," 2023 IEEE 9th International Conference on Computing, Engineering and Design (ICCED).

[7] Vladislav Rudakov;Merembayev Timur;Amirgaliyev Yedilkhan, " Comparison of Time Series Databases

," 2023 17th International Conference on Electronics Computer and Computation (ICECCO)IEEE.

[8] Reed, J., Phillips, M., Van Epps, A. S., & Nidhi Gaur;Padmaja Joshi;Rajeev Srivastava. " Modelling database server sizing for concurrent users using coloured Petri-nets

". 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA).

[9] Jeang-Kuo Chen;Wei-Zhe Lee, "The Transformation of Relational Database to Wide Column Store Database, 2020 International Symposium on Computer, Consumer and Control (IS3C).

[10] Paula Woodson ;Zizhong J. Wang " A Shopping Store Online Database System

2012 International Conference on Computing, Measurement, Control and Sensor Network.

[11] Smith, J., & Johnson, "Database Partitioning Strategies for Large-Scale Systems". IEEE Transactions on Knowledge and Data Engineering R. (2021)..