UDC 004.652.1:004.6
DOI: https://doi.org/ 10.17721/3041-2323.2024.440-448

Ramin SAMEDOV, PhD (Engin.), Assoc. Prof.
ORCID ID: 0009-0002-9384-6541
e-mail: ramin.samedov@gmail.com
Baku State University, Baku, Azerbaijan,

INCREASING FAULT TOLERANCE IN TRANSACTIONAL DATABASES BY ADDING AN ADDITIONAL SECONDARY DATABASE

This article explores the enhancement of fault tolerance in transactional databases through the implementation of an additional secondary database. It highlights the critical importance of data reliability and availability in modern business operations, especially in sectors such as finance and healthcare, which are subject to stringent regulatory requirements. The paper discusses Oracle's database solutions, emphasizing their utility in maintaining high availability and resilience against cyber-attacks and hardware failures. By deploying a backup database and configuring a real-time replication system using triggers, the study demonstrates an effective approach to ensure data consistency and system reliability. Experiments validate the system's capability to seamlessly switch to a backup database and maintain operational continuity during failures, thereby minimizing downtime and data loss. The findings underscore the essential role of robust data management strategies in supporting uninterrupted business processes and maintaining competitive advantages.

Keywords: ***transactional databases, fault tolerance, uninterrupted payment systems, reliability assurance.***

Background

Modern companies generate and process a massive amount of data, and this volume is continually increasing. Ensuring continuous access to this data is critically important for decision-making, customer service, and business operations. Data is one of the most valuable assets of any company. Data loss or temporary unavailability can lead to severe financial and reputational losses. The reliability and fault tolerance of databases directly affect the continuity of business processes. With the digitalization of processes, threats related to cyberattacks, hardware failures, and human errors are also increasing. The reliability and fault

tolerance of database management systems become key factors in protecting against these threats (Date, & Darwen, 2006).

For many companies, ensuring business continuity is a priority task. IT system downtimes can lead to work disruptions, financial losses, and reduced reputation. Oracle provides solutions that help minimize downtime and ensure high data availability.

In some industries, there are strict regulatory requirements for data storage and protection, such as in the financial sector, healthcare, and government administration. Compliance with these requirements is only possible with high database reliability and fault tolerance.

In a highly competitive environment, companies with reliable and fault-tolerant database management systems have significant advantages. They can adapt faster to market changes, respond quickly to customer requests, and minimize downtime risks (Oracle Corporation, 2020).

Database technologies are constantly evolving, offering new opportunities to enhance fault tolerance and reliability. Researching and implementing these innovations can significantly improve the performance and reliability of an IT infrastructure.

In today's payment system market, when choosing a database, a database from Oracle is often selected. The main reason for choosing Oracle is that it offers powerful backup and disaster recovery features, ensuring minimal downtime in the event of failures. This database has technologies that allow load distribution across multiple servers, ensuring fault tolerance and high availability. There are also automatic failover and recovery capabilities that provide real-time data protection. In addition, there are built-in monitoring and management tools that allow quick detection and elimination of failures. Oracle's flexibility in deployment, both in the cloud and on local servers, allows companies to adapt to their unique fault tolerance requirements. Combined with high performance and scalability, these features make Oracle a preferred choice for companies with critical workloads.

With the growing volume of data, increased risks, and changing regulatory requirements, companies must invest in technologies and solutions that ensure high availability and data protection. Research and development of effective methods and strategies for fault tolerance and reliability in Oracle database management systems are important

for ensuring competitive advantages and business stability (Gray, & Reuter, 1993).

The main operations in an Oracle database are INSERT, UPDATE, and DELETE, which are the basic operations performed on the data, each having its characteristics and applicability.

The INSERT operation is used to add new records to a database table. When an INSERT command is executed, a new row is added to the specified table with given values for the relevant columns. This event often generates automatic value assignment for fields with DEFAULT or AUTOINCREMENT constraints. INSERT can trigger cascading actions in dependent tables if such actions are configured.

An UPDATE event occurs when an existing record in a table changes one or more of its values. An UPDATE command requires specifying both the target columns for change and the criteria (e.g., WHERE clause) used to determine which records should be updated. This operation is critical for maintaining data relevancy and consistency. As with INSERT, cascading updates affecting related tables might be involved with UPDATE.

The DELETE command is used to remove one or more records from a table. It requires a condition (e. g., WHERE clause) to determine which rows will be deleted. Data deletion is irreversible, so this operation must be undertaken with care. DELETE can also activate cascading deletions in related tables, if such relationships are defined, to maintain data integrity.

All three operations described above interact with transactions, providing the ability to revert changes before they are committed using the COMMIT command. Transactional integrity is particularly crucial for systems requiring data reliability, such as banking and financial applications (Elmasri, & Navathe, 2016).

Transactional integrity, in the context of databases, refers to the assurance that all parts of a transaction are executed and recorded correctly. It is upheld by adhering to ACID principles: Atomicity, Consistency, Isolation, and Durability. Atomicity guarantees that all operations in a transaction are completed entirely or not at all, leaving no intermediate state of data. Consistency requires the database to transition from one correct state to another, maintaining all constraints (Connolly, & Begg, 2015).

Isolation ensures independent execution of transactions, preventing mutual influence until changes are finalized, thus avoiding such issues as lost updates. Durability guarantees that the results of a transaction remain in the system even in the event of failures like power outages or system errors. In payment systems, transactional integrity is critically important as it safeguards the integrity of financial operations.

Without atomicity, a payment may be deducted from one account but not deposited into another, leading to financial losses. Consistent state ensures all transactions comply with rules, such as proper calculation of fund balances. Isolation protects concurrent execution of transactions, for example, during peak loads, preventing sequence breaches. Durability is essential for customer confidence as it guarantees successful payments remain fixed even in system failures (Kumar, & Zhang, 1996).

Maintaining transactional integrity in payment systems not only strengthens user trust but also supports the reliability and resilience of financial institutions.

The goal of this research is to create a mechanism by adding an additional database to ensure fault tolerance and reliability in the operation of Oracle databases (McCarthy, & Stonebraker, 1989).

Problem Description. Using a single database in application architecture can lead to several problems. First and foremost, having a single point of failure means that if the database becomes unavailable due to hardware or software failure, the entire application will also cease to function. This is because most applications rely on database access to perform critical operations, such as user authentication and transaction processing. Additionally, a single database can become overloaded as data volumes and user numbers increase, leading to performance degradation. Over time, this can cause delays and slowdowns in application performance, degrading user experience. For scheduled database maintenance, the application also becomes unavailable if mechanisms for failover to backup resources are not provided.

As data becomes increasingly important for business decision-making, its loss due to a single point of failure can lead to significant financial loss. Database unavailability also eliminates the possibility of data searching and retrieval, halting many business processes. Without a database, it is impossible to ensure data integrity and relevance, which is critical for many applications, like in finance and healthcare.

Finally, recovery from a single database failure can take significant time, during which users will be deprived of access to the application.

Results

A solution to the single point of failure issue for databases may lie in creating a backup database and configuring a replication system. The process begins with deploying a second database, which will serve as a backup copy of the primary one. This approach, known as hot standby, ensures data availability even if the primary database fails. A data replication mechanism can be used to maintain synchronization between the two databases. This is achieved by copying changes occurring in the primary database to the backup, often in real time.

Using triggers in the database, automatic updates in the backup database can be initiated upon data change in the primary. Triggers are specialized procedures in the database that execute automatically in response to certain events, such as data insertion, update, or deletion in a table. They allow implementing automatic reactions to data changes, which can be used to maintain data integrity and consistency. Triggers are often employed for data validation, logging changes, or performing additional modifications in other tables when data changes occur. They can perform complex business logics without the need to alter client applications, reducing code duplication, and providing centralized control. Creating triggers requires a cautious approach as misplanning may lead to suboptimal performance and even looping changes in the database. For example, when in the primary database an insertion, update, or deletion of a record happens in a specific table, triggers may initiate a similar operation in the backup database. This allows ensuring data consistency and integrity across both databases.

Replication logic must account for potential conflicts and provide resolution methods. This is especially important in asynchronous replication, where changes are transmitted with some delay. It is also recommended to implement a monitoring system that will track the status of both databases and warn of potential failures or data desynchronization.

Consideration should also be given to a strategy for switching to the backup database in case of primary database failure. Automatic switchover may be implemented through cluster management systems, ensuring application continuity. It is crucial to test the switchover

and recovery processes in case of failure to confirm their reliability and efficiency.

Applying these solutions enhances system fault tolerance and reliability, allowing applications to remain available to users even in critical situations. It is essential to evaluate the overall system performance, as replication may add an overhead to both databases. Finally, regular testing and updates of the backup and replication strategy ensure the system will be resilient to failures and ready for increased load in the future.

The following outlines the process for creating a data replication system from the primary database to a backup database using triggers:

Step 1. Initialize Backup Database: Initially, create the structure of the backup database identical to that of the primary database, with necessary tables and indexes. Ensure that tables in both databases have the same schemas.

Step 2. Create Triggers on the Primary Database: Write triggers for each table in the primary database that needs replication. Each trigger should activate on events INSERT, UPDATE, and DELETE to record data changes.

Step 3. Handle INSERT Operations: In the trigger for INSERT operations, include logic that records inserted data into the backup database. This can be done by executing an INSERT command in the corresponding backup database table.

Step 4. Handle UPDATE Operations: Create logic in the triggers that copies changes to the backup database. Virtual tables like Inserted and Deleted should be used to capture old and new values and perform the corresponding UPDATE in the backup database.

Step 5. Handle DELETE Operations: Add logic to the triggers to reflect data deletions. This may involve executing a DELETE command in the backup database, using data from the Deleted virtual table to ensure correct removal of the necessary records.

Step 6. Testing and Monitoring: Thoroughly test the triggers to ensure that all changes in the primary database are accurately reflected in the backup. Set up a monitoring system to track data synchronization between databases, timely identifying and correcting data transmission failures.

After creating a fault-tolerant system consisting of two databases, experiments and verification of execution results should be conducted. Below are the experiments carried out and the results obtained.

Experiment 1. Testing Fault Tolerance Upon Primary Database Failure. Ensure that the system properly switches to the backup database upon primary database failure. During the operation of the primary database, the primary is disabled. Confirm that all transactions have transitioned to the backup; operations may still be ongoing on the backup as it might take a little time for transactions to be applied to the necessary tables based on defined logic. After confirming that there are no unapplied transactions, switch the backup database to the primary mode and continue passing transactions through the backup database. After the switch, perform a check for data loss by comparing the number of records and their content in both databases post-switch. The following time was spent on this experiment: Stopping the primary database – 34 seconds Waiting for all transactions to apply – 107 seconds Switching the backup database as primary – 53 seconds Checking database integrity – 304 seconds

Experiment 2. Checking for Network Failures and Continuation of Backup Operation Post Network Restoration. The main goal of this experiment is to verify that the entire system will continue to operate in an unplanned event of connection loss. After restoring connection, all processes should continue to work normally. Stopping the network between the two databases – 10 seconds Waiting for all transactions to apply on the backup – 57 seconds Pause waiting – 300 seconds Restore network between the two databases – 10 seconds Checking integrity and process continuation – 160 seconds.

Discussion and conclusions

The experiments also identified areas for improvement, such as optimizing switchover time and minimizing delays under high load. The overall test results indicate successful implementation of the fault tolerance architecture, which is critical for ensuring the stable operation of applications requiring high availability, such as payment systems. Continuous system state monitoring and regular checks will help maintain its readiness for any changes in the operating environment. This approach reinforces user and business confidence that data will be available and protected even in the face of serious system failures.

References

Connolly, T., & Begg, C. (2015). *Database systems: A practical approach to design, implementation, and management.* Pearson Education.

Date, C. J., & Darwen, H. (2006). *Databases, types, and the relational model: The third manifesto.* Addison-Wesley Professional.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems.* Pearson.

Gray, J., & Reuter, A. (1993). *Transaction processing: Concepts and techniques.* Morgan Kaufmann Publishers.

Kumar, V., & Zhang, T. (1996). *High-performance fault-tolerant systems.* IEEE Computer Society Press.

McCarthy, D. R., & Stonebraker, M. (1989). The design of Postgres. *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data.* ACM.

Oracle Corporation. (2020). *Oracle database concepts.*

Отримано редакцією журналу / Received: 12.09.24
Прорецензовано / Revised: 24.09.24
Схвалено до друку / Accepted: 01.10.24

Рамін САМЕДОВ, канд. техн. наук, доц.
ORCID ID: 0009-0002-9384-6541
e-mail: ramin.samedov@gmail.com
Бакинський державний університет, Баку, Азербайджан

ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ ТРАНЗАКЦІЙНИХ БАЗ ДАНИХ ЗАВДЯКИ ДОДАВАННЮ ДОДАТКОВОЇ ВТОРИННОЇ БАЗИ ДАНИХ

Досліджено підвищення відмовостійкості транзакційних баз даних шляхом упровадження додаткової резервної бази даних. Підкреслено критичну важливість надійності та доступності даних у сучасних бізнес-операціях, особливо у сферах фінансів та охорони здоров'я, які підпадають під суворі регуляторні вимоги. Розглядаються рішення баз даних Oracle, акцентується їхня корисність у забезпеченні високої доступності та захисту від кібератак і збоїв обладнання. Дослідження демонструє ефективний підхід до забезпечення узгодженості даних і надійності системи шляхом розгортання резервної бази даних і налаштування системи реплікації в реальному часі за допомогою тригерів. Експериментальні результати підтверджують здатність системи безперебійно перемикатися на резервну базу даних під час збоїв, мінімізуючи час простою та втрати даних. Висновки підкреслюють ключову роль стратегії управління даними у підтримці безперервності бізнес-процесів і збереженні конкурентних переваг.

Ключові слова: ***транзакційні бази даних, відмовостійкість, безперервні платіжні системи, гарантія надійності.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

.