Investigate Database Architecture Issues

Question 1 (20 Points)

What are cursors? Why are they useful in application architectures? What are some of the benefits? Are there drawbacks to cursors? How do MySQL and SQLite support cursors and how would you (in one of them) use them? Provide a 300-500 word explanation of cursors.

Answer for Q1

Cursors are programming constructs(objects) that allow for the traversal and manipulation of database query results row by row in a database management system (DBMS). They are particularly useful in application architectures because they provide fine-grained control over the retrieval and processing of data from a database, making it possible to efficiently handle large datasets or perform complex operations on data.

Cursors enable developers to retrieve and process database query results one row at a time, rather than fetching the entire result set at once. This can greatly improve performance and memory usage in cases where the result set is large, as it allows for efficient processing of data in chunks. Cursors also provide the ability to navigate through query results in a controlled manner, allowing for precise control over the order of data retrieval and manipulation. Moreover, cursors can be useful in scenarios where data manipulation is complex and requires multiple operations on each row. For example, in cases where data needs to be transformed or aggregated before being processed further, cursors can facilitate this by allowing developers to iterate over the data and apply the necessary operations row by row. Cursors also enable the ability to update or delete rows in a result set, which can be useful in cases where data needs to be modified based on certain conditions.

However, cursors can introduce additional complexity and overhead in application code, as they require explicit management of the cursor state and resources. They also require careful handling to avoid issues such as cursor leaks or excessive use of system resources, which can impact performance and stability.

MySQL supports cursors through stored procedures, which are blocks of code that can be created and executed within the database. Cursors can be declared and used within stored procedures in MySQL to fetch, update, or delete rows from query results. On the other hand, SQLite, being a serverless database engine, supports cursors through the use of the SQLite C/C++ library, which provides an API for cursor-based operations. Cursors in SQLite are typically managed through programming languages such as C/C++, Python, or other languages that have SQLite bindings. Cursors can be used in SQLite to fetch and manipulate rows from query results, and they provide flexibility in terms of data retrieval and processing.

To use cursors in MySQL, we can create a stored procedure that declares a cursor, opens it, fetches rows, and processes them one by one. We can then close the cursor once we're done with it. In SQLite, we should use the SQLite C/C++ library to create a cursor, execute a query, fetch rows, and perform operations on each row as needed(For example, when we're developing Android applications, we can use SQLite database and utilize the cursor to implement DAO layer).

Citations

- MySQL Tutorial. (n.d.). MySQL Cursor. https://www.mysqltutorial.org/mysql-cursor/
- Google. (n.d.). SQLiteCursor. Android Developers. https://developer.android.com/reference/android/database/sqlite/SQLiteCursor

- WebCodeExpert. (n.d.). What is Cursor? Advantages and Disadvantages of Cursor. https://www.webcodeexpert.com/2013/11/what-is-cursor-advantages-and.html

Question 2 (30 Points)

What are connection pools? Why are they useful in application architectures? What are some of the benefits? Are there any potential drawbacks? What are some of the main issues with using connection pools? Provide a 300-500 word explanation of connection pools.

Answer for Q2

Connection pools are a technique used in application architectures to manage a pool of pre-established database connections that can be reused by multiple clients or threads, rather than creating a new database connection for each client request. Connection pools are typically used in applications that require frequent database access and need to handle a large number of concurrent connections efficiently.

Connection pools can improve performance and scalability. Establishing a new connection to a database can be a time-consuming process, involving overhead such as authentication, handshaking, and resource allocation. Connection pools help reduce this overhead by reusing existing connections from the pool, eliminating the need to create a new connection for each client request. This can result in faster response times, reduced latency, and improved overall performance of the application.

Moreover, connection pools make resource management more efficient. Managing a large number of concurrent connections to a database can put a strain on system resources, such as memory and network sockets. Connection pools help manage these resources effectively by limiting the number of concurrent connections to a predefined pool size. When a client is done using a connection, it is returned to the pool instead of being closed, allowing other clients to reuse the connection. This helps prevent resource exhaustion and ensures efficient utilization of system resources.

Connection pools also provide better control over the life cycle of database connections. By keeping a pool of pre-established connections, the application can manage the creation, reuse, and destruction of connections in a controlled manner. This can help avoid issues such as connection leaks, where connections are not properly closed, and can help ensure proper cleanup and release of resources.

However, there are potential drawbacks to using connection pools. Connection pools can introduce additional complexity in application code, as the application needs to manage the pool of connections and handle scenarios such as connection failures or timeouts. Connection pools also require careful configuration and tuning to ensure optimal performance and scalability, as an improperly sized pool or mis-configured settings can impact application performance.

Another potential issue with connection pools is that they can result in stale connections. Connections in the pool may become stale or invalid due to factors such as database server restarts or network failures. This can result in errors or unexpected behavior in the application if not properly handled. Proper mechanisms for detecting and handling stale connections need to be implemented to ensure the reliability of the connection pool.

Citations

- Custer, C. (2021, November 30). What is Connection Pooling? Cockroach Labs Blog. https://www.cockroachlabs.com/blog/what-is-connection-pooling/
- IBM. (2023, January 26). Connection pooling. IBM Knowledge Center. https://www.ibm.com/docs/en/was/9.0.5?topic=architecture-connection-pooling
- Aboagye, M. (2020, October 14). Improve Database Performance with Connection Pooling. Stack Overflow Blog. https://stackoverflow.blog/2020/10/14/improve-database-performance-with-connection-pooling/

Question 3 (50 Points)

Reading two articles and then write an 800 - 1000 word review of them (combine the two reviews; do not write two separate reviews). What are the main issues they address? List three things that you learned or three key take-aways (from either of the two papers). Are there statements that you do not agree with? - Article 1: A Hybrid Technique for SQL Injection Attacks Detection and Prevention - Article 2: A Classification of SQL Injection Attacks and Countermeasures

Answer for Q3

The two articles address similar issues related to SQL injection attacks, which are a type of security vulnerability that can allow attackers to manipulate or access sensitive data stored in a database. SQL injection attacks can be difficult to detect and prevent, and they can have serious consequences for organizations that rely on databases to store and manage their data. The two articles focus on different aspects of the problem. The first article proposes a new hybrid technique for detecting and preventing SQL injection attacks that combines static and runtime analysis. The authors conduct a simulation to evaluate the effectiveness of their approach using a large set of SQL queries, and compare it to other detection methods. One unique feature of the suggested hybrid technique is its ability to detect and prevent SQLIAs that use built-in functions to perform attacks. This makes it a more comprehensive defense strategy than other methods that may only focus on certain types of attacks. In contrast, the second article proposes a classification system for SQL injection attacks based on their characteristics and behaviors. The authors do not conduct any simulations or experiments to evaluate their proposed system; instead, they provide an overview of different types of attacks that fall into each category, and discuss various countermeasures that can be used to prevent or mitigate these attacks.

I've learned a lot from the second article. First of all, the article explains seven types of SQL injection attacks: (1) Tautologies: These attacks involve injecting a statement that is always true, such as “1=1”, into a query in order to bypass authentication or access control mechanisms; (2) Illegal/Logically Incorrect Queries: These attacks involve injecting a statement that is not syntactically correct or does not make logical sense, such as “SELECT * FROM users WHERE username = ‘admin’ OR 1=1 DROP TABLE users”; (3) Union Query: These attacks involve using the UNION operator to combine the results of two or more queries into a single result set. By carefully crafting their queries, attackers can use this technique to extract sensitive data from the database; (4) Piggybacked Queries: These attacks involve injecting additional queries into an existing query in order to execute them along with the original query. For example, an attacker might inject a SELECT statement that retrieves sensitive data from another table in addition to the original query; (5) Stored Procedures: These attacks involve exploiting vulnerabilities in stored procedures, which are pre-compiled SQL statements that can be executed repeatedly with different parameters. Attackers can use these vulnerabilities to execute arbitrary code on the database server or extract sensitive data from the database; (6) Inference Attacks: These attacks involve using the application's response to infer information about the database structure or contents. For example, an attacker might submit a query that returns different results depending on whether a particular condition is true or false, allowing them to deduce information about the data; (7) Alternate Encodings: These attacks involve encoding special characters in different ways in order to bypass input validation and sanitization mechanisms and inject malicious code into SQL statements.

Moreover, the second article also discusses various techniques to prevent or mitigate SQL injection attacks: (1) Black Box Testing: This involves testing an application with various inputs to identify vulnerabilities to SQL injection attacks; (2) Static Code Checkers: This involves analyzing the source code of an application to identify potential vulnerabilities to SQL injection attacks; (3) Combined Static and Dynamic Analysis: This involves combining static analysis of the source code with dynamic analysis of the application in order to identify potential vulnerabilities to SQL injection attacks; (4) Taint-Based Approaches: This involves tracking user input throughout an application and identifying any points where it is used in a database query without proper validation or sanitization; (5) New Query Development Paradigms: This involves using new query development paradigms that are less susceptible to SQL injection attacks, such as object-relational mapping (ORM) frameworks or stored procedures; (6) Intrusion Detection Systems (IDS): This involves using an IDS to monitor network traffic for signs of a SQL injection attack; (7) Proxy Filters: This involves using a proxy filter to intercept and analyze incoming requests for signs of a SQL injection attack; (8) Instruction Set Randomization: This involves randomizing the instruction set used by a database server in order to make it more difficult for attackers to exploit vulnerabilities.

Finally, one interesting point made by the authors is that many SQL injection attacks are not actually targeted at stealing data from a database; instead, they may be used to perform other malicious actions such as modifying data or executing arbitrary code on the server. This highlights the importance of protecting against all types of SQL injection attacks, not just those that involve data theft.

There's one statement in the first article that I found somewhat questionable was when the authors claimed that their hybrid technique was "the only one" that could detect and prevent certain types of SQL injection attacks (using built-in functions). Nowadays, under the rapid development of computer technologies, it seems unlikely that there are no other techniques or tools available that could also be effective against it. Technologies never imagined before may be born tomorrow and have a significant impact on the progress of human society. Therefore, we should keep learning and utilizing the emerging techniques so that we can address the potential problems in a better way.