Investigate Database Architecture Issues

Fangying Li, Yangli Liu

2023-04-25

Question 1

What are cursors? Cursor provides a mechanism that allows the application to process the records row by row from a set of results retrieved by a query. It is similar to an iteration pointer. We can declare and open cursors associated with any SQL statements, then fetch the desired tuple by moving the positions of cursors.

Why are they useful in application architectures? In the application layer, cursors help provide a way for the application server to traverse the query result. Instead of holding the entire result set and processing all data at a time, the server can use the cursor as a pointer to iterate through each row, accessing individual rows and performing validation or modification at each tuple.

What are some of the benefits?

- The cursor can save time because the application doesn't have to wait for the entire result set to be loaded before rendering any rows. By using cursors, the application can display the first few tuples within a short response time.
- Cursors allow an application to update the table simultaneously while fetching the result set row by row. Without cursors, the application must send a separate SQL statement to the database to update the records. Thus, it could lead to concurrency issues if the result set has changed after the client's previous query.

Are there drawbacks to cursors?

- Creating cursors will occupy working memory. If the cursors are not correctly closed, the occupied resources won't be freed for other processes.
- Each fetch of the cursor requires the transmission of every row. Thus, the resulting network round trip takes more bandwidth than other single SQL statements, such as SELECT or DELETE.

How do MySQL and SQLite support cursors and how would you (in one of them) use them? MySQL supports cursors through stored procedures, using `DECLARE`, `OPEN`, `FETCH`, and `CLOSE` statements. While SQLite supports cursors through the implementing program languages, such as Python `sqlite3` library.

Here is an example using MySQL cursor to fetch each book's title from the Books table:

1. Declare a cursor inside a store procedure:

```
DELIMITER //
CREATE PROCEDURE use_cursor()
BEGIN
DECLARE  book_title VARCHAR(255);
DECLARE book_cursor CURSOR FOR SELECT title FROM Books;
...
```

2. Also, declare a `NOT FOUND` handler to handle the situation when the cursor cannot find any row from the result set:

```
...
DECLARE done INT DEFAULT FALSE;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
...
```

3. Then, open the cursor by using the `OPEN` statement:

```
...
OPEN book_cursor ;
...
```

4. After initializing the cursor, use the `FETCH` statement inside a `LOOP` to retrieve each book's title. If there is no row left to be read, leave the loop.

```
...
fetch_loop: LOOP
    FETCH book_cursor  INTO book_title;
    IF done THEN
        LEAVE fetch_loop;
    END IF;
         -- Perform operations on the fetched row, e.g., print the book_title
END LOOP;
...
```

5. Finally, close the cursor using the `CLOSE` statement and end the procedure declaration:

```
...
 CLOSE book_cursor  ;
END;
//
DELIMITER ;
```

Question 2

What are connection pools? Connection pools provide a way to manage and optimize the use of connections to the database server. The pools track all open connections that can be reused across different operations, reducing the overhead of opening and closing new connections.

Why are they useful in application architectures? A single connection costs time and computing resources. For example, it requires creating a socket, finishing the TCP handshake, server authentication, etc. For a small application, it's not expensive to initiate a single connection for every query. However, as the application scales up, the costs of opening and closing connections repeatedly or concurrently become larger, degrading the application's performance. Therefore, using connection pools to cache open connections and optimize the reusing is helpful in scaled applications.

What are some of the benefits?

- Improve performance. Reduce the time and resource consumption by reusing open connections from the pool, resulting in a faster response time and better performance in the application layer.
- Load balancing. In distributed databases, overloading the subset of databases may cause poor performance and transaction failures. Connection pooling helps optimize the use of connections, distributing the operation load across multiple databases.
- It can also increase reliability by providing redundant connections in the pool, reducing the risks of a single-point failure.

Are there any potential drawbacks?

- Extra resource consumption. Managing connection pools also consumes additional resources. In the worst case, the excessive usage of resources could offset the benefits of reducing latency, leading to degraded overall performance.
- Complicated implementation and configuration. Unlike opening and closing a single connection, connection pooling increases the implementation complexity. For example, users must take care of various pool settings, such as determining the pool size and handling idle connections.

What are some of the main issues with using connection pools?

- Be careful of the resource used by connection pools. For some applications running in an environment with limited resources, using connection pools requires occupying a large number of resources at an early time.
- Validate the connections before using them. The inactive connections in the pool could be timed out and become stale. To avoid the potential issue of using these corrupted connections, users should test the connections before utilization.

Question 3

In this question you will venture into literature (an important skill for graduate students). Start with the designated paper: "A hybrid technique for SQL injection attacks detection and prevention".

The second paper I chose from the references is "A Survey on Attacks due to SQL Injection and their prevention method for web application".

In the first paper, the writers introduced the concept of SQL injection attacks (SQLIAs): in any data-driven system, whether online or offline, web-based or not, SQL injection is a type of attack that is used to gain access to, alter, or delete information. It points out that this kind of system attack is one of the core issues for web applications. The writer in the second paper, Shubham stated that as web application code is not secure during the process of application development, this specific security vulnerability makes it possible for some malicious code to attach to the SQL. Then the hacker can use this SQL Injection hacking technique to perform different types of SQL Injection. Hence, the backend database is exposed to hackers for them to execute their SQL commands for stealing/gathering data, exposing sensitive information, injecting their desired codes and functions into the database, making buffer overflow or even dropping the tables.

Through the first paper, we realize that SQL injection attacks are not only very effective but also easy to learn, and simple to be executed. The potential dangers can be applied to more than databases but to the victim machines as well. In both papers, writers claimed that limiting access to the database by assigning access rights to the appropriate users is the first mechanism and possibly the best way for securing the application. In the background section of the first paper, the writers present three aspects for coming up with a good solution for SQLIAs detection and prevention techniques. They are Static Analysis – the principle for finding malicious codes in the system source code; Runtime Analysis – a technique to track the events of the system by observing the execution process; Static and Runtime Analysis – a type of combination solution by the previous two techniques.

In both papers, writers look into the architecture of the systems. It is the most common architecture as (1) Presentation Tier (front end, like a web browser): displays information to users and communicates results with other tiers (2) Logic Tier also called Application Tier (middle tier, like a server code): perform detailed processing (3) Storage Tier, also known as Data Tier (Backend, like MySQL): stores and retrieves information. Figure 1. in the first paper is a straightforward infographic illustrating this three-tier architecture.

What are the main issues they address?

The core discussion subject for the first paper is that the writers suggested a hybrid technique for detecting and preventing SQLIAs. By performing an application to simulate the suggested approach, writers presented their evaluation of this hybrid technique based on the performance and accuracy of SQLIA detection. And the main issue for the second paper is identifying types of SQLIA and sharing insights about the most relevant detection and prevention techniques.

List three things that you learned or three key takeaways (from either of the two papers).

Six stages of the hybrid approach

- If looking closely, the hybrid approach in the first paper is providing a security controlling method that can be performed on the server side to make sure no database hacking or fabrication is possible. If breaking up this approach, we can see six different stages for rejecting dangerous queries to reach the database. First, writers suggest duplicating a small set of sample data as a new replicate system database. Second, all system database queries, and possible query execution behaviors should be placed in a separate database. Third, redirect the SQL query to the replicated database and delay the execution in the actual database. Fourth, using encoding analysis, simple White-Box validation, and parameters replacement to check the SQL query in the replicated database. Fifth, monitor the behaviors of the SQL query. Sixth, create a list of affected objects by the SQL query and compare it with the second stage. As this finally determines whether this SQL query can be accepted into the real database or rejected as a suspicious query from execution.

Nine types of the SQL injection attack

- The main body of the second paper mentioned five SQL Injection Attack types including Tautology attack, Logically incorrect query attack, Union query attack, Piggy-Backed Queries, and Stored Procedure. Besides these five types, the first paper also listed other four types: Built-in Functions, Inference, Alternate Encoding, and Direct Attack. Based on Table 1 in the first paper, writers believe that their hybrid technique covers all types of SQLIA.

Thirteen existing relevant works

- Shubham made a list that discusses the most relevant works for detecting and preventing the potential vulnerabilities that might lead to SQLIAs.

(1) Ali et al.'s Scheme: create hash values for the first-time user's username and password.
(2) William G.J.Halfond et al.'s Scheme: generate a model of the legitimate queries and then monitor the generation for checking compliance with the previous model.
(3) SAFELI: identify the SQL injection attack during the compile time.
(4) Thomas et al.'s Scheme: an algorithm to remove possible vulnerabilities.
(5) Ruse et al.'s Approach: automatic test case generation to detect vulnerabilities.
(6) Haixia and Zhihong's Scheme: a database testing design for web applications.
(7) Roichman and Gudes's Scheme: fine-grained access control to the database.
(8) SQL-IDS Approach: a query-specific detection for detection.
(9) AMNESIA: an approach for tracing SQL input flow and emphasizes attack input precision.
(10) SQLrand Scheme: a de-randomization framework.
(11) SQLIA Prevention Using Stored Procedures: stored procedures by using static analysis and runtime analysis.
(12) Parse Tree Validation Approach: the parse tree framework.
(13) Dynamic Candidate Evaluations Approach: dynamically extracting the query structure for making prepared statements.

Are there statements that you do not agree with?

I don't think there are particular statements I do not agree with, but I am a little bit skeptical about the conclusion from the first paper as it states that their novel hybrid approach can detect and prevent all types of SQLIAs. They mentioned about 250 SQL queries that cover all types of attacks have been tested but do not show any particular example. And the approach is trying to create an "intercepting" system for malicious queries before their execution in the database. What if hackers can create some attacks that do create expected behaviors like in normal cases as their purpose is to gain access to data rather than manipulate databases? Will this hybrid approach still work?