CS5200 | Investigate Database Architecture Issues

Nickhil Tekwani and Anurag Arasan

2022-06-27

Question 1: SQL Cursors

Cursors serve as iterators in database management systems, allowing for the iteration through the resultant set of a SELECT query within a stored procedure.

Cursors have plenty of benefits, namely that cursors allow us to update certain parts of rows depending on their contents by nature of being used in a stored procedure. Such a process may take multiple queries in native SQL to accomplish the same goal. In some instances, cursors may increase performance for large SELECT queries as the cursor only returns a single row at a time. These performance increases are similar to those observed when using the LIMIT constraint in a standard SELECT query. Still, cursors provide more flexibility in allowing the selection of n rows until a specific condition is met via a while loop. However, cursors have some drawbacks as well. Depending on the situation, cursors may be more resource-intensive than a simple SELECT query. As previously mentioned, a cursor might save time by only needing to return a few rows. However, suppose the cursor iterates through almost as many rows as the SELECT query itself. In that case, multiple network requests will have taken up more resources than just a single SELECT query. Additionally, through being used in a stored procedure, extra memory is being used that a SELECT query does not require.

In MySQL, cursors can be declared using the syntax "DECLARE cursor CURSOR FOR select_query;" within a stored procedure. Following this declaration, the cursor needs to be opened by "OPEN cursor," Then, the stored procedure can use the FETCH command to obtain the next row from the SELECT query. By placing this FETCH command in a while loop, the following selection row can iteratively be fetched until there are no more rows or the loop's condition is met. Cursors also need to be closed using the CLOSE command after they have finished being used (often at the end of the stored procedure).

As SQLite does not support stored procedures, cursors are not available in native SQLite.

Question 2: Connection Pools

Connection pools are caches of database connections that are maintained such that the database connections can be reused, after they are originally set up, when future requests to the database are needed to complete some task. Connection pools enhance the database's performance when executing specific commands on said database.

In regards to application architecture, they are useful as their implementation allows for users to alleviate connection management overhead, as well as decrease development tasks when accessing data. This way, there is less redundancy in development and less resources required (both in the architecture and on the human side), since each time an application submits a request to a datastore, there are more added resources required. Thus, connection pooling streamlines the resources to create, maintain, and release the database connection to the backend.

Beyond these benefits, there are a few potential drawbacks and issues. One of the most common issues when dealing with connection pooling is that pooled connections can end up becoming stale. This will happen when a connection is not used in a while, it becomes inactive and thus network devices will time it out. This creates a stale connection which is inefficient an ineffective. Additionally, one thing to note with connection

pooling is that it is not an umbrella solution for all business cases. One must look at the application at hand and decide whether it will actually save any resources to set up connection pooling — meaning, does the approach of said application structured in such a way that it is submitting the same request multiple times over a long period of time, and would need to take advantage of a reusable database connection?

All in all, connection pooling is a technique of caching database connections in database management, and can prove to be very useful for certain business cases. While not always the perfect solution, connection pooling can increase the efficiency of resources and decrease connection management overhead.