Ways to avoid Data Skew

- July 12, 2014

Ways to avoid Account Data Skew:

You should always strive to recognize and avoid account data skew situations that produce unwanted lock contention and diminished database concurrency. Correcting a data skew can be painful because you're changing ownership, which in turn triggers sharing calculations. Here are some tips to help you avoid account data skews.

PREVENT ACCOUNT DATA SKEW SITUATIONS THAT PRODUCE UNWANTED LOCK CONTENTION AND DIMINISHED DATABASE CONCURRENCY

1. Design architecture to limit account objects to 10,000 children. Some possible methods include creating a pool of Accounts and assigning children in a round robin fashion or using Custom Settings for the current Account and the number of children.

1. Consider a Public Read/Write sharing model in which the parent account stays locked, but sharing calculations don't occur.

1. If you have a skewed account, re-distribute child objects in chunks during down hours to reduce the impact of record-level lock contention. Batch Apex or the Bulk API are useful ways to re-parent.

Ways to avoid Lookup Skew:

You can address the lock exceptions associated with lookup skew in many ways. Any time you encounter any type of lock exception, the mitigation strategy always revolves around reducing the frequency and duration of the offending locks.

Strategies for Mitigating Problems Related to Lookup Skew:

- Reducing Record Save Time.
- Distributing the skew.
- Using a picklist field.
- Reducing the load.

Reducing Record Save Time

- Increase the performance of synchronous apex code–tune you triggers for peak performance by consolidating code into a single trigger per object and following Apex best practices.
- Remove unnecessary workflow or consolidate into existing trigger code–workflow lengthens the lock duration, which in turn increases the lock failure rate. Removing or consolidating workflow into your Apex code can increase your save performance.
- Only process what's required–move any code that isn't required immediately upon save to asynchronous processing. Locks are held the entire time your custom code is executing, so processing logic that is not critical during the save will increase the lock failure rate.

Distributing the Skew

For lookup skew, the root of the problem is that a large number of records look up to a single record. We can distribute the skew to resolve the problem. The first step is to identify which lookup records are heavily skewed. If you add additional lookup values to distribute the skew, you can significantly reduce or even eliminate your lock exceptions.

In some situations, the lookup skew is on a single generic value, which is created as a catchall. Assuming in the previous example that most customers have not taken training at XYZ Company, the "Silver" achievement level falls into this catchall category because it represents customers who have not yet attended any training. Instead of heavily skewing many customers who will never take any training with this lookup value, we could have just left the lookup blank for those customers. Note that there might still be a problem if XYZ Company customers all start taking classes, and the majority of accounts wind up achieving "Gold" and "Platinum" levels. Over time, a lookup skew would be created, which could cause long-term scalability issues with XYZ Company's architecture.

Using a Picklist Field:

When you have a relatively low number of lookup values, it's generally a good idea to use a picklist field rather than a lookup field to define those values. By defining those values as a picklist, you can eliminate locks associated with lookups on those records, which eliminates any kind of locking issues related to lookup skew. In many cases, you cannot substitute a picklist for a lookup field if you have additional fields and other data on the lookup records. The main point here is that you should generally avoid using a lookup field to represent data that can be accommodated by a picklist.

Reducing the Load

In most cases, end user load will not be heavy enough to cause lock exceptions when lookup skew exists. If it is, carefully consider your save performance. It is far more likely that automated processes or integrations running in parallel would create problems for themselves and other users in the system. One way to address this problem is

to reduce the load created by the automated process, which you can do by running the automated processes serially during non-peak periods. If automated processing must occur during your end user operations, and your end users are encountering locks, then you can reduce the batch size to reduce the lock duration and allow your end users to obtain the lock. You can prioritize the end users' access to the locks to ensure they are not impacted by the automated processes.