# Designing Record Access for Enterprise Scale

Salesforce, Spring ’18

@salesforcedocs

Last updated: February 7, 2018

# CONTENTS

# DESIGNING RECORD ACCESS FOR ENTERPRISE SCALE

## Preface

This document introduces advanced topics in user record access for the Salesforce Sales, Customer Service, and Portal applications, and provides guidance on how to configure your organization to optimize access control performance.

### Audience

This document is for expert architects working on Salesforce implementations with complex record access requirements or large-scale sales organization realignments.

### Assumptions

This document assumes expertise in Salesforce administration and security, and knowledge of SQL and relational database concepts. It also assumes a familiarity with the content of its companion paper, *Record-Level Access: Under the Hood*, which explains the inner workings of the flexible and powerful Salesforce record access infrastructure.

## Introduction

This paper focuses primarily on the effects of group maintenance on Sharing performance and the built-in sharing behaviors that support Salesforce applications.

It also:

- Addresses how to avoid common configuration traps that can drag down the performance of your record-level access system
- Introduces some key platform features that can help you speed up large-scale sharing realignments

## Group Membership Operations and Sharing Recalculation

The Salesforce Role Hierarchy, Public Groups, and Territories are closely connected to sharing rules and the special security features of Salesforce applications. Because of these relationships, seemingly simple changes to groups and group membership can sometimes involve substantial recalculations of users' access rights.

For example, when an administrator moves a user from one branch of the hierarchy to another, Salesforce performs all of the following actions to ensure that other users have correct access to data owned by that relocated user.

- If the user:
  - Is the first member in his or her new role to own any data, Salesforce adds or removes access to the user's data for people who are above the user's new or old role in the hierarchy.
  - Has a new role with different settings for accessing contacts, cases, and opportunities, Salesforce does the following to reflect the change in settings.
    - Adds shares to those child objects where the new settings are more permissive
    - Removes existing shares where the new settings are more restrictive

- Owns any accounts that have been enabled for either the Customer or Partner portals, Salesforce removes any child portal roles from the user's original role and adds them as children to the user's new role.

  Note:

  - Salesforce also adjusts *boss-implicit shares*, which provide access in the hierarchy to records owned by or shared to portal users. See Implicit Sharing on page 4.
  - Salesforce must perform these tasks for *every* portal-enabled account the user owns.

- Salesforce also recalculates all sharing rules that include the user's old or new role in the source group. It removes all of the user's records from the scope of sharing rules where the old role is the source group and adds those records to the scope of rules where the new role is the source. Depending on the sharing rule settings for accounts, Salesforce might also add or remove shares to account child records.

  Note: If the user owns portal accounts, and there are sharing rules that use portal roles as the source group, Salesforce might need to recalculate those rules. Some sharing rules might no longer be valid given the user's new location in the hierarchy, in which case an administrator might need to modify or delete them.

During the user's move, the managers in the branch above the user's old role lose access to all the data that the user owns, as well as to child records shared through the managers' role settings. Managers in the branch above the user's new role will gain access to the user's accounts and to child records according to their own role settings.

## Common Group and Data Updates

So a lot can happen under the hood when an administrator takes what looks like a simple action, such as changing the role of a user. We chose this operation to illustrate all the possible types of sharing maintenance, but other common group and data updates can have a similar impact.

**Moving a role to another branch in the hierarchy**

One benefit to moving a whole role is that any portal accounts simply move along with their parent role, and Salesforce doesn't have to change the related sharing. On the other hand, Salesforce must do all of the work involved in moving a single user for *all* users in the role being moved and for all of those users' data.

**Changing the owner of a portal account**

The effort required for what looks like a simple data update—changing the name of the user in the `Account Owner` field—can be surprising. When the old and new owners are in different roles, Salesforce is not only moving the portal roles to a new parent role but also adjusting the sharing for all the data associated with the portal account.

## Ownership Data Skew

Even with all of the work that Salesforce does to maintain correct access for security groups, most customers will never encounter performance issues unless they are performing updates that affect many users or large amounts of data. However, there are certain common configurations that greatly increase the probability of performance problems. When a single user owns more than 10,000 records of an object, we call that condition *ownership data skew*. One of the common patterns involves customers concentrating ownership of data so that a single user or queue, or all the members of a single role or public group, owns most or all of the records for a particular object.

For example, a customer can assign all of his or her unassigned leads to a dummy user. This practice might seem like a convenient way to park unused data, but it can cause performance issues if those users are moved around the hierarchy, or if they are moved into or out of a role or group that is the source group for a sharing rule. In both cases, Salesforce must adjust a very large number of entries in the sharing tables, which can lead to a long-running recalculation of access rights.

Distributing ownership of records across a greater number of users will decrease the chance of long-running updates occurring..

Tip: You can take the same approach when dealing with a large amount of data that is owned by or visible to the users under a single portal account—changing the owner of that account or moving those users in the hierarchy requires the system to recalculate all the sharing and inheritance for all the data under the account.

If you do have a compelling reason for assigning ownership to a small number of users, you can minimize possible performance impacts by not assigning the user(s) to a role.

If the user(s) must have a role to share data, we recommend that you:

- Place them in a separate role at the top of the hierarchy
- Not move them out of that top-level role
- Keep them out of public groups that could be used as the source for sharing rules

## Group Membership Locking

When updating the role hierarchy or group membership through integration or the administration console, customers might occasionally receive a "could not acquire lock" error and have to repeat the operation. This error occurs because the sharing system locks the tables holding group membership information during updates to prevent incompatible simultaneous updates or timing issues, both of which could lead to inaccurate data about users' access rights. Typically, these locks are held only very briefly, so most customers will never see a lock conflict error. In some scenarios—such as a change in role triggering a sharing rule recalculation—locks might be held for a longer time, and conflicts might occur.

Customers who experience these locking errors are typically executing large-scale data loads or integrations with other internal systems that are making changes to role and group structure, user assignments to roles and groups, or both. When these processes are running—and an administrator tries to change a user's role, or the customer tries to provision a new portal user—one of these simultaneous operations might be unable to secure the lock it requires. The most likely time for this failure to occur is during periodic organizational realignment events, such as end-of-year or end-of-quarter processing, where many account assignments and user roles are changing.

Customers can lessen the chance of locking errors by:

- Scheduling separate group maintenance processes carefully so they don't overlap
- Implementing retry logic in integrations and other automated group maintenance processes to recover from a failure to acquire a lock
- Using the granular locking feature to allow some group maintenance operations to proceed simultaneously

## Takeaway: Tuning Group Membership for Performance

Understand the performance characteristics of the various group maintenance operations that you are performing and *always* test substantial configuration changes in a sandbox environment so you know what to expect in production.

Here are some specific suggestions.

- Identify user and group updates that are complex, such as user role and portal account ownership changes, or updates that involve a large amount of associated data. Allow for additional time to process these changes.
- When making changes to the hierarchy, process changes to the bottom (leaf) nodes first, then move upward to avoid duplicate processing.
- Limit the number of records of an object owned by a single user to 10,000.
- Run group maintenance operations single threaded to prevent locking. Investigate whether the use of granular locking will allow some of your operations to run simultaneously.
- Tune your updates for maximum throughput by experimenting with batch sizes and using the bulk API, where possible.
- Remove redundant paths of access, such as sharing rules that provide access to people who already have it through the hierarchy.

# Object Relationships, Bulk Loading, and Sharing Recalculation

Choices that Salesforce administrators make when designing their data models can have a major impact on sharing performance when data is loaded, updated, or transferred between users. Understanding how Salesforce handles the relationships between objects and protects data integrity during updates can help administrators optimize the performance of their data operations.

## Implicit Sharing

The sharing capabilities of the Lightning Platform platform include a wide variety of features that administrators can use to explicitly grant access to data for individuals and groups. In addition to these more familiar functions, there are a number of sharing behaviors that are built into Salesforce applications. This kind of sharing is called *implicit* because it is not configured by administrators; it is defined and maintained by the system to support collaboration among members of sales teams, customer service representatives, and clients or customers.

This table describes the different kinds of implicit sharing built into Salesforce applications and the record access that each kind provides.

| Type of Sharing | Provides | Details |
|---|---|---|
| Parent | Read-only access to the parent account for a user with access to a child record | • Not used when sharing on the child is controlled by its parent<br>• Expensive to maintain with many account children<br>• When a user loses access to a child, Salesforce needs to check all other children to see if it can delete the implicit parent. |
| Child | Access to child records for the owner of the parent account | • Not used when sharing on the child is controlled by its parent<br>• Controlled by child access settings for the account owner's role<br>• Supports account sharing rules that grant child record access<br>• Supports account team access based on team settings<br>• When a user loses access to the parent, Salesforce needs to remove all the implicit children for that user. |
| Portal | Access to portal account and all associated contacts for all portal users under that account | Shared to the lowest role under the portal account |
| High Volume[1] | Access to data owned by high volume users associated with a sharing set for users member of the sharing set's access group | All members of the sharing set access group gain access to every record owned by every high volume user associated with that sharing set |

| Type of Sharing | Provides | Details |
|---|---|---|
| High Volume Parent | Read only access to the parent account of records shared through a sharing set's access group for users member of the group | Maintains the ability to see the parent account when users are given access to account children owned by high volume users |

[1]To allow portal users to scale into the millions, Community users have a streamlined sharing model that does not rely on roles or groups, and functions similarly to calendar events and activities. Community users are provisioned with the Service Cloud Portal or Authenticated Website licenses.

## Parent-Child Data Skew

These implicit sharing behaviors simplify the task of managing security for users in Salesforce applications. They handle the most common data access use cases without requiring administrators to configure additional roles, groups, and sharing rules. Like data ownership skew, some parent-child configurations can slow the performance of large data loads and updates, and sometimes even of single-record operations.

A common configuration that can lead to poor performance is the association of a large number of child records (10,000 or more) with a single parent account. For example, a customer can have tens or hundreds of thousands of contacts generated by marketing campaigns or purchased from mailing lists—without any association to formal business accounts. If a contact is required to have an associated account, what should an administrator do? It might be convenient to park all those unallocated contacts under a single dummy account until their real business value and relationship can be determined.

While this option seems reasonable, this kind of *parent-child data skew* can cause serious performance problems in the maintenance of implicit sharing.

### Problem #1: Losing Access to a Child Record Under a Skewed Account

Assume that you have 300,000 unallocated contacts all under the same account. A user with access to one of these contacts will also have a parent implicit share in the account sharing table that gives him or her access to that account. Now what happens if that user loses access to the contact?

In order to determine whether to remove his or her sharing to the account, Salesforce needs to scan all of the other 299,999 contacts to ensure that the user doesn’t have access to them either. This practice can become expensive if Salesforce is processing a lot of visibility changes on these highly skewed accounts.

### Problem #2: Losing Access to the Skewed Parent Account

Consider the opposite scenario: The user has access to all 300,000 contacts because of his or her access to their parent account. What happens when the user loses access to the account?

This situation is not as problematic because the user must lose access to all the child records. Salesforce can query that list very quickly, but if there are very many child records, it might still take substantial time to delete all the relevant rows from the sharing tables for all the child objects.

Configuring a severe data skew on an account can also cause issues when customers make large-scale changes in sharing or realign sales assignments in Territory Management. For example, if the account is part of the source group for a sharing rule, and the administrator recalculates sharing on accounts, the work required to adjust the child entity access for that one account can cause the recalculation to become a long-running transaction or, in extreme cases, to fail altogether. Similar problems can occur when a territory realignment process attempts to evaluate assignment rules for a skewed account.

## Record-Level Locking

Many customers regularly upload large amounts of data to the service, and maintain integrations with other systems that update their data in scheduled batches or continuously in real time. Like other transactional systems, the Lightning Platform platform employs record-level database locking to preserve the integrity of data during these updates. The locks are held very briefly and don't present the same performance risks as some of the other organization locks. However, they can still cause updates to fail, so customers must still be careful not to run updates to the same collections of records in multiple threads.

In addition to taking this standard precaution, developers and administrators should know that when they are updating child records in Salesforce, the system locks the parent and the child records to prevent inconsistencies, such as updating a child record whose parent has just been deleted in another thread. When objects being processed have a parent-child relationship, two situations in particular pose a risk of producing locking errors.

- Updates to parent records and their children are being processed simultaneously in separate threads.
- Updates to child records that have the same parent records are being processed simultaneously in separate threads.

Because Salesforce holds these locks very briefly, customers who are experiencing a small number of locking errors might be able to handle the problem by adding retry logic to their integration code. Customers who experience frequent locking from integrations and mass updates should sequence batches so that the same records are not updated in multiple threads simultaneously.

## Takeaway: Tuning Data Relationships and Updates for Performance

Understand the performance characteristics of the various maintenance operations that you are performing and *always* test substantial data uploads and changes to object relationships in a sandbox environment so you know what to expect.

Here are some specific suggestions.

- Use a Public Read Only or Read/Write organization-wide default sharing model for all non-confidential data.
- To avoid creating implicit shares, configure child objects to be `Controlled by Parent` wherever this configuration meets security requirements.
- Configure parent-child relationships with no more than 10,000 children to one parent record.
- If you are encountering only occasional locking errors, see if the addition of retry logic is sufficient to solve the problem.
- Sequence operations on parent and child objects by `ParentID` and ensure that different threads are operating on unique sets of records.
- Tune your updates for maximum throughput by working with batch sizes, timeout values, the Bulk API, and other performance-optimizing techniques.

# Tools for Large-Scale Realignments

The most demanding maintenance activity that customers perform is a large-scale realignment of sales teams, territories, and account assignments. Whether customers do realignments annually, quarterly, or more frequently, the realignments typically involve extensive changes to an organization's structure and updates to large amounts of data, both of which result in many changes to record access.

At the same time, sales realignments are very time sensitive—failing to complete them quickly can adversely affect revenue. Optimizing the performance of sales realignments is naturally a key concern of many enterprise administrators, and the Lightning Platform platform includes several features to help with the planning and execution of realignments.

Note: To enable any of the following features in your organization, contact Salesforce customer support.

## Parallel Sharing Rule Recalculation

Normally, when an administrator creates, deletes, or edits a sharing rule, the recalculation required to make those changes take effect is processed synchronously. The same is also true when sharing rules are recalculated because of updates, such as a movement of roles or changes to the membership of the source group for rules. The calculation proceeds as a single job and typically completes within a few minutes.

However, when a sharing rule change affects access rights to a very large amount of data, the recalculation can run longer. In addition, a recalculation job can get killed if it is running when Salesforce performs a scheduled feature or patch release.

If you have experienced long-running processing times or jobs that were killed during realignments, consider using parallel sharing rule recalculation. When this feature is turned on, sharing rules are processed asynchronously and split into multiple simultaneous execution threads based on load. The processing is also more resilient; during a server restart, the jobs will be reinstated on the queue, and the process will continue when the server comes back online.

## Deferred Sharing Maintenance

In addition to all the technical concerns administrators must manage to perform a major realignment, they must also coordinate closely with the business to ensure that end users are not adversely affected when access rights are being adjusted. In an enterprise environment in which multiple systems are continually processing updates, it can be difficult to schedule an organization or sharing rule change that might take substantial time to complete. In order to increase the predictability of these kinds of updates, the Lightning Platform platform has recently introduced the concept of *deferred sharing maintenance*.

Here's how deferred sharing maintenance works in practice.

1. Based on requests from the business, an administrator identifies a number of changes to the role hierarchy and group membership, or updates to sharing rules.
2. Given best estimates of the remaining overall work, the administrator negotiates a maintenance window for completing the processing.

   Tip: This window should be modelled in a sandbox environment to get the best estimate possible.

3. Instead of processing each separate update and waiting for it to complete, the administrator prepares all the information required to perform all updates ahead of the planned maintenance window.
4. At the start of the maintenance window, the administrator uses the deferral feature to essentially "turn off" processing of group maintenance operations, and then makes all the desired changes to role and group membership at the same time.

   Note: Sharing rule processing is also deferred at this time so the administrator can perform all sharing rule updates.

5. Once the changes have completed, the administrator resumes processing group maintenance, and the system performs a recalculation to make all the role and group changes take effect.
6. At this point, the system is in a state that requires a *full* recalculation of all sharing rules for user access rights to be complete and accurate. The administrator can resume sharing rule processing immediately or wait to start the process at a later time. After the sharing rule recalculation has completed, all the access changes take effect.

When using the deferred sharing features, it is especially important to test the whole process in a sandbox environment.

This practice helps:

- Benchmark how long the overall recalculation is likely to take in production
- Smooth out any kinks in orchestrating deferred sharing maintenance

Note: Deferred sharing maintenance does not defer the recalculation of implicit sharing as described in the implicit sharing table on page 4. The cascading effects to implicit shares continue to be processed immediately when sharing rules are changed by administrators or through the code.

### Who's a Good Candidate for Deferred Sharing?

There are two main criteria for determining whether deferred sharing maintenance is the right tool for your organization: the size and complexity of your realignment activities, and the flexibility you have to arrange a maintenance window with your customers. If you find that organizational changes and sharing rule updates typically complete quickly enough to be scheduled into the workday and weekend slack times in your use of the service, you are unlikely to benefit substantially from this feature. On the other hand, if you are able to negotiate downtime with your business customers and have been struggling to complete updates in a timely fashion, deferred sharing might be a great solution to your problem.

## Granular Locking

By default, the Lightning Platform platform locks the entire group membership table to protect data integrity when Salesforce makes changes to roles and groups. This locking makes it impossible to process group changes in multiple threads to increase throughput on updates. When the *granular locking feature* is enabled, the system employs additional logic to allow multiple updates to proceed simultaneously if there is no hierarchical or other relationship between the roles or groups involved in the updates. Administrators can adjust their maintenance processes and integration code to take advantage of this limited concurrency to process large-scale updates faster, all while still avoiding locking errors.

The key advantages of granular locking are that:

- Groups that are in separate hierarchies are now able to be manipulated concurrently.
- Public groups and roles that do not include territories are no longer blocked by territory operations.
- Users can be added concurrently to territories and public groups.
- User provisioning can now occur in parallel.
  - Portal user creation requires locks only if new portal roles are being created.
  - Provisioning new portal users in existing accounts occurs concurrently.
- A single-long running process, such as a role delete, blocks only a small subset of operations.

This table lists all of the operations that can occur in parallel when granular locking is enabled. Note that certain operations, such as *reparenting* (moving roles within the role hierarchy), still block almost all other group updates.

| Group Operation | Can be Performed Concurrently with... |
|---|---|
| Role creation | • User role change[1]<br>• Territory reparenting<br>• Territory deletion<br>• Territory creation<br>• Removal of user from territory<br>• Addition of user to territory<br>• User provisioning[2] |
| Role deletion | • Territory reparenting |

| Group Operation | Can be Performed Concurrently with... |
|---|---|
| | • Territory deletion<br>• Territory insertion<br>• Removal of user from territory<br>• Addition of user to territory |
| Role reparenting (includes change of portal account owner) | Territory creation |
| Adding user to territory | • Role deletion<br>• Role insertion<br>• Territory creation<br>• Addition of user to territory<br>• User provisioning[3] |
| Removing user from territory | • Role deletion<br>• Role insertion<br>• Territory creation<br>• User provisioning[3] |
| Territory reparenting | • Role deletion<br>• Role insertion<br>• User provisioning[3] |
| Territory deletion | • Role deletion<br>• Role insertion<br>• User provisioning[3] |
| Territory creation | • Role reparenting<br>• Role deletion<br>• Role insertion<br>• User role change[1]<br>• Addition of user to territory<br>• Removal of user from territory<br>• User provisioning[3] |
| Provisioning internal user with an existing role | • Role insertion<br>• User role change[1]<br>• Territory reparenting<br>• Territory deletion<br>• Territory creation<br>• Removal of user from territory |

| Group Operation | Can be Performed Concurrently with... |
|---|---|
| | • Addition of user to territory<br>• User provisioning[3] |
| Changing user role (User must not own any portal accounts.) | • Role insertion<br>• Territory insertion<br>• User provisioning[3] |
| Provisioning first non-Community portal user under an account | • User role change[1]<br>• Territory reparenting<br>• Territory deletion<br>• Territory creation<br>• Removal of user from territory<br>• Addition of user to territory<br>• User provisioning[2] |
| Creating second non-Community portal user under an account | • Role insertion<br>• User role change[1]<br>• Territory reparenting<br>• Territory deletion<br>• Territory creation<br>• Removal of user from territory<br>• Addition of user to territory<br>• User provisioning[3] |
| Provisioning Community user | Any group membership operation |
| Changing portal account owner | Territory creation |
| Changing role of a user who owns a portal account | Territory creation |

[1] The user must not own any partner or customer portal accounts.

[2] Provisioning standard user or portal user in an existing portal role

[3] Provisioning any standard or portal user, including the first portal user under an account

## Who's a Good Candidate for Granular Locking?

Customers may consider using granular locking if they experience frequent and persistent locking that severely restricts their ability to manage manual and automated updates at the same time, or severely degrades the throughput of integrations or other automated group maintenance operations. Customers who only occasionally receive an error message indicating that they have encountered a group membership lock are probably not good candidates for this feature.

## Takeaway: Making Realignment Smoother

Understand the pros and cons of the performance tools, and make sure they fit well with the process and timing of your realignment. *Always* test these tools and new realignment processes in a sandbox environment so you know what to expect.

Here are some specific suggestions.

- If the time required to recalculate sharing is affecting your overall realignments schedule, consider using parallel recalculation of sharing rules.
- Consider whether it is more efficient to:
  - Set aside specific maintenance windows
  - Defer organizational or sharing rule maintenance while processing your updates

    Note: While making your decision, remember that deferring organizational maintenance will require recalculating sharing rules for all objects.

- If you have encountered issues with organizational locking, compare the maintenance you perform regularly with the increased concurrency allowed by granular locking. You might be able to increase throughput by allowing some operations to safely run in multiple threads with granular locking turned on.

## Conclusion

The record-level access controls at the heart of the Lightning Platform platform are extremely flexible and powerful, and serve the collaboration and security needs of all customers—from those working in small sales teams to those working in very large enterprises. With the knowledge and features described in this paper, Salesforce developers and administrators can optimize system performance while continuing to deliver the flexibility their companies require in access control.