Scalability

Last Updated:2026.02.25

CONTENTS

Scalability

Optimize and test your implementations. Troubleshoot errors, identify issues with application performance, and improve how you scale.

Plan Trailhead: Explore Scalability Salesforce Architects: Well-Architected Framework	**Build** Online Help: ApexGuru Overview Online Help: ApexGuru FAQ Trailhead: ApexGuru	**Test** Online Help: Scale Test Overview Online Help: Scale Test FAQ Trailhead: Scale Test Quick Look Knowledge: Large Data Volume (LDV) Org Benefit for ISV Partners
Release Knowledge: Understand Org Throttling TrailBlazer Community: Scalability	**Observe** Online Help: Scale Center Overview Online Help: Scale Center FAQ Trailhead: Scale Center Quick Look Blog: Analyze Concurrent Errors in Scale Center	

Explore Scalability

Scale Test
Use metrics to plan scale tests. Book a slot in your sandbox instance calendar, and test at production peak load. Compare performance data from recent tests.

Scale Center

Diagnose root causes and act on scale issues earlier in your development cycle. View org performance metrics and analyze performance reports.

Scale Test

Use metrics to plan scale tests. Book a slot in your sandbox instance calendar, and test at production peak load. Compare performance data from recent tests.

Access Scale Test

To get access, contact your customer success representative or account executive. From Setup, in the Quick Find box, enter `Scale`, then click **Scale Test**.

Considerations

- We recommend Scale Test when you're expecting production traffic above 120 requests per second. It lets you validate that your implementation can scale with no issues before you go live with a large rollout or peak event.
- Scale Test is available as a paid add-on for customers with a Full Sandbox, which includes the Test Scheduler and Test Execution features.
- Even if your production instance doesn't reside on Hyperforce, you can use Scale Test if your sandbox does.
- Scale Test is available in all Hyperforce regions except Singapore.

Steps

Test Plan Creation	Use this data to plan your scale tests. See how long pages are taking to load and optimize them to reduce Experienced Page Time (EPT). Use the **View Server Side Traffic** feature to get workload metrics such as triggers and platform events. Include these backend operations in your test scripts and test data.
Test Environment Setup	Create a sandbox trial run using the same code from production and review your test setup for accuracy. In the **Trial Accuracy Checker** section, a green color indicates accurate peak replication. Yellow indicates issues with your test setup that require attention. Red indicates that the sandbox test setup doesn't match production and you must fix it before your scale test.
Test Scheduler	View available test slots and book a slot. During the slot, we upgrade and enhance your sandbox infrastructure to match a particular production instance. • New slot bookings require 2 weeks of notice, based on capacity

	availability. Cancellations require at least 1 week of notice. • To see updated bookings after you delete or cancel a scale test, refresh the page.
Test Execution	After the testing slot ends, compare performance metrics from two or three recent tests and optimize your next test run. Use the **Open in Scale Center** link to access other performance and scale-related insights. Test Execution also contains an optional checklist of tasks that you can use to prepare for future testing.

Explore Scale Test

Scale Test's Limits and Suggested Usage
Read information about the technical limitations and recommended usage of Scale Test.

Scale Test FAQ
Get answers to common questions about using Scale Test.

Scale Test's Limits and Suggested Usage

Read information about the technical limitations and recommended usage of Scale Test.

- Features are enabled only after the purchase of a Scale Test license. The Restricted Product Approver (RPA) process includes a certain number of days for lead time.
- After you visit Scale Test for the first time, you're automatically opted into the collection of telemetry data. We use this data to provide scale test and execution insights. It takes 2 or 3 days after your first visit to populate.
- All communications about Scale Test booking are sent in the app or from the account team.
- The Scale Test environment isn't an exact replica of your production but rather a production-like configuration to the best of Salesforce's capabilities.
- The instance isn't an isolated setting. Other customers in the instance perform low or medium-scale activities.
- You can only book scale test days in the same sandbox where you intend to test.
- The information that you request when booking a test slot is considered final and must coincide with the actual test stats, including:
  - Requests per Second (RPS)
  - Max user logins per minute, as inputs beyond the maximum number of 60 are rejected

Capacity Evaluations

- You have the option of requesting a Capacity Evaluation from your account team to make sure you have the right environment for your updates.
- If you've run a scale test, including those results in your request is especially useful.

Key Event Management for Signature Support Customers

- While not required, we recommend that our Signature support customers request a Customer Peak Readiness (CPR) review at least 3 months before their peak date.
- A CPR review triggers the Key Event Management (KEM) engagement that prepares customers in need of additional support during their single most important event of the year.
- Conduct your scale test and share the results with KEM at least 2 weeks before your peak date.

Scale Test FAQ

Get answers to common questions about using Scale Test.

What is Scale Test and how does it differ from traditional performance testing tools?

Scale Test is a tool designed exclusively for Salesforce that enhances your sandbox infrastructure to a production-scale configuration, allowing you to test the performance of your Salesforce org under heavy load. Unlike traditional tools like JMeter or LoadRunner, which only generate load against existing environments, Scale Test actually scales your sandbox to a production-like configuration.

What types of performance tests does Scale Test support?

Scale Test supports all major performance tests, including load, stress, endurance, and spike testing. The tool handles up to 26,000 Transactions Per Second (TPS) in Hyperforce environments and scales beyond 4,500 Requests Per Second (RPS) based on customer requirements.

How quickly can I set up and run my first performance test?

Scale Test simplifies the setup process, allowing developers and architects to schedule tests in minutes. It includes a Chrome Plugin for recording user journeys that auto-generates Playwright TypeScript scripts, making it easier to test Lightning components without extensive customization.

How does Scale Test simulate real-world user interactions?

Scale Test offers a Chrome Plugin for UI journey recording and supports both Playwright (frontend) and JMeter (API-heavy) workloads. It also includes AI-powered script healing to fix common script issues and build complex workloads.

How do I calculate my Requests Per Second (RPS)?

Visit the Org Performance page in Scale Center. View the Request Volume/10 min graph, and then divide it by 600 to convert it to requests per second. If you don't have any production volume, run a small test with 10 or 20 users on your sandbox and review the graph in Scale Center.

What cloud products does Scale Test scale up to a production-like level?

Scale Test supports Sales Cloud, Service Cloud, Community Cloud, and Experience Cloud.

What performance metrics and reports are available?

Scale Test provides live client-side metrics and can differentiate between Lightning rendering and Salesforce backend performance. It includes a Test Plan Creation report that identifies scale hotspots, live observability with metrics, logs, errors, and screenshots, and integration with Scale Center for detailed investigations.

How long is my test data retained?

You can retrieve Test Plan Creation reports from the last 30 days and Trial Accuracy reports from the last 15 days.

Does Scale Test impact org performance or access my data?

No, the logs collected by Scale Test have no impact on your org's performance. Scale Test only accesses observability data collected from execution and doesn’t access your data.

How is Scale Test licensed and booked?

Scale Test is a standalone SKU priced per day, with each day providing 24 hours of enhanced, production-like sandbox infrastructure. A Scale Test day begins at 00:00 and ends at 23:59 local organization time, as configured in your Setup. New bookings require two weeks of notice. Choose a point of contact with access to the Scale Test features. That person receives email reminders and in-app notifications about upcoming activities.

Why are there unavailable scale test days on my calendar?

Another customer in your instance could be testing on those days, or they were blocked for internal reasons.

What should I do if my sandbox is refreshed or deleted?

If you refresh or delete a sandbox, you must cancel any booked Scale Test days to avoid being charged. Cancellations require at least one week of notice. If the test environment upgrade validation fails for a particular scale test day, you can request an additional test from your Salesforce account executive. Bookings are for a specific 24-hour window and can't be extended. You must adhere to the confirmed schedule. If you test beyond that, Salesforce reserves the right to stop your test.

Can minor sandbox releases impact scale tests?

Yes, minor sandbox releases are normal and can occur during scale tests. We can't prevent them, but you can run your test again to mitigate the impact, especially for standard 24-hour tests. To help us investigate performance issues, provide the request duration, normal response times, and details about the affected requests.

What happens if I exceed my booked RPS or if Salesforce cancels my test?

You must provide your expected maximum RPS when booking your test. If you test beyond that, or if your test causes issues in the overall instance, Salesforce reserves the right to stop it. If Salesforce cancels your test day, you receive at least 12 hours of notice, won't be charged for that day, and can book the next available slot. Contact your account executive.

How do I get access to Scale Test features?

A system admin can assign you to either the `ScaleTestingServiceStandardUserPS` (basic access without Test Scheduler) or the `ScaleTestingServiceAdvancedUserPS` permission set (full access). Also, see Match production and sandbox licenses without a sandbox refresh.

Scale Center

Diagnose root causes and act on scale issues earlier in your development cycle. View org performance metrics and analyze performance reports.

Access Scale Center

- From Setup, in the Quick Find box, enter *`Scale`*, then click **Scale Center**.
- To enable Scale Center for your org, turn on **Enable Scale Center**. Performance metrics appear within a few hours.

Considerations

- The Org Overview page shows you what's going well, top issues, volume insights, and recommended fixes.
- Scale Center is Generally Available in Salesforce Lightning.
- Scale Center is a free product in full sandboxes and production orgs for all Unlimited Edition (UE), Enterprise Edition (EE), Professional Edition (PE), Signature, and Scale Test customers.
- Some features in Scale Center allow you to hide and unhide URLs and other fields. All Scale Center users in your org can view these fields and any customer data they include.
- Scale Center runs on its own infrastructure, separate from your other services. Scale Center can have different privacy and security protections and it can be hosted in different physical locations.

Explore Scale Center

See What's New in Scale Center
View changes to Scale Center and added features.

Scale Center FAQ
Get answers to common questions about using Scale Center.

Enable Scale Center for Standard Users
Use the Scale Center Standard Permission Set to enable access for standard users who aren't System Administrators.

View and Compare Org Performance Metrics
Monitor and analyze your org's performance in Scale Center.

Analyze Performance Reports
To troubleshoot errors or spikes reported in an org performance query, create a performance analysis report.

Apply Scale Insights
Get insights about your Apex code, reports, and search. View database and deployment recommendations.

See Also
Trailhead: Scale Center: Quick Look
Trailhead: Explore Scalability
Knowledge Article: Understand Org Throttling
Salesforce Architects: Well-Architected Framework

See What's New in Scale Center

View changes to Scale Center and added features.

Date	What's New
February 2026	Use on-demand Database Insights to get recommendations for optimizing slow queries and improving database performance.
December 2025	Scale Center is now available in Government Cloud Plus with Org Performance, Performance Comparison, and Performance Analysis features.
October 2025	• Use on-demand Lightning Experience Insights to identify slow pages, components, and actions. Improve your End-to-End Performance Time (EPT). • Scale Center is now Generally Available for all Enterprise and Professional Edition production and full copy sandbox orgs. • The Customer Success Score (CSS) dashboard for Signature

Date	What's New
	customers now includes a direct link to Scale Center in the Technical Health section. • Scale Center is now available in LDV orgs for Partners and ISVs.
September 2025	• Use on-demand Report Insights to identify slow reports and improve their performance. • Use on-demand Search Insights to get recommendations and improve your search performance.
July 2025	Scale Center is available for all EU customers who aren't on Hyperforce EU Operating Zone (OZ).
May 2025	• Generate improved Apex Concurrency and Apex Summary investigations with actionable insights, prioritization, and recommendations. • Provide feedback on these investigations in the app.
October 2024	The Customer Success Score (CSS) dashboard includes a deep link to Scale Center for Signature customers.
September 2024	View unencrypted URLs in the Callouts Summary section of the Integrations Performance report.
August 2024	View Report Insights to see slow reports over the last week and ways to improve performance.
June 2024	• View more org performance metrics for connection pools, Visualforce, and sync/async callout errors. • Get recommendations for improving SOQL performance in the database CPU investigation.
January 2024	• View Apex insights. • Download ApexGuru insights in PDF format.
October 2023	Scale Center is available for all Performance Edition customer orgs.
September 2023	• Enable Scale Center for 5 Standard (non-SysAdmin) users per org.

Date	What's New
	• Run 15 Deep Dive investigations per week, per org.
August 2023	• All Unlimited Edition (UE) Full Sandbox orgs can access Scale Center. • Scale Center is available for all Signature customer orgs.
June 2023	Scale Center is Generally Available for all UE production orgs.

Scale Center FAQ

Get answers to common questions about using Scale Center.

How much does Scale Center cost?

It’s free.

I have an org with scale issues, but it’s not Unlimited Edition (UE) or Signature. How do I get access?

Scale Center is available in Production and Full Copy sandbox environments for all major Salesforce editions, including Enterprise and Professional.

After enabling Scale Center, why can't I see it in my UE full sandbox?

If your sandbox org license expired, match it with your UE production org.

How far back in time can I retrieve data?

For performance metrics and analysis reports, Scale Center retrieves data from the last 30 days.

Can I review more than one 24-hour period at a time?

No, the maximum time span that you can select to retrieve Org Performance metrics is 24 hours. When triaging, a shorter time span gives better results.

Which user profiles can access Scale Center?

System Administrators can access Scale Center and give access to 5 Standard (non-SysAdmin) users per org.

Is there a limit to the number of investigations I can run?

You can run up to 15 investigations per organization every 7 days. This limit uses a 7-day rolling window based on when each investigation starts. While your reports are saved for 30 days, each investigation only counts toward your limit for the first 7 days.

Is there a limit to the number of on-demand insights I can run?

For each of the main insight pages, you can run up to 3 on-demand insights per week, per org.

Can I request an increase to this weekly limit?

No, this limit is the same for all orgs.

Does a Consolidated investigation count as one or eight investigations?

A Consolidated investigation counts as one and we recommend using it to help stay under the weekly limit.

How do I get started with ApexGuru?

ApexGuru is a feature in the Scale Insights section of Scale Center.

Are Scale Center and ApexGuru available in the EU Operating Zone?

No, due to data governance and compliance requirements, customers in the EU Operating Zone aren’t able to access Scale Center or ApexGuru.

Is Scale Center available in Hyperforce?

Yes, orgs that migrate to Hyperforce have access to Scale Center.

Does Scale Center affect performance?

No, the logs used by Scale Center are collected with no impact on performance.

How do I share a report from Scale Center?

To share an analysis report with other users in the same org, copy and send them the report URL.

Does Scale Center count against Salesforce platform limits, add to API usage, or require storage space?

No.

Does Scale Center use underlying event log files, shield monitoring events, or any other data?

Scale Center aggregates and processes data from multiple internal data sources.

Why must I accept the message about backend services?

Some Scale Center backend services are hosted on Hyperforce infrastructure. All data is part of the trust layer and we follow security and data governance best practices.

My org recently migrated from 1P to Hyperforce. Can I create investigations on 1P?

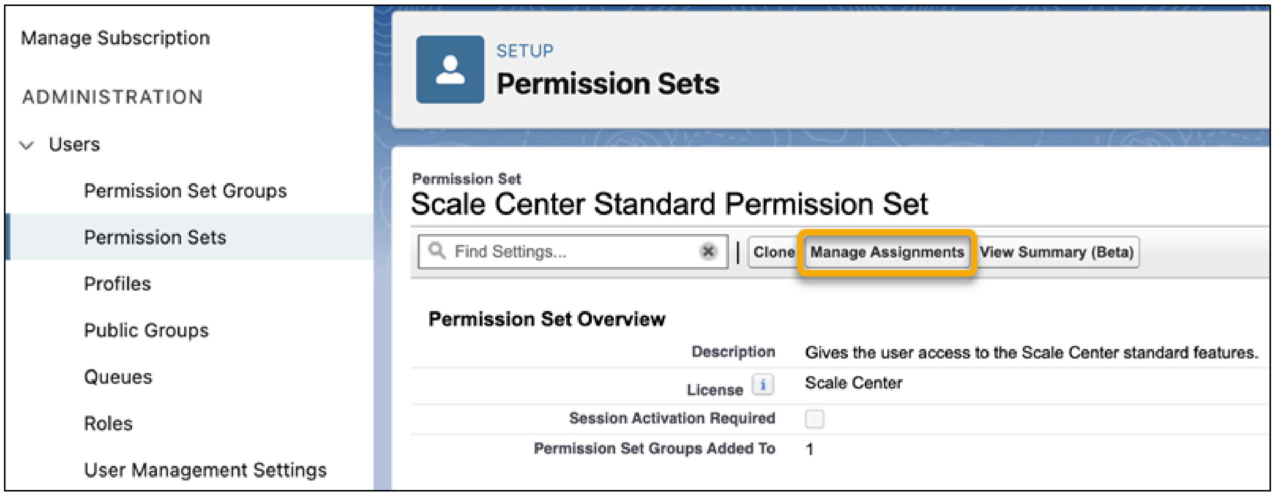
No, once you migrate to Hyperforce, you can’t create investigations on 1P. To compare metrics and data points, create investigations for 1P before migration.

Enable Scale Center for Standard Users

Use the Scale Center Standard Permission Set to enable access for standard users who aren’t System Administrators.

Your org’s admin can enable access for 5 additional users and see how many licenses are being used. Only your org’s admin can see the Permission Set Licenses section in your Company Information settings.

1. From Setup, go to Administration, then Users, and then click **Permission Sets**.
2. Click **Scale Center Standard Permission Set**.
3. Click **Manage Assignments**.

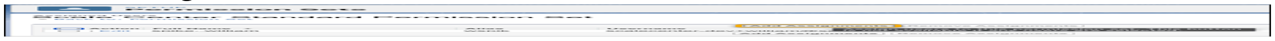


Permission Set
Scale Center Standard Permission Set

Permission Set Overview

Description	Gives the user access to the Scale Center standard features.
License	Scale Center
Session Activation Required	
Permission Set Groups Added To	1

4. Click **Add Assignments**.



5. Select 1 or more users.
6. Click **Assign**, and then save your changes.

View and Compare Org Performance Metrics

Monitor and analyze your org's performance in Scale Center.

See a snapshot of your org's performance metrics over a defined time range or compare two time ranges. Time ranges longer than 12 hours can take several minutes to load.

1. From Setup, in the Quick Find box, enter `Scale Center`, and then click **Org Performance**.
2. To change the time range covered, select a new time range.
3. You can compare two time ranges for data from the last 30 days. To choose a second time range, select **Compare**. The first time range is the Baseline Time Range, and the second is the Comparison Time Range.
4. Click **Submit**. Scale Center returns results and time series charts for key performance metrics in your selected time range.
5. To select a time range and launch an analysis, click and drag any chart.

Org Performance Charts

Summary Metric Definitions

Metric	Definition
Successful Logins	Total number of successful logins.
Failed Logins	Total number of unsuccessful logins. An unusually high number can indicate network problems.
Concurrent Apex Errors	Number of synchronous Apex requests that ran longer than 5 seconds and exceeded governor limits for your org. Possible

Metric	Definition
	causes include inefficient SOQL queries, poorly coded Apex, and synchronous callouts from Visualforce pages.
Concurrent UI Errors	Number of UI requests that ran longer than 10 seconds and exceeded governor limits for your org. Possible causes include multiple dashboard refreshes and long-running reports or dashboards. To learn more, see Proactive Alert Monitoring: Concurrent UI Errors.
Row Lock Errors	Number of exceptions raised when there are multiple attempts to update a specific record simultaneously. Possible causes include parent-child data skew, failed API or Apex triggers, and multiple Email-to-Case updates timing out. To learn more, see Proactive Alert Monitoring: Rowlock Timeout Errors.
Total Callout Errors	Number of errors when making a callout to an external, third-party package or managed package.

Click to toggle each metric, or double-click to isolate the metric.

- Total Execution Errors (errors/10 mins)
- Average Request Time (ms)
- Total Request Volume (requests/10 mins)
- Total CPU Time (ms)
- Total Logins (logins/10 mins)
- Average Callout Time (ms)
- Total Callout Errors (errors/10 mins)

Analyze Performance Reports

To troubleshoot errors or spikes reported in an org performance query, create a performance analysis report.

Start a new analysis report, see report status, and view the last 100 reports for your org. You can create 15 analysis reports per week. Reports expire after 30 days.

To share an analysis report with other users in the same org, copy and send them the report URL.

1. From Setup, in the Quick Find box, enter `Scale Center`, and then click **Performance Analysis**.
2. Click **New Analysis**.
3. Select the type, time zone, start date, and end date.
4. Give the report a name.
5. Click **Submit**.

Performance Analysis Types

- Apex Concurrency
- Apex Summary
- Database Performance
- Failed Logins
- Flow Performance
- Governor Limits
- Integrations Performance
- List Views and Reports
- Row Lock

Apex Investigations FAQ
Get answers to common questions about using Apex investigations in Scale Center.

Apex Investigations FAQ

Get answers to common questions about using Apex investigations in Scale Center.

What does Org Throttling Detection show?

Org Throttling Detection shows when your org experienced throttling due to high infrastructure usage. If throttling occurred, the report displays the reason for the throttle, start and end time, and the URI that was throttled.

How do I understand Long-Running Apex Transactions?

Time spent for All Long-Running (> 5 s) Apex Transactions illustrates how time is distributed across various components for transactions exceeding 5 seconds. These components can include:

- Database calls
- Apex execution
- Workflow time
- Compile time
- Other implementation-specific transactions

How are long-running transactions organized?

Prioritization logic combines the frequency (count) of entry points with their corresponding run time. The list of long-running transactions sorts in descending order based on their count and run time.

What do the red highlights in transaction tables indicate?

The red highlighting in specific columns indicates the area where the most run time was spent within that particular transaction, including:

- Database time
- Apex execution time
- Workflow time
- Other relevant time components

How do I provide feedback about this feature?

Click the **Thumbs Up** or **Thumbs Down** icon.

How do I see the latest version of Apex investigations?

The latest version is provided by default. To see the older version, click **Switch back to the old version**. Then to go back to the latest version, click **Try the new version**.

Apply Scale Insights

Get insights about your Apex code, reports, and search. View database and deployment recommendations.

ApexGuru Insights
Use AI and machine learning models to analyze Apex classes and entry points, giving you critical code recommendations that improve your implementation.

Lightning Experience Insights
Use on-demand insights to identify slow pages and improve their performance. Starting in Spring ‘26, this feature is generally available for production and sandbox orgs.

Database Insights
Fix expensive SOQL queries and reduce database time.

Report Insights
Use on-demand insights to identify slow reports and improve their performance.

Search Insights
Use on-demand insights to get recommendations and improve your search performance.

ApexGuru Insights

Use AI and machine learning models to analyze Apex classes and entry points, giving you critical code recommendations that improve your implementation.

- ApexGuru has no impact on org performance or storage limits.

- The Machine Learning (ML) models in ApexGuru don't use customer code.
- Sometimes, a method inaccurately appears as used or unused. Examples include System.PageReference and methods that implement specific interfaces.
- ApexGuru's automated code optimization features are directly integrated into the Code Analyzer VS Code extension. To learn more and get started, see Scan Your Apex Code for Performance Issues Using ApexGuru.

To generate ApexGuru recommendations:

1. From Setup, in the Quick Find box, enter `Scale Insights`, then click **ApexGuru Insights**.
2. Select a **Report Creation Date**. ApexGuru analyzes your code and consolidates different categories of recommended fixes.

ApexGuru Insight Categories

Insight	Definition
Code Recommendations	Ranked by expected impact, prescriptive recommendations for optimizing sObject types, methods, and memory.
SOQL/DML Analysis	Ways to optimize your SOQL and avoid performance issues.
Expensive Methods	Expensive methods consume 2–5% of CPU time, and critically expensive methods consume at least 5% of CPU time.
Unused Methods	Methods that your currently deployed Apex code doesn't call.

ApexGuru Antipatterns
The antipatterns highlighted in ApexGuru help you detect avoidable inefficiencies in your Apex code. Where practical, we also share recommendations for revising your approach to improve the efficiency of your code.

ApexGuru Test Case Antipatterns
The test cases highlighted in ApexGuru help you find places in your Apex test code that aren't valid, maintainable, or production-ready.

Activate and Use ApexGuru for Apex Code Analysis
Learn how to activate ApexGuru and analyze your Salesforce org's Apex code securely using the Einstein Trust Layer.

ApexGuru FAQ
Get answers to common questions about ApexGuru.

See Also
Get AI-Powered Insights for Your Apex Code with ApexGuru
Limitation: ApexGuru "Unused Methods"

ApexGuru Antipatterns

The antipatterns highlighted in ApexGuru help you detect avoidable inefficiencies in your Apex code. Where practical, we also share recommendations for revising your approach to improve the efficiency of your code.

ApexGuru Antipattern: Aggregating Records in Apex
SOQL queries that manually compute aggregate functions in Apex are inefficient and lead to performance issues. This approach wastes CPU resources and results in bloated, less readable code that's harder to maintain. Instead, use SOQL's built-in aggregate functions like SUM(), AVG(), COUNT(), and COUNT_DISTINCT() to perform the work at the database level.

ApexGuru Antipattern: Busy Loop Delay
Busy loops that introduce a delay aren't efficient and lead to performance issues. They waste CPU resources due to continuous loop execution. Governor limit breaches cause transaction failures. For delayed execution, use Salesforce alternatives like scheduling systems in Apex. If the delay is due to an async job, use System.enqueueJob with a delay parameter.

ApexGuru Antipattern: Copying List or Set Elements Using a For Loop
Using a For loop to copy parts of a list or set takes more CPU time and can cause governor limit breaches, especially with large datasets. Instead, Apex provides the addAll() method to copy elements efficiently in a single operation, improving both performance and code clarity.

ApexGuru Antipattern: DML in a Loop
Executing DML operations (insert, update, and delete) inside loops causes excessive resource consumption and can hit governor limits. Instead, batch DML operations by storing records in a list and performing the DML operation one time outside of the loop.

ApexGuru Antipattern: Expensive Debug Statements
Overusing System.debug() slows down execution, especially in production environments. Writing large amounts of data to logs increases the execution time and affects performance. Plus, the logs fill, making it harder to identify important issues.

ApexGuru Antipattern: Expensive Methods
Hot methods that consume over 5% of CPU time can cause performance bottlenecks, breach governor limits, and decrease system performance. Expensive methods consume more than 2% but less than 5% of CPU time. Critically expensive methods consume more than 5% of CPU time. To reduce CPU consumption, refactor expensive methods.

ApexGuru Antipattern: Expensive String Comparison
The == operator for string comparison invokes java.text.Collator.compare(), which is slower than using String.equals() or String.equalsIgnoreCase(). Longer strings and larger loops increase overhead.

ApexGuru Antipattern: Limits.getHeapsize() Methods
The Limits.getHeapSize() method monitors memory consumption during execution. This operation increases resource consumption, especially when used in production environments. When used in debug statements or logs, this method is evaluated even if the logs aren't printed in production and this causes more performance degradation.

ApexGuru Antipattern: Redundant SOQL
Executing multiple SOQL queries on the same sObject with identical or nearly identical filter

conditions creates inefficiencies. This practice wastes governor limits and increases transaction time because it requires more database round-trips. Processing separate query responses wastes CPU time and heap space.

ApexGuru Antipattern: Schema.getGlobalDescribe() Called Multiple Times Within a Class
Multiple calls to Schema.getGlobalDescribe() in a class cause performance issues and are computationally expensive. Calling this method multiple times within the same method results in redundant schema retrieval. Increased CPU usage can cause you to go over your governor limits.

ApexGuru Antipattern: Schema.getGlobalDescribe() In a Loop
The Schema.getGlobalDescribe() method collects schema information on all SObjects, which is computationally expensive. Using Schema.getGlobalDescribe() inside a loop can lead to performance issues, including breaking governor limits.

ApexGuru Antipattern: Schema.getGlobalDescribe() Not Efficient
Using Schema.getGlobalDescribe() to retrieve metadata for a single known SObject isn't efficient. This method causes unnecessary overhead, decreases performance, and increases the latency of the associated entry point.

ApexGuru Antipattern: SObject Map in a For Loop
Building Map<Id, sObject> in a For loop inserts one record at a time. This query fetches all Custom Object__c records without filtering, which causes extra CPU usage. Filtering later in Apex boosts memory use, risks overflow, and slows performance. Instead, use Apex's map constructor or putAll().

ApexGuru Antipattern: SOQL in a Loop
Executing SOQL queries inside loops can quickly exhaust the governor limits and decrease system performance. Salesforce enforces a limit on the number of queries per transaction. Instead, batch SOQL queries outside of loops.

ApexGuru Antipattern: SOQL in a Loop (One Hop)
The SOQL query is placed in a method in one class and the loop that invokes this method is in a different class. This structure leads to excessive SOQL queries, potentially hitting governor limits. Repeated query execution across class boundaries causes performance bottlenecks. Refactor the code to batch SOQL queries outside the loop, even if they're spread across multiple classes.

ApexGuru Antipattern: SOQL with Apex Filter
Executing SOQL queries without a WHERE clause or LIMIT statement and filtering them in Apex retrieves unnecessary data. This process increases memory and CPU usage and can lead to overall system performance and governor limit issues.

ApexGuru Antipattern: SOQL with Negative Expressions
SOQL queries that use negative expressions such as NOT IN or != don't index effectively, leading to full-table scans. Less-efficient database operations increase query time and CPU usage. Avoid negative expressions. Use positive expressions wherever possible.

ApexGuru Antipattern: SOQL with Unused Fields
Retrieving unused fields in SOQL queries increases overhead, resource consumption, and query execution time. It also increases heap size and causes governor limits to be exceeded, especially with large datasets or complex objects. Only include fields directly required by your logic. Regularly review and improve SOQL queries to remove unused fields.

ApexGuru Antipattern: SOQL with Wildcard Filter
Using a wildcard such as LIKE '%term in SOQL queries prevents the use of indexes and causes full-

table scans. Full-table scans increase query execution time and CPU usage. Use indexed fields for filtering and optimized search strategies.

ApexGuru Antipattern: SOQL Without a WHERE Clause or LIMIT Statement
Executing SOQL queries without a WHERE clause or LIMIT statement can retrieve too many records, leading to governor-limit breaches for the number of rows returned. To restrict the amount of data returned, include a WHERE clause or LIMIT statement in SOQL queries.

ApexGuru Antipattern: SOQL Without Platform Cache
Running the same SOQL query repeatedly during a transaction or across requests can slow down performance and put an unnecessary load on the database. Instead, use Platform Cache to eliminate the need for custom settings or persistent storage.

ApexGuru Antipattern: Sorting in Apex
Sorting records in Apex code uses more central processing unit (CPU) time and can exceed governor limits. This is especially true for large datasets. Using Apex to sort can also lead to heap-size overflows and degraded query performance.

ApexGuru Antipattern: Unused Methods
Unused Apex methods contribute to code bloat and increase the cognitive load for developers. These pieces of code make it harder to maintain the system and can also increase the size of the deployment package, making it more difficult to stay within the Salesforce Apex code size limits. Identify and remove unused methods to improve code clarity and performance.

ApexGuru Antipattern: Aggregating Records in Apex

SOQL queries that manually compute aggregate functions in Apex are inefficient and lead to performance issues. This approach wastes CPU resources and results in bloated, less readable code that's harder to maintain. Instead, use SOQL's built-in aggregate functions like SUM(), AVG(), COUNT(), and COUNT_DISTINCT() to perform the work at the database level.

Scope

Detection only

Detection Example

The developer is querying individual records and performing the aggregation (summing `Points__c`) manually in Apex, likely within a loop per user, rather than letting the database engine handle the aggregation efficiently.

```
for(User u : allActiveUsers){
    Decimal i = 0;
    List<Achievement__c> achievements = [
        SELECT Id, Points__c, User__c, Statut__c, CreatedDate
        FROM Achievement__c
        WHERE Status__c = 'Valid'
        AND CreatedDate > :currentChallenge.StartDate__c
    ];
    for(Achievement__c achievement: achievements){
            i += achievement.Points__c;
    }
    System.debug('Total: ' + i);
}
```

Recommendation

Use SOQL aggregate functions like `SUM()` to shift computation to the database for better scalability.

```
for(User u : allActiveUsers){
    Decimal i = 0;
    AggregateResult ar = [
        SELECT SUM(Points__c) totalPoints
        FROM Achievement__c
        WHERE Status__c = 'Valid'
        AND CreatedDate > :currentChallenge.StartDate__c
        AND User__c = :u.Id
    ];
    if(ar != null && ar.get('totalPoints') != null){
        i = (Decimal)ar.get('totalPoints');
    }
    System.debug('Total: ' + i);
}
```

ApexGuru Antipattern: Busy Loop Delay

Busy loops that introduce a delay aren't efficient and lead to performance issues. They waste CPU resources due to continuous loop execution. Governor limit breaches cause transaction failures. For delayed execution, use Salesforce alternatives like scheduling systems in Apex. If the delay is due to an async job, use System.enqueueJob with a delay parameter.

Scope

Detection only

Detection Example

```
Apex
public class JobScheduler {
    public void scheduleJobWithDelay() {
        Integer delayTime = 10; // Delay in seconds
        DateTime startTime = System.now();
        // Busy loop to create a delay
        while (System.now() < startTime.addSeconds(delayTime)) {
            // Do nothing (busy wait)
        }
        // Enqueue the Queueable job after the delay
        ID jobId = System.enqueueJob(new MyQueueableJob());
    }
}
```

ApexGuru Antipattern: Copying List or Set Elements Using a For Loop

Using a For loop to copy parts of a list or set takes more CPU time and can cause governor limit breaches, especially with large datasets. Instead, Apex provides the addAll() method to copy elements efficiently in a single operation, improving both performance and code clarity.

Scope

Detection and recommendation

Detection Example

The developer is manually iterating over `attachmentPresentIdsCassic` to copy each `Id` into `attachmentPresentIds`.

```
for (Id ids : attachmentPresentIdsCassic){
    attachmentPresentIds.add(ids);
}
```

Recommendation

The recommended fix replaces this manual loop with the built-in `addAll()` method, which copies the entire collection in one step.

```
attachmentPresentIds.addAll(attachmentPresentIdsCassic);
```

ApexGuru Antipattern: DML in a Loop

Executing DML operations (insert, update, and delete) inside loops causes excessive resource consumption and can hit governor limits. Instead, batch DML operations by storing records in a list and performing the DML operation one time outside of the loop.

Scope

Detection only

Detection Example

```
apex
 for (Account acc : accounts) {
      insert acc;
```

ApexGuru Antipattern: Expensive Debug Statements

Overusing System.debug() slows down execution, especially in production environments. Writing large amounts of data to logs increases the execution time and affects performance. Plus, the logs fill, making it harder to identify important issues.

Scope

Detection and recommendation

Detection Example

```
apex
  System.debug('Heap Size: ' + Limits.getHeapSize());
```

Recommendation

To make sure that expensive operations aren't evaluated unless necessary, perform a conditional check before invoking the call. Minimize debug statements in production code. Use conditional checks before invoking expensive calls, especially if the debug statement involves expensive computations such as `Limits.getHeapSize`.

```
apex
// Commenting out the debug log in production environments
// System.debug('Heap Size: ' + Limits.getHeapSize());
```

ApexGuru Antipattern: Expensive Methods

Hot methods that consume over 5% of CPU time can cause performance bottlenecks, breach governor limits, and decrease system performance. Expensive methods consume more than 2% but less than 5% of CPU time. Critically expensive methods consume more than 5% of CPU time. To reduce CPU consumption, refactor expensive methods.

Scope

Detection only

ApexGuru Antipattern: Expensive String Comparison

The == operator for string comparison invokes java.text.Collator.compare(), which is slower than using String.equals() or String.equalsIgnoreCase(). Longer strings and larger loops increase overhead.

Scope

Detection and recommendation

Detection Example

```
apex
  if (content.respKey == key) {
      // Expensive string comparison
  }
```

Recommendation

Use `String.equals()` for case-sensitive comparisons and `String.equalsIgnoreCase()` when ignoring capitalization. Avoid using `==` for a string comparison when locale-based comparison isn't required. To prevent NullPointerException during comparison, add a null check.

```
apex
    if (content.respKey != null && content.respKey.equals(key)) {
      // Optimized string comparison
  }
```

ApexGuru Antipattern: Limits.getHeapsize() Methods

The Limits.getHeapSize() method monitors memory consumption during execution. This operation increases resource consumption, especially when used in production environments. When used in debug statements or logs, this method is evaluated even if the logs aren't printed in production and this causes more performance degradation.

Scope

Detection and recommendation

Detection Example

```
apex
System.debug('Current Heap Size: ' + Limits.getHeapSize());
```

Recommendation

Avoid using `Limits.getHeapSize()` in production environments unless necessary. Use this method with conditions or remove it if monitoring heap size isn't required during normal execution.

```
apex
// Commenting out the Limits.getHeapSize() call in production environments
// System.debug('Current Heap Size: ' + Limits.getHeapSize());
```

ApexGuru Antipattern: Redundant SOQL

Executing multiple SOQL queries on the same sObject with identical or nearly identical filter conditions creates inefficiencies. This practice wastes governor limits and increases transaction time because it requires more database round-trips. Processing separate query responses wastes CPU time and heap space.

Scope

Detection and recommendation

Detection Example

```
//Apex
// Bad: Two round-trips to the database
```

```
User userA = [SELECT Id FROM User WHERE ContactId = :idA];
User userB = [SELECT Id FROM User WHERE ContactId = :idB];
```

Recommendation

Merge these queries into fewer SOQL statements, followed by in-memory filtering in Apex. Query all records at the same time and map them for easy access. This scales regardless of whether you need 2 records or 200.

```
//Apex
// Good: One round-trip using a Set and Map
Set<Id> contactIds = new Set<Id>{ idA, idB };
// 1. Bulk Query
List<User> users = [SELECT Id, ContactId FROM User WHERE ContactId IN :contact
Ids];
// 2. In-Memory Organization (The Map Pattern)
Map<Id, User> userMap = new Map<Id, User>();
for(User u : users) {
    userMap.put(u.ContactId, u);
}
// 3. Retrieve efficiently
User userA = userMap.get(idA);
User userB = userMap.get(idB);
```

ApexGuru Antipattern: Schema.getGlobalDescribe() Called Multiple Times Within a Class

Multiple calls to Schema.getGlobalDescribe() in a class cause performance issues and are computationally expensive. Calling this method multiple times within the same method results in redundant schema retrieval. Increased CPU usage can cause you to go over your governor limits.

Scope

Detection and recommendation

Detection Example

```
public void pro(String objectName) {

    // Get the describe result for the provided object name
    Schema.DescribeSObjectResult describeResult = Schema.getGlobalDescribe().g
et(objectName).getDescribe();
```

```
    // Log object label and plural label
    System.debug('Object Label: ' + describeResult.getLabel());
    System.debug('Object Plural Label: ' + describeResult.getLabelPlural());

    // Dynamically create a new instance of the SObject
    SObject newRecord = Schema.getGlobalDescribe().get(objectName).newSObjec
t();

    newRecord.put('Name', 'Address');

    // Insert the record
    insert newRecord;

    // Log the newly created record ID
    System.debug('New record created with ID: ' + newRecord.Id);
}
}
```

Recommendation

When you optimize logic design, avoid multiple `Schema.getGlobalDescribe()` calls. Focus on optimizing the functionality itself. Use alternatives like caching or metadata to eliminate the need for these calls. Retrieve schema information one time at the start of the method and reuse it to improve performance.

```
public void getObjectDescribe(String objectName) {
    // Get the describe result for the provided object name
    Schema.DescribeSObjectResult describeResult = null
    try {
         List<Schema.DescribeSObjectResult> describes = Schema.describeSObject
s(new String[] { objName}, SObjectDescribeOptions.DEFERRED);
         result = describes[0];
     } catch (InvalidParameterValueException ipve) {
          result = null;
     }
     Schema.DescribeSObjectResult objectResult = result;

    // Log object label and plural label

    if (objectResult != null) {
        System.debug('Object Label: ' + describeResult.getLabel());
        System.debug('Object Plural Label: ' + describeResult.getLabelPlura
l());
    }

    // Dynamically create a new instance of the SObject
    SObject newRecord = (SObject)Type.forName( objectName ).newInstance()

    newRecord.put('Name', 'Address');

    // Insert the record
    insert newRecord;

    // Log the newly created record ID
    System.debug('New record created with ID: ' + newRecord.Id);
}
```

ApexGuru Antipattern: Schema.getGlobalDescribe() In a Loop

The Schema.getGlobalDescribe() method collects schema information on all SObjects, which is computationally expensive. Using Schema.getGlobalDescribe() inside a loop can lead to performance issues, including breaking governor limits.

Scope

Detection and recommendation

Detection Example

```
apex
public class ObjectMetadataHandler {

    public void processObjectNames(List<String> objectNames) {
        for (String objectName : objectNames) {
            if (Schema.getGlobalDescribe().containsKey(objectName)) {
                Schema.SObjectType sObjectType = objectMap.get(objectName);
                System.debug('Object found: ' + objectName);
            } else {
                System.debug('Object not found: ' + objectName);
            }
        }
    }
}
```

Recommendation

Cache the results of `Schema.getGlobalDescribe()` outside of the loop and reuse them within the loop.

```
public class ObjectMetadataHandler {
    // Cache the result of Schema.getGlobalDescribe() to avoid redundant calls
    private static Map<String, Schema.SObjectType> objectMap = Schema.getGloba
lDescribe();

    public void processObjectNames(List<String> objectNames) {
        for (String objectName : objectNames) {
            // Use the cached objectMap to access SObjectType
            if (objectMap.containsKey(objectName)) {
                Schema.SObjectType sObjectType = objectMap.get(objectName);
                System.debug('Object found: ' + objectName);
            } else {
                System.debug('Object not found: ' + objectName);
            }
        }
    }
}
```

See Also

getGlobalDescribe() in the Apex Reference Guide

ApexGuru Antipattern: Schema.getGlobalDescribe() Not Efficient

Using Schema.getGlobalDescribe() to retrieve metadata for a single known SObject isn't efficient. This method causes unnecessary overhead, decreases performance, and increases the latency of the associated entry point.

Scope

Detection and recommendation

Detection Example

```
apex
Schema.DescribeSObjectResult dsr = Schema.getGlobalDescribe().get('Case').getD
escribe();
```

Recommendation

Use the `getSObjectType()` method to retrieve the metadata.

```
Schema.DescribeSObjectResult dsr = Case.sObjectType.getDescribe();
```

ApexGuru Antipattern: SObject Map in a For Loop

Building Map<Id, sObject> in a For loop inserts one record at a time. This query fetches all Custom Object__c records without filtering, which causes extra CPU usage. Filtering later in Apex boosts memory use, risks overflow, and slows performance. Instead, use Apex's map constructor or putAll().

Scope

Detection and recommendation

Detection Example

This code manually loops through each sObject and inserts it into the map using `put()`, which adds unnecessary CPU overhead.

```
Map<Id, Account> accMap = new Map<Id, Account>();
for (Account acc : [SELECT Id, Name FROM Account]) {
    accMap.put(acc.Id, acc);
}
```

Recommendation

Increase map creation efficiency by using list constructor.

```
List<Account> accs = [SELECT Id, Name FROM Account];
Map<Id, Account> accMap = new Map<Id, Account>(accs);
```

If you're working with large results or paginated queries, use `putAll()` outside of the loop.

```
Map<Id, Account> accMap = new Map<Id, Account>();
for (List<Account> accs : [SELECT Id, Name FROM Account]) {
    accMap.putAll(accs);
}
```

ApexGuru Antipattern: SOQL in a Loop

Executing SOQL queries inside loops can quickly exhaust the governor limits and decrease system performance. Salesforce enforces a limit on the number of queries per transaction. Instead, batch SOQL queries outside of loops.

Scope

Detection only

Detection Example

```
apex
for (Integer i = 0; i < 10; i++) {
      List<Account> accounts = [SELECT Id FROM Account WHERE Name = :someValu
e];
  }
```

See Also

SOQL For Loops in the Apex Developer Guide

ApexGuru Antipattern: SOQL in a Loop (One Hop)

The SOQL query is placed in a method in one class and the loop that invokes this method is in a different class. This structure leads to excessive SOQL queries, potentially hitting governor limits. Repeated query execution across class boundaries causes performance bottlenecks. Refactor the code to batch SOQL queries outside the loop, even if they're spread across multiple classes.

Scope

Detection only

Detection Example

```
apex
public class FirstClass {
      public static List<Contact> getContactsForAccount(Id accId) {
          return [SELECT Id, Name FROM Contact WHERE AccountId = :accId LIMIT
5];
      }
  }
  public class SecondClass {
      public static void processAccounts() {
          List<Account> accounts = FirstClass.getAccounts();
          for (Account acc : accounts) {
              List<Contact> contacts = FirstClass.getContactsForAccount(acc.I
d);
              // Process contacts
          }
      }
  }
```

ApexGuru Antipattern: SOQL with Apex Filter

Executing SOQL queries without a WHERE clause or LIMIT statement and filtering them in Apex retrieves unnecessary data. This process increases memory and CPU usage and can lead to overall system performance and governor limit issues.

Scope

Detection and recommendation

Detection Example

```
List<Custom_Object__c> records = [
    SELECT Id, Name, Category__c, Amount__c, Status__c,
        Priority__c, Type__c
    FROM Custom_Object__c
];
List<Custom_Object__c> filteredRecords = new List<Custom_Object__c>();
for (Custom_Object__c record : records) {
    if (record.Status__c == 'Open' || record.Status__c == 'Pending') {
        filteredRecords.add(record);
    }
}
```

Recommendation

To minimize data volume and improve query efficiency, avoid postprocessing and filtering in Apex. Apply filters directly in the SOQL query using `WHERE` clauses and restrict results using `LIMIT` statements. This recommendation prevents Central Processing Unit (CPU) and memory limit breaches.

```
List<Custom_Object__c> filteredRecords = [
     SELECT Id, Name, Category__c, Amount__c, Status__c,
        Priority__c, Type__c
    FROM Custom_Object__c
    WHERE Status__c = 'Open'
       OR Status__c = 'Pending'
];
```

ApexGuru Antipattern: SOQL with Negative Expressions

SOQL queries that use negative expressions such as NOT IN or != don't index effectively, leading to full-table scans. Less-efficient database operations increase query time and CPU usage. Avoid negative expressions. Use positive expressions wherever possible.

Scope

Detection only

Detection Example

```
apex
  [SELECT Id FROM Opportunity WHERE StageName NOT IN ('Closed Won', 'Closed Lo
st')];
```

ApexGuru Antipattern: SOQL with Unused Fields

Retrieving unused fields in SOQL queries increases overhead, resource consumption, and query execution time. It also increases heap size and causes governor limits to be exceeded, especially with large datasets or complex objects. Only include fields directly required by your logic. Regularly review and improve SOQL queries to remove unused fields.

Scope

Detection and recommendation

Detection Example

```
apex
SELECT id, KnowledgeArticleId, PublishStatus, Article_Tag__c
FROM Knowledge__kav
```

```
WHERE id = :recordId
```

Recommendation

Remove the unused `KnowledgeArticleId` field to optimize the query. Include only fields directly required by your logic. Regularly review and optimize SOQL queries.

```
SELECT id, PublishStatus, Article_Tag__c
FROM Knowledge__kav
WHERE id = :recordId
```

ApexGuru Antipattern: SOQL with Wildcard Filter

Using a wildcard such as LIKE '%term in SOQL queries prevents the use of indexes and causes full-table scans. Full-table scans increase query execution time and CPU usage. Use indexed fields for filtering and optimized search strategies.

Scope

Detection only

Detection Example

```
apex
SELECT Id, LastName, FirstName FROM Contact WHERE LastName LIKE '%smi'
```

ApexGuru Antipattern: SOQL Without a WHERE Clause or LIMIT Statement

Executing SOQL queries without a WHERE clause or LIMIT statement can retrieve too many records, leading to governor-limit breaches for the number of rows returned. To restrict the amount of data returned, include a WHERE clause or LIMIT statement in SOQL queries.

Scope

Detection only

Detection Example

```
apex
```

```
[SELECT Id, Name FROM Account];
```

ApexGuru Antipattern: SOQL Without Platform Cache

Running the same SOQL query repeatedly during a transaction or across requests can slow down performance and put an unnecessary load on the database. Instead, use Platform Cache to eliminate the need for custom settings or persistent storage.

Scope

Detection only

Cache query results for faster data reuse across sessions or your entire org using Platform Cache. This approach is particularly helpful for frequently used queries on semistatic data invoked by various flows or trigger paths.

Detection Example

```
// Code without Cache
List<BranchContact__c> conlist = [SELECT Id,BillingState FROM Contact];

// Code with Cache
String cacheQuery = 'SELECT Id,BillingState FROM Contact';
conlist = BrContactPlatformCacheUtility.fetchFromCache('BrContactObjectCache',
'uid_query', cacheQuery);

// Platform Cache implementation
public class BrContactPlatformCacheUtility implements Cache.CacheBuilder {
   public static String query;
   public static String key;

   // Method to fetch from Cache
   public static List<sObject> fetchFromCache(String cacheName, String key, St
ring query) {
      List<sObject> lstSObjValue = new List<sObject>();
      try {
         Cache.OrgPartition orgPartitionVar = Cache.Org.getPartition(cacheNam
e);
         BrContactPlatformCacheUtility.query = query;
         BrContactPlatformCacheUtility.key = key;
         lstSObjValue = (List<sObject>)
orgPartitionVar.get(BrContactPlatformCacheUtility.class, key);
```

```
        catch (Exception ex) {
            // Handle Exception
        }
    return lstSObjValue;
    }

    // Method to Clear the Cache
    public static void clearCacheKey(String cacheName, String key) {
        Cache.OrgPartition orgPartitionVar = Cache.Org.getPartition(cacheName);
orgPartitionVar.remove(BrContactPlatformCacheUtility.class, key);
    }
}
```

ApexGuru Antipattern: Sorting in Apex

Sorting records in Apex code uses more central processing unit (CPU) time and can exceed governor limits. This is especially true for large datasets. Using Apex to sort can also lead to heap-size overflows and degraded query performance.

Scope

Detection and recommendation

Detection Example

```
List<Contact> conlist = [SELECT Id,BillingState FROM Contact];

conlist.sort();
```

Recommendation

Use the `ORDER BY` clause in SOQL to delegate sorting to the database layer.

```
List<sObject> conlist = [SELECT Id, BillingState FROM Contact ORDER BY Billing
State];
```

ApexGuru Antipattern: Unused Methods

Unused Apex methods contribute to code bloat and increase the cognitive load for developers. These pieces of code make it harder to maintain the system and can also increase the size of the deployment package, making it more difficult to stay within the Salesforce Apex code size limits. Identify and remove unused methods to improve code clarity and performance.

Scope

Detection only

See Also

Limitation: ApexGuru "Unused Methods"

Apex Governor Limits in the Salesforce Developer Limits and Allocations Quick Reference

ApexGuru Test Case Antipatterns

The test cases highlighted in ApexGuru help you find places in your Apex test code that aren’t valid, maintainable, or production-ready.

ApexGuru Test Case Antipattern: Using the testMethod Keyword
The testMethod syntax is outdated. Avoid using it to preserve confidence in your test suite and deployment quality.

ApexGuru Test Case Antipattern: Writing Filler Statements
Using meaningless code to inflate test coverage undermines code quality. Filler statements cause technical debt, missed bugs, and false code confidence. Focus on quality over quantity. Write meaningful tests for actual business logic and use coverage as a guide, not the goal.

ApexGuru Test Case Antipattern: Using the testMethod Keyword

The testMethod syntax is outdated. Avoid using it to preserve confidence in your test suite and deployment quality.

Scope

Detection and recommendation

Detection Example

```
public class MyTestClass {
   static testMethod void myTest() {
      System.debug('This test will be versioned out soon as it has testMethod
keyword');
   }
}
```

Recommendation

Always annotate test classes with `@isTest`. Avoid legacy syntax. Adopt modern standards that properly recognize and execute all test methods.

```
@isTest
private class MyTestClass {
   @isTest
   static void myTest() {
      System.debug('This test follows Salesforce recommended practices');
   }
}
```

ApexGuru Test Case Antipattern: Writing Filler Statements

Using meaningless code to inflate test coverage undermines code quality. Filler statements cause technical debt, missed bugs, and false code confidence. Focus on quality over quantity. Write meaningful tests for actual business logic and use coverage as a guide, not the goal.

Scope

Detection only

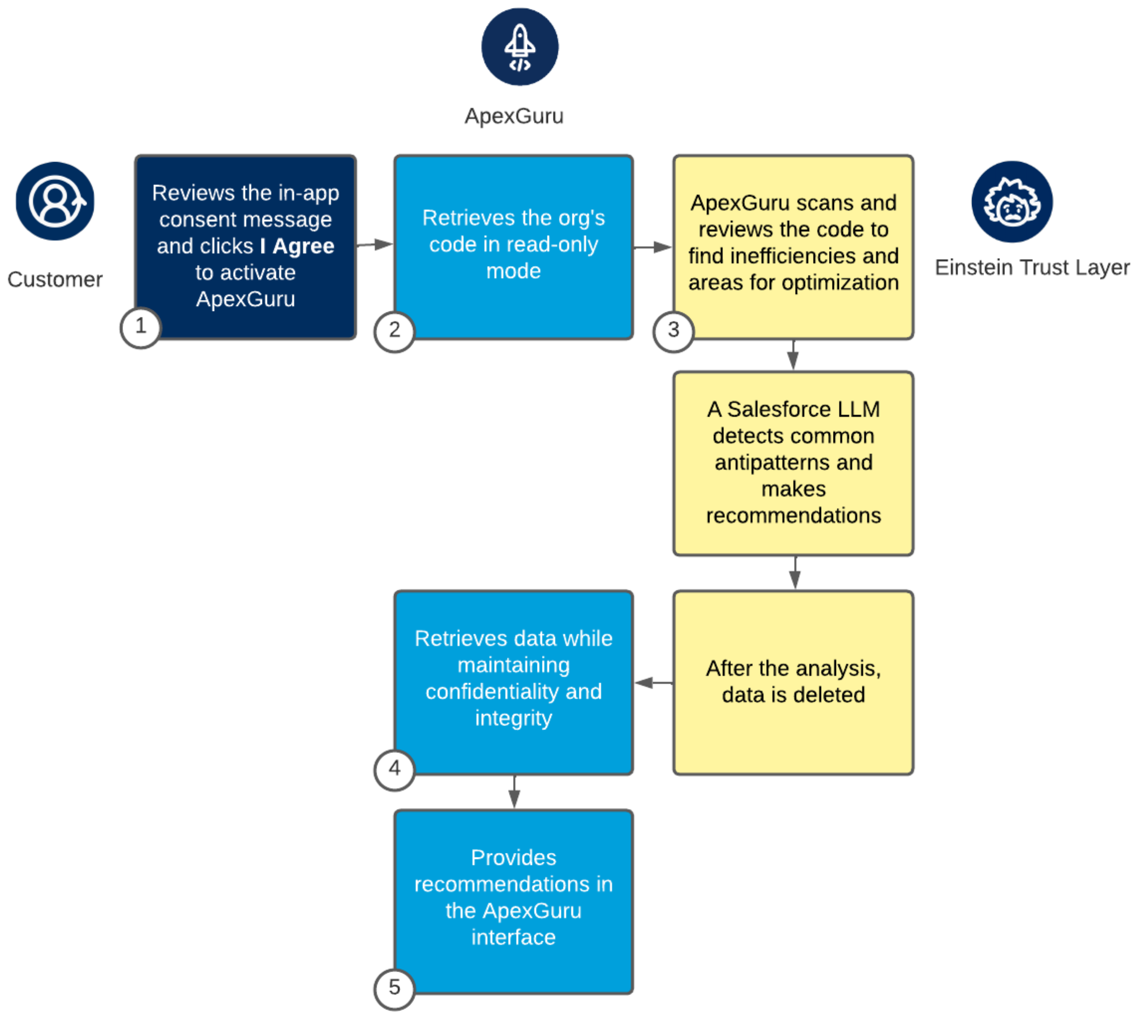
Detection Example

```
public class CodeCoverageInflationExample1 {
   public static void CodeCoverageInflation1() {

      //Filler code
      String str1 = 'Placeholder ';
      String str2 = 'Code';
      String result = str1 + str2;
      assertEquals('Placeholder Code', result);

      integer j = 0;
      j++;
      j++;
      j++;
      j++;
      j++;
      j++;
      j++;
```

```
        j++;
        j++;
        j++;
        j++;
        j++;
        j++;
        j++;
        j++;
        j++;

    }
}
```

Activate and Use ApexGuru for Apex Code Analysis

Learn how to activate ApexGuru and analyze your Salesforce org's Apex code securely using the Einstein Trust Layer.


ApexGuru
Customer
Reviews the in-app consent message and clicks **I Agree** to activate ApexGuru
1
Retrieves the org's code in read-only mode
2
ApexGuru scans and reviews the code to find inefficiencies and areas for optimization
3
Einstein Trust Layer
A Salesforce LLM detects common antipatterns and makes recommendations
After the analysis, data is deleted
Retrieves data while maintaining confidentiality and integrity
4
Provides recommendations in the ApexGuru interface
5

1. Review the legal terms for ApexGuru and click **I Agree** to activate the service.
2. Allow ApexGuru to securely retrieve your org's Apex code in read-only mode for dynamic grounding.
3. Let the Einstein Trust Layer provide data masking, secure gateway, and zero data retention during the analysis process.
   a. ApexGuru scans and reviews the uploaded Apex code to identify inefficiencies and areas of optimization.

      These hotspots and performance bottlenecks can impact overall performance.

   b. A Salesforce-owned LLM within the Einstein Trust Layer detects common antipatterns in the code and prioritizes recommendations based on their potential impact.
   c. Ensure that the data is immediately deleted after the analysis is complete.
4. Allow ApexGuru to retrieve data while maintaining the confidentiality and integrity of the Apex code during processing and transmission.
5. Review the recommendations generated by the model in the ApexGuru interface for further analysis.

You have securely activated ApexGuru, analyzed your Apex code, and received prioritized recommendations for optimization in your ApexGuru interface.

ApexGuru FAQ

Get answers to common questions about ApexGuru.

What is ApexGuru?

ApexGuru is an AI-powered tool that analyzes your Apex code to find and fix performance-related issues. ApexGuru provides insights within 24 hours of opt-in and refreshes weekly. Retrieve insights for the last 30 days.

How does ApexGuru analyze my code?

ApexGuru requires read-only access to your Apex code. It makes an encrypted copy for analysis and then immediately deletes it. It does not store your code long-term.

What does ApexGuru evaluate my code for?

ApexGuru checks for various governor limits, including CPU time, SOQL queries and statement length, DML statements, and the total Apex character limit. It also identifies antipatterns like SOQL and DML statements within loops.

Are there any usage limits for ApexGuru?

Insight generation is limited to **3** per rolling week. If you're using the ApexGuru integration with Code Analyzer, you're limited to **50** scans per org per hour.

How should I configure my network security for ApexGuru?

To ensure uninterrupted connectivity, we recommend that you use domain-based allowlisting (for example, salesforce.com) instead of IP-based restrictions. This approach is more reliable for cloud apps. In most configurations, ApexGuru traffic appears as standard Salesforce traffic. Make sure that your network allows traffic from the standard Salesforce IP ranges.

Can I allowlist specific IP addresses?

No, ApexGuru doesn’t support strict IP allowlisting for individual app instances. Because our cloud infrastructure is dynamic, it assigns and rotates addresses without notice. Even providing static ranges can create a false sense of security and lead to service interruptions. If you hard code dynamic IP addresses into a firewall, connectivity will break whenever the infrastructure automatically scales or updates.

What is the Test Cases feature?

The Test Cases feature scans your Apex code to identify antipatterns and missing best practices, ensuring your test code is valid and ready for production.

What do the Test Case severity levels mean?

ApexGuru categorizes issues with a static ruleset:

- **Critical**: Issues likely to cause production problems or stop deployments. An example is using filler statements to inflate code coverage.
- **Major**: Non-blocking issues that are important for improving test reliability and maintainability, such as using `testMethod` without the `@isTest` class declaration.
- **Minor**: Style or efficiency issues that affect readability, like non-standard naming conventions or redundant variables.

Is ApexGuru available in Hyperforce?

Yes, ApexGuru is available in Hyperforce. If it was enabled before migration, it remains active afterward.

Does enabling ApexGuru affect my org's performance?

No, ApexGuru uses logs that are collected without any impact on your org's performance. It also doesn't count towards platform limits.

Is ApexGuru available in all Salesforce orgs?

No. ApexGuru is available only in Production and Full Copy Sandbox orgs for Unlimited, Enterprise, Professional, and Signature Editions. It's not available in Developer Edition orgs.

ApexGuru Insights are available via Code Analyzer in orgs that have already activated ApexGuru (Prod or Full Copy). You can see these insights in the following org types:

- Developer Sandbox
- Partial Copy Sandbox
- Scratch Orgs (where the Developer Hub is Prod or Full Copy)

How do I use ApexGuru with Salesforce Code Analyzer?

ApexGuru's automated code optimization features are directly integrated into the Code Analyzer VS Code extension. To learn more and get started, see Scan Your Apex Code for Performance Issues Using ApexGuru.

Lightning Experience Insights

Use on-demand insights to identify slow pages and improve their performance. Starting in Spring ‘26, this feature is generally available for production and sandbox orgs.

Considerations

- Lightning Experience Insights shows you performance metrics for your Lightning Experience apps and components and excludes metrics for build-in framework pages and actions.
- As a part of Scale Center, Lightning Experience Insights isn't supported in Government Cloud Plus.

To generate Lightning Experience insights:

1. From Setup, in the Quick Find box, enter `Scale Insights`, then click **Lightning Experience Insights (Beta)**.
2. Click **Generate Report** for on-demand analysis. Lightning Experience Insights analyzes your Lightning pages and apps and summarizes performance metrics. The rest of the report recommends overall improvements and provides detailed metrics for your slowest pages and actions.

See Use Lightning Experience Insights to Improve Page Performance for additional details and guidelines for interpreting your report insights.

Use Lightning Experience Insights to Improve Page Performance
As a Salesforce administrator, you want your users to have a fast and responsive experience. Lightning Experience Insights, located within Scale Center, helps you analyze and improve the performance of your custom Lightning pages and Lightning components. It provides a high-level overview of your users’ experience, and detailed metrics to identify bottlenecks and optimize page load times. Starting in Spring ‘26, this feature is generally available for production and sandbox orgs.

Understand the High Impact, Slow Performing Pages List
Use the details provided in the High Impact, Slow Performing Pages table to diagnose and optimize performance of your most important pages.

Use Lightning Experience Insights to Improve Page Performance

As a Salesforce administrator, you want your users to have a fast and responsive experience. Lightning Experience Insights, located within Scale Center, helps you analyze and improve the performance of your custom Lightning pages and Lightning components. It provides a high-level overview of your users’ experience, and detailed metrics to identify bottlenecks and optimize page load times. Starting in Spring ‘26, this feature is generally available for production and sandbox orgs.

Key Terminology

Before diving into the report, it's helpful to understand how Salesforce measures a few key performance metrics:

- *Experienced Page Time (EPT)*: This is the primary metric Salesforce uses to measure page load performance in Lightning. EPT measures the total time it takes for a page to load to a state where a user can fully interact with it. A lower EPT means the page loads faster, providing a better user experience.

- *P75 EPT*: This value represents the 75th percentile for EPT. It means that 75% of page loads for your users were this fast or faster. We use P75 instead of the average (or 50th percentile) because it more accurately represents the experience of users who encounter slower performance. This helps you to identify and focus on issues that affect a significant portion of your user base.

Overview of the Lightning Experience Insights Report

The Lightning Experience Insights report gives you a 7-day rolling window of your org's performance, ending three days before each report was generated.

If you don't already have a report, click **Generate Report**. Once a report is generated, select the report you want to review and analyze.

Note It can take some time for a report to become available. You can click **Refresh**, or return to Lightning Experience Insights a little later to review your report.

The report is organized into several sections designed to guide you from high-level trends to specific, actionable details.

Performance Overview

Lightning Experience Insights focuses on four core metrics, which are measured and presented in different ways to provide useful insights. These four metrics are presented at the top of your report.

First Page EPT

Page load time for a complete page load, including Lightning Experience bootstrap (framework initialization), measured in milliseconds. (Divide by 1000 to get the number of seconds.) Typically, Lightning Experience bootstrap is required only for the first page a user visits in Lightning Experience.

You can't affect the time it takes for Lightning Experience bootstrap – that's our job, and we're always working on it! Since it adds significant time to the EPT, we measure and share it, but it's the next metric that represents the more typical user experience.

Subsequent Page EPT

Page load time for additional pages, after Lightning Experience bootstrap, measured in milliseconds. Typically, this is any page a user navigates to after loading Lightning Experience. Without the need to reload Lightning Experience, this metric is typically faster. While First Page EPT represents a first impression or start of session experience, Subsequent Page EPT is the better measurement of how your users experience Salesforce as they use it throughout a working session.

Note When not explicitly separated, we blend EPT across first and subsequent page views, and simply refer to it as "EPT." This definition omits Lightning Experience bootstrap, and combines non-bootstrap EPT from both first and subsequent page navigations. Since you can't directly optimize bootstrap, this represents the portion of EPT you can influence. It provides a simplified metric for further breakdowns across geographies, browsers, and slow pages.

Total Page Views

This is the total number of page views over the seven day period. You can use it as a way to track increasing usage of your Salesforce implementation.

First Page Views, % of Total

Proportion of all page views that are first page views. This ratio gives you a meaningful insight into how your users experience working with Salesforce in Lightning Experience. This number should normally be low, a minority of overall page views. If it isn't, that might indicate you need to spend some time thinking about the business processes that are performed in Salesforce, and optimize navigation or otherwise make it easier for your users to stay in Lightning Experience.

Key Insights

Key Insights provides a plain-language summary of the most important performance takeaways. Rather than just presenting raw numbers, this section interprets the data for you, highlighting critical issues and applicable best practices.

Insight	Details
Number of components	The number of custom components on a page affects page load time, especially when a component is complex or inefficient. Make sure every component on your pages has a purpose.
Secured Browser Caching	Secure data caching in the browser improves page reload performance by avoiding extra round trips to the server. This setting is enabled by default.
Lightning CDN	The Lightning content delivery network (CDN) serves static content for the Lightning component framework. This setting is enabled by default.
Dynamic Boxcar'ing	Dynamic boxcar optimization improves Aura application performance by grouping Aura actions into boxcars.
Apex caching	Percentage of Apex actions that are served from client cache, instead of requiring a round-trip to the server. Make sure your Apex classes are taking full advantage of the @AuraEnabled annotation to allow caching of server-side action results.

Insight	Details
Slowest Apex actions	These Apex methods are the slowest actions used by your custom components. Check ApexGuru Insights for opportunities to optimize these actions.

Key Insights serves as your starting point, immediately drawing your attention to the most significant performance problems and opportunities for your org.

Performance Trends

Below Key Insights, you'll find charts that visualize your org's P75 EPT over the selected week, as well as breaking EPT down by region and browser. This graph helps you identify trends and understand performance patterns. You can see when EPT is spiking, whether performance is degrading over time, or if recent changes have improved page load times. The Performance Trends chart shows trends for both the initial page load (1st Page EPT) and subsequent page navigations within the application.

High Impact, Slow Performing Pages

To help you pinpoint the exact sources of performance degradation, the report includes a detailed breakdown of the slowest pages in your org. This section lists the ten pages with the highest combination of the number of page views and the highest P75 EPT, allowing you to focus your optimization efforts where they will have the most impact.

Lightning Experience Insights goes deeper by identifying the specific pages, components, and actions that might make Salesforce feel slow for your users. For a given page, you can expand the row to see the individual components and their actions–such as complex Apex calls or inefficient component logic–that are the most resource-intensive. This allows you and your development team to move directly from analysis to action, addressing the root cause of the performance bottleneck.

Note Only custom components and actions are displayed, to focus your attention on impactful changes you can make. Components that can't be determined from metrics are aggregated together, and are displayed as "UNKNOWN". You can still expand the "component" row, and review the individual actions for optimization opportunities.

See Understand the High Impact, Slow Performing Pages List for details of the various aspects of page, component, and action request processing that are measured in this list.

Understand the High Impact, Slow Performing Pages List

Use the details provided in the High Impact, Slow Performing Pages table to diagnose and optimize performance of your most important pages.

When you view a Lightning Experience Insights report, the High Impact, Slow Performing Pages displays a brief list of ten pages that have the most impact on your users. We select these ten pages based on the combination of the number of page views and the highest P75 EPT. In other words, these are the ten slow pages that are used the most.

To make effective use of this list, dive into the details for each page. Click the disclosure chevron left of the page URL to expand that entry, and display details about the components used on that page. Expand the chevron for each component to see details of the actions used by the component. Now you can review the page's performance characteristics at a very fine grained level, and focus on specific opportunities for optimization.

Untimed Action Metrics

The following non-time-based metrics are provided. These metrics offer insight into which component actions are executed the most frequently, and whether and how efficiently they're reused.

Action Count
: The number of times this action was executed during the selected time period.

Boxcar Count
: The number of actions that were batched together in a single network request. Lightning Experience can group multiple actions into one request for efficiency. Higher counts mean more efficient batching, but potentially longer individual action times if the batch becomes too large.

  See Batching of Server-side Actions in the *Lightning Aura Components Developer Guide* for a detailed description of the boxcar'ing process.

Cache Eligible
: Indicates if an action can be served from a cache. If it can, it loads faster because it doesn't require new processing from the server. Check the cache hit ratio to see how often this actually happens.

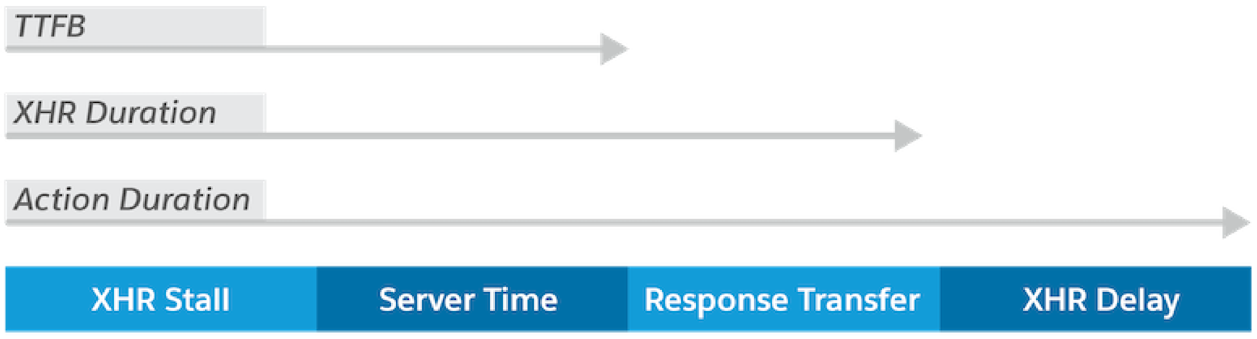
  See Client-Side Caching of Apex Method Results in the *Lightning Web Components Developer Guide* and Storable Actions in the *Lightning Aura Components Developer Guide* for additional details.

Cache Hit Ratio
: The percentage of actions that returned results from cache rather than requiring server processing. Higher ratios indicate better caching efficiency, and generally result in faster response times for repeated actions.

Timed Action Metrics

Interpreting the details provided in the High Impact, Slow Performing Pages list is easier when you understand how actions are processed, from the time a component fires the action, to the different phases of the round-trip to Salesforce, until the action completes and any updates are applied to the page.

The following diagram illustrates the various phases of an action request.



The following time-based metrics are provided.

XHR Request Stall Time

Measure of the initial network connection delay before a request is sent to the server. This includes the time needed to look up the server's address (DNS lookup) and establish the network connection (TCP setup).

Server Time

The time spent executing the action on the server, including database operations and business logic. This excludes network transfer time and client-side processing, and can help identify when delays are due to server processing.

TTFB

Time to First Byte (TTFB) is the time from when a request is sent to the server until the first byte of the response is received. This indicates how quickly the server responds to the action, and can help identify when delays are due to server processing or network issues.

XHR Duration

The time between when a network request is sent to the server and when the full response is received by the client. This measures pure network communication time, including the server execution time, but not client-side processing.

XHR Response Stall Time

Delay in processing the response, between when a network response is received from the server and when the JavaScript callback can execute due to the main thread being busy. This metric helps identify when the client-side application is the bottleneck, rather than network or server performance issues.

Client-side processing issues that prevent immediate response handling are often caused by heavy JavaScript execution or UI thread blocking. High values can indicate that the action or component should be reviewed, and possibly redesigned to handle responses more independently or asynchronously.

Duration

The total time for this action to complete, from when the action is first requested until the client-side callback completes. This includes all waiting time, network transfer, server processing, and client-side execution. It's the complete end-to-end time for an action to finish.

Database Insights

Fix expensive SOQL queries and reduce database time.

Considerations

- Database Insights shows you SOQL insights and your top 15 expensive SOQL queries.
- Use the Takeaways section to see the impact on your org and follow the recommendations.

To generate database insights:

1. From Setup, in the Quick Find box, enter `Scale Insights`, then click **Database Insights**.
2. Click **Generate Insight Report** for on-demand analysis. Database Insights analyzes your queries and provides recommendations to improve database time.

Database Insight Categories

Insight	Definition
Top 15 Expensive SOQL Queries	SOQL queries running during peak business hours consume high CPU cycles. Optimize slow queries to reduce CPU consumption and improve transaction speeds during peak hours.
Takeaways & Impact	Use the recommendations to optimize suboptimal queries, reducing database time and improving scale.
Query Plan Analyzer Integration	Use Query Plan Analyzer to get recommendations for speeding up SOQL queries against large volumes and optimizing for scale.

Report Insights

Use on-demand insights to identify slow reports and improve their performance.

Considerations

- Report Insights shows you public access reports and excludes private reports.
- As a part of Scale Center, Report Insights is supported in Government Cloud Plus.

To generate report insights:

1. From Setup, in the Quick Find box, enter `Scale Insights`, then click **Report Insights**.
2. Click **Generate Insight Report** for on-demand analysis. Report Insights analyzes your reports and recommends improvements for the slowest ones.

Report Insight Categories

Insight	**Definition**
Slow Reports by Average Run Time	Ranked by average run time.
Recommendations	Ways to optimize your slow reports and avoid performance issues.

Search Insights

Use on-demand insights to get recommendations and improve your search performance.

Considerations

- Search Insights provides insights, recommendations, and best practices about your custom entities with search enabled.
- As a part of Scale Center, Search Insights is supported in Government Cloud Plus.

To generate search insights:

1. From Setup, in the Quick Find box, enter `Scale Insights`, then click **Search Insights**.
2. Click **Generate Insight Report** for on-demand analysis. Search Insights analyzes your entities and provides recommendations to improve your search performance.

Search Insight Categories

Insight	**Definition**
Entities Never Queried	Custom objects aren't queried even though Salesforce maintains search indexes. Operations related to these objects are slower.Use previous reports to identify the most unused entities. Go to these custom objects in Object Manager and disable search.
Entities Queried and Never Clicked	Custom objects are searched for but never actually clicked once search results are returned.Ask your business team if these objects need to be searchable and consider disabling search in Object Manager.