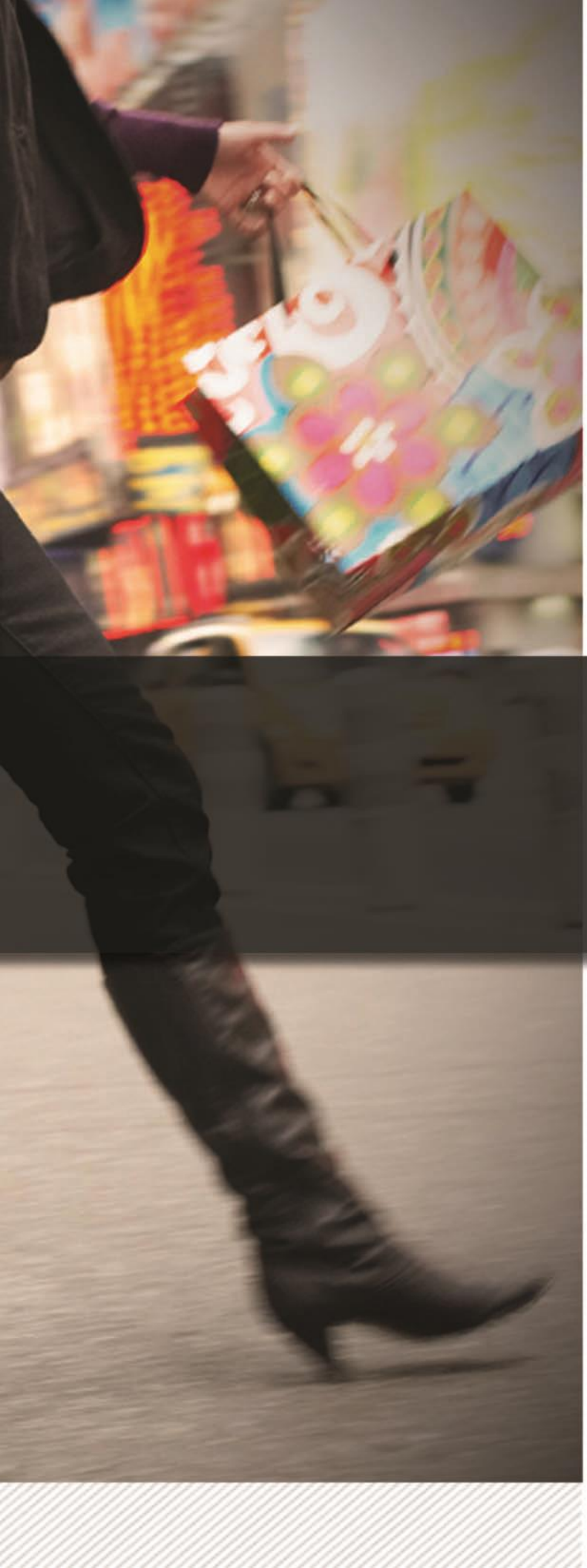


demandware
move faster, grow faster

DEVELOPING FOR PERFORMANCE AND SCALABILITY

Student Guide

Table of Contents

demandware move faster, grow faster.

Introduction

About the Course

Audience	Developers
Duration	4 hours
Prerequisites	To successfully meet the objectives of this course, we strongly suggest that participants complete the *Developing in Demandware* course. The associated certification is a requirement for this course. You'll also need a BitBucket account with access to the Demandware repository. Create your BitBucket account, then email developer.community@demandware.com with your BitBucket user ID to get access.
System Requirements	Laptop or desktop with UX Studio installed and access to a sandbox.
Course Materials	*Developing for Performance and Scalability* Student Guide The scripts and templates you need for the exercises are included in the `Developing for Performance and Scalability Exercises.zip` file provided by your instructor. **Note**: If you are taking the eLearning version of the course, the zip file is on the **Course Materials** tab.

Course Objectives

Welcome to *Developing for Performance and Scalability*. After completing this course, you'll be able to:

- Discuss factors that impact performance on the platform and develop strategies to address them.
- Use caching techniques and tools to improve performance, efficiency, and stability on the platform.
- Optimize client-side and server-side processing using coding best practices.
- Optimize transactional jobs and integrations.
- Describe how Demandware uses quotas to enforce appropriate data usage policies.
- Review Quota Status reports.
- Analyze and troubleshoot performance issues using Business Manager Analytics, Pipeline Profiler, and Control Center.

Module Objectives

The following table describes the objectives for each module:

Module	Objectives
It's All about Scalability	▪ Discuss how performance, efficiency, and stability affect a site as it scales. ▪ Describe the impact of designing transactions to be processed on the web, app, and database tiers.
Implementing Effective Caching Strategies	▪ Describe the caching techniques that are available on the Demandware platform. ▪ Set up page and partial page caching. ▪ Determine appropriate cache settings for site pages. ▪ Review cache settings using the Business Manager Toolkit.
Optimizing Performance During Development	▪ Optimize client-side and server-side processing using coding best practices. ▪ Manage data efficiently and monitor data storage usage.
Analyzing Performance Issues	▪ Analyze performance using Pipeline Profiler. ▪ Analyze performance using Business Manager Analytics.
Optimizing Transactional Jobs and Integrations	▪ Create and schedule jobs that limit resource consumption on the platform. ▪ Diagnose issues with job performance. ▪ Use web services efficiently.
Adhering to Quotas	▪ Describe platform quotas and the benefits of using them. ▪ Describe the Demandware data usage policy. ▪ Identify quota violations using the Business Manager Quota Status and quota log files. ▪ Create email notifications for quota violations. ▪ Describe the types and levels of quotas and explain the purpose of each. ▪ Describe the process of overriding platform quotas.
Summary of Diagnostic Tools	▪ Summarize the diagnostics tools and identify under which circumstances they should be used. ▪ Identify external tools for analyzing performance.

About the Course Exercises

This course includes scripting examples that you'll run on your storefront site. You can find these files in the `Developing for Performance and Scalability Exercises.zip` file provided by your instructor. You'll run the scripts and use the Pipeline Profiler to analyze the performance of the scripts.

Note: If you are taking the eLearning version of the course, the zip file is on the **Course Materials** tab.

demandware move faster, grow faster

Module 1: It's All about Scalability

Learning Objectives

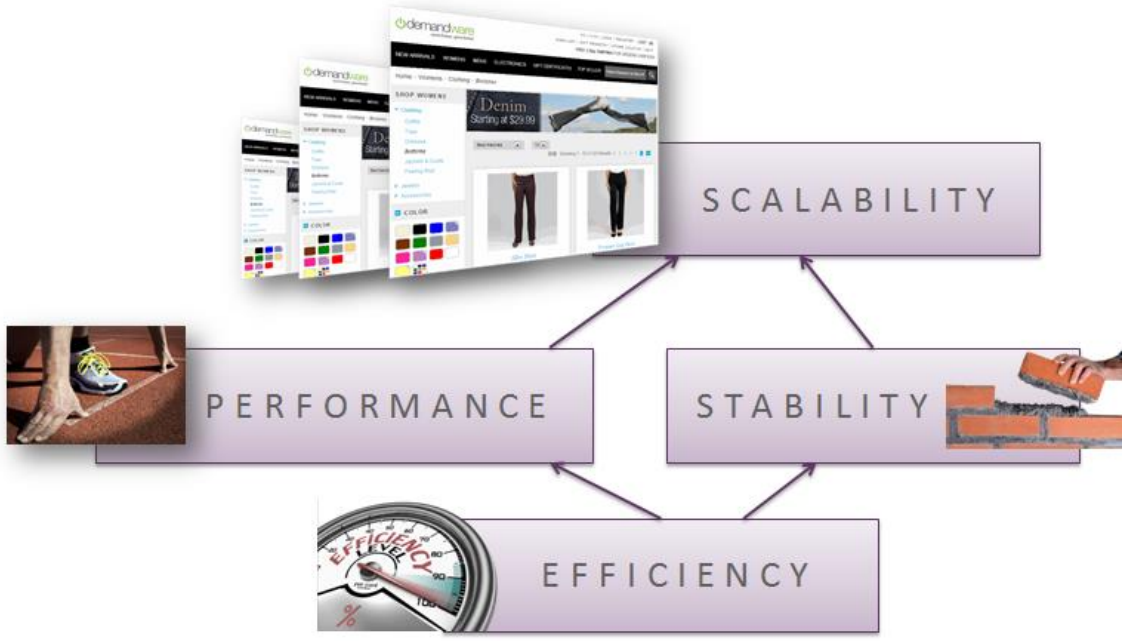
After completing this module, you will be able to:

- Discuss how performance, efficiency, and stability affect a site as it scales.
- Describe the impact of designing transactions to be processed on the web, app, and database tiers.

Introduction

Site performance is critical to sales on your site. If customers experience slowdowns when they search for an item or attempt to check-out on your site, there's a good chance they'll leave without completing a purchase. Also, when customers do complete purchases on slow sites, their basket totals tend to be lower. The speed of your site affects customer brand perception, as well. Customers are accustomed to immediate web responses; your site must meet or exceed their expectations for site performance.

Developing eCommerce sites in a Software-as-a-Service (SaaS)-based environment requires thoughtful design that takes resource usage into consideration as sites scale. The Demandware platform's extensive customization capabilities make it possible to accidentally exceed the limits of Demandware controls. For example, misuse of an API or a business object can lead to overages that reduce performance and stability, and increase cost.



To ensure that your site scales effectively on the Demandware platform, your team must:

Improve efficiency by optimizing:

- Data transfer—Determine optimal approaches.
- Storage—Avoid churn.
- Processing operations—Use appropriate APIs.

By improving efficiency, you:

- **Optimize performance**, measured in terms of response time and data transmission speed.
- **Ensure platform stability**, critical in SaaS-based architectures to ensure that sites run smoothly, measured in site availability.

If you meet these goals of optimized performance and platform stability, your site will scale easily as you add sites and locales.

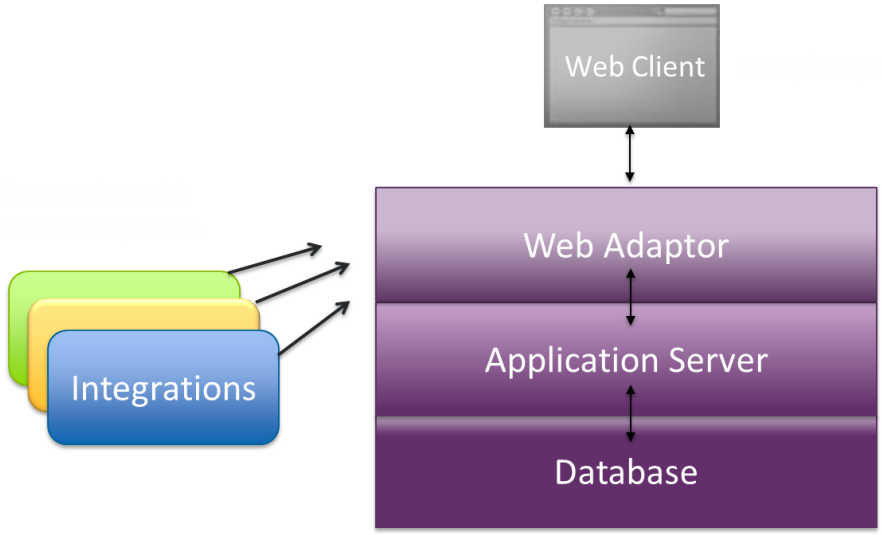
To help ensure that your site meets performance goals, Demandware provides a quota framework that indicates when your site resources exceed recommended limits. The quota framework helps detect inefficiencies in:

- Data processing
- Data storage
- Data transfer

Staying within the quota limits ensures that your site will scale as your business scales. In this course, you'll learn about the different types of quotas, how to monitor them, and how to diagnose performance issues that result in quota overages. The goal is to prevent quota overages by designing and developing your site with efficient best practices in mind. In this course, you'll learn how to design your site to take advantage of the Demandware platform while optimizing for scalability.

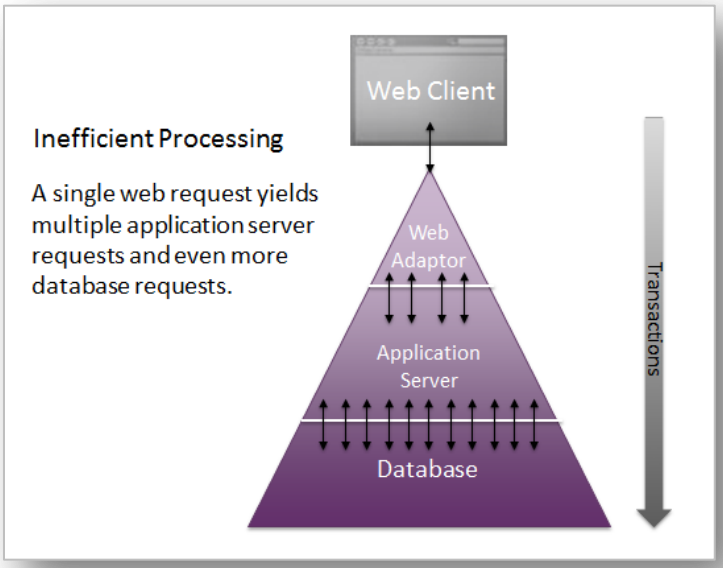
Lesson 1.1: Optimizing Web, App, and Database Tier Interactions

Web site efficiency is influenced by many factors, including integration with external systems, the processing time spent on the web adaptor, application server, and database tiers, and the complexity of the HTML pages the browser renders to create the customer experience.



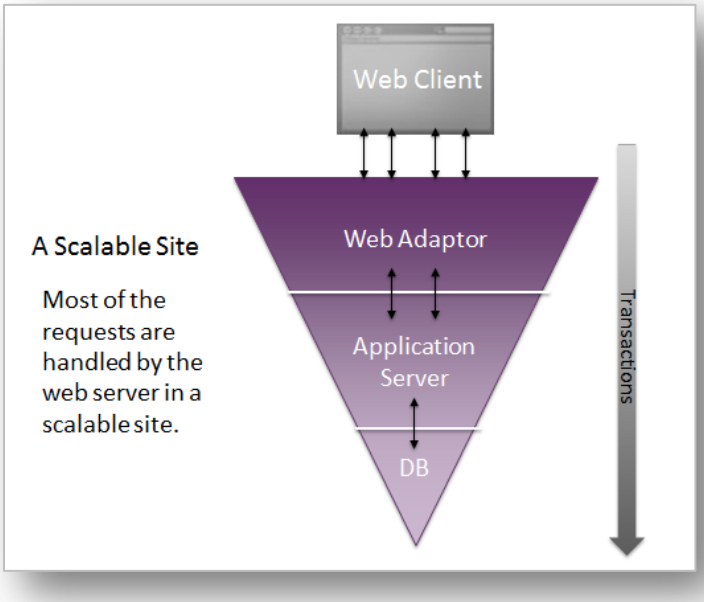
The platform scales to meet traffic and order peaks most effectively when the web tier handles the majority of transaction requests. As you design your site, it's important to minimize the number of transactions that pass through the web tier to the application tier and then from the application tier to the database. To do so you'll use efficient programming and caching techniques.

In the following example, individual requests coming from the web create multiple requests to the application layer; each of those requests creates multiple additional requests to the database layer:



Too many requests on the database layer causes latency, leading to weak performance and the inability of your site to scale effectively.

In a scalable site, most of the requests are handled by the web server:



A recommended ratio for transactions is 100:10:1 where there are 100 transactions on the web tier for every 10 transactions on the app tier for every single transaction on the database. While 100:10:1 is an

ideal ratio for web-to-app-to-database transactions, even a ratio of 100:90:90 is a large improvement when scaled exponentially.

In this course, you'll learn how to improve the ratio of transactions from web tier to app tier to the database tier by using effective strategies to:

- Cache page and partial page requests.
- Cache static content.
- Limit database transactions through the effective use of the Demandware APIs.
- Select the most efficient storage techniques for each type of data.
- Minimize the effects of transactional jobs on your production, staging, and development environments.

Knowledge Check

Which of these statements about site performance and scalability are true? *Select all that apply. The Knowledge Check answer key is on the following page*.

a. It's important to minimize the transactions that pass from the database to the data grid. F

T

b. Web site efficiency is influenced by integrations with external systems. T

T

c. Scalability is a result of platform stability and optimal performance. True

T

d. The platform scales best when the web tier handles most requests. T

e. An optimal ratio of database-to-app-server transactions and from app-server-to-web-server transactions is 100:10:1. F

Knowledge Check Answers

Which of these statements about site performance and scalability are true? *Select all that apply. The Knowledge Check answer key is on the following page*.

a. It's important to minimize the transactions that pass from the database to the data grid.

b. Web site efficiency is influenced by integrations with external systems.

c. Scalability is a result of platform stability and optimal performance.

d. The platform scales best when the web tier handles most requests.

e. An optimal ratio of database-to-app-server transactions and from app-server-to-web-server transactions is 100:10:1.

Answers: b, c, and d.

Module 2: Implementing Effective Caching Strategies

Learning Objectives

After completing this module, you will be able to:

- Describe the caching techniques that are available on the Demandware platform.
- Set up page caching and partial page caching.
- Determine appropriate cache settings for site pages.
- Review cache settings using the Business Manager Toolkit.

Introduction

The Demandware platform supports various caching mechanisms to improve site performance and minimize customer wait times. The platform manages some aspects of caching, but as a developer, you can fine-tune caching mechanisms through cache settings and coding best practices. For dynamic data, you'll optimize page caching. For static content, you'll optimize use of the Content Delivery Network (CDN).

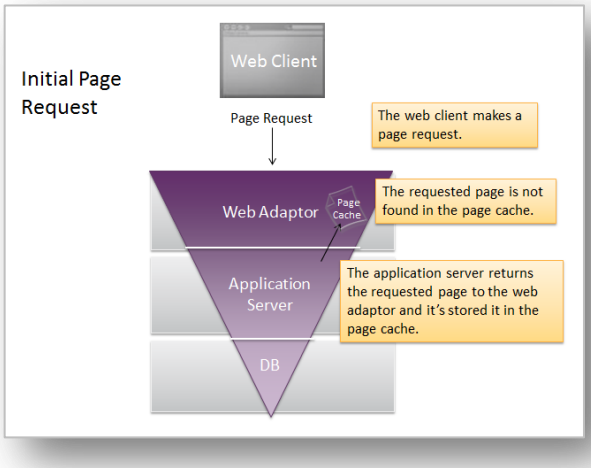
Lesson 2.1: Page Caching

How Page Caching Works

The Demandware platform manages the page cache as shown in the following diagram.

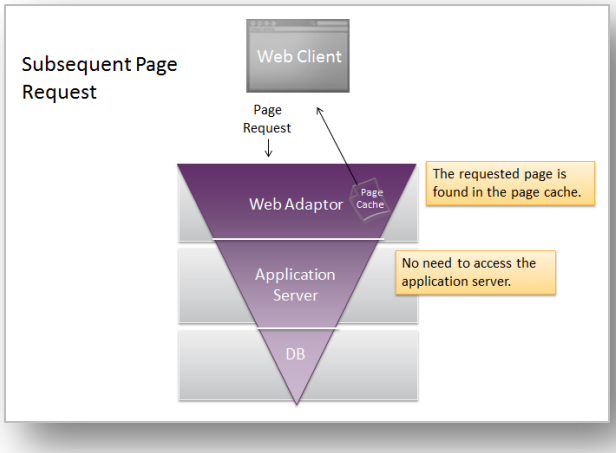
For an initial page request:

1. The web client makes a page (or partial page) request.
2. The web adaptor checks the page cache to see whether a cached response exists for the requested page. Because the content is not in the page cache initially, the web adaptor routes the request to the application server.
3. The application server handles the request, sending a request to the database if necessary.
4. The application server returns the requested page to the web adaptor.
5. If the page is flagged as cacheable, the web adaptor stores a copy of the content in the page cache.



For a subsequent page request:

1. The web client makes a subsequent page request for the same content.
2. The web adaptor finds the requested content in the page cache and returns it immediately to the web client.
3. The web adaptor handles the transaction without accessing the application server or database tier.



Page Caching on the Demandware Platform

Your site should leverage the page cache to ensure efficient handling of dynamic data. Page download time is a critical factor in keeping visitors on your storefront site. The longer it takes to download a page, the higher your risk of losing a sale. Pages that process many business objects and perform complex calculations are costly. Since this information generally does not change from one user to another, caching the results speeds up the site for all users—and not just for the requested pages. Job processing will be more efficient, too.

To optimize your use of the page cache and minimize page download times, ensure that the most frequently visited pages are cacheable. Typically, the most heavily used pages are:

- Category and search result pages (`Search-Show` pipeline)
- Product detail pages (`Product-Show` pipeline)
- Home pages (`Default-Start` and `Home-Show` pipelines).

In addition to caching pages, you can cache partial pages. You can develop individual components of a page, cache them separately, and assemble them as needed. You can display components of a page dynamically as needed.

Use caching:

- For the pages that most customers will access
- For the most heavily visited pages
- On production instances to ensure optimal performance.
- On development instances to help Quality engineers validate caching behavior.
- For "personalized" data—data customized for a particular user group.

Don't use caching:

- For information specific to a particular customer. For example, don't use caching for pages that show a particular customer's purchasing details or session information.
- For pages or portions of pages with data that updates frequently.
- On sandbox and staging instances so that you can see the result of your updates immediately.
- For portions of a page that customers access infrequently.
- For portions of a page that have very low cache-hit ratios due to customer group personalization.

Knowledge Check

When should you use page caching? *Select all that apply. The Knowledge Check answer key is on the following page*.

10 Points

a. On sandbox, development, and staging instances

b. For customer-specific information

T c. For static content T

d. For pages that are updated frequently

T e. For the most heavily visited pages T

Knowledge Check Answers

When should you use page caching? Select all that apply.

a. On sandbox, development, and staging instances
b. For customer-specific information
c. For static content
d. For pages that are updated frequently
e. For the most heavily visited pages

Answer: c, e.

Lesson 2.2: Enabling Page Caching

You enable page caching at two levels:

- In the ISML Templates
- In the Business Manager

You must enable the cache setting at both levels in order for page caching to occur.

Enabling Page Cache in ISML Templates

Demandware controls caching is controlled on a per-page basis within the ISML template for the page using the `<iscache>` tag:

```
<iscache type="relative" hour="24"/>
```

See the documentation (https://info.demandware.com) to review the parameters of the `<iscache>` tag.

When using the `<iscache>` tag,

- If there is no `<iscache>` tag defined, Demandware doesn't cache the template.
- If the `<iscache>` tag occurs multiple times in a template or in its locally included templates, Demandware uses the shortest duration. It's best to cache tags in the outermost template and not overwrite the cache setting within the local includes.
- If you use absolute caching times (for example, `<iscache type="daily" hour="23"/>` where the time is in UTC), keep in mind that the application server and web server might have slight time differences.

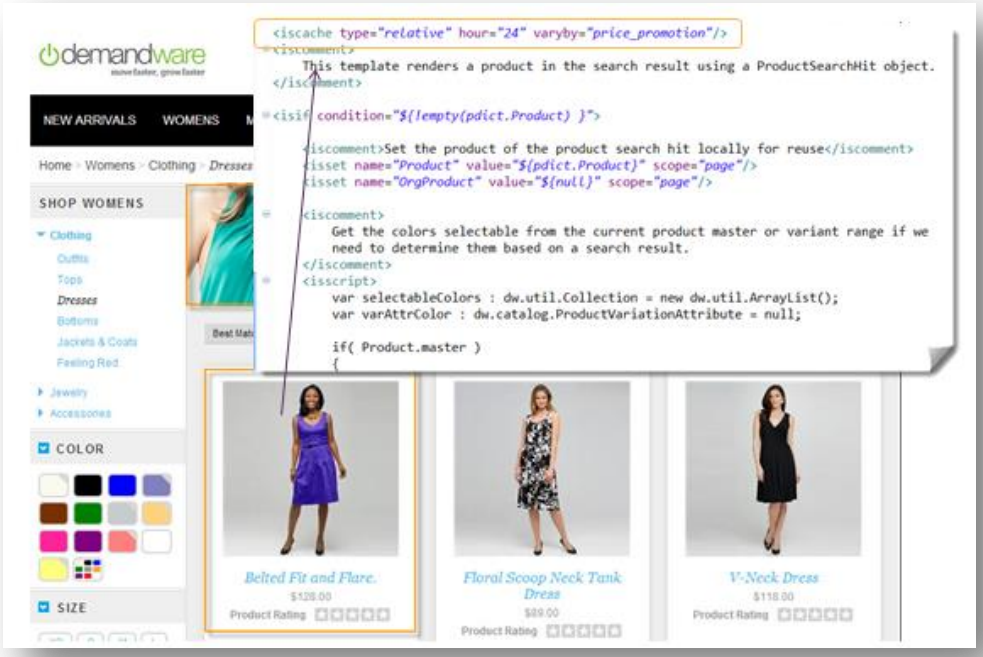
Caching for Personalized Pages

Demandware sites can contain pages that display "personalized data." Personalized data is not customer-specific, but is instead specific to particular groups of customers. Merchants can set up personalized pages to implement A/B testing, as well as unique promotions for specific customer groups.

Implement personalized page caching using the `varyby="price_promotion"` parameter to the `<iscache>` tag.

- Each personalized version of the page is cached so performance is optimized.
- Different customer groups see different versions of the same page. For example, this `varyby` parameter is necessary for product pages since a customer belonging to a customer group might get special promotions that other customer groups don't get.
- These different versions of the page might display different prices, promotions, sorting rules, or A/B test segments. The system recalculates the customer groups each time it updates the session attributes.

The following example shows the `productsearchhittile.isml` template from SiteGenesis which renders a product on a search results page. The template uses the `varyby` parameter to implement a personalized page.



Important: Set the `varyby="price_promotion"` parameter for a page only if the page is personalized; otherwise, the performance will degrade unnecessarily.

Partial Page Caching

Usually it doesn't make sense to cache a whole page. Instead, you'll cache portions of pages separately. For each portion of the page that needs unique caching characteristics, implement a remote include, for example:

```
<isinclude url="${URLUtils.url('Page-Include', 'cid', 'COOKIE_TEST')}">
```

1. Each remote include makes a pipeline request.
2. The web adaptor handles the requests and assembles all page portions.
3. The web adaptor returns the assembled page to the client.

In addition to remote includes, you can also use the `<iscomponent>` tag to specify unique cache settings for a portion of a page, for example:

```
<iscomponent pipeline="Product-Show" productid="1234" name="Wide-screen
television">
```

Enabling Page Cache in Business Manager

In addition to including the `<iscache>` tag in your ISML templates, you need to ensure that page caching is enabled in Business Manager. For production instances, Demandware enables caching by default.

To enable page caching, in Business Manager, select **Administration > Sites > Manage Sites**. Select the **Cache** tab and select the **Enable page caching** option.

Note: If you disable page caching in Business Manager, the Demandware platform ignores the `<iscache>` tags in the ISML templates.

Lesson 2.3: Invalidating Page Cache

Invalidating the page cache clears the cache and resets the caching "clock" for relative caching. The page cache is invalidated at the following times:

- When the defined caching time is exceeded
- When merchants invalidate the page cache explicitly using Business Manager—the entire page cache or just the cache corresponding to specific pipelines
- When there is code replication

After the cache is invalidated, subsequent requests are slow until the cache is filled again. It's best to invalidate the cache only when necessary and during periods of low traffic.

Invalidating Page Cache Using Partitions

Occasionally, merchants need to make unplanned edits to content on the production system. For example, they might need to adjust an incorrect price for an item. If the page displaying the content is cached, they'll need to invalidate the page cache to display the new content. Invalidating the entire page cache has a significant impact on performance because the whole page cache needs to be regenerated. Instead of invalidating the entire page cache, merchants can use page cache partitions to invalidate selected pipelines. Demandware provides a number of partitions containing common pipelines.

To create page cache partitions:

- Specify the start node of each pipeline to be included in the partition—whether they're local pipelines or remote includes. Each pipeline node can be in one page cache partition only.
- Once the partitions are set up, merchants can invalidate them by name in the Business Manager.

Exercise: Enable Page Caching

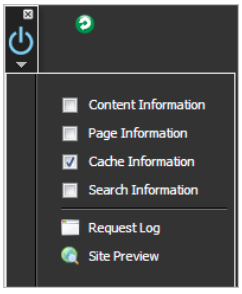
In this exercise, you'll enable page caching in your sandbox. Normally, you'd disable caching in your sandbox so that you can see the results of your updates immediately. However, for the exercises in this course, you'll enable page caching.

1. Select **Administration > Sites > Manage Sites**.
2. For **Instance Type**, select **Sandbox / Development**.
3. Select the **Cache** tab and select the **Enable page caching** option.

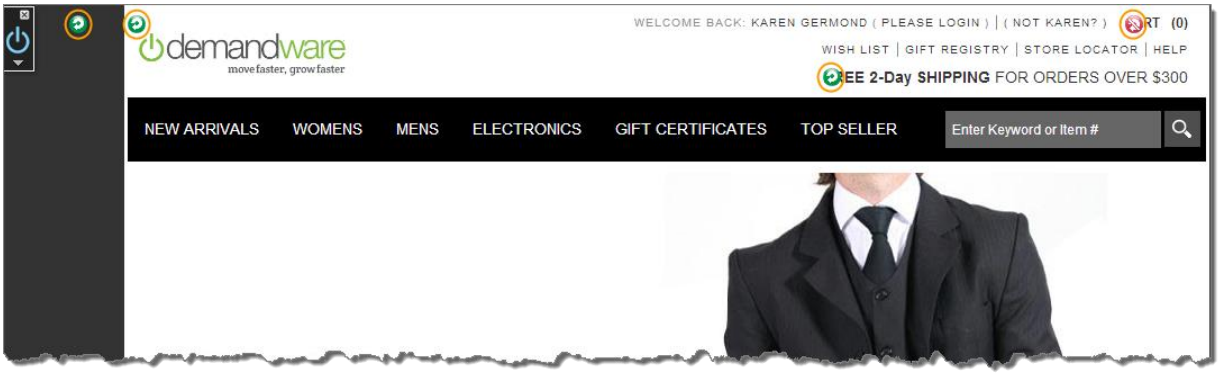
Exercise: Explore the Cache Settings

In this exercise, you'll use the Storefront Toolkit to verify cache settings.

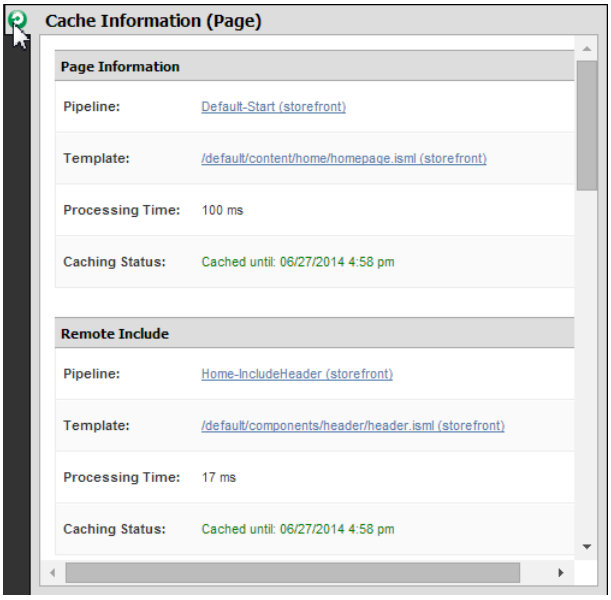
1. In your Storefront Toolkit, enable **Cache Information**.



The storefront displays green cache icons for cached portions of the page and red cache icons for portions of the page that are not cached.

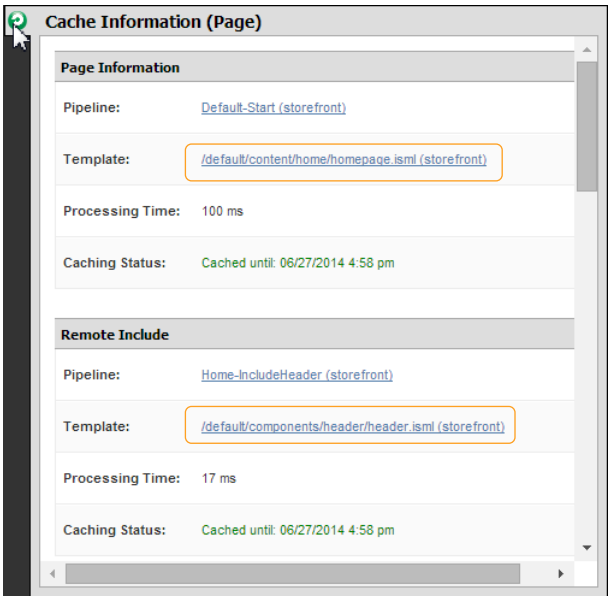


2. To review the cache settings for the whole page, move your mouse over the green cache icon in the upper left of the window.

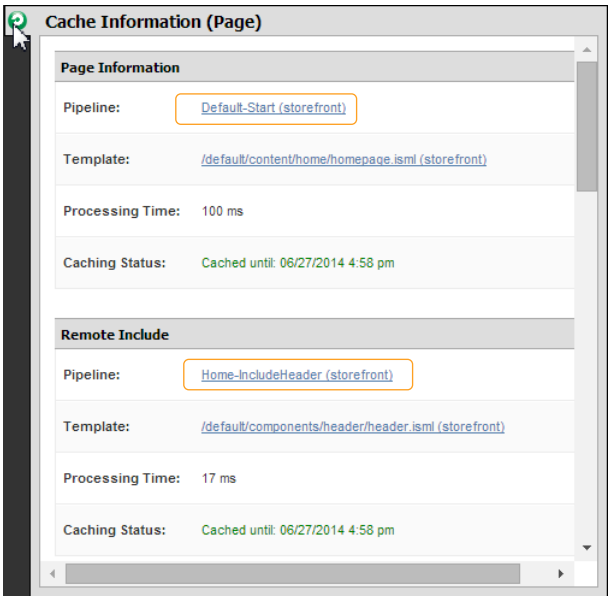


You can use the scrollbar on the Cache Information (Page) window to view the cache settings of each remote include on the page. Move your mouse over any of the green cache icons to review the cache settings for a particular page portion.

3. With UX Studio running, click a Template link to view the associated ISML template code.



4. Click a Pipeline link to view the associated pipeline code:



Exercise: Set Up a Page Cache Partition

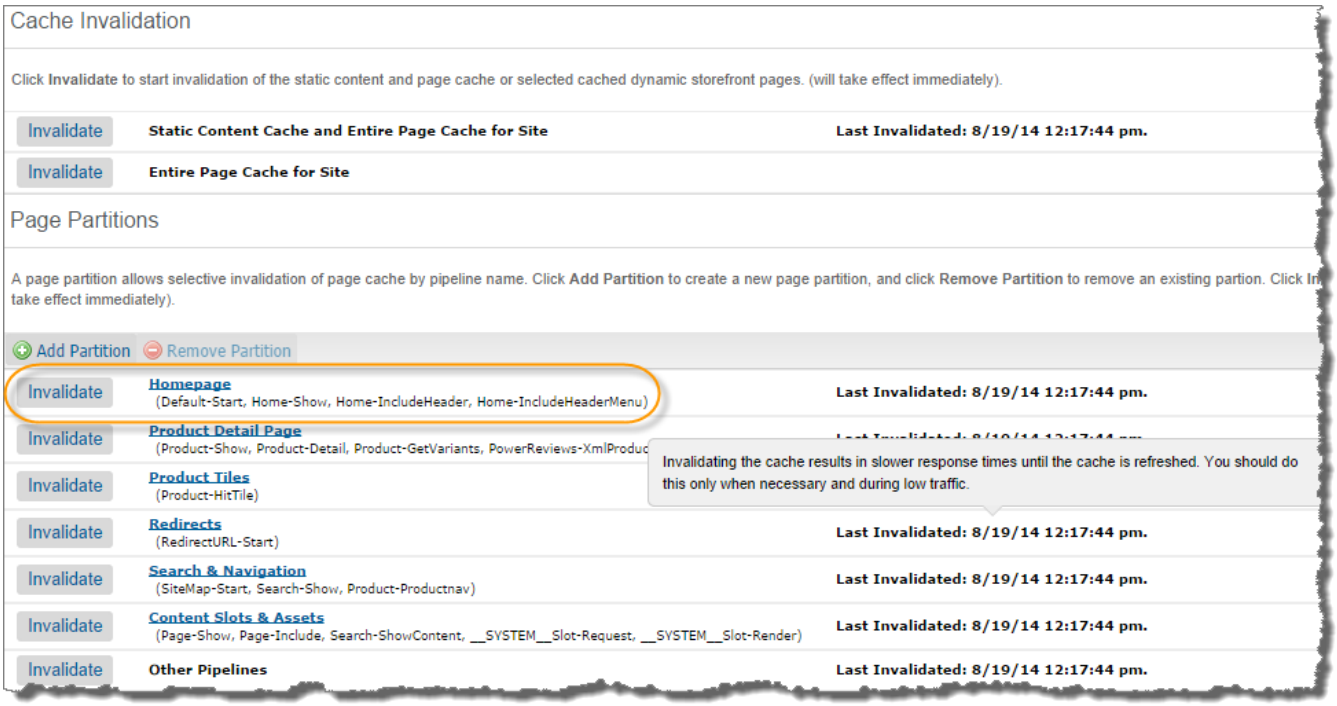
In this exercise, you'll remove the Homepage page cache partition that Demandware provides. You'll create the Homepage partition and invalidate it.

1. In the **Cache** tab, under Page Partitions, select the Homepage row and click **Remove Partition**.
2. Click **Yes** to confirm the delete.
3. Now you'll create the Homepage page cache partition by clicking **Add Partition** in the **Cache** tab:
   a. In the **ID** field, enter a unique identifier for the partition, in this case, **Homepage**.
   b. In the **Name** field, enter **Homepage**.
4. Click **Add Pipeline**.
5. In the field that displays, enter the name of a pipeline node to be included in the page cache partition, in this case, `Default-Store`.
6. Click **Add Pipeline** for each of the other pipeline nodes that belong in the Homepage partition, enter the following pipelines, and click **Save**.

   ```
   Home-Show
   Home-IncludeHeader
   Home-IncludeHeaderMenu
   ```

7. On the storefront site, reload the page and move your mouse over the green cache icon in the header. Note the Caching Status time.

8. In the Business Manager **Cache** tab, click **Invalidate** next to the Homepage partition you created.



9. On the storefront site, reload the page and move your mouse over the green cache icon in the header again.

   Note that the Caching Status time has been reset.

10. Move your mouse over other green cache icons to see the Caching Status for other portions of the page.

    Note that the Caching Status has only changed for the partition you invalidated and not for the other portions of the page.

Exercise: Invalidate the Page Cache for the Site

In this exercise, you'll invalidate the cache and view the updated Caching Status using the Storefront Toolkit.

1. With **Cache Information** enabled in the Storefront Toolkit, move your mouse over the green cache icons to note the Caching Status time for each portion of the page.
2. In Business Manager, invalidate the cache for the entire page:
   a. Select **Administration > Sites > Manage Sites** and select **SiteGenesis**.
   b. Select the **Cache** tab.

c. For **Instance Type**, select **Sandbox/Development**.

d. Click **Invalidate** next to **Entire Page Cache for Site**.

3. On your storefront site, reload the page and move your mouse over any green cache icon.

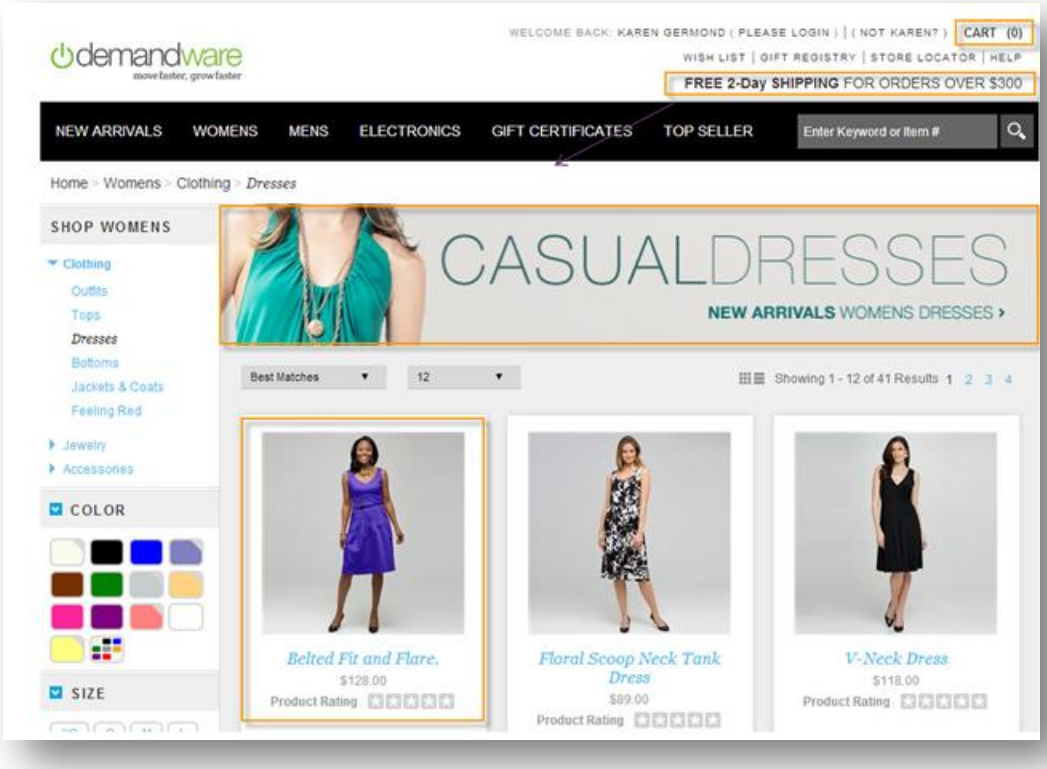
   Notice that the Caching Status time has been updated to 24 hours from now.

   Note: The time is shown in GMT, so the cache status time will vary depending on your time zone.

You've finished the exercises in this section.

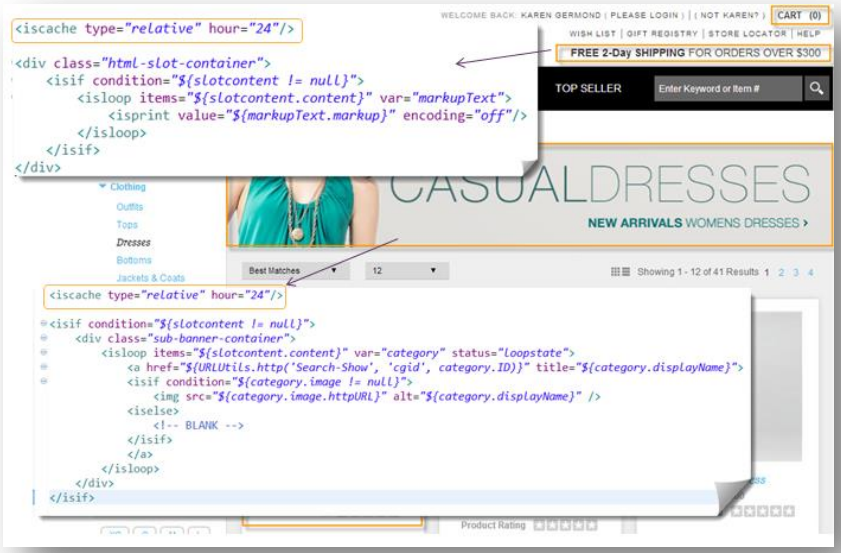
Just Thinking

Which cache settings would you use for the selected portions of the following storefront page? For example, what cache time would you use for each? Does the page portion include personalized content?

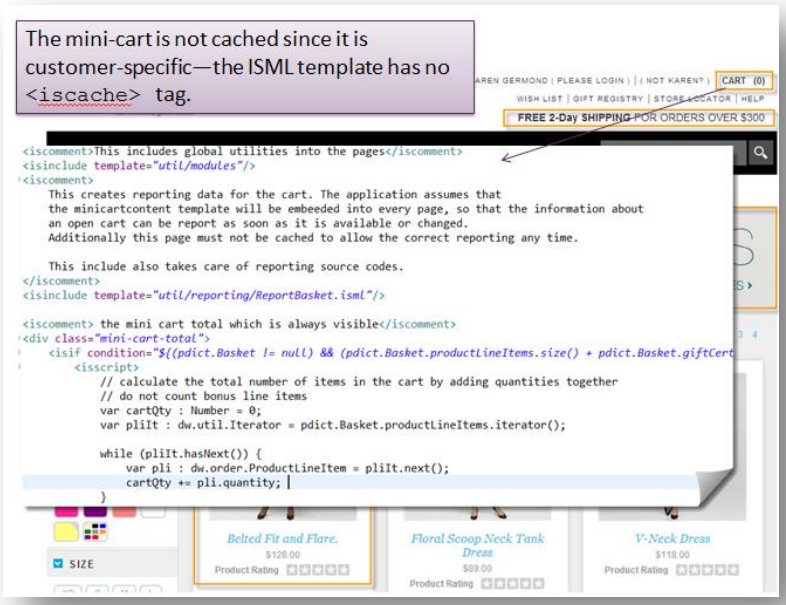


The following are the cache settings you'll find in your SiteGenesis sandbox for this page.

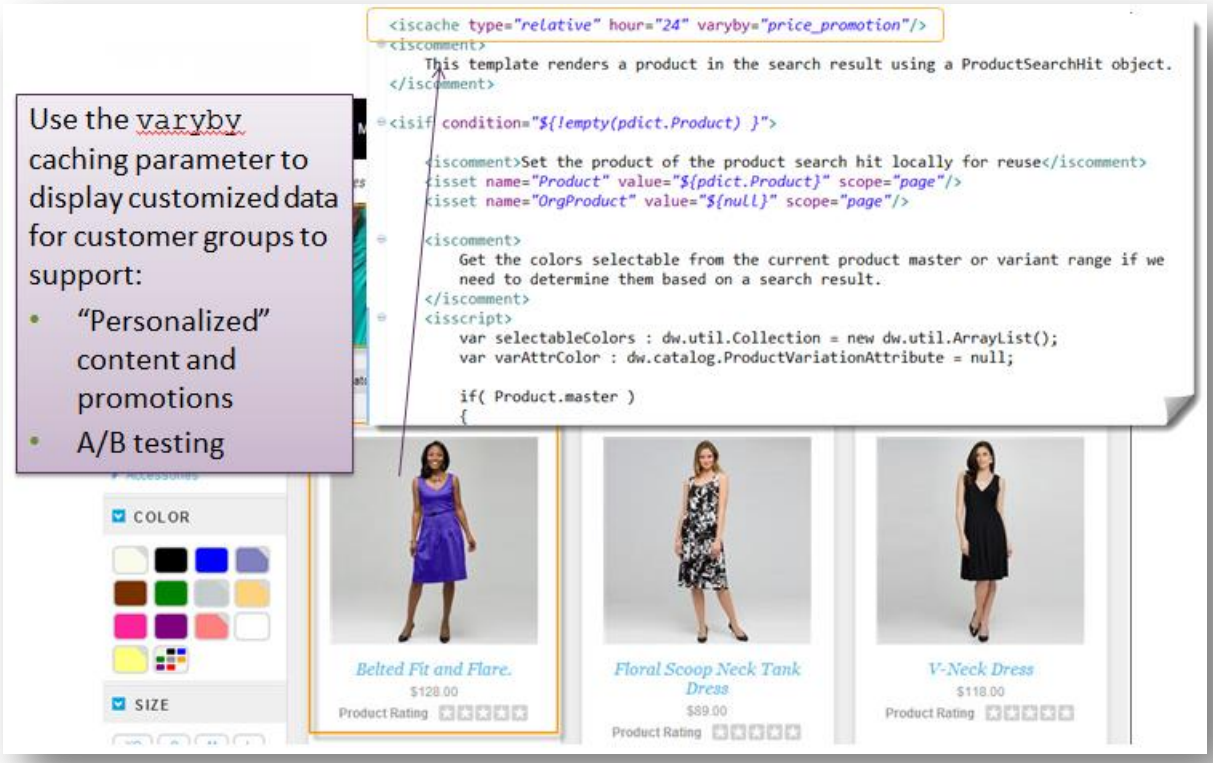
For portions of the page that most customers visit, you can use a relative or daily caching strategy. This example uses the `relative` parameter:



Don't cache the portions of a page that are specific to an individual customer, like the mini-cart, cart, and profile pages:



Use the `varyby="price_promotion"` for personalized content—content that is specific to a group of customers, for example, for promotions or A/B testing.



Best Practices

Caching Tips

- Be sure to cache the result pages of redirect templates, especially heavily loaded pages like those created with the `RedirectURL-Start` pipeline.
- If possible, use `HTTP GET` rather than `POST` for form submissions with cached results. If you use the `POST` form method, you bypass the page cache and make a server-side call.

Caching for Dynamic Pages

Give special thought to highly dynamic pages, for example, search pages:

- Build pages with static and dynamic areas using remote includes.
- Cache each product tile on its own because this data isn't likely to change. Caching product tiles for 24 hours makes sense.

- If your inventory or price information changes regularly, cache for 30 minutes or even as few as five minutes. You might even avoid caching if the information changes very frequently.
- Attributes on product detail pages change during catalog replication only. Because organizations replicate the catalog once every 24 hours typically, set product detail page cache to 24 hours.

Caching for Remote Includes

Remote includes let you cache requests individually for sections of a page. This strategy becomes inefficient, however, if you have too many remote includes:

- Do not use more than 20 remote include sections on a page.
- You can use a depth of up to seven remote includes, but limit the depth to three to ensure good performance. For each level, determine the number of remote includes needed and enforce this limit.
- Keep in mind that each `<isslot>` creates one remote include for the configuration and one remote include for the actual content.

Caching the Refinement Panel

Calculating refinements is expensive, so it's a good idea to cache the refinement panel (also referred to as the "guided navigation menu"). To use the refinements panel, customers select a category and refinements. Next, they select products and visit the product detail page. They might then return to the product list which is still governed by the refinements that are in place. Use a remote include for the refinement panel so that the site can serve the cached refinement panel until the customer changes the refinements.

Knowledge Check

The Knowledge Check answer key is on the following page.

The following diagram represents a storefront page with partial pages A-F described in the table.

(Partial Page) A contains a header with a logo.
B contains a mini-cart.
C contains a refinement panel.
D contains a category sub-banner.
E contains product tiles.
F contains a footer content asset.

In the following question, select the ISML code segments you can use to cache the refinement panel (represented as C in the diagram)? *Select all that apply*.

a. Use `<iscomponent pipeline =“MyRefine-Start”/>` where the `MyRefine-Start` pipeline generates an ISML template that contains `<iscache type=“relative” hour="24"/>`.

b. Use `<isinclude template=“search/components/myrefinebar”/>`. where the included template, `myrefinebar`, contains `<iscache type=“relative” minute=“30” varyby=“price_promotion”/>`.

c. Use `<isinclude template=“search/components/myrefinebar”/>` where the included template, `myrefinebar`, contains `<iscache type=“relative” hour=“24”/>`.

d. Use `<isinclude url=“$URLUtils('MyRefine-Start')}”/>` where the `MyRefine-Start` pipeline generate an ISML template that contains `<iscache type=“relative” hour=“4”/>`.

Knowledge Check Answers

In the following question, select the ISML code segments you can use to cache the refinement panel (represented as C in the diagram)? *Select all that apply*.

a. Use `<iscomponent pipeline ="MyRefine-Start"/>` where the `MyRefine-Start` pipeline generates an ISML template that contains `<iscache type="relative" hour="24"/>`.

b. Use `<isinclude template="search/components/myrefinebar"/>`. where the included template, `myrefinebar`, contains `<iscache type="relative" minute="30" varyby="price_promotion"/>`.

c. Use `<isinclude template="search/components/myrefinebar"/>` where the included template, `myrefinebar`, contains `<iscache type="relative" hour="24"/>`.

d. Use `<isinclude url="$URLUtils('MyRefine-Start')}"/>` where the `MyRefine-Start` pipeline generate an ISML template that contains `<iscache type="relative" hour="24"/>`.

Answers: a and d.

Use remote includes to cache partial pages using the `<isinclude url="">` or the `<iscomponent pipeline=…>` syntax.

Best Practice

Sometimes improving the cache hit ratio is not always the best strategy for a pipeline. Sometimes reducing the processing time of a pipeline is the best approach. Optimize expensive pieces of code—sluggish pipelines, as well as heavily used ones. You'll learn code optimization tips later in this course.

Lesson 2.4: Managing Static Content

Demandware leverages a Content Delivery Network (CDN) that provides static caching for content assets using Akamai servers. Demandware caches images, icons, CSS style sheets, and other non-dynamic assets. You can set the length of time for static content to be cached by the web server. This value is called the Time to Live (TTL). By default, the TTL for content assets is 24 hours (86,400 seconds). At the end of the TTL, the cache is emptied and content is retrieved from the server.

To set the TTL value for static content, in Business Manager, select **Administration > Sites > Manage Sites**. Select your site and select the **Cache** tab. Select an instance type. You select an instance type because some settings are available for sandbox instances, but not for production and development instances.

For **Time to Live (TTL) of Static Content**, enter the amount of time you want to cache static assets. Enter the time in seconds. Demandware caches the asset for the time specified, then retrieves a new version of the asset from the server.

You can invalidate the static content cache for sandbox instances by clicking **Invalidate** next to the **Static Content Cache and Entire Page Cache for Site**.

URLs to Static Content

Demandware provides a utilities class (`dw.web.URLUtils`) for generating URLs to static content.

For production instances, Demandware routes requests for static content to the nearest Akamai server and the server caches the resource.

For development instances, Demandware routes requests for static content to the nearest Akamai server, but the Akamai server doesn't cache the resource. Instead the resource is filtered through Akamai (a pass-through each time). Note that Akamai technology is not supported for sandbox instances.

To ensure that a static resource is configured for caching on an Akamai server, view the resource URL in the HTML on a development or production instance. If the resource is cached, it will have a fully qualified URL using the Akamai domain: `demandware.edgesuite.net`.

This example shows a URL to a site header logo that is served and cached by Akamai:

```
<img src="http://demandware.edgesuite.net/aako_prd/on/demandware.static/
Sites-SiteGenesis-Site/-/default/v1408703042541/images/logo.png">
```

Notice that the URL contains the `Demandware.edgesuite.net` domain.

This example shows a different URL for the same image:

```
<img src="/on/demandware.static/Sites-anilo_us-Site/-
/default/v1408736578363/images/logo.png">
```

In this case, the image is not served or cached by Akamai. It's a relative URL and doesn't include the `Demandware.edgesuite.net` domain.

Analyzing Content Asset Usage

To determine which content assets consume the most space, you can use the WebDAV Size Analyzer. Download the utility from Bitbucket:

https://bitbucket.org/demandware/dataresourceusage

You'll need a Bitbucket account and access to the Demandware repository to download the Size Analyzer. For instructions on setting up and using the utility, see https://bitbucket.org/demandware/dataresourceusage/wiki/Home/

Size Analyzer traverses all directories on the Demandware WebDAV servers and generates a tree structure that indicates the size of each subdirectory, for example:

```
--- Impex folder ---
Resource: Impex/ (0MB)
+--Resource: log/ (0MB)
+--Resource: src/ (0MB)
|  +--Resource: customization/ (0MB)
|  +--Resource: customobject/ (0MB)
|  +--Resource: instance/ (0MB)
|  +--Resource: marketing/ (0MB)
|  +--Resource: order/ (0MB)
+--Resource: upload/ (0MB)

--- Catalogs ---
Resource: Catalogs/storefront-catalog-en/ (10MB)
+--Resource: default/ (10MB)
|  +--Resource: images/ (9MB)
|  |  +--Resource: slot/ (9MB)
|  +--Resource: pwr/ (0MB)
|  |  +--Resource: content/ (0MB)
|  |  +--Resource: engine/ (0MB)
|  |  +--Resource: p6879prc/ (0MB)

--- Log folder ---
Resource: Logs/ (17MB)
+--Resource: dbinit-sql/ (0MB)
|  +--Resource: data_kernel/ (0MB)
+--Resource: deprecation/ (0MB)
+--Resource: soa/ (0MB)
|  +--Resource: log_archive/ (0MB)
```

Best Practice

To take advantage of the CDN caching, do not use system objects to deliver static resources. By doing so, you bypass the optimized static caching that Demandware provides. For example, to display a different background image that varies based on the category a customer is browsing, you can use JavaScript dynamic processing.

Module 3: Optimizing Performance During Development

Learning Objectives

After completing this module, you will be able to:

- Optimize client-side and server-side processing using coding best practices.
- Manage data efficiently and monitor data storage usage.

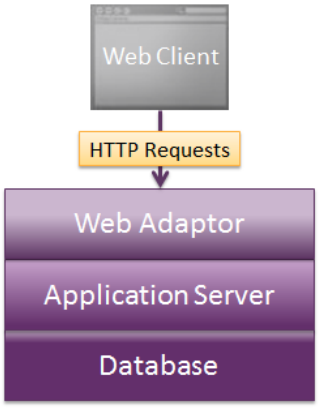
Introduction

eCommerce applications built on the Demandware Commerce platform will run fast and perform reliably if you use the platform within its capabilities. By using coding best practices for your customizations, you can improve page load times and ensure that your site stays within quota limits.

This module provides tips so that you can select the optimal approach for your site customizations, including client-side and server-side customizations.

Lesson 3.1: Improving Client-Side Performance

To improve client-side performance, you'll need to minimize the size and the number of elements on pages. Additionally, you'll want to reduce the amount of JavaScript and avoid complex style sheets and HTML markup. The key to improving client-side performance is reducing the number of HTTP requests to the server. The less data transmitted and the fewer requests performed, the better the performance.



You can reduce the number of HTTP requests using these methods:

- Separate HTML from JavaScript.
- Compress and concatenate CSS and JavaScript.
- Use "CSS sprites" to combine visual elements into a single image.

The following table describes methods to reduce HTTP requests:

Separate HTML from JavaScript.
To reduce the number of HTTP requests, use unobtrusive JavaScript, the strict separation of the HTML and the JavaScript layer. Separating the HTML from the JavaScript leads to a more robust site. Because browsers stop, load, and interpret JavaScript where the code occurs on the page, issues with inline JavaScript blocks can cause the browser to stall when loading images or rendering a page. Separating out the HTML and JavaScript also makes the Document Object Model (DOM) smaller and improves page load times. Avoiding inline JavaScript also makes it easier to understand both the HTML and JavaScript, as the markup is plain HTML without any embedded chunks of functionality.
Compress and concatenate CSS and JavaScript.
To reduce the number of HTTP requests, you should minify your CSS and JavaScript files. BuildSuite has tools for minifying code. Implement CSS in a layered way—start with an overall CSS, then develop more granular CSS on a per-page basis. You can concatenate your CSS and JavaScript using BuildSuite to: ▪ Reduce the number of requests ▪ Speed up the download ▪ Speed up caching of resources It's best to concatenate these files during deployment. If you concatenate the files during development, you lose the modularity of your code making the code more difficult to update and debug. These types of tools merge JavaScript files, insert generated variable and function names, and strip comments, line breaks, and other unnecessary characters to reduce the size of the files. For details on BuildSuite, see https://xchange.demandware.com/docs/DOC-5728.

Compress Images, Use CSS Sprites
If your site is rich in images, you'll need to optimize image processing. The first step is to compress the images. You can also use "CSS sprites" to further optimize image processing. A CSS sprite is an aggregation of a group of images, stacked vertically or horizontally. Instead of retrieving multiple image files, the client accesses the single aggregated image, the sprite. A client request for an image specifies the coordinates of one of the images included within the sprite. By using CSS sprites, you can load a single image rather than loading images individually. Consequently, you reduce the number of HTTP requests and speed up page loads.

Knowledge Check

The Knowledge Check answer key is on the following page.

1. Which of the following are methods of reducing the number of HTTP requests to the server? *Select all that apply.*
   a. Use aggregated images to optimize image processing.
   b. Separate the templates from the pipelines.
   c. Concatenate CSS and JavaScript files.
   d. Cache the CSS style sheets.

2. Match the technique on the left with its benefit to performance on the right.

BuildSuite	Retains readability of code during development.
CSS sprite	Minimizes code to speed up download and caching of resources.
Unobtrusive JavaScript	Lets you load a single image file rather multiple image files.
Minimize Document Object Model (DOM)	Separates HTML and JavaScript layers.
Concatenation during deployment	Improves page load times.

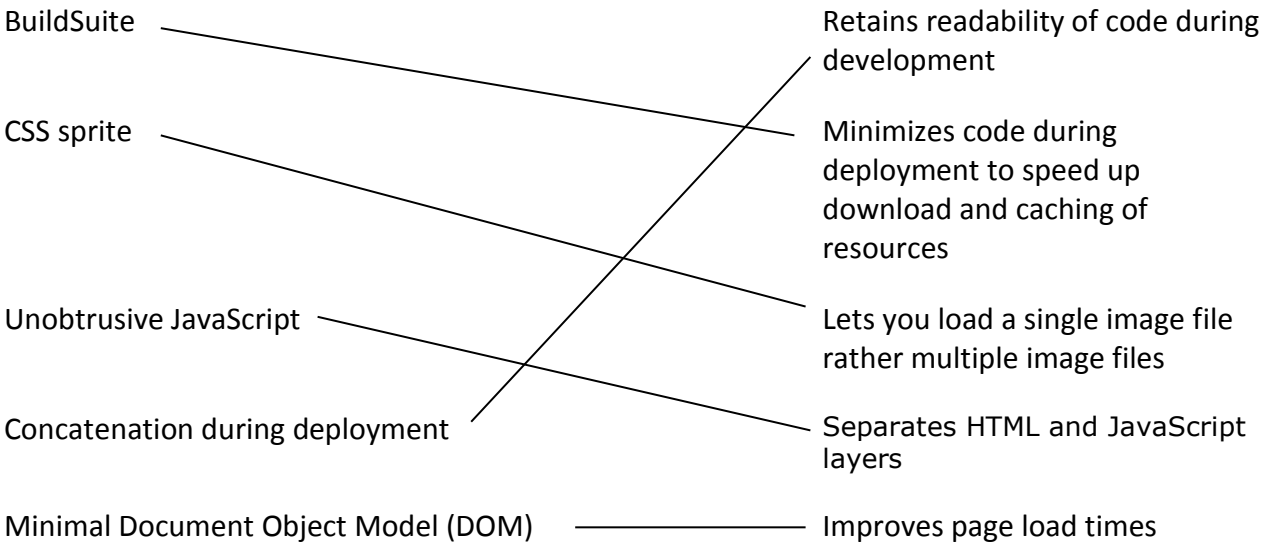
XEM ĐÁP ÁN Ở DƯỚI CHO CHẮC NHA

Knowledge Check Answers

1. Which of the following are methods of reducing the number of HTTP requests to the server?
   a. Use aggregated images to optimize image processing.
   b. Separate the templates from the pipelines.
   c. Concatenate CSS and JavaScript files.
   d. Aggregate server requests.

Answers: a, c

2. Match the technique on the left with its benefit to performance on the right.



Lesson 3.2: Improving Server-Side Performance

The Demandware platform provides the resources necessary to meet client requirements and handle customer load automatically. The platform optimizes your site's performance. However, scaling hardware resources can't always make up for inefficient coding. By following the best practices in this lesson, you'll ensure that your site and the underlying systems—web servers, application servers and database servers—will scale effectively.

Performance improvements on the server side can greatly improve your site's performance. You can minimize the number of transactions that are passed through to the application layer and to the database following these guidelines (described in the following table):

- Avoid iterating over products and categories.
- Minimize post-processing of search results.
- Optimize the display of search results.
- Avoid URL diversity by keeping positional parameters out of URLs.
- Ensure efficiency of scripts within templates.
- For long-running operations like imports and exports, use jobs.
- Do not trigger live calls to external systems on frequently visited pages.

Avoid Iterating Over Products and Categories

It's natural to expect to use the Catalog and Category APIs to iterate over products and categories, for example, variants of a master. However, these methods access the database and cause latency between the server and the database.

The most efficient way to access product and category information is actually through the Search API. You can obtain a great deal of product information from the Search API, which accesses the search index rather than the database tier.

The following table shows examples of inefficient API methods that return large collection objects from the database and alternative Search API methods.

Inefficient API Methods
▪ `Category.getProducts()` ▪ `Category.getOnlineProducts()` ▪ `Category.getCategoryAssignments()` ▪ `Category.getOnlineCategoryAssignments()` ▪ `ProductMgr.queryAllSiteProducts()` These methods execute queries against the database and typically return large collections of objects. Often developers use these methods to traverse category trees and iterate over objects to decide whether to use the objects. Iterating over these objects causes database churn, where the system must rotate objects in memory frequently. And if you add these collections onto the pipeline dictionary as opposed to using them in a pipelet script, the performance degrades further. The system keeps these objects loaded into memory until it processes the request completely. If you include them in a pipelet script, the system can make them available for garbage collection after the script executes.
More Efficient Search API Methods
▪ `ProductSearchModel.orderableProductsOnly(flag)` ▪ `ProductSearchRefinements.getNextLevelCategoryRefinementValues(Category)` ▪ `ProductSearchHit.getRepresentedVariationValues(ProductVariationAttribute)` ▪ `ProductSearchHit.getMinPrice(), ProductSearchHit.getMaxPrice()` `ProductSearchModel`, `ProductSearchRefinements`, and `ProductSearchHit` let you access products and categories without having to access the database. The `ProductSearchModel` returns only online objects, so you never have to check whether a product is online.

You can ensure that storefront pipelines are efficient by selecting optimal API methods rather than developing custom code to process query results. For instance, instead of manually counting the number of products in a category using a loop, use `ProductSearchRefinementValue.getHitCount()`.

The following inefficient code example implements a global navigation scheme that renders category links for each category with orderable products. The script calls `dw.catalog.CatalogMgr.getSiteCatalog()` and iterates over the site's categories and the

products within each category to search for online products. If the category has at least one orderable product, the script adds the category or sub-category name to the `cats` array.

Don't Iterate Over Products Using the Category Class

```
/**
*
*   @input ExampleIn : String This is a sample comment.
*   @output outputCategories : dw.util.ArrayList
*
*/
importPackage( dw.system );
importPackage( dw.catalog );

function execute( args : PipelineDictionary ) : Number
{

    // get root category of current site's navigation catalog
    var siteCatalog:Catalog = CatalogMgr.getSiteCatalog();
    var root:Category = null;
    if(siteCatalog!=null) {root = siteCatalog.getRoot();}

    var cats : dw.util.ArrayList = new dw.util.ArrayList();
    var slcat:Category;
    var tlcat:Category;
    var flcat:Category;

    for each ( flcat in root.getOnlineSubCategories()) {
          for each ( slcat  in flcat.getOnlineSubCategories()) {
               for each ( tlcat  in slcat.getOnlineSubCategories()) {
                    if(checkProduct(tlcat)) {
                    cats.add(tlcat.ID);
                    if(!cats.contains(slcat.ID)) cats.add(slcat.ID);
                    if(!cats.contains(flcat.ID)) cats.add(flcat.ID);
                    }
               }
               }
          if(checkProduct(slcat)) {
               if(!cats.contains(slcat.ID)) cats.add(slcat.ID);
               if(!cats.contains(flcat.ID)) cats.add(flcat.ID);
          }
          if (checkProduct(flcat)) if(!cats.contains(flcat.ID)) cats.add(flcat.ID);

    }

   args.outputCategories=cats;
    return PIPELET_NEXT;
}
```

Uses `CatalogMgr` class to get the site catalog.

Iterates over categories and products using the `Category` class.

Calls `checkProduct` function which iterates through the category searching for orderable products.

This code sample instantiates all products in every category until it finds an orderable product—a costly algorithm.

Use the Search APIs for Efficient Scripting

You can implement the same functionality more efficiently using the `ProductSearchModel` shown in the following code example. This script doesn't have to access the database after the initial search because the availability information is already included in the availability index.

```
/**
*   @input ProductSearchModel : dw.catalog.ProductSearchModel
*   @output outputCategories : dw.util.ArrayList
*
*/
importPackage( dw.system );
importPackage (dw.catalog );

function execute( args : PipelineDictionary ) : Number
{

    var searchcats : dw.util.ArrayList = new dw.util.ArrayList();

    for each (var flref: ProductSearchRefinementValue in
        args.ProductSearchModel.refinements.getNextLevelCategoryRefinementValues(args.ProductSearchModel.category)) {
        var flcat : Category = dw.catalog.CatalogMgr.getCategory(flref.value);

        for each (var slref : ProductSearchRefinementValue in
            args.ProductSearchModel.refinements.getNextLevelCategoryRefinementValues(flcat)) {
            var slcat : Category = CatalogMgr.getCategory(slref.value);

            for each (var tlref : ProductSearchRefinementValue in
                args.ProductSearchModel.refinements.getNextLevelCategoryRefinementValues(slcat)) {
                var tlcat: dw.catalog.Category = dw.catalog.CatalogMgr.getCategory(tlref.value);
                searchcats.add(tlcat.ID);
            }
            searchcats.add(slcat.ID);
        }
        searchcats.add(flcat.ID);
        }
    args.outputCategories=searchcats;
   return PIPELET_NEXT;
}
```

Minimize Post-Processing of Search Results

Using the Search APIs is efficient, but be sure not to post-process the search results inefficiently. The following script uses custom code to return all products variants that are in stock. The script uses the Search APIs, but then fetches all the variation data and all the inventory data. If the variant is in stock, the script adds it to the results.

```
function execute( pdict : PipelineDictionary ) : Number
{

    var psm : ProductSearchModel = pdict.SearchResult;
    var results: ArrayList = new ArrayList();;
    var psmit : Iterator = psm.getProductSearchHits();
    var variantIterator : Iterator;
    var productHit : ProductSearchHit;
    var variant : Variant;
    var pvm : ProductVariationModel;
    var instock : Boolean;

    while (psmit.hasNext())
    {
       productHit = psmit.next();
       pvm = getVariationModel(productHit);
       variantIterator=getVariants(pvm);
       instock = true;

       while (variantIterator.hasNext()){
          variant = getVariant(variantIterator);
          if(!isInStock(variant))
          {
               instock = false;
               break;
          }
       }

       if(instock){
           results.add(productHit);
       }
    }
```

Best Practice

Search results can be large sets. If a site has 10 thousand t-shirts and you search for yellow variants, the custom algorithm causes a huge processing hit. Custom search result processing is the most prevalent cause of performance issues.

Rather than post-processing product or content search results using custom code, include all search criteria in a single query for efficient execution.

Knowledge Check

A developer in your group created the following script and has found that the performance is poor:

```
function execute( args : PipelineDictionary ) : Number
{

     var productIterator : SeekableIterator = dw.catalog.ProductMgr.queryAllSiteProducts();
     var prodDone : dw.util.ArrayList = new dw.util.ArrayList();

     while(productIterator.hasNext()) {

          var product : Product = productIterator.next();
          var productIDColor : String;
          var color : String;

          if(!product.master) {

               // if the product is not a variant, add the product ID
               // if the product is a variant without color variations, add the master ID
               if(!product.variant || product.variant && empty(product.custom.color)) {
                    productIDColor = product.variant ? product.master.ID : product.ID;

               // else create an ID from the (master) product ID + color value
               }else if (product.variant) {
                    color = product.custom.color;
                    var master : dw.catalog.Product = product.variationModel.master;
                    productIDColor = master.ID + color;
               }

               // if the list does not contain the ID, add it to the list
               if (!empty(productIDColor) && !prodDone.contains(productIDColor) && product.availabilityModel.orderable) {
                    // execute some code to add this product to the feed
                    prodDone.add(productIDColor);
               }
          }
     }


   return PIPELET_NEXT;
}
```

1. What are some possible reasons for the poor performance? *Select all that apply.*
   a. The script checks for master products using `product.master`.
   b. The script iterates over products to determine if a product is a variant.
   c. The script returns a custom color by iterating over custom objects.
   d. The script post-processes search results returned using the `ProductMgr` class.
   e. The script uses the Search API's availability model to determine if the product variation exists.

2. To improve the performance of the script, which of the following API methods might you use instead of the methods in the script? *Select all that apply.*
   a. `ProductSearchModel.orderableProductsOnly()`
   b. `GetProductList` pipelet
   c. `ProductMgr.getRepresentedVariationValues()`
   d. `ProductSearchHit.getRepresentedVariationValues()`
   e. `Search` pipelet

Knowledge Check Answers

1. What are some possible reasons for the poor performance? *Select all that apply.*
   a. The script checks for master products using `product.master`.
   b. The script iterates over products to determine if a product is a variant.
   c. The script returns a custom color by iterating over custom objects.
   d. The script post-processes search results returned using the `ProductMgr` class.
   e. The script uses the Search API's availability model to determine if the product variation exists.

Answers: b, d

2. To improve the performance of the script, which of the following API methods might you use instead of the methods in the script? *Select all that apply.*
   a. `ProductSearchModel.orderableProductsOnly()`
   b. `GetProductList` pipelet
   c. `ProductMgr.getRepresentedVariationValues()`
   d. `ProductSearchHit.getRepresentedVariationValues()`
   e. `Search` pipelet

Answers: a, d, e

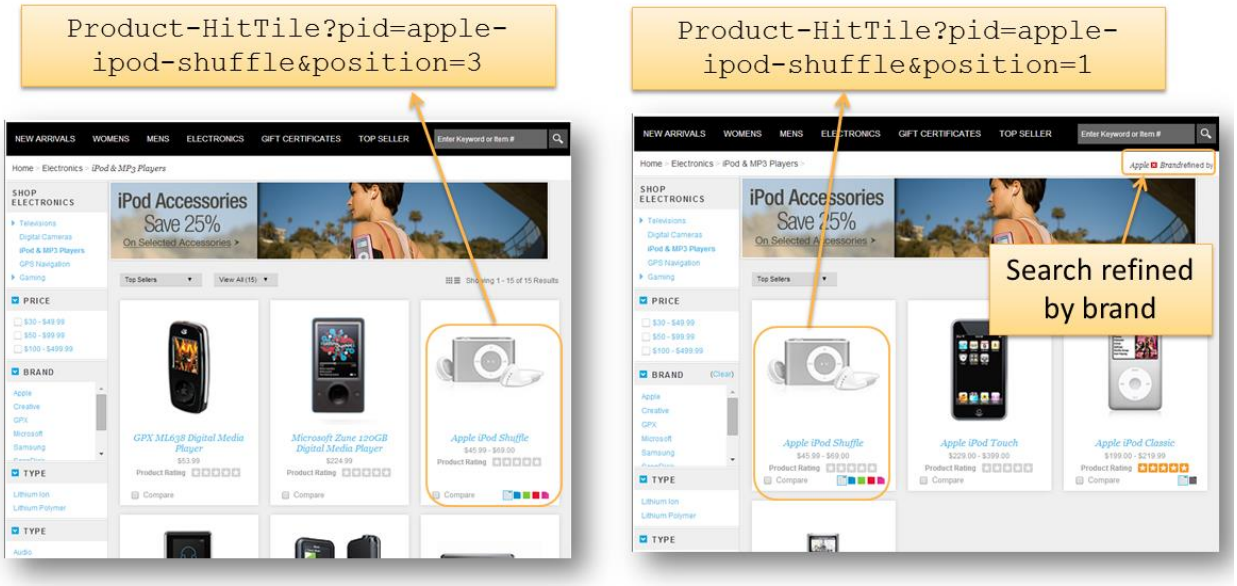
Optimize the Display of Search Results

Searches with many matches can result in slow search grid pages. To optimize the display of search results, break the search results into pages using the `Paging` pipelet before sending them to the client. Set the `PageSize` parameter for the `Paging` pipelet—a reasonable value is 24 products. Displaying more products than that on grid pages is very expensive.

View the parameters for the `Paging` pipelet by accessing the Demandware online help: `https://info.demandware.com` and selecting **Demandware API > Demandware Pipelets > Common**. Select the `Paging` pipelet. Alternatively, you can review the `Search-Show` pipeline in SiteGenesis.

Avoid URL Diversity by Keeping Positional Parameters Out of URLs

When the system caches a pipeline response, it hashes the pipeline request URL and uses it as a caching key to retrieve the results from the cache for subsequent requests. Because request URLs are hashed, they must be identical to take advantage of caching. The pages in the following example reuse the iPod product tile. Because the URLs include different positional parameters, they are cached separately:



Because the URLs are generated with different parameters, the system doesn't make use of the cached URLs. Caching actually impedes performance in this case. Use the same URL to ensure proper reuse.

To view the URLs and their parameters called for your site, within Business Manager, select **Site > Analytics > Traffic Reports**. Select the **Top Pages** report. The Analytics reports are available only on production instances.

Ensure Efficiency of Template Scripts

In general, you should use DWScripts rather than using `<isscript>` blocks in your ISML templates. If you must use `<isscript>` blocks, be sure to keep them small. To ensure that you can diagnose performance issues, break large `<isscript>` blocks in templates into multiple pieces. With smaller chunks of DW Script, you'll be able to pinpoint bottlenecks more efficiently. Smaller chunks also help with readability.

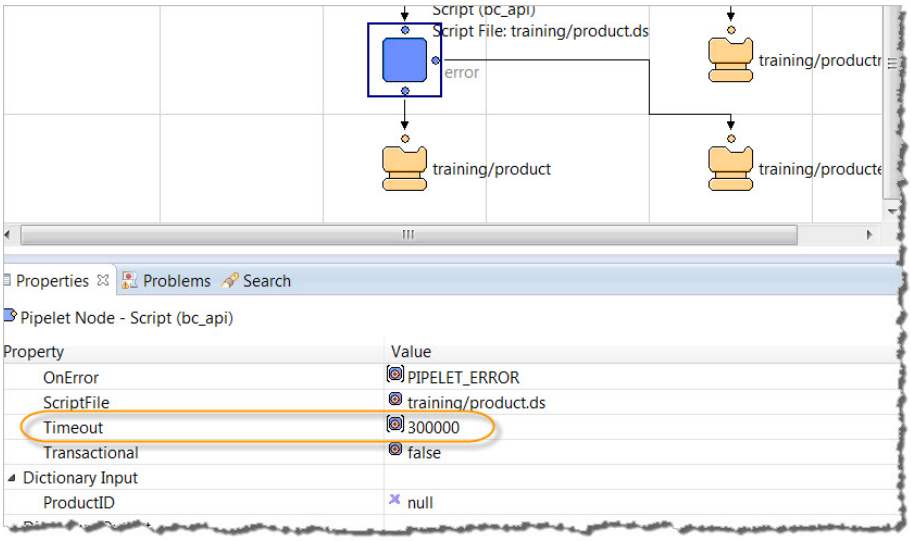
Avoid Long-Running Operations in Storefront Pipelines

- Avoid executing long running operations such as imports and exports in storefront pipelines.

  Demandware implements script timeouts to protect site stability, but the web tier closes the browser connection after a specified timeout, by default, 30 seconds. The maximum script timeout you

can define is five minutes. Even if you set the timeout of a script to the maximum of five minutes, it might not be finished after five minutes. An endless loop can exhaust threads and require your operations team to restart the instance.

To set the script timeout, use the `Timeout` property of the script node.



- Instead of using pipeline scripts for long running operations, use jobs.

  The maximum timeout for jobs is 60 minutes. You'll learn about optimizing jobs later in the course.

- Another way to prevent thread exhaustion is to avoid using live calls to external systems on frequently visited pages.

  For example, do not use live calls on the home page, product detail pages, category pages, and search results pages. Where live calls are needed, specify a low timeout value, for example, one second. A Demandware application server thread waiting for a response from an external system cannot serve other requests. Many threads waiting for responses can make the entire cluster unresponsive for all pages.

Lesson 3.3: Implementing Optimal Data Management Strategies

One of the most important development tasks in a Demandware implementation is to determine which information should be stored on the platform and how it should be stored. Plan a data storage strategy that uses the most appropriate type of storage for each type of data. To do so, you'll need to think of the lifetimes and states needed for each type of data object.

Exercise: Select a Storage Strategy

Following are storage strategies you can use with the platform. Review these strategies and answer the questions that follow.

External Systems
Decide whether the data should be stored in the Demandware database or in an external system, such as an Enterprise Resource Planning (ERP) system or a Product Information Management (PIM) system. ▪ You can use asynchronous import and export jobs to synch up with the external systems. ▪ You can also develop synchronous integrations using web services. Some implementations store legacy order data and email subscription data in custom objects, but in this case it's more efficient to keep the data in external systems and use web services for real-time access to the data.
Custom Attributes and Objects
You can extend the Demandware platform by creating: ▪ Custom objects ▪ Custom attributes for system objects The system stores custom attributes and localizable system attributes in the database in tables with the system objects. You access these using a compound key of the attribute ID, the locale, and the corresponding system object ID. Accessing these attributes in the database is expensive—especially as the data set grows over time. Defining many custom attributes and creating object queries with many attribute conditions impedes performance. The system processes custom objects similarly, so use custom objects judiciously, as well. For example, if you implement custom analytics using custom objects, you'll have to write to these objects for each request. This implementation might seem fine in your sandbox, but on production, customers can generate hundreds or thousands of these custom objects, degrading performance. A valid use of custom objects is to store small, temporary data sets or data that an administrator manages using Business Manager. A common use case for custom objects is to store temporary data to configure analytics integrations. Be sure to access custom objects and system object attributes using primary keys rather than secondary keys. You can determine the primary key for each object in Business Manager. The primary key is the object's attribute shown with the key icon.

Sessions

One way to limit the use of custom objects is to use session storage. When you calculate the cart contents or a wish list, you can create a session basket stored as JSON. When the customer accesses the mini-cart, the system checks if the JSON object exists and, if so, parses it. Using the session storage in this way minimizes database access. When the customer views a new page, the system does not have to access the basket in the database to make sure the basket is up-to-date.

Cookies

Cookies let you store data across sessions. Use cookies for information that is required for every server-side request. An effective use of cookies is to save geolocation information. Sites can use geolocation to present region-specific content or experiences based on the customer's location. To do this, store the geolocation data in a cookie on the browser. If you use cookies in this way, you won't need to make subsequent calls to the geolocation system for each request.

Keep in mind that the cookie header size is limited in the web adapter, so use cookies sparingly. If a cookie grows with each request (for example, if you're appending data to a cookie based on a customer's actions), the cookie header will eventually exceed the limit and the web adapter will no longer accept the request. Since cookies are sent with every request, problems with cookies slow down client-side performance. It is preferable to use `sessionStorage` or `localStorage` to cookies if possible.

HTML5

Like cookies, you can use HTML5 to store data locally within a user's browser. HTML5 has the advantage that the system utilizes the stored data without having to send the data with every web request. HTML5 lets you access large data sets quickly (typically up to 5 MB).

Keep in mind that local storage has a separate database per protocol in the browser (HTTP versus HTTPS).

Select a Storage Strategy

1. Suppose you need to store data across sessions. Which storage strategy or strategies would you use? *Select all that apply.*

	Sessions
	External Systems
	Custom and System Objects
	Cookies
	HTML5

2. Suppose you need to store data across sessions but you don't want to send the data with each request. Which storage strategy or strategies would you use? *Select all that apply*.

	Sessions
	External Systems
	Custom and System Objects
	Cookies
	HTML5

3. Which storage strategy or strategies let you save state information within a session? *Select all that apply*.

	Sessions
	External Systems
	Custom and System Objects
	Cookies
	HTML5

4. Which storage strategy is most efficient for storing legacy order data?

	Sessions
	External Systems
	Custom and System Objects
	Cookies
	HTML5

Answers: Select a Storage Strategy

1. Suppose you need to store data across sessions. Which storage strategy or strategies would you use? *Select all that apply.*

	Sessions
	External Systems
	Custom and System Objects
✓	Cookies
✓	HTML5

2. Suppose you need to store data across sessions but you don't want to send the data with each request. Which storage strategy or strategies would you use? *Select all that apply*.

	Sessions
	External Systems
	Custom and System Objects
	Cookies
✓	HTML5

3. Which storage strategy or strategies let you save state information within a session? *Select all that apply*.

✓	Sessions
	External Systems
✓	Custom and System Objects
	Cookies
	HTML5

4. Which storage strategy is most efficient for storing legacy order data?

	Sessions
✓	External Systems
	Custom and System Objects
	Cookies
	HTML5

You've finished the exercise.

Object Creation

Delay creating baskets, wish lists, and custom objects until the customer actually needs them. Create a basket only when a customer adds an item for the first time. If you create the basket automatically for a customer's session or request, the customer has to wait for the database to create the object. If you avoid creating these objects until they're needed, you create fewer objects, so your system cleanup jobs will be faster.

Note that when you use a transactional pipelet, you're accessing the database, so limit the number of transactional pipelets you use in your pipelines.

Input/Output

Do not use the `dw.io` classes unless you absolutely have to.

- With the `dw.io` classes, you risk making concurrent writes by different application servers to the same file.
- For example, to generate log files, use the `dw.system.Logger`.

Avoid Concurrent Processing

You should prevent concurrent changes to the same object:

- For shared data like catalogs and prices, use read-only processing.
- For customer-specific data like customer profiles, shopping carts, and orders, you can use read and write processing.

You do not have to safeguard inventory processing against concurrent changes. The inventory framework is designed to support concurrent change, for example, two customers buying the same product at the same time or a customer buying a product while the inventory import is running. The storefront pipeline marks the order with `EXPORT_STATUS_READY` as the last step in order creation. Once the `EXPORT_STATUS_READY` flag is set, order processing jobs can start modifying the order object. Concurrent requests for the same session are serialized at the application server.

Object Churn

The Demandware platform uses an Object Relational Mapper (ORM)—an object cache—to avoid duplicate fetches from the database, for example, when fetching product information. This cache is transparent to the custom application, but improves performance significantly. The system fetches the items initially, then looks them up easily using their primary keys.

The terms “ORM churn” and “object churn” refer to situations when the system must rotate objects frequently in memory. ORM churn occurs when a script or job accesses many database objects one-by-one without reusing them. Rather than using complex object queries that hit the database repeatedly, look up objects by ID wherever possible to leverage the ORM caching capability.

Although some object churn is expected, inefficient practices result in excessive object churn. If your site is experiencing slow checkouts, object churn might be the cause. The Object Churn Trends report shows trends in object creation, updates, and deletions. The report analyzes these objects:

- Sessions (creation only)
- Baskets
- Orders
- Products
- Wish Lists
- Customers

Module 4: Analyzing Performance Issues

Learning Objectives

After completing this module, you will be able to:

- Analyze performance using Pipeline Profiler.
- Analyze performance using Business Manager Analytics.

Lesson 4.1: Diagnosing Performance with the Pipeline Profiler

The Pipeline Profiler in Business Manager lets you analyze execution times of scripts, pipelines, and templates. To ensure that you develop efficient code, it's important to use the Pipeline Profiler during development. You can activate the Profiler on production instances, as well, but using it during development and testing can head off issues.

To diagnose your pipelines and scripts, you:

1. Enable the Profiler.
2. Run the pipelines you want to test.
3. View the Pipeline Profiler page.

To ensure that you obtain meaningful averages, run the pipelines and scripts a number of times before viewing the data. Performance times vary depending on factors like caching and site traffic.

The Profiler captures all requests that hit the application server and detects high hit counts and high average runtimes. These statistics help you identify the root causes of slow pages or pipelines, so you know which code to optimize.

Note: The Profiler analyzes code executed by the application server only. The Profiler doesn't monitor cached pages—the web server manages cached pages, bypassing the application server.

Detecting Pipeline Hot Spots

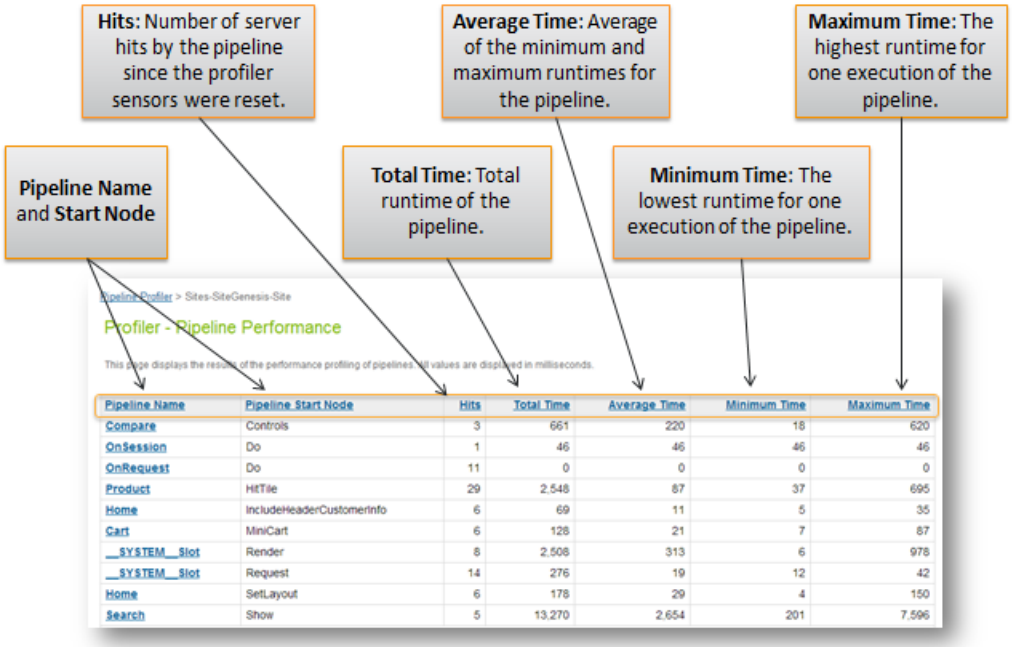
Use the Pipeline Profiler to tune the performance of pipelines. The Profiler provides the number of hits and execution times of pipelines. The Profiler helps you detect long-running pipelines. Demandware recommends that:

- Non-cached pipelines run for a maximum of 3 seconds.
- Cached pipelines run for a maximum of 250 milliseconds.

To analyze pipelines, select **Administration > Operations > Pipeline Profiler** and click the green start button to activate the pipeline profiler. On your storefront site, perform operations you want to profile. On the Pipeline Profiler page, click **Reset** and click your storefront site link.

If the Pipeline Profiler indicates that it has run for the maximum allowed duration, click **Reset Sensors**, rerun the pipeline, and click **Refresh**.

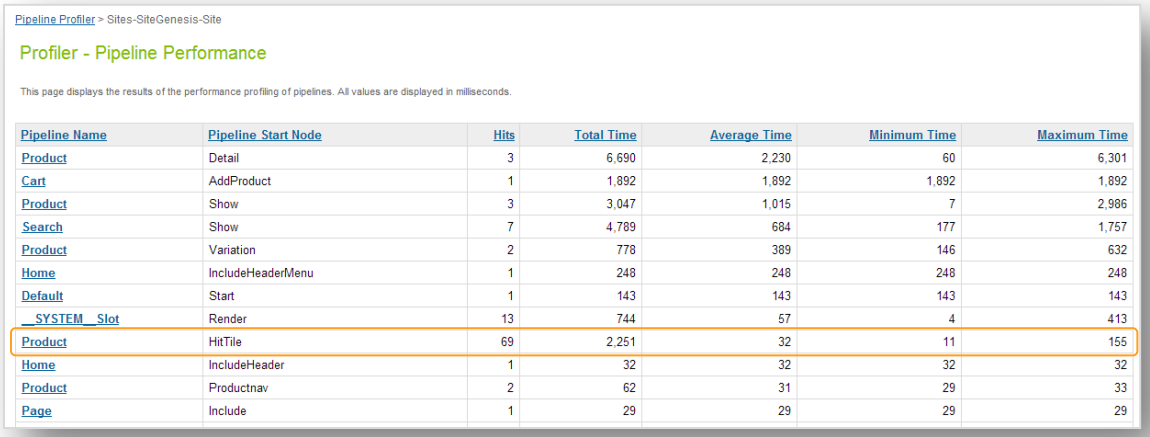
The Pipeline Profiler displays data for all pipelines:



Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Compare	Controls	3	661	220	18	620
OnSession	Do	1	46	46	46	46
OnRequest	Do	11	0	0	0	0
Product	HitTile	29	2,548	87	37	695
Home	IncludeHeaderCustomerInfo	6	69	11	5	35
Cart	MiniCart	6	128	21	7	87
__SYSTEM__Slot	Render	8	2,508	313	6	978
__SYSTEM__Slot	Request	14	276	19	12	42
Home	SetLayout	6	178	29	4	150
Search	Show	5	13,270	2,654	201	7,596

When you use the Pipeline Profiler, monitor the pipelines that your site calls most frequently. Performance issues in these pipelines have more of an effect on site performance than pipelines called infrequently. Optimizing frequently-used pipelines is often an effective method for improving site performance.

In this example, the `Product-HitTile` pipeline is used most frequently—69 times:



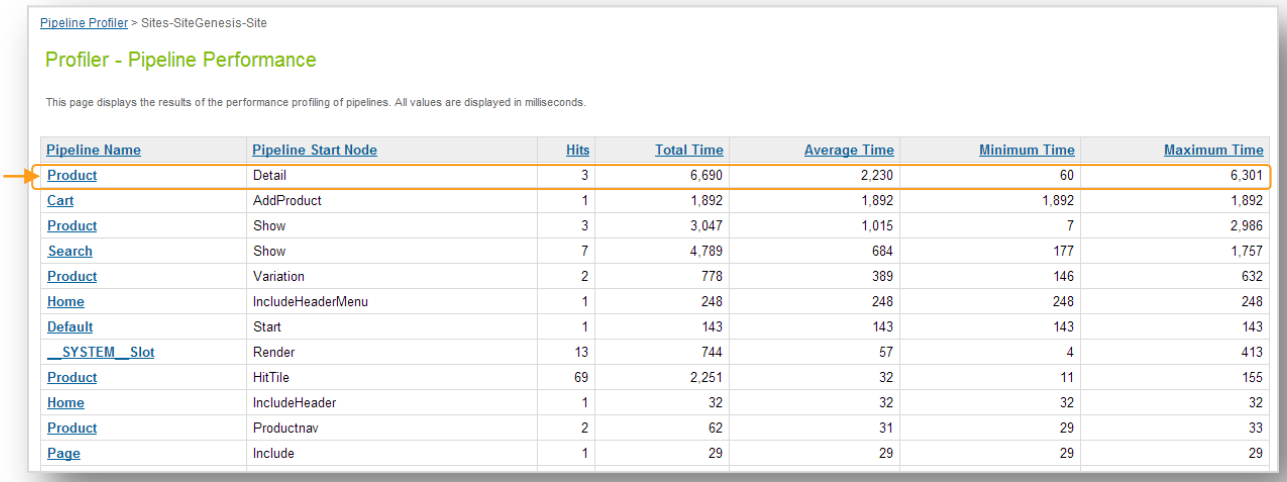
Pipeline Profiler > Sites-SiteGenesis-Site

Profiler - Pipeline Performance

This page displays the results of the performance profiling of pipelines. All values are displayed in milliseconds.

Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Product	Detail	3	6,690	2,230	60	6,301
Cart	AddProduct	1	1,892	1,892	1,892	1,892
Product	Show	3	3,047	1,015	7	2,986
Search	Show	7	4,789	684	177	1,757
Product	Variation	2	778	389	146	632
Home	IncludeHeaderMenu	1	248	248	248	248
Default	Start	1	143	143	143	143
__SYSTEM__Slot	Render	13	744	57	4	413
Product	HitTile	69	2,251	32	11	155
Home	IncludeHeader	1	32	32	32	32
Product	Productnav	2	62	31	29	33
Page	Include	1	29	29	29	29

The `Product-Detail` pipeline is used less frequently than the `Product-HitTile` pipeline. For the same period, the site called the `Product-Detail` pipeline 3 times only:



Pipeline Profiler > Sites-SiteGenesis-Site

Profiler - Pipeline Performance

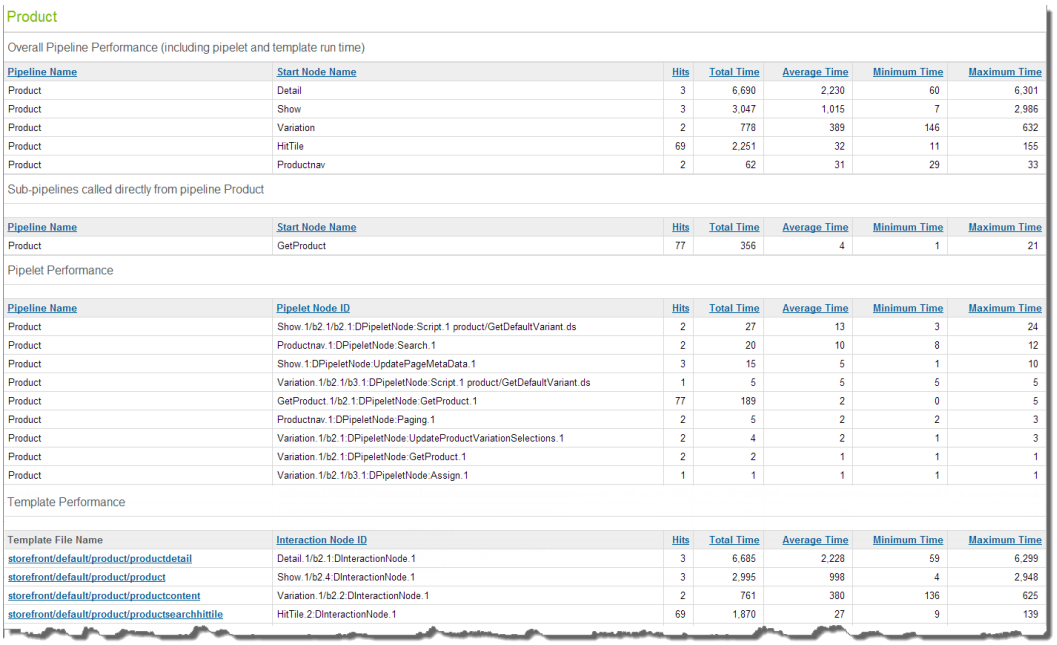
This page displays the results of the performance profiling of pipelines. All values are displayed in milliseconds.

Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Product	Detail	3	6,690	2,230	60	6,301
Cart	AddProduct	1	1,892	1,892	1,892	1,892
Product	Show	3	3,047	1,015	7	2,986
Search	Show	7	4,789	684	177	1,757
Product	Variation	2	778	389	146	632
Home	IncludeHeaderMenu	1	248	248	248	248
Default	Start	1	143	143	143	143
__SYSTEM__Slot	Render	13	744	57	4	413
Product	HitTile	69	2,251	32	11	155
Home	IncludeHeader	1	32	32	32	32
Product	Productnav	2	62	31	29	33
Page	Include	1	29	29	29	29

However, the **Total Time** for the `Product-Detail` pipeline is 6,990 ms, longer than for the `Product-HitTile` pipeline, which is 2,251 ms. Even though the `Product-HitTile` pipeline is called frequently, its **Total Time** is far less than the `Product-Detail` pipeline.

Optimizing long-running pipelines is critical to improving site performance. In this example, the priority is to optimize the `Product-Detail` pipeline, a long-running pipeline with an average runtime of 2,230 ms. To investigate the `Product-Detail` pipeline further, click the **Product** link corresponding to the `Detail` start node.

By selecting a pipeline, in this case `Product-Detail`, you can view its performance, including its sub-pipelines, pipelet scripts, and templates:



Product

Overall Pipeline Performance (including pipelet and template run time)

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Product	Detail	3	6,690	2,230	60	6,301
Product	Show	3	3,047	1,015	7	2,986
Product	Variation	2	778	389	146	632
Product	HitTile	69	2,251	32	11	155
Product	Productnav	2	62	31	29	33

Sub-pipelines called directly from pipeline Product

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Product	GetProduct	77	356	4	1	21

Pipelet Performance

Pipeline Name	Pipelet Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Product	Show.1/b2.1/b2.1:DPipeletNode:Script.1 product/GetDefaultVariant.ds	2	27	13	3	24
Product	Productnav.1:DPipeletNode:Search.1	2	20	10	8	12
Product	Show.1:DPipeletNode:UpdatePageMetaData.1	3	15	5	1	10
Product	Variation.1/b2.1/b3.1:DPipeletNode:Script.1 product/GetDefaultVariant.ds	1	5	5	5	5
Product	GetProduct.1/b2.1:DPipeletNode:GetProduct.1	77	189	2	0	5
Product	Productnav.1:DPipeletNode:Paging.1	2	5	2	2	3
Product	Variation.1/b2.1:DPipeletNode:UpdateProductVariationSelections.1	2	4	2	1	3
Product	Variation.1/b2.1:DPipeletNode:GetProduct.1	2	2	1	1	1
Product	Variation.1/b2.1/b3.1:DPipeletNode:Assign.1	1	1	1	1	1

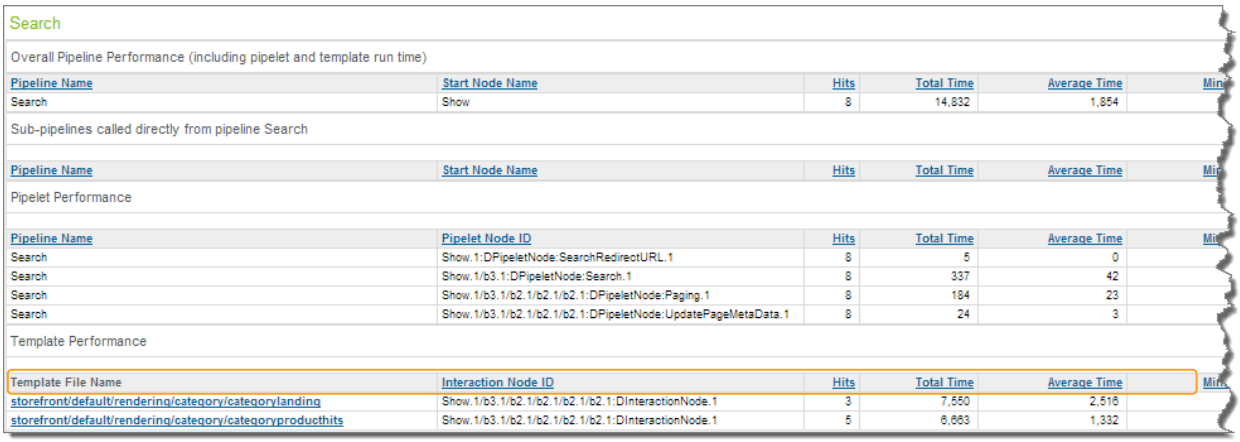
Template Performance

Template File Name	Interaction Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
storefront/default/product/productdetail	Detail.1/b2.1:DInteractionNode.1	3	6,685	2,228	59	6,299
storefront/default/product/product	Show.1/b2.4:DInteractionNode.1	3	2,995	998	4	2,948
storefront/default/product/productcontent	Variation.1/b2.2:DInteractionNode.1	2	761	380	136	625
storefront/default/product/productsearchhittile	HitTile.2:DInteractionNode.1	69	1,870	27	9	139

Processing times factor in includes and script code. The Pipelet Node IDs and the Interaction Node IDs are represented as paths from the start node through branches to the nodes.

Detecting Template Hot Spots

You can use the Pipeline Profiler to identify hot spots in templates, too. In the Template section of the Profiler report, select the templates with the highest average times to see their local includes.



Search

Overall Pipeline Performance (including pipelet and template run time)

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Min
Search	Show	8	14,832	1,854	

Sub-pipelines called directly from pipeline Search

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Mi

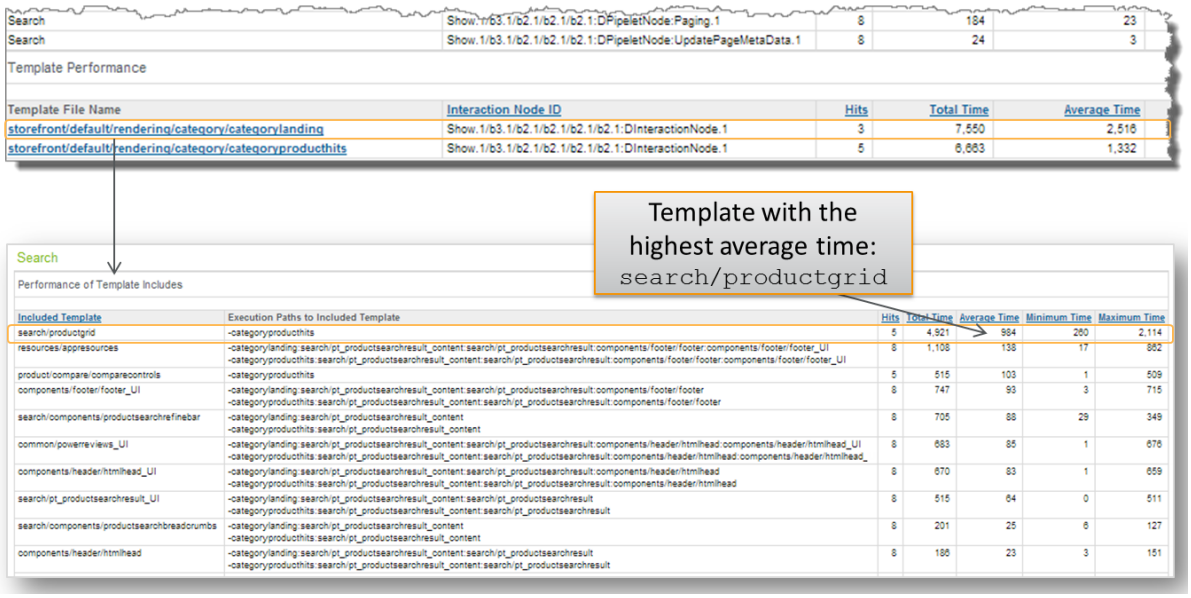
Pipelet Performance

Pipeline Name	Pipelet Node ID	Hits	Total Time	Average Time	Mi
Search	Show.1:DPipeletNode:SearchRedirectURL.1	8	5	0	
Search	Show.1/b3.1:DPipeletNode:Search.1	8	337	42	
Search	Show.1/b3.1/b2.1/b2.1/b2.1:DPipeletNode:Paging.1	8	184	23	
Search	Show.1/b3.1/b2.1/b2.1/b2.1:DPipeletNode:UpdatePageMetaData.1	8	24	3	

Template Performance

Template File Name	Interaction Node ID	Hits	Total Time	Average Time	Min
storefront/default/rendering/category/categorylanding	Show.1/b3.1/b2.1/b2.1/b2.1/b2.1:DInteractionNode.1	3	7,550	2,516	
storefront/default/rendering/category/categoryproducthits	Show.1/b3.1/b2.1/b2.1/b2.1/b2.1:DInteractionNode.1	5	6,663	1,332	

You can view local template includes to find template hot spots:



However, you can't drill into included templates to analyze their performance.

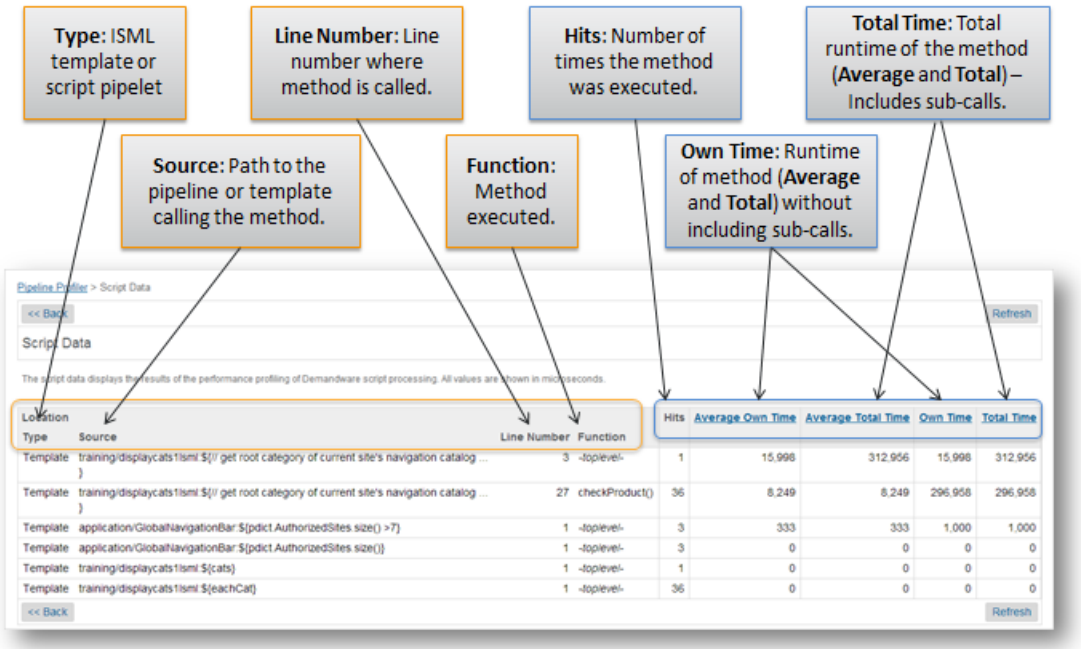
Detecting Script Hot Spots

In addition to pipelines and templates, you can use the Pipeline Profiler to identify hot spots in DW scripts.

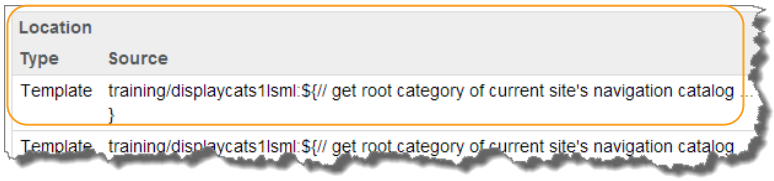
To analyze script data, select **Administration > Operations > Pipeline Profiler** and click the green start button to activate the pipeline profiler. Run the pipeline containing the script. On the Pipeline Profiler page, click **Reset**. Under **Browse Captured Script Data**, click the **Script Data** link.

If the Pipeline Profiler indicates that it has run for the maximum allowed duration, click **Reset Sensors**, rerun the pipeline, and click **Refresh**.

The Pipeline Profiler displays script data for all methods the pipeline calls, as shown in this example:



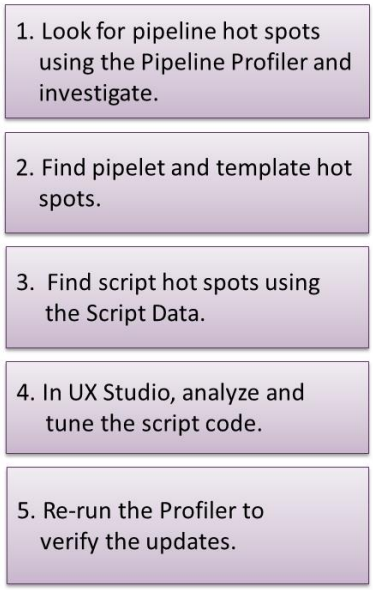
The **Location** in the Script Data report shows the template or script that calls the method.



Important: The Profiler analyzes down to the function-level. The Profiler cannot detect issues within a function—if you have an expensive loop within a function, the Profiler can't detect it.

Troubleshooting Performance Workflow

The process of troubleshooting a slow site using the Pipeline Profiler includes these steps:



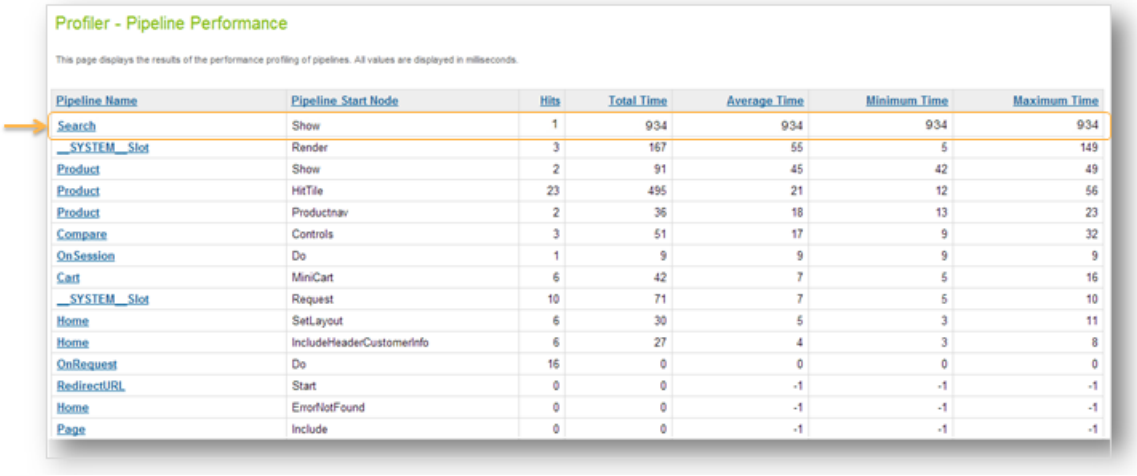
You'll follow these general steps to troubleshoot during development:

1. Use the Pipeline Profiler to look for pipeline hot spots—frequently-used pipelines, slow pipelines, or both—and select a pipeline to investigate.
2. Compare the performance of the pipeline's pipelets and templates to find pipeline hot spots.
3. Analyze the Pipeline Profiler Script Data to find script hot spots.
4. In UX Studio, analyze and tune the script code using the tips you learned in the "Optimizing Performance during Development" module.
5. Reset the Pipeline Profiler sensors, run the pipeline, and check the Profiler again to compare the performance of the updated script against the previous version.

1. Look for Pipeline Hot Spots and Investigate

To analyze pipelines, select **Administration > Operations > Pipeline Profiler** and click the green start button to activate the pipeline profiler. On the storefront site, run the pipeline and view the results in the pipeline profiler.

In this example, the `Search-Show` pipeline has the highest average runtime.



Profiler - Pipeline Performance

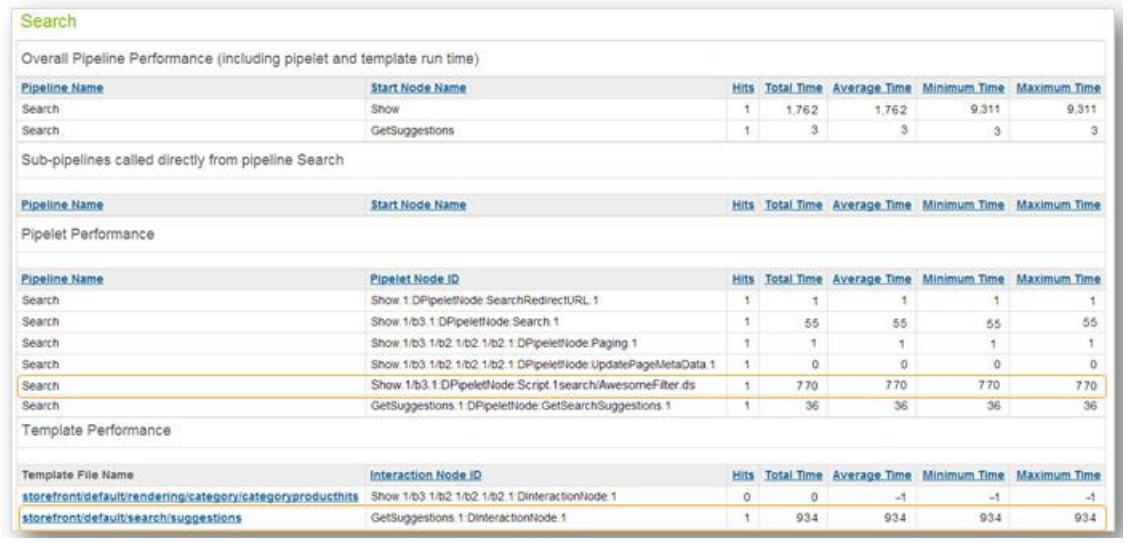
This page displays the results of the performance profiling of pipelines. All values are displayed in milliseconds.

Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Search	Show	1	934	934	934	934
__SYSTEM__Slot	Render	3	167	55	5	149
Product	Show	2	91	45	42	49
Product	HitTile	23	495	21	12	56
Product	Productnav	2	36	18	13	23
Compare	Controls	3	51	17	9	32
OnSession	Do	1	9	9	9	9
Cart	MiniCart	6	42	7	5	16
__SYSTEM__Slot	Request	10	71	7	5	10
Home	SetLayout	6	30	5	3	11
Home	IncludeHeaderCustomerInfo	6	27	4	3	8
OnRequest	Do	16	0	0	0	0
RedirectURL	Start	0	0	-1	-1	-1
Home	ErrorNotFound	0	0	-1	-1	-1
Page	Include	0	0	-1	-1	-1

2. Find Pipelet and Template Hot Spots

Within the `Search` pipeline in this example, the pipelet script `AwesomeFilter.ds` has an average time of 770 ms. The `AwesomeFilter.ds` is an inefficient search script.

The `GetSuggestions.isml` template has an average time of 934 ms, which is also not efficient.



Search

Overall Pipeline Performance (including pipelet and template run time)

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Search	Show	1	1,762	1,762	9,311	9,311
Search	GetSuggestions	1	3	3	3	3

Sub-pipelines called directly from pipeline Search

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time

Pipelet Performance

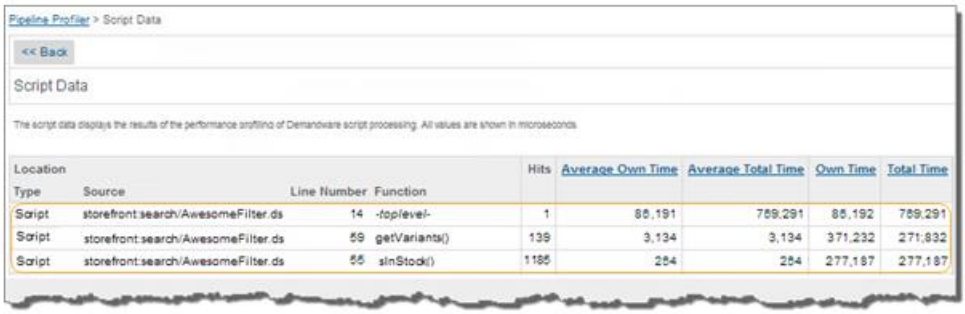
Pipeline Name	Pipelet Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Search	Show.1:DPipeletNode:SearchRedirectURL.1	1	1	1	1	1
Search	Show.1/b3.1:DPipeletNode:Search.1	1	55	55	55	55
Search	Show.1/b3.1/b2.1/b2.1/b2.1:DPipeletNode:Paging.1	1	1	1	1	1
Search	Show.1/b3.1/b2.1/b2.1/b2.1:DPipeletNode:UpdatePageMetaData.1	1	0	0	0	0
Search	Show.1/b3.1:DPipeletNode:Script.1search/AwesomeFilter.ds	1	770	770	770	770
Search	GetSuggestions.1:DPipeletNode:GetSearchSuggestions.1	1	36	36	36	36

Template Performance

Template File Name	Interaction Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
storefront/default/rendering/category/categoryproducthits	Show.1/b3.1/b2.1/b2.1/b2.1:DInteractionNode.1	0	0	-1	-1	-1
storefront/default/search/suggestions	GetSuggestions.1:DInteractionNode.1	1	934	934	934	934

3. Find Script Hot Spots

To access the script data, under **Browse Captured Script Data** on the Pipeline Profiler page, click the **Script Data** link. Click the **Average Total Time** to sort the results based on total time, in descending order.


Pipeline Profiler > Script Data

<< Back

Script Data

The script data displays the results of the performance profiling of Demandware script processing. All values are shown in microseconds.

Location				Hits	Average Own Time	Average Total Time	Own Time	Total Time
Type	Source	Line Number	Function					
Script	storefront:search/AwesomeFilter.ds	14	-toplevel-	1	86,191	769,291	86,192	769,291
Script	storefront:search/AwesomeFilter.ds	59	getVariants()	139	3,134	3,134	371,232	271,832
Script	storefront:search/AwesomeFilter.ds	56	isInStock()	1185	284	284	277,187	277,187

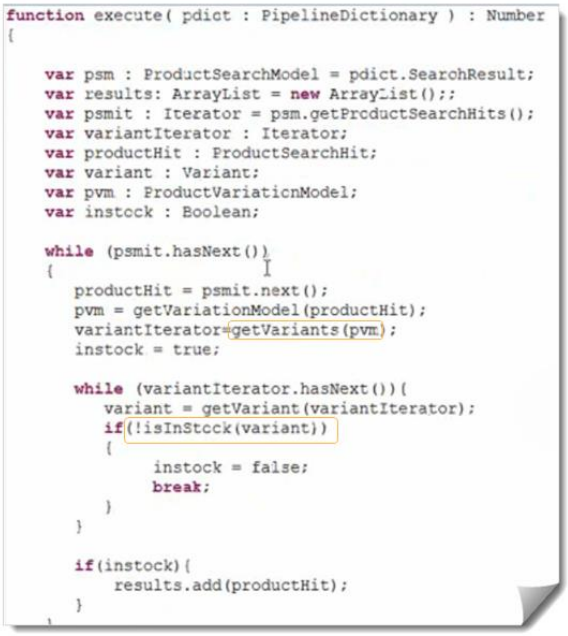
In this Script Data report:

- The main `AwesomeFilter.ds` (*toplevel*) script runs for 769 ms but only uses 86 ms of that itself. (**Note**: the script data is shown in µs.)
- The `getVariants()` method runs for 272 ms of those 769 ms.
  It's called 139 times.
- The `isInstock()` method uses 277 ms. It's called 1185 times.

The `getVariants()` and `isInstock()` methods are causing the bottleneck, so next we inspect the `AwesomeFilter.ds` script to investigate these calls.

4. In UX Studio, Analyze and Tune the Script Code

The `AwesomeFilter.ds` script uses efficient Search APIs to access the search index rather than the database. However, for each search hit, the script gets all variants, creates a database object for each, then checks that each is in stock. The `getVariants()` and `isInstock()` methods are inefficient because they need to access the database. Post-processing search results causes the `Search-Show` pipeline's sluggish performance.



```
function execute( pdict : PipelineDictionary ) : Number
{

    var psm : ProductSearchModel = pdict.SearchResult;
    var results: ArrayList = new ArrayList();;
    var psmit : Iterator = psm.getProductSearchHits();
    var variantIterator : Iterator;
    var productHit : ProductSearchHit;
    var variant : Variant;
    var pvm : ProductVariationModel;
    var instock : Boolean;

    while (psmit.hasNext())
    {
       productHit = psmit.next();
       pvm = getVariationModel(productHit);
       variantIterator=getVariants(pvm);
       instock = true;

       while (variantIterator.hasNext()){
          variant = getVariant(variantIterator);
          if(!isInStock(variant))
          {
               instock = false;
               break;
          }
       }

       if(instock){
           results.add(productHit);
       }
```

Next, you'll update the script to resolve performance issues.

5. Re-Run the Profiler to Verify the Updated Script

Once you've updated your code, you'll run the Pipeline Profiler again on your sandbox to check the performance of your updated pipeline. To ensure that you're not measuring compilation times, run the code for a while and clear the report by clicking **Reset Sensors** on the Pipeline Profiler page. Then you can start testing the pipeline's performance.

Be sure to test your pipelines with a number of different masters and variants. Sometimes a particular result is cached, which skews the pipeline performance reporting.

Exercise: Diagnose Performance Using the Pipeline Profiler

In this exercise, you'll analyze pipeline performance using the Pipeline Profiler.

1. In Business Manager, select **Administration > Operations > Pipeline Profiler**.
2. Click the start button to activate the Profiler.

   The stop button displays while the profiler captures performance data.
3. On your sandbox storefront, trigger pipelines by navigating through categories, for example, **Mens > Clothing > Dress Shirts**.

   You can trigger other pipelines by performing other operations on your storefront.
4. Return to the **Administration > Operations > Pipeline Profiler** page and click the stop button to stop profiling.
5. Under **Browse Captured Pipeline Data**, click your storefront site link, **Sites-SiteGenesis-Site** if you're using SiteGenesis.
6. Click the **Average Time** header link twice to list the average pipeline runtimes in descending order.

   Which pipeline has the highest average time?

   _____

   Which pipeline has the most hits?

   _____

Exercise: Compare Script and Pipeline Performance with the Profiler

In this exercise, you'll use the Pipeline Profiler to compare the two scripts shown in the previous lesson.

The scripts and templates you need for the exercise are included in the `Developing for Performance and Scalability Exercises.zip` file provided by your instructor.

Note: If you are taking the eLearning version of the course, the zip file is on the **Course Materials** tab.

Import the Exercise Cartridge

The exercises cartridge contains the scripts you'll use in this exercise and subsequent exercises in the course. Follow these steps to import the cartridge:

1. Unzip the `Developing for Performance and Scalability Exercises.zip` file provided by your instructor.
2. In UX Studio, import the `trainingPerformance` cartridge by selecting **File > Import**.

3. Select **Existing Projects into Workspace** and click **Next**.
4. With **Select root directory** enabled, click **Browse** and navigate to the `Developing for Performance and Scalability Exercises/trainingPerformance` folder.

   The `trainingPerformance` cartridge is displayed under Projects.
5. Ensure that the `trainingPerformance` cartridge is enabled and click **Finish**.
6. If prompted to link the training cartridge to the Demandware Server, click **Yes**.
7. In Business Manager:
   a. Add the `trainingPerformance` cartridge to the cartridge path by navigating to **Administration > Sites > Manage Sites** and selecting your site. Select the **Settings** tab, add `trainingPerformance` to the **Cartridges** path, and click **Apply**.
   b. Select the **Cache** tab and make sure the **Enable Page Caching** option is disabled. Invalidate the page cache if necessary.

Run the `DisplayCats1` Pipeline

In this exercise, you'll run the inefficient `DisplayCats1` pipeline that calls `dw.catalog.CatalogMgr.getSiteCatalog()` to access the database. It then post-processes the categories and products one by one.

1. In the `trainingPerformance` cartridge, open the `DisplayCats1` pipeline in UX Studio.

   The `DisplayCats1` pipeline calls the `displayCats1.ds` script which obtains and post-processes category and product data. The `displayCats1.isml` template prints the categories that contain online products.
2. Run the `DisplayCats1.xml` pipeline on your storefront site, for example:

   ```
   https://student01.training-
   na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-
   Site/default/DisplayCats1-Start
   ```

   The pipeline prints the results as follows:


category 1 : newarrivals-electronics
category 2 : newarrivals
category 3 : womens-outfits
category 4 : womens-clothing
category 5 : womens
category 6 : womens-clothing-tops
category 7 : womens-clothing-dresses
category 8 : womens-clothing-bottoms
category 9 : womens-clothing-jackets
category 10 : womens-clothing-feeling-red
category 11 : womens-jewelry-earrings
category 12 : womens-jewelry
category 13 : womens-jewlery-bracelets
category 14 : womens-jewelry-necklaces
category 15 : womens-accessories-scarves
category 16 : womens-accessories
category 17 : womens-accessories-shoes
category 18 : mens-clothing-suits
category 19 : mens-clothing
category 20 : mens
category 21 : mens-clothing-jackets
category 22 : mens-clothing-dress-shirts
category 23 : mens-clothing-shorts
category 24 : mens-clothing-pants
category 25 : mens-accessories-ties
category 26 : mens-accessories

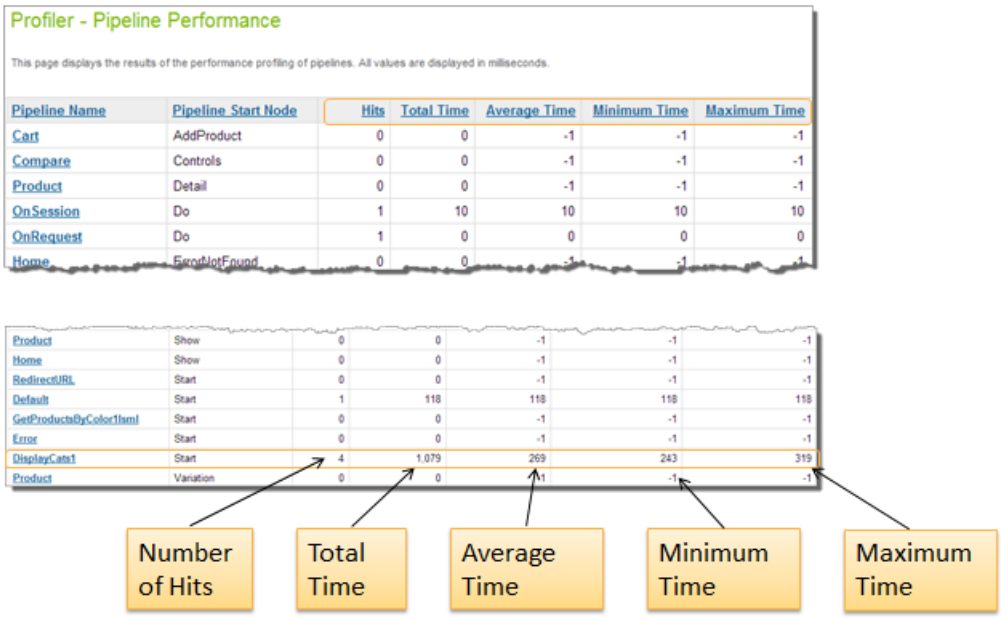
Analyze the `DisplayCats1` Pipeline and Script

Now you'll use the Pipeline Profiler to analyze the `DisplayCats1` pipeline and corresponding script.

1. In Business Manager, select **Administration > Operations > Pipeline Profiler**.
2. In the Pipeline Profiler, click the start button to activate the profiler.

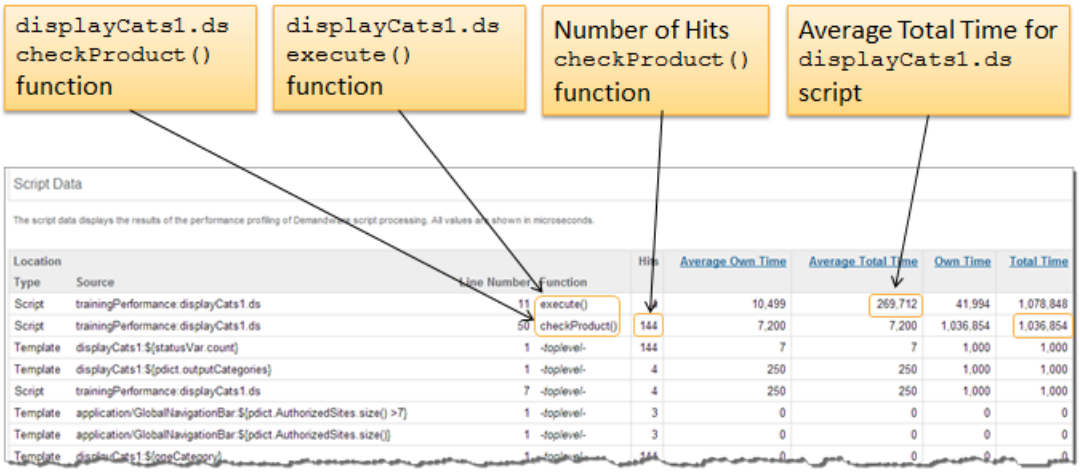
   The stop button displays while the profiler captures performance data.
3. Run your `DisplayCats1` pipeline a few times now that the profiler is capturing data.
4. Under **Browse Captured Pipeline Data**, click your storefront site link, for example, **Sites-SiteGenesis-Site**.

   In the Profiler-Pipeline Performance page, you'll see the profiling results of the `DisplayCats1` pipeline. The results include the number of hits, the total, average, minimum, and maximum time, for example:



5. Under **Browse Captured Script Data**, click **Script Data**.

   The script data shows the pipelines, scripts, and templates used on the instance. Note the average time for the script. Note also the number of hits on the `checkProduct()` method and the total time it uses. Running the `checkProduct()` method on all categories is expensive.



Run the DisplayCats2 Pipeline

In this exercise, you'll run the more efficient `DisplayCats2` pipeline that uses the Search API to find categories with orderable products.

1. In the `trainingPerformance` cartridge, open the `DisplayCats2.xml` pipeline in UX Studio.

   The `DisplayCats2.xml` pipeline calls the `displayCats2.ds` script and the `displayCats2.isml` template which prints the categories that contain online products.

2. Run the `DisplayCats2.xml` pipeline on your storefront site, adding a search parameter, for example, q=shirts:

   ```
   https://student01.training-
   na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-
   Site/default/DisplayCats2-Start?q=shirts
   ```

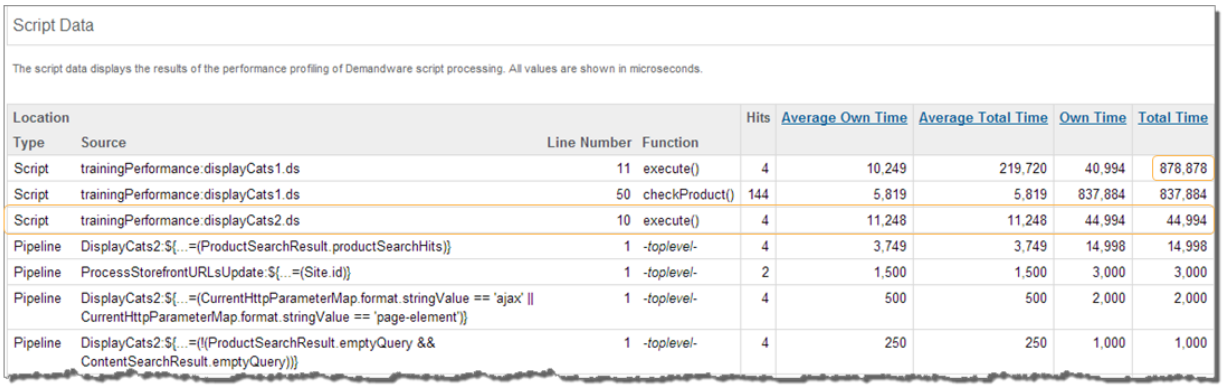
   The search parameter causes the platform to create a search index.

Analyze the DisplayCats2 Pipeline and Script

Now you'll use the Pipeline Profiler to analyze the `DisplayCat2` pipeline and corresponding script.

1. In Business Manager, select **Administration > Operations > Pipeline Profiler**.
2. In the Pipeline Profiler, click **Reset Sensors** or if the profiler is not capturing data, click the start button to activate the profiler.
3. Run your `DisplayCats2` pipeline a few times using q=shirts as the query:

   https://student01.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/DisplayCats2-Start?q=shirts
4. Under **Browse Captured Script Data**, click **Script Data**.



Script Data

The script data displays the results of the performance profiling of Demandware script processing. All values are shown in microseconds.

Location Type	Source	Line Number	Function	Hits	Average Own Time	Average Total Time	Own Time	Total Time
Script	trainingPerformance:displayCats1.ds	11	execute()	4	10,249	219,720	40,994	878,878
Script	trainingPerformance:displayCats1.ds	50	checkProduct()	144	5,819	5,819	837,884	837,884
Script	trainingPerformance:displayCats2.ds	10	execute()	4	11,248	11,248	44,994	44,994
Pipeline	DisplayCats2:${...=(ProductSearchResult.productSearchHits)}	1	-toplevel-	4	3,749	3,749	14,998	14,998
Pipeline	ProcessStorefrontURLsUpdate:${...=(Site.id)}	1	-toplevel-	2	1,500	1,500	3,000	3,000
Pipeline	DisplayCats2:${...=(CurrentHttpParameterMap.format.stringValue == 'ajax' || CurrentHttpParameterMap.format.stringValue == 'page-element')}	1	-toplevel-	4	500	500	2,000	2,000
Pipeline	DisplayCats2:${...=(!(ProductSearchResult.emptyQuery && ContentSearchResult.emptyQuery))}	1	-toplevel-	4	250	250	1,000	1,000

The `DisplayCats2` pipeline is much more efficient than the `DisplayCats1` pipeline. The `DisplayCats1` pipeline instantiates and post-processes all products in the catalog. The `DisplayCats2` pipeline accesses the pre-built search index with orderable data, which is much faster than accessing the database for every product.

You've finished the exercises in this section.

Lesson 4.2: Diagnosing Performance with Business Manager Analytics

Business Manager Analytics lets you run reports against the data the Demandware platform collects. Use the Analytics reports to diagnose performance issues in production, but not on sandboxes or development instances.

The Demandware operations team must enable Demandware Analytics on your instance for you to access the reports. If enabled, Demandware runs the Analytics reports once a day. The reports are based on data from the web server log files of the Demandware web tier.

Note: Although Analytics reports are not available generally on sandbox instances, the sandboxes do have sample Analytics reports for the year 2012. You can access this data from your sandbox to familiarize yourself with the reports.

The Analytics module generates reports for merchants, as well as developers, operations personnel, and administrators. These reports help to diagnose production site performance issues:

- **Object Churn Trends Reports** provide data about objects created, updated, and deleted. You can review trends for sessions, baskets, orders, products, wish lists, and customers. See the "Optimizing Performance during Development" module for details on Object Churn Trends reports.
- **Traffic Reports** provide statistics on site traffic, for example, the frequency of visits and requests, as well as the runtimes of requests and visit durations.
- **Technical Reports** provides pipeline details.

Depending on the report type, you can generate daily, weekly, monthly, quarterly, or yearly reports. You can export reports to XLS files for further analysis. Be sure to check the Business Manager Analytics reports frequently, especially after code deployments.

Lesson 4.3: Diagnosing Performance with Traffic Reports

Use the Analytics Traffic Reports to review site traffic statistics for site traffic.

To view the analytics traffic reports, in Business Manager, select **Site > Analytics > Traffic Reports**. Select one of the report types. Analytics are supported only on production instances.

These reports answer questions such as:

- *How many visitors enter the site during particular time periods?*
- *How long do visitors stay?*
- *What are the top pages for the site?*
- *How many requests are handled day to day?*
- *Where is the traffic to the website coming from? Is it coming from a competitor's site?*
- *Are there caching issues on the site?*
- *Are there negative performance trends that correspond to code releases?*

Best Practices

Select the Top Pages report on the Traffic Reports page to list the top 100,000 URLs used in a day.

For the top pages:

- Check that each of the top pages is being cached.
- Look for ways to improve the scripts used on the page.
- Look for ways to optimize the images and the JavaScript on the page.

Lesson 4.4: Diagnosing Performance with Technical Reports

You can review pipeline performance data on production instances using the Analytics Technical Reports. In addition to drilling into pipeline data, the Technical Reports let you analyze pipeline includes and cache hit ratios, as well.

To view the analytics technical reports, in Business Manager, select **Site > Analytics > Technical Reports**. Select one of the report types. Analytics are supported only on production and load testing instances.

The Analytics Technical Reports answers questions such as:

- *How many times were the pipelines called in the requested time period?*
- *How many pipelines were retrieved from cache?*
- *How many pipelines were stored in cache, but never retrieved?*
- *How many pipelines were not cached?*
- *How many incoming requests were made to the site?*

For each pipeline, you can view:

- The Call Count: The number of times each pipeline is called.
- The Processing Time: The time it takes to process pipelines and their includes. Processing time does not include network overhead and latency during response delivery. Check that heavily used pipelines are optimized. For example, the `Product-Show` pipeline should be below the average of 320 ms and the `Search-Show` pipeline should be below the average of 400 ms. You can see the processing time of just the pipeline and not its includes by looking at the **Avg Own** column in the Pipeline Performance report.

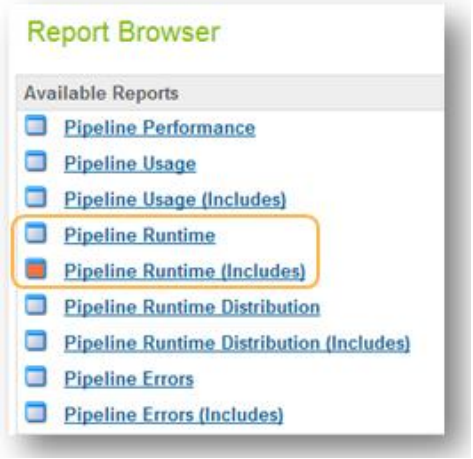
- The Request Time: The time it takes for the server to deliver the response. The request time includes the processing time plus the time needed to assemble the request by pulling together the cached snippets from the web adaptor. The request time includes the time requests are waiting on the web adaptor or the application server, so response times are affected by server traffic.
- The Cache Hit Ratio: The percentage of pipeline requests: those retrieved from cached, stored in cache but never retrieved, or never stored in cache.
- Errors: List of errors that occurred during the reporting period. For each error, the reports include the pipeline name, the error count, and the error codes.

Using Technical Reports to Analyze Page Caching Issues

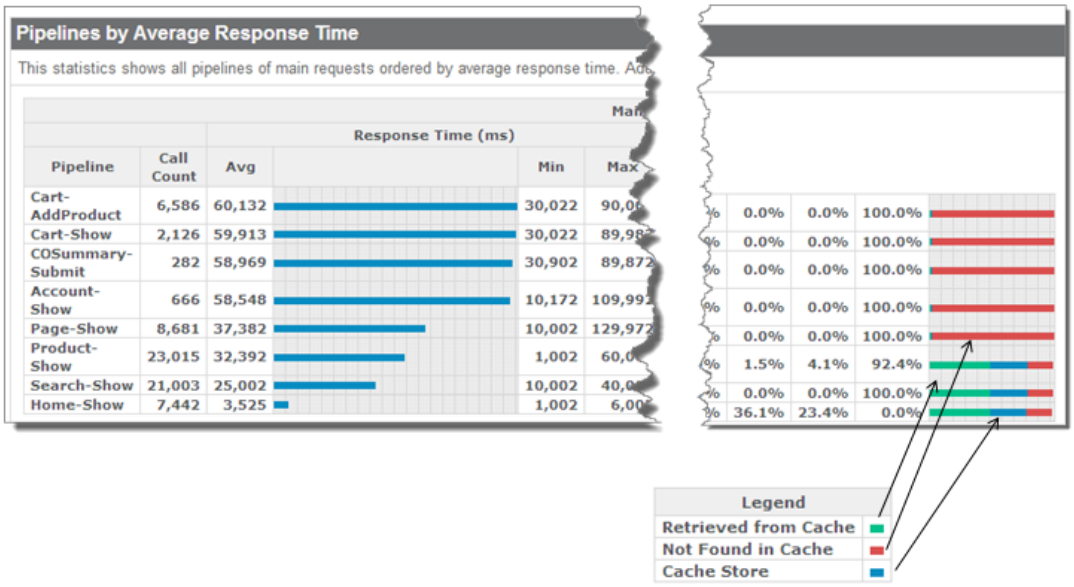
Demandware provides many different reports and analytics tools to help you diagnose performance issues. To review caching behavior on production instances, you can use the Report Browser available in the Business Manager Analytics module (only available on production instances).

To analyze page caching issues on production, in Business Manager, select your site and select **Site > Analytics > Technical Reports**.

Select one of the pipeline reports. Each type of report has a version for includes, as well, for example, there is a **Pipeline Runtime** report for pipelines and a **Pipeline Runtime (Includes)** report for remote includes in the pipeline.



In the Pipeline Runtime report, you can see the caching behavior per pipeline:



Retrieved from Cache: The green bar shows the percentage of requests that the web server retrieves from the page cache. The goal is to maximize this value to keep all transactions on the web tier.

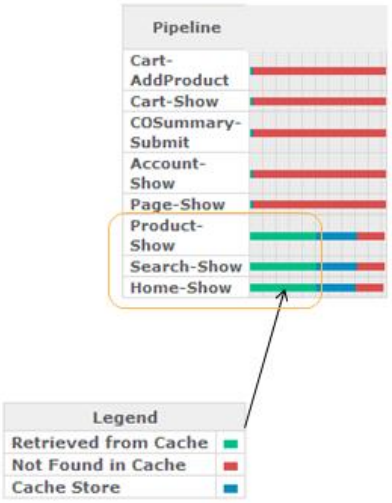
Not Found in Cache: The red bar shows requests that were not retrieved from cache. If a template does not have the `<iscache>` tag, the content is not cached. If a template does the have the <iscache> tag but the cache has expired, the page needs to be regenerated.

For the percentage of cached pipelines and includes, combine the green bar (retrieved from cache) and the blue bar (cache store).

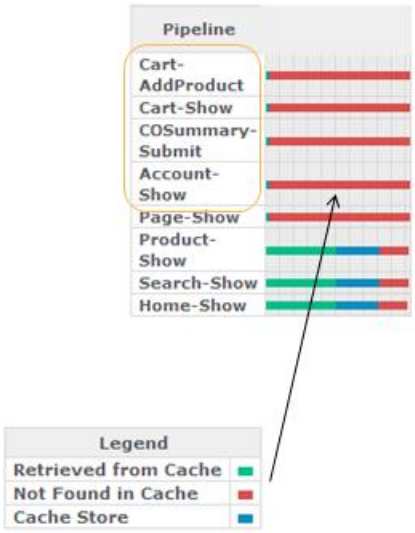
Cache Store: The blue bar shows the percentage of requests that were cached but never retrieved again within the caching time. Caching these requests is wasteful since they're never accessed again. To reduce the percentage of the requests stored in cache and not retrieved:

- Analyze pipelines and includes with high Cache Store percentages to determine whether it makes sense to cache them. Sometimes instead of trying to increase the cache hit ratio, you should tune the performance of the pipeline or include code and avoid caching the page or partial page.
- If it makes sense to cache these pipelines and include responses, you might need to lengthen the caching time.
- Use methods that increase the likelihood that cached pages and partial pages are reused. For example, ensure that URLs are reused by avoiding varied parameters that cause URL mismatches. Another method is to reuse snippets and templates for similar behaviors. For example, if you're displaying the same data using a gallery or a listing, create and cache the snippet and template once, then change the styling using JavaScript and style sheets.

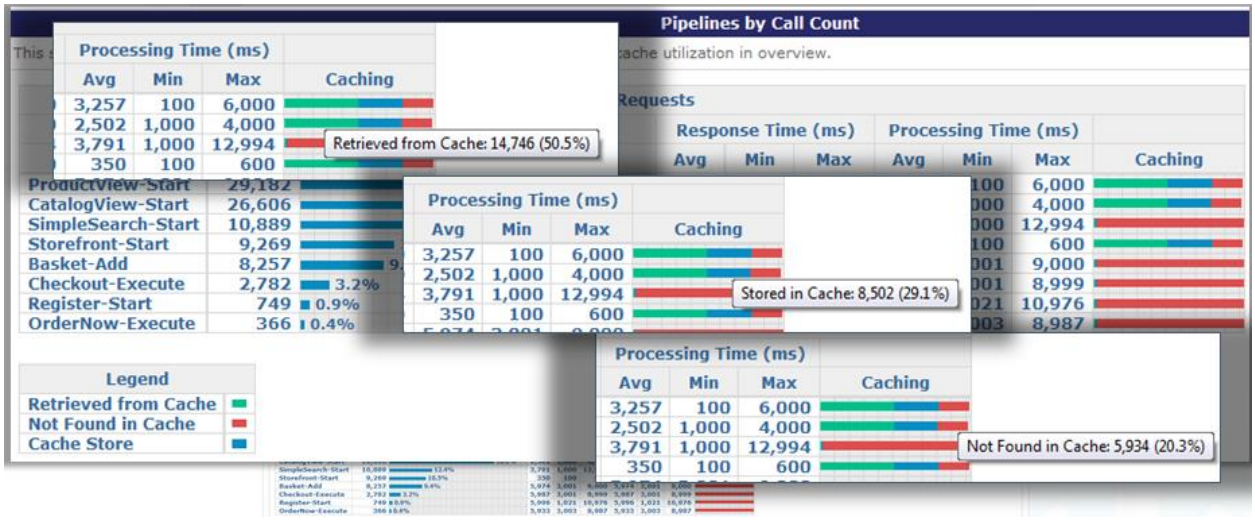
In this example, the `Product-Show`, `Search-Show` and `Home-Show` pipelines are cached. All customers use these pages.



On the other hand, pipelines related to the cart and user accounts are not cached since they're customer-specific.



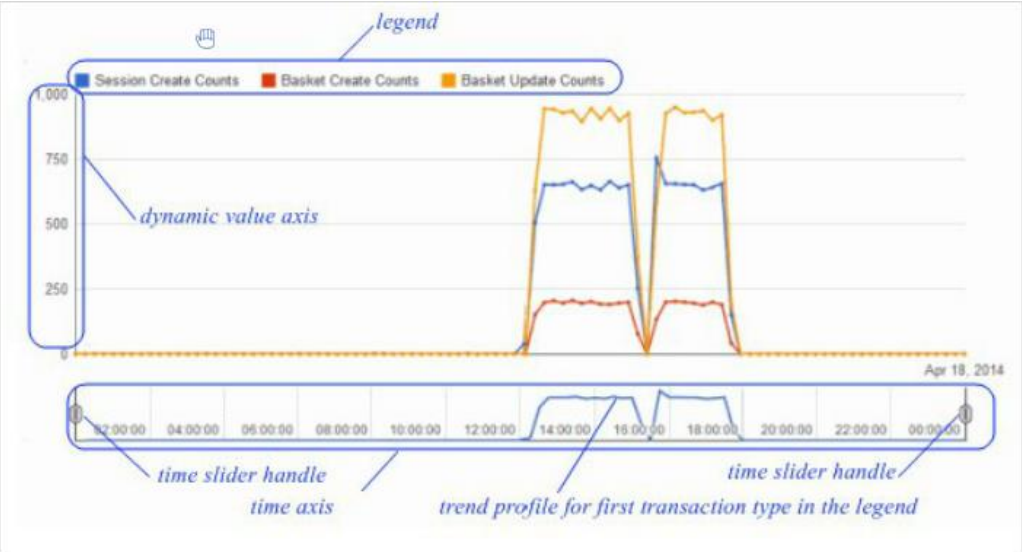
If you move your mouse over the cache status bar, you can see the actual hit count for the requests retrieved from cache (green bar), the requests stored in cache but not retrieved (blue bar), and the requests that were not found in cache (red bar):



Lesson 4.5: Diagnosing Performance with Object Churn Trends

The Object Churn Trends report shows data for the last 24 hours for all sites on an instance, rather than just a single site. Like the rest of the Business Manager Analytics, the Object Churn Trends report is generated on production instances only.

To analyze database performance, in Business Manager, select **Site > Analytics > Object Churn Trends** and choose the object types you're investigating: sessions, baskets, orders, products, wish lists, or customers.



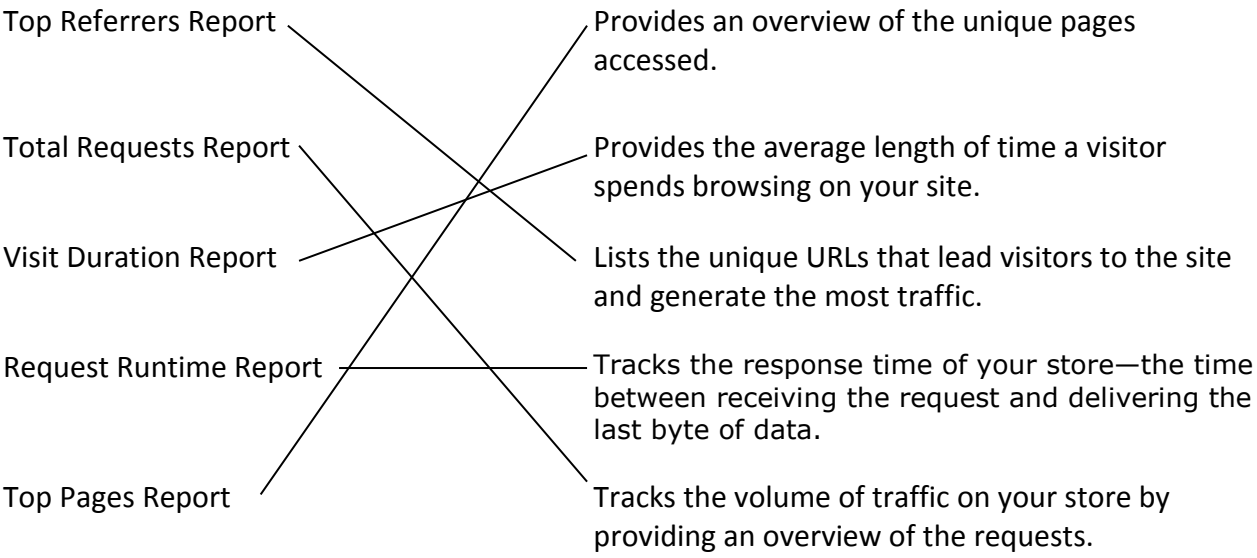
The Object Churn Trends report shows the numbers of object creations, updates, and deletes in a particular time period. Use the vertical time slider to view the time period. You can view daily, weekly, monthly, quarterly, yearly, or real time trends (the past hour).

Exercise: Review Traffic Reports

Review the documentation (https://info.demandware.com) and *match the reports on the left with their contents on the right*. *(Answers on following page)*

Top Referrers Report	Provides an overview of the unique pages accessed.
Total Requests Report	Provides the average length of time a visitor spends browsing on your site.
Visit Duration Report	Lists the unique URLs that lead visitors to the site and generate the most traffic.
Request Runtime Report	Tracks the response time of your store—the time between receiving the request and delivering the last byte of data.
Top Pages Report	Tracks the volume of traffic on your store by providing an overview of the requests.

Exercise Answers: Review Traffic Reports



Exercise: Locate Usage Data in Traffic Reports

In this exercise, you'll use the Report Browser to review caching behavior by pipeline.

1. In Business Manager, select your site and select **Site > Analytics > Traffic Reports**.
2. Answer the following questions about Traffic Reports.

   Note: Although Analytics reports are not available generally on sandbox instances, the sandboxes do have sample Analytics reports for the year 2012.

   What was the maximum number of requests during the month of August in 2012?

   _____

   On which day was the longest visit during time span, 8/12/12 – 8/18/12?

   _____

   On February 13, 2012, of the average request runtimes, when was the longest and how many seconds was it?

   _____

   Which date in December, 2012 had the most visits to the site?

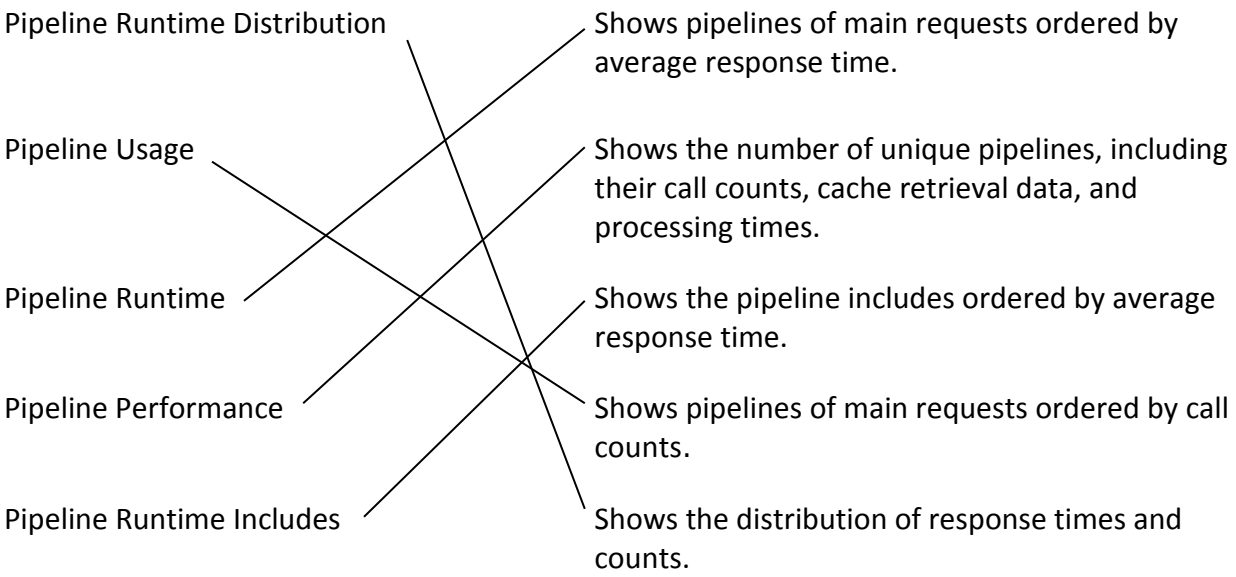
   _____

Exercise: Review Technical Reports

Review the documentation (https://info.demandware.com) and *match the reports on the left with their contents on the right*. *(Answers on following page)*

Pipeline Runtime Distribution	Shows pipelines of main requests ordered by average response time.
Pipeline Usage	Shows the number of unique pipelines, including their call counts, cache retrieval data, and processing times.
Pipeline Runtime	Shows the pipeline includes ordered by average response time.
Pipeline Performance	Shows pipelines of main requests ordered by call counts.
Pipeline Runtime Includes	Shows the distribution of response times and counts.

Exercise Answers: Review Technical Reports



Exercise: Locate Pipeline Data in Technical Reports

In this exercise, you'll use the Report Browser to review pipeline data.

1. In Business Manager, select your site and select **Site > Analytics > Technical Reports**.
2. Answer the following questions about Technical Reports.

   Note: Although Analytics reports are not available generally on sandbox instances, the sandboxes do have sample Analytics reports for the entire year or 2012.

   Which pipeline was called the most?

   _____

   Which pipeline had the shortest response time?

   _____

   What percentage of the main request pipelines were retrieved from cache?

   _____

   What percentage of main request pipelines were stored in cache but never retrieved?

   _____

   What percentage of main request pipelines were not found in cache?

   _____

   Were there errors in any of the pipelines and if so, what were the error codes?

   _____

Exercise: Analyze Caching Behavior

In this exercise, you'll use the Report Browser to review caching behavior by pipeline.

1. In Business Manager, select your site and select **Site > Analytics > Technical Reports**.
2. Select a date or range of dates to filter the technical reports.

   Note: Although Analytics reports are not available generally on sandbox instances, the sandboxes do have sample Analytics reports for the entire year or 2012. Select a date in 2012 to view sample reports.

3. Select a tab to choose the frequency of reporting—**Daily Report**, **Weekly Report**, **Monthly Report**, **Quarterly Report**, or **Yearly Report**.
4. In the Report Browser, select the **Pipeline Runtime** report.

   Which pipeline has the longest response time?

   _____

5. In the Report Browser, select the **Pipeline Performance** report.

   What percentage of requests were cached?

   _____

   What percentage of requests were not found in cache?

   _____

You've finished the exercises in this section.

Best Practices: Tips for Troubleshooting Performance on Production

To troubleshoot your site, use Business Manager Analytics to rank the top five slowest or most heavily used pipelines on production. These are the pipelines you should investigate for potential optimizations. Once you determine the top five pipelines, you can use the Pipeline Profiler to analyze them further.

Follow these tips to look for slow or heavily used pipelines:

- Use **Traffic Reports > Top Pages** to view the top 100 URLs with their parameters. Use these to rank the areas of the site with slow performance.
- Use **Technical Reports > Pipeline Performance** to analyze pipelines with high average processing times and weak cache hit ratios.
- Use **Technical Reports > Pipeline Runtime Distribution** to see another view of request times because average runtimes do not always reflect optimal runtimes; good caching can make a pipeline appear faster than it really is.
- Use **Technical Reports > Pipeline Runtime (Includes)** to determine which pipelines are calling which includes.
- Use **Object Churn Trends** to investigate database performance issues.

Module 5: Optimizing Transactional Jobs and Integrations

Learning Objectives

After completing this module, you will be able to:

- Create and schedule jobs that limit resource consumption on the platform.
- Diagnose issues with job performance.
- Use web services efficiently.

Introduction

The number and type of integrations affect the performance of your site. The data models your team develops are a key factor in optimizing storefront processing. You can make your storefront faster by developing a good data structure.

You'll need to determine the optimal way to handle each type of data being transferred. You can use:

- Storefront pipelines and scripts
- Asynchronous integration using jobs
- Synchronous integration using web services

The following table compares these methods for transferring data, as well as the timeouts for each method. The timeouts represent the amount of time the system waits for a response before closing the browser connection.

Method	Purpose	Timeouts
Storefront Pipelines	Use for immediate storefront responses.	Default timeout: 30 seconds Maximum timeout: 5 minutes To set the timeout for scripts in storefront pipelines, set the `Timeout` property of the script node in the pipeline. **Note**: The maximum debugging timeout for a script is 30 minutes. If a script stays on a breakpoint for more than 30 minutes, the script is terminated.
Jobs	Use for long-running batch operations such as imports and exports. Jobs are file-based so they can update large numbers of objects in Demandware, such as product and pricing data. Administrators can schedule jobs for opportune times when traffic on the production server is light.	Default timeout: 15 minutes Maximum timeout: 1 hour
REST-Based Web Services	Use for real-time integrations with third-party systems when you need to transfer small amounts of data at a time. Web services are less resource-intensive than jobs. Typical uses of web services include ratings and reviews, payment processing, analytics, and taxes. To obtain data from the platform, you can implement web services using Demandware's Open Commerce APIs and you can develop your own custom web services, as well.	Recommended timeout: 5 seconds The average response time should be beneath this threshold. Use the Services Dashboard in the Web Service Framework to set the timeout value for each web service—the time the server will wait for an answer from the calling service. You can also use the `dw.net.HTTPClient setTimeout()` and `getTimeout()` methods to work with REST-based web service timeouts.

Lesson 5.1: Optimizing Job Performance

For asynchronous integrations you'll set up jobs using the Integration Framework. Asynchronous integrations let you import and export large amounts of data. These jobs can be resource-intensive, so this lesson provides tips to optimize job performance.

Exercise: Job Optimization Benefits

Review the tips for optimizing job performance in the following table. Then complete the table that follows.

Tips	Explanation
Schedule jobs strategically.	Large import or export jobs use a great deal of system resources, so schedule your jobs strategically. Schedule jobs for non-peak hours and disperse job start times to balance the job load. Ideally, try to stagger job start times so that only one job at a time runs on an instance group. As product imports are time and resource intensive, be strategic about their use, especially if you have multiple sites. Try to leverage synergies between sites. Not every site needs to import its own catalog, price book, and inventory. When you import products, consolidate data during transformation, so that import jobs touch each changed product only once. In general, limit the frequency of imports, exports, and search index builds. Each of these operations clears the cache which slows processing until the cache is filled again.
Set up jobs and index builds on the staging instance.	Do not perform frequent catalog imports and search index builds on the production server. To reduce resource consumption on the production server, use staging and then replicate to production—Import catalogs to staging, build the search index there, then replicate to production. Replicate catalogs, prices, and search indexes only once a day. If you need to update the availability of products regularly, you can perform incremental search index builds on production throughout the day. Some third-party integrations require imports of data directly to the production instance. As long as you limit the frequency of these jobs, they won't hinder performance. As with catalog imports, run sanity and plausibility checks on staging rather than production. Sanity and plausibility checks ensure that your

	feed entries are valid. For example, you might check for products with: ▪ A price of $0 ▪ No name ▪ Incomplete attribution ▪ Offline status Instead of running these checks on production, review catalog data on a staging instance. You can filter criteria such as availability with the `Search` pipelet. For example, you can perform a search to filter out those products that are unavailable or have a price of $0 rather than setting them to offline. Efficient platform features like the Search API can help you limit long-running batch jobs. Although it's best to import feeds onto staging, be aware that all transactional jobs on production, staging, and development use the same Oracle install and thus compete for resources on that tier. Since excessive database consumption on staging and development instances affects production instance performance, monitor aggregate database traffic across your entire ecommerce environment. When using the Site Backup feature on the staging instance, configure it to run no more than once a day and do not keep more archives than necessary (usually five is enough).
Use delta processing.	To reduce the number of objects being imported or exported, use delta feeds. Delta feeds include only those objects and attributes that have changed. For example, if a client typically sells 1000 items a day, it should not be necessary to import 100,000 inventory records every 30 minutes. You can set up a delta feed in Integration Framework by selecting the MERGE import mode. The MERGE mode is more efficient than the UPDATE mode and it creates objects if they do not exist. For best performance, avoid the REPLACE import mode as it is less efficient than MERGE and UPDATE because it deletes then recreates attributes and relationships. To prepare for delta processing, use standard Demandware import formats as defined in the Demandware object schema files. The schema governs the XML import files you'll create for each system object type. The Demandware online help (`https://info.demandware.com`) provides an Import/Export Object cheat sheet with links to the Demandware object schema files on which you'll base your XML import files.

Avoid concurrent changes to objects.	As with storefront pipelines, avoid concurrent changes to the same object. Use the Integration Framework which provides a locking mechanism to ensure exclusive access. Specify named resources for job schedules.
Optimize the memory footprint for large data sets.	Standard Demandware imports can process arbitrary feed sizes. When processing these large data sets, pay attention to the memory footprint. Design your loop logic so memory consumption doesn't increase as the result set increases. Be sure to keep only currently processed objects in memory and refrain from performing operations like sorting on collections in memory. To ensure that objects can be freed from memory, don't retain references to objects in memory. Avoid reading an entire file into memory by reading feeds one record at a time. Also, stream data to a file at regular intervals to avoid building large structures in memory. If you need to create multiple feeds, query the objects once and write records to all of the feeds as you iterate over the results. This method saves time because you only need to create the objects in memory once.
Group operations in explicit pipeline transactions.	Demandware commits changes to the database for each business object. To improve performance, commit related changes in a single transaction by enclosing the import pipelets in an explicit pipeline transaction. You must ensure that the transaction size limit of 1000 is not exceeded, so use this approach only as an exception.
Adhere to the suggested job load factor.	Keep application server utilization by jobs to a minimum. Calculate the job load factor—the total number of seconds per day of job execution time on an instance (staging or production) divided by 86,400 (the number of seconds in a day). Try to keep the job load factor below 0.2.
Develop a recovery strategy.	Develop a recovery strategy for your jobs. Jobs that end abnormally can cause server restarts or application server failures. Design each job so that it recovers automatically, for example, by recognizing which files have not yet been imported during the subsequent execution.
Clean up after import and exports.	For imports, implement proper cleanup and delete files that are older than two weeks. Compress import files after they are processed. Be sure to delete site exports when you no longer need them. To minimize the size of site exports, exclude product images.

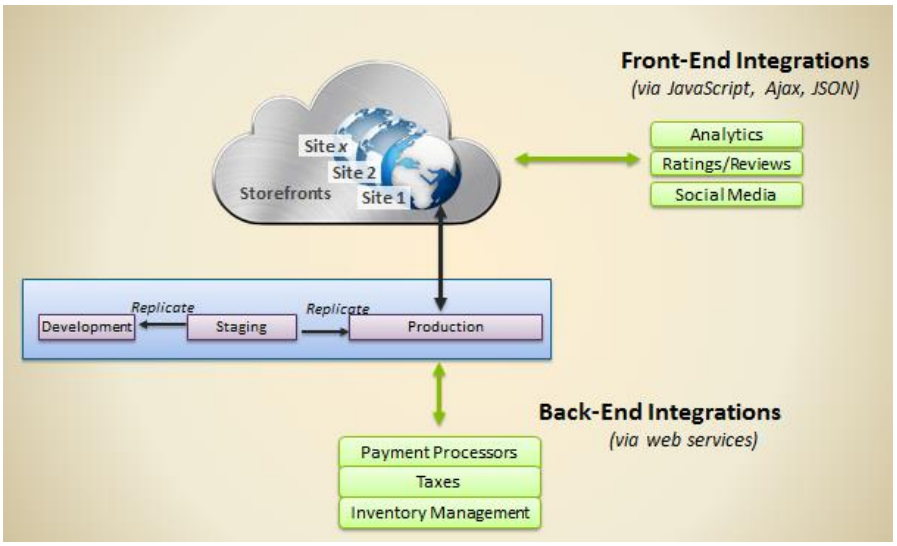
Write down the benefit(s) of each job optimization tip in the following table.

Job Optimization Tip	Benefit
Schedule jobs strategically.	
Set up feeds, search index builds, and sanity checks on the staging instance.	
Use delta processing.	
Avoid concurrent changes to objects.	
Optimize the memory footprint for large data sets.	
Group operations in explicit pipeline transactions.	
Adhere to the suggested job load factor.	
Develop a recovery strategy for jobs.	
Clean up after imports and exports.	

You've finished the exercise.

Lesson 5.2: Optimizing Synchronous Integrations

All commerce sites have external integration points, and these external integrations heavily impact site performance. Front-end integrations typically implement features like analytics, ratings, and reviews using JavaScript, Ajax, or JSON.



You learned coding tips to optimize client-side front-end integrations in the *Optimizing Performance during Development* module. This lesson focuses on optimizing back-end integrations. On the back-end, developers typically implement synchronous integrations using web services. Examples of back-end integrations include payment processors, tax calculation software, and inventory management systems.

To optimize the performance of synchronous back-end integrations:

- Specify proper timeouts for back-end services.
- Manage unresponsive services.
- Optimize REST-based caching.

Specify Proper Timeouts for Backend Services

Ensure the stability of integrations by setting low timeout values for service calls. If your timeouts are set too high, your site can be bogged down by slow or unresponsive web services. Each request to the unresponsive service creates an application thread that has to wait for the service. If all the threads provided by the storefront site are exhausted, the site can no longer process traffic.

Ideally, your timeouts should be one second. If you use web services for checkout, you might need a timeout of 5-10 seconds. For recommendations and availability, the timeout should be very low. Use

creative approaches to deal with timeouts. For example, if a recommendation request times out, a good strategy is to display random products.

Note: When setting timeouts for web service calls, set the timeout on the object itself, for example, `httpClient.setTimeout()`, as well as in the pipelet properties.

To investigate timeouts, use error logs, the Pipeline Profiler, or Business Manager Analytics. The error logs list the web service timeouts. The Pipeline Profiler and Business Manager Analytics identify long or unresponsive web services. Use these tools to investigate which pipelines are using web services and are taking up a lot of time.

Manage Unresponsive Services

To ensure that unresponsive services don't block all storefront application server threads, the system prevents calls to these unresponsive services. You configure the locking strategy using the Service Framework. The Service Framework is covered in detail in the *Integrating with Demandware* course. To summarize, locking works as follows:

1. When a service is unresponsive, the system locks the service.
2. After a specified amount of time, the system tries again to access the service.
3. If the system can't access the service, it renews the lock.

If a service is locked, the system rejects all other requests for the service, thus improving performance and releasing threads. Use the Services Dashboard in the Service Framework to specify the length of time locks should last and the maximum number of service access attempts before the system locks the service.

The Service Framework helps to handle unresponsive services, but be sure to design offline processes to handle situations when one or all of your external systems have limited availability or are offline completely:

- Set up processes for monitoring integration points and switching automatically to offline processing if a service is unresponsive.
- Use data redundancy to enable the shop to respond if a web service is offline.
- Allow administrators to turn off services in the Business Manager if necessary, using a site preference.

Optimize REST-Based Caching

The best external integration is a call you can cache. Remember not to cache personal information.

Use the caching capabilities of REST-based web services. The Demandware APIs support caching of responses depending on time, usage, and size. Use the `dw.net.HTTPClient.enableCaching(ttl)` capabilities and set the caching time-to-live (TTL) to 30 minutes, for example. When you use the `enableCaching(ttl)` method, be sure that the response is independent of state or time information and depends only on the URL. For example, be careful when using session information and

personalized content. Also make sure that the application sends the exact same URL each time to take advantage of the cache.

The Open Commerce API also provides caching capabilities. You can specify a unique cache time for each cacheable resource, including products, categories, prices, promotions, and product searches.

To configure OCAPI resources, go to **Administration > Site Development > Open Commerce APIs** and enter the client-specific configuration settings.

Following is a sample OCAPI configuration (specified in JSON), which includes cache settings for each specified resource. The `cache_time` attribute is specified in seconds.

```
{
  "_v":"14.2",
  "clients":
  [
    {
      "client_id":"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
      "resources":
      [
        {
          "resource_id":"/categories/(id)",
          "read_attributes":"(**)",
          "write_attributes":"(**)",
          "cache_time":900
        },
        {
          "resource_id":"/products/(id)",
          "read_attributes":"(**)",
          "write_attributes":"(**)",
          "cache_time":900
        },
        {
          "resource_id":"/products/(id)/availability",
          "read_attributes":"(**)",
          "write_attributes":"(**)",
          "cache_time":60
        },
        {
          "resource_id":"/products/(id)/prices",
          "read_attributes":"(**)",
          "write_attributes":"(**)",
          "cache_time":300
        },
        ...
      ]
    }
  ]
}
```

Knowledge Check

The Knowledge Check answer key is on the following page.

1. Displaying random products is a creative approach for dealing with:
   a. Unresponsive checkout services
   b. Caching failures
   c. Recommendation timeouts
   d. Open Commerce API exceptions
   e. Locked OCAPI services

2. Select all that apply to the Open Commerce API:
   a. OCAPI lets you set a different timeout for each service.
   b. The timeout for OCAPI is 1 second by default.
   c. To configure OCAPI, you use the Service Framework.
   d. The OCAPI configuration is specified in JSON.
   e. Disable OCAPI web services using Business Manager site preferences.

Knowledge Check Answers

1. Displaying random products is a creative approach for dealing with:
    a. Unresponsive checkout services
    b. Caching failures
    c. Recommendation timeouts
    d. Open Commerce API exceptions
    e. Locked OCAPI services

Answer: c

2. Select all that apply to Open Commerce API:
    a. OCAPI lets you set a different timeout for each service.
    b. The timeout for OCAPI is 1 second by default.
    c. To configure OCAPI, you use the Service Framework.
    d. The OCAPI configuration is specified in JSON.
    e. Disable OCAPI web services using Business Manager site preferences.

Answers: a, d

demandware move faster, grow faster.

Module 6: Adhering to Quotas

Learning Objectives

After completing this module, you will be able to:

- Describe platform quotas and the benefits of using them.
- Describe the Demandware data usage policy.
- Identify quota violations using the Business Manager Quota Status and quota log files.
- Create email notifications for quota violations.
- Describe the types and levels of quotas and explain the purpose of each.
- Describe the process of overriding platform quotas.

Lesson 6.1: What Is a Quota?

The Demandware Quota Framework measures usage on all instances of your realm against a set of quotas. Quotas define programmatic boundaries and best practices for using the Demandware API and business objects. Quotas ensure that you stay within the platform limits for memory usage, resource consumption, API calls, and the number of business objects. Demandware provides explicit usage controls—for example, the frequency of script and pipelet API calls and the number of objects per-instance. These limits prevent issues such as out-of-memory exceptions, thread exhaustion, database churn, and overuse of garbage collection. In preventing these issues, quotas ensure that your custom implementations can safely operate and scale on the Demandware platform.

If an implementation exceeds quota boundaries, the system throws an exception. The system handles quota exceptions as follows:

Exception	**Action**
WARN threshold (usually 60% of limit)	The system logs a quota warning.
ERROR threshold (100% of limit)	The system logs an exception.
Quotas with **limit 0** have no WARN threshold.	The system throws an exception immediately for quotas with **limit 0**. **Note**: Quotas with limit 0 should not be used with custom application code.

These exceptions eventually lead to platform instability and even disruption of the operating environment. Quotas provide the Demandware development community with explicit quantitative specifications to ensure platform stability.

Quota Enforcement

Quotas are either "enforced" or "not enforced."

If an enforced quota is exceeded, the system throws an exception preventing the current operation from completing. The Quota Framework writes these types of exceptions directly to the quota logs. You'll learn how to access the quota logs later in this module.

Examples of quota violations can include:

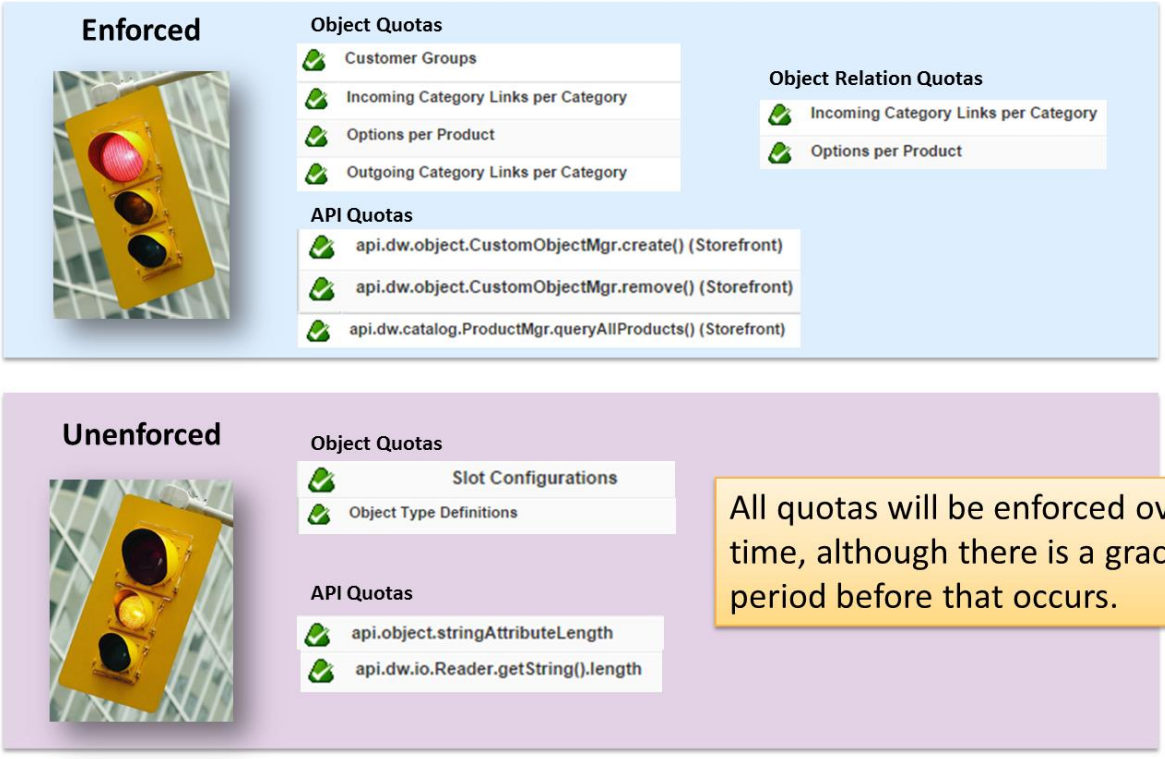
- A collection that creates too many objects in memory
- Too many persistent objects of a particular object type created in an instance

When an enforced quota is exceeded in a storefront request, the general error page appears. If an enforced quota is exceeded during a standard import, the system reports a data warning or data error.

When a quota is not enforced, the platform provides a warning, but does not take any other action.

Important: Demandware does not enforce all quotas initially. Over time Demandware will enforce these quotas. This policy gives developers a chance to refactor code to prevent quota violations. Exceeding enforced quotas can lead to additional fees.

Examples of Enforced and Unenforced Quotas



Benefits

Adhering to platform quotas while designing and developing custom code is beneficial because quotas prevent the following system-level issues from occurring:

- High database activity
- High network usage
- Memory violations
- Concurrent file modifications

Lesson 6.2: Data Usage

It's important to keep custom implementations within platform usage limits. Demandware's standard contract includes an addendum that outlines this Data Usage Policy (see https://xchange.demandware.com/docs/DOC-1481).

This document details specific data usage components, including data processing (DP), data transfer (DT), and data storage (DS).

The parameters within the policy include:

- A baseline monthly allocation for all customers, independent of gross merchandise volume (GMV)
- An additional monthly allocation based on GMV
- A per-unit fee for resource utilization in excess of the total monthly entitlement

Although most sites stay within their monthly resource allocations, sometimes sites utilize resources beyond their allocations. For example, some clients gain significant business value from the additional resources consumed during a particular period of time. In this case, the client pays an additional fee for the benefit of extra resources.

Lesson 6.3: Viewing Quota Status

You can view the status of quotas to identify issues within your code and correct the issues before they affect site stability. Use the Business Manager Quota Status page, as well as the quota log files, to view the status of quotas.

Quota Status

You use the Business Manager Quota Status page to view API and object usage levels, as well as the number of quota violations on a particular instance. You can also see which quotas the system enforces as well as the maximum values and the limits for quotas.

To view quota status data within Business Manager, select **Administration > Operations > Quota Status**.

The Quota Status page displays quota usage and violations for the last 24 hours or in the time period since the server was started. Some quotas specify warning thresholds to indicate that your site is nearing a quota limit. Quotas display with a green icon if there are no quota issues, an orange icon for warnings, and a red icon for violations.

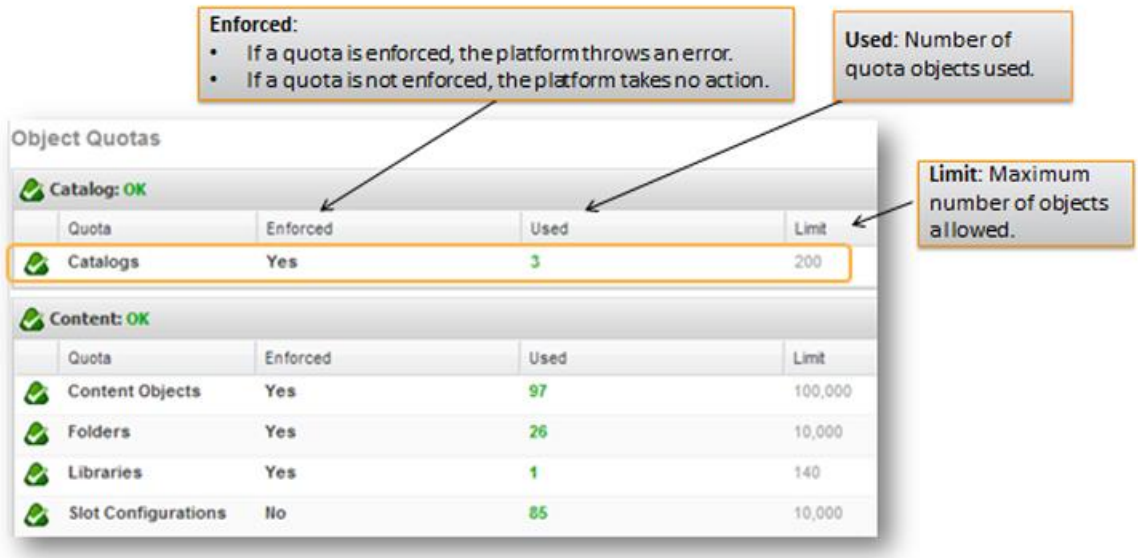
The Quota Status shows:

- Object quotas
- Object relation quotas
- API quotas

Object Quotas

Object quotas limit the number of objects of a particular type per instance. The platform updates object quotas within 20 minutes of the quota being exceeded.

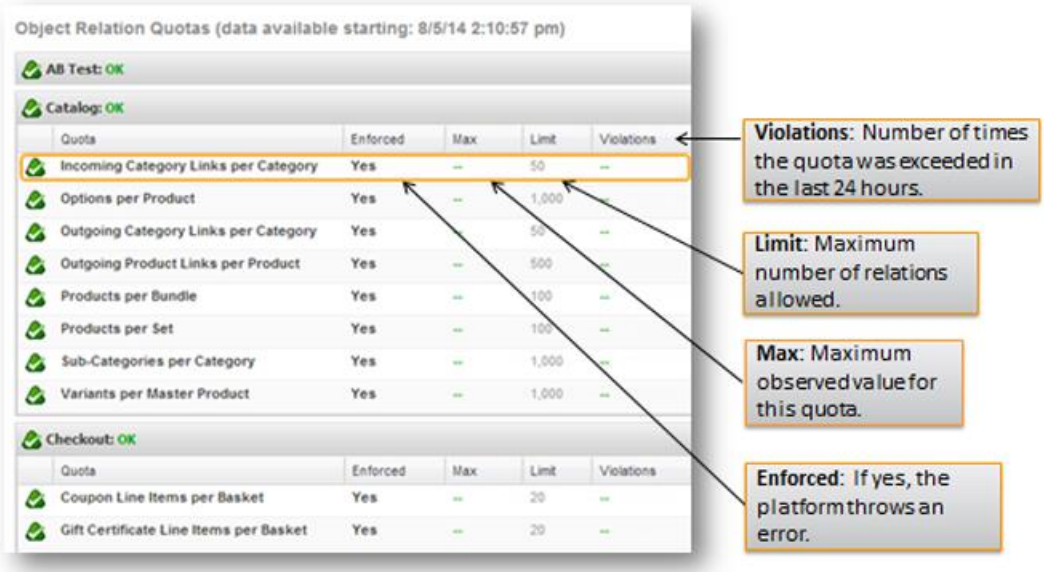
In this example, the number of catalogs allowed is 200. The instance uses three catalogs.



Object Relation Quotas

Object relation quotas limit the number of objects associated with an object type, for example, variants per master product. The Quota Framework increments the number of object relations when the system modifies relations—not when it accesses them.

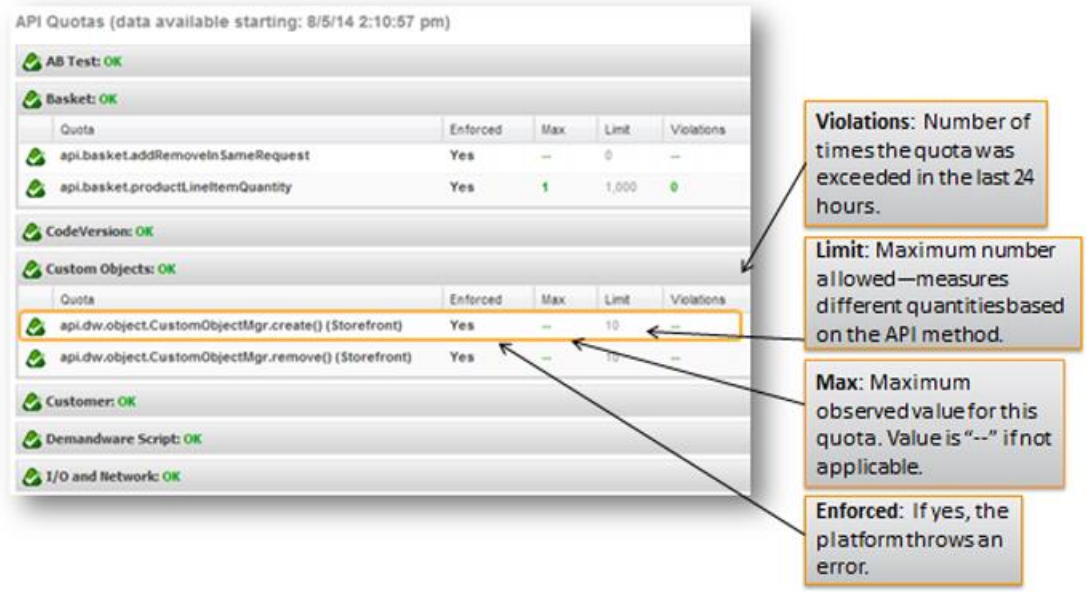
In this example, there are 50 incoming category links allowed for each category.



API Quotas

API quotas set limits on API usage, for example, the number of object instances, object sizes, runtimes, and the number of executions per page.

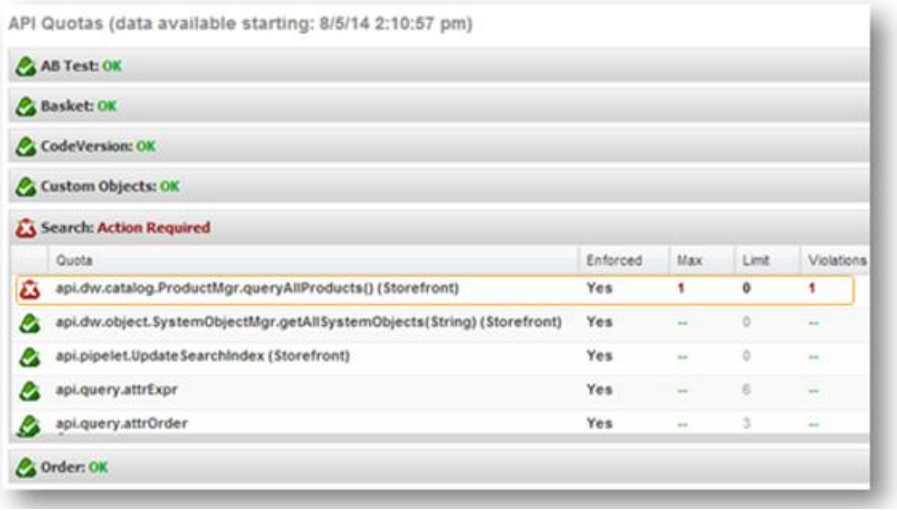
In the example below, the platform limits the number of custom objects created to 10 per page:



Quota Violations

Some API quotas restrict the execution of particular DW script methods or pipelets on the storefront. For example, the system prevents the `dw.catalog.ProductMgr.queryAllProducts()` method from being used on the storefront. In this case, the storefront limit for the `api.dw.catalog.ProductMgr.queryAllProducts()` quota is 0.

This example shows the `queryAllProducts(Storefront)` API quota violation (identified by the red icon).



Email Alerts

You can set up email alerts to notify yourself or others about quota violations. Once a day, the system sends email alerts listing all quotas above the warning threshold or above the limit. This example shows an email notifying recipients that an API quota limit was exceeded.

Subject: Demandware Quota Violation real time alert:kgermond.inside-na03.dw.demandware.net

Demandware Quotas Alert

Instance: kgermond.inside-na03.dw.demandware.net
Date Time: Thu Jul 31 15:43:02 GMT 2014

==========
API Quotas Exceeded:

[Category]	[Quota]	[Max]	[Limit]	[Warn]	[Enforced]	[Status]
Search	api.dw.catalog.ProductMgr.queryAllProducts() (Storefront)	1	0	0	true	limit exceeded

To configure an email alert within Business Manager, select **Administration > Operations > Quota Status**. Under **Quota Alert Settings**, enter the email addresses of the alert recipients and select the **Enabled** option.

Lesson 6.4: Viewing Quota Log Files

When a quota's warn threshold is exceeded, or when a specific quota limit is exceeded, messages are written to the quota log files, along with the code segment where the exception occurred. This happens for all quotas, enforced and not enforced. Quota log files are created using a filename prefix of "quota", for example, `quota-blade2-5-appserver-20140723.log`.

To access quota logs within Business Manager, select **Site Development > Development Setup**. Under **WebDAV Access**, click the link under **Log Files**.

Log File Format

Quota log files use the following format:

- **Timestamp**: Includes the date and time of the event.
- **Thread**: Contains the type of request, the site name, the pipeline name, and other parameters.
- **Message**: Provides details of the quota issue, such as the quota type, whether it's enforced, the warn threshold, the limit, and the highest observed reading.

Quota Entry

This example shows an entry in a quota log file:

```
[2014-07-23 13:00:22.577 GMT] [RequestHandlerServlet|14320460|Sites-
SiteGenesis-Site|Product-
Show|PipelineCall|pRLNN0PXCp7rqIClq7IcPUvuEEjtkOWUAMBdUWhniPRFlV4Xvim3HwhDw9
6bRZ7e5BUxhnkC2ExrVYZVBFLJtw==] Quota api.jsStringLength (enforced, warn
600000, limit 1000000): warn threshold exceeded 1 time(s), max actual was
600010, current location: request/site Sites-SiteGenesis-Site/top pipeline
Product-Show/interaction node/template refapp.default_.CustomDisplay
```

This log file entry contains the data in the following table:

Quota Log File Entry Component	Quota Log File Entry Data (from preceding example)
Time stamp	`[2014-07-23 13:00:22.577 GMT]`
Thread	`[RequestHandlerServlet|14320460|Sites-SiteGenesis-Site|Product-Show|PipelineCall|pRLNN0PXCp7rqIClq7IcPUvuEEjtkOWUAMBdUWhniPRFlV4Xvim3HwhDw96bRZ7e5BUxhnkC2ExrVYZVBFLJtw==]`
Message	`Quota api.jsStringLength (enforced, warn 600000, limit 1000000): warn threshold exceeded 1 time(s), max actual was 600010, current location: request/site Sites-SiteGenesis-Site/top pipeline Product-Show/interaction node/template refapp.default_.CustomDisplay`

The details of the message are separated by colons and include the following:

- Quota type
- Whether the quota is enforced or not enforced
- The warn threshold (if one exists)
- The limit
- The number of times since the last log entry for this quota that the WARN threshold or limit was exceeded
- The highest observed reading (max actual)
- The current location, which shows where the warn threshold or limit was exceeded

The log system collects information about quota violations and aggregates quota log messages over a period of time. After the time period ends, the system writes to the quota log file the next time a WARN threshold or limit is exceeded. Demandware writes the code location of the most recent event to the log file, but does not expose the code locations of the aggregated events.

Lesson 6.5: Overriding Quotas

Quota overrides are a transitional solution to handle legacy quota exceptions. These overrides soften quotas and quota status actions by not enforcing violations. Quota overrides downgrade quotas from error to warn.

For quota issues on primary instances (production, staging, and development), you must request a temporary quota override via a support ticket. Demandware overrides the quota if the impact justifies a SEV-1 issue. Quota overrides are typically reserved for use with realms created before the Demandware Quota Framework was introduced (November 2011). Demandware never removes quota overrides unilaterally.

Exercise: Configure an Email Alert

In this exercise, you'll set up an email alert, which will send you an email alert when a quota violation occurs.

1. In Business Manager, select **Administration > Operations > Quota Status**.
2. Under **Quota Alert Settings**, enter your email address in the **Email to Addresses** field.
3. Select the **Enabled** field and click **Apply**.

   The system sends an email alert once a day if there are quota violations.

Exercise: Diagnose Performance Issues Using Quota Status

Because the Quota Status monitors object, object relation, and API violations, you can use the report to diagnose performance issues.

In this exercise, you'll diagnose performance issues using the Quota Status report. You'll compare the quota results of two different scripts, `getProdByColor1.ds` and `getProdByColor2.ds`.

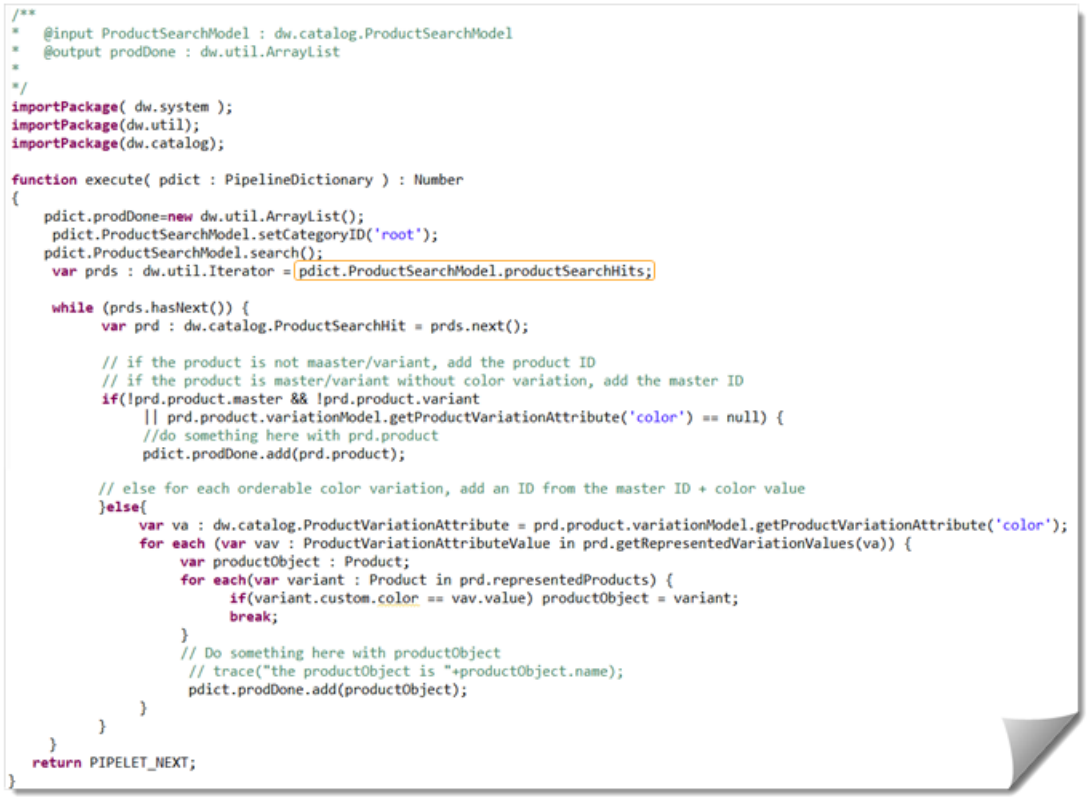
This script, `getProdByColor1.ds`, causes an API quota violation:

```
/**
*
*   @output prodDone : dw.util.ArrayList
*
*/

importPackage( dw.system );
importPackage(dw.util);
importPackage(dw.catalog);

function execute( args : PipelineDictionary ) : Number
{

    var productIterator : SeekableIterator = dw.catalog.ProductMgr.queryAllSiteProducts();
    var prodDone : dw.util.ArrayList = new dw.util.ArrayList();

    while(productIterator.hasNext()) {

        var product : Product = productIterator.next();
        var productIDColor : String;
        var color : String;

        if(!product.master) {

            // if the product is not a variant, add the product ID
            // if the product is a variant without color variations, add the master ID
            if(!product.variant || product.variant && empty(product.custom.color)) {
                productIDColor = product.variant ? product.master.ID : product.ID;

            // else create an ID from the (master) product ID + color value

            // if the list does not contain the ID, add it to the list
            if (!empty(productIDColor) && !prodDone.contains(productIDColor) && product.availabilityModel.orderable) {
                // execute some code to add this product to the feed
                prodDone.add(productIDColor);
            }
        }
    }


  return PIPELET_NEXT;
}
```

The script generates a product feed for product search engines. The feed contains only online products, represented as variants of each orderable color. So, instead of adding one entry for the master, the feed should have one variant entry for each different color. This script provides external product search engines with relevant color matches.

The script attempts to generate the feed by obtaining all objects using the `dw.catalog.ProductMgr.queryAllSiteProducts()` method on the entire storefront, then querying for online variations. The Demandware platform prevents the `queryAllSiteProducts()` method from being used on the storefront, so this script fails with an API quota violation.

The `getProdByColor2.ds` script performs the same functionality without eliciting a quota violation:



```
/**
*   @input ProductSearchModel : dw.catalog.ProductSearchModel
*   @output prodDone : dw.util.ArrayList
*
*/
importPackage( dw.system );
importPackage(dw.util);
importPackage(dw.catalog);

function execute( pdict : PipelineDictionary ) : Number
{
    pdict.prodDone=new dw.util.ArrayList();
     pdict.ProductSearchModel.setCategoryID('root');
    pdict.ProductSearchModel.search();
     var prds : dw.util.Iterator = pdict.ProductSearchModel.productSearchHits;

     while (prds.hasNext()) {
           var prd : dw.catalog.ProductSearchHit = prds.next();

           // if the product is not maaster/variant, add the product ID
           // if the product is master/variant without color variation, add the master ID
           if(!prd.product.master && !prd.product.variant
                || prd.product.variationModel.getProductVariationAttribute('color') == null) {
                //do something here with prd.product
                pdict.prodDone.add(prd.product);

          // else for each orderable color variation, add an ID from the master ID + color value
          }else{
                var va : dw.catalog.ProductVariationAttribute = prd.product.variationModel.getProductVariationAttribute('color');
                for each (var vav : ProductVariationAttributeValue in prd.getRepresentedVariationValues(va)) {
                     var productObject : Product;
                     for each(var variant : Product in prd.representedProducts) {
                          if(variant.custom.color == vav.value) productObject = variant;
                          break;
                     }
                     // Do something here with productObject
                      // trace("the productObject is "+productObject.name);
                      pdict.prodDone.add(productObject);
                }
           }
     }
   return PIPELET_NEXT;
}
```

This more efficient script uses `productSearchHits` from the Search API to access the online products.

In this exercise, you'll run the `GetProductsByColor1.ds` script.

View the Quota Status for the GetProductsByColor1 Pipeline

Follow this procedure to view the quota violation caused by the `getProdByColor1.ds` script:

1. In the `trainingPerformance` cartridge, open the `GetProductByColor1` pipeline in UX Studio.

   Note: If you haven't yet imported the `trainingPerformance` cartridge, see the "Optimizing Performance During Development" module for steps.

   The `GetProductByColor1` pipeline calls the `getProdByColor1.ds` script which creates the feed contents and the `GetProdByColor1.isml` template to display the feed contents.

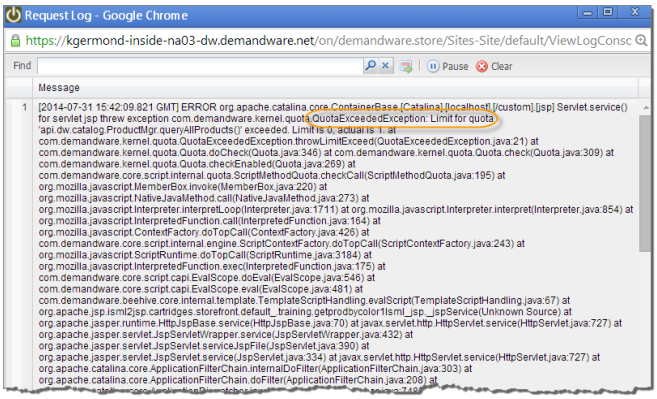
2. Run the `GetProductByColor1` pipeline on your storefront site, for example:

   `https:// student01.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/GetProductsByColor1-Start`

   The pipeline fails to run and a storefront page displays the message: "An Error Occurred."

3. Open the Toolkit Request Log.

   The request log shows that the script exceeded the quota limit:



4. In Business Manager, select **Administration > Operations > Quota Status**.
5. Click the red icon with the “Action Required” label.

   Which quota shows a violation?

   _____

6. Click the **Catalog** entry under **Object Relation Quotas**.

   How many sub-categories can a category have?

_____

7. Check to see if you received an email notification due to the error.

   (The previous exercise showed you how to set up an email alert.)

View the Quota Status of the GetProductByColor2 Pipeline

Now, you’ll run the `GetProductByColor2` pipeline which calls the more efficient `getProdByColor2.ds` script:

1. In the `trainingPerformance` cartridge, open the `GetProductByColor2` pipeline in UX Studio.

   The `GetProductByColor2` pipeline calls the `getProdByColor2.ds` script which creates the feed contents and the `GetProdByColor2.isml` template to display the feed contents.

2. Run the `GetProductByColor2` pipeline on your storefront site, for example:

   ```
   https:// student01.training-
   na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-
   Site/default/GetProductsByColor2-Start?q=shirts
   ```

   The search parameter causes the platform to create a search index.

   The pipeline prints the results as follows:

   Paisley Shirt
   Striped Shirt
   Striped Shirt
   Striped Shirt
   No-Iron Textured Dress Shirt
   The White Dress Shirt
   Modern Dress Shirt in Pink
   Modern Dress Shirt
   Modern Striped Dress Shirt
   Button Down Shirt
   Button Down Shirt
   Button Front Shirt
   Sleeveless Button Down Shirt
   Classic Button Front Shirt
   Classic Button Front Shirt
   Ruffle Sleeveless Shirt
   Roll Sleeve Shirt
   Belted Shirt Dress
   Floral Tie Front Shirt
   Floral Shirt Dress.
   Floral Print Shirt Jacket.
   Roll Sleeve Floral Shirt
   Denim Shirt Jacket
   Button Front Shirt
   Stripped Button Down Shirt
   Long Sleeve Seamed Shirt

3. In Business Manager, select **Administration > Operations > Quota Status**.
4. Check that there are no new quota violations caused by the `GetProductByColor2` pipeline.

Exercise: View Quota Violations in Log Files

In this exercise, you'll review a quota log file.

1. Access today's quota log file by selecting **Site Development > Development Setup** in Business Manager.
2. Under **WebDAV Access**, click the link under **Log Files**.
3. Click today's quota log file (named `quota-`*`bladenumber`*`-appserver-`*`date`*`.log`).

   Which pipeline caused the most recent quota violation?

   _____

   Which type of quota did the pipeline violate?

   _____

You've finished the exercises in this section.

Knowledge Check

1. Which of the following correctly describes the object quotas limit?
   a. The number of objects of a particular type per site.
   b. The amount of memory used to store objects.
   c. The number of object of a particular type per instance.
   d. The number of objects used in a pipeline.

Knowledge Check Answers

1. Which of the following correctly describes the object quotas limit?
   a. The number of objects of a particular type per site.
   b. The amount of memory used to store objects.
   c. The number of object of a particular type per instance.
   d. The number of objects used in a pipeline.

Answer: c

demandware move faster, grow faster.

Module 7: Summary of Diagnostic Tools

Learning Objectives

After completing this module, you will be able to:

- Summarize the diagnostic tools and identify under which circumstances they should be used.
- Identify external tools for analyzing performance.

Lesson 7.1: Diagnostic Tools for Analyzing Performance and Data Usage

This table shows the diagnostic tools available to help you troubleshoot performance issues on the Demandware platform:

Diagnostic Tool	Capabilities	How to Access Tool	Instance Type Supported by Tool
Pipeline Profiler	Profiles pipelines and scripts to detect: ▪ Slow pipelines ▪ Inefficient coding practices ▪ Bottlenecks on production instances, but run the Profiler on production for only about 15 minutes.	Business Manager: **Administration > Operations > Pipeline Profiler**	All instance types
Technical Reports	Provides analytics on: ▪ Performance and usage of pipelines and includes ▪ Errors in pipelines and includes ▪ Effective use of caching	Business Manager: **Site > Analytics > Technical Reports**	Production
Traffic Reports	▪ Provides request runtime averages. ▪ Displays the number of requests in	Business Manager: **Site > Analytics > Traffic Reports**	Production

	specified time periods. ▪ Provides the top pages. ▪ Provides the number and duration of site visits.		
Object Churn Trends	▪ Diagnoses slow operations, such as checkouts. ▪ Identifies spikes in database operations related to creating, updating, and deleting database objects, such as sessions and baskets.	Business Manager: **Site > Analytics > Object Churn Trends**	Production
Logs	▪ Monitors errors and warnings on the platform. ▪ Monitors errors and warnings based on custom categories developers create.	Log files: **Site Development > Development Setup**. Under **WebDAV Access**, click the link under **Log Files**. To enable custom logging: **Administration > Operations > Custom Log Settings**.	All instance types
Quotas	▪ Identifies misuse of data objects and APIs. ▪ Indicates when the quotas specified by Demandware are nearing or exceeding limits.	In Business Manager: Select **Administration > Operations > Quota Status**. Log files: **Site Development > Development Setup**. Under **WebDAV Access**, click the link under **Log Files**. Click the log file in the format: `quota-`*`bladenumber`*`-appserver-`*`date`*`.log`.	All instance types

Demandware Control Center	▪ Reports overall resource consumption for an instance. ▪ Determines whether the site will scale—to verify scalability, compare the Control Center resource consumption data with the Analytics Traffic Report data. **Note**: Control Center is available to users with administrator access only.	In Demandware Control Center, access the Instance Manager.	All instance types
External Tools (such as WebPagetest, YSlow, and PageSpeed)	▪ WebPagetest analyzes page performance by browser and location. ▪ Firebug YSlow analyzes site performance and suggests how to improve site performance. ▪ PageSpeed suggests how to improve site performance for mobile and desktop platforms.	www.webpagetest.org www.getfirebug.com https://developers.google.com/speed/pagespeed	All instance types

Lesson 7.2: Reviewing Resource Consumption Using Control Center

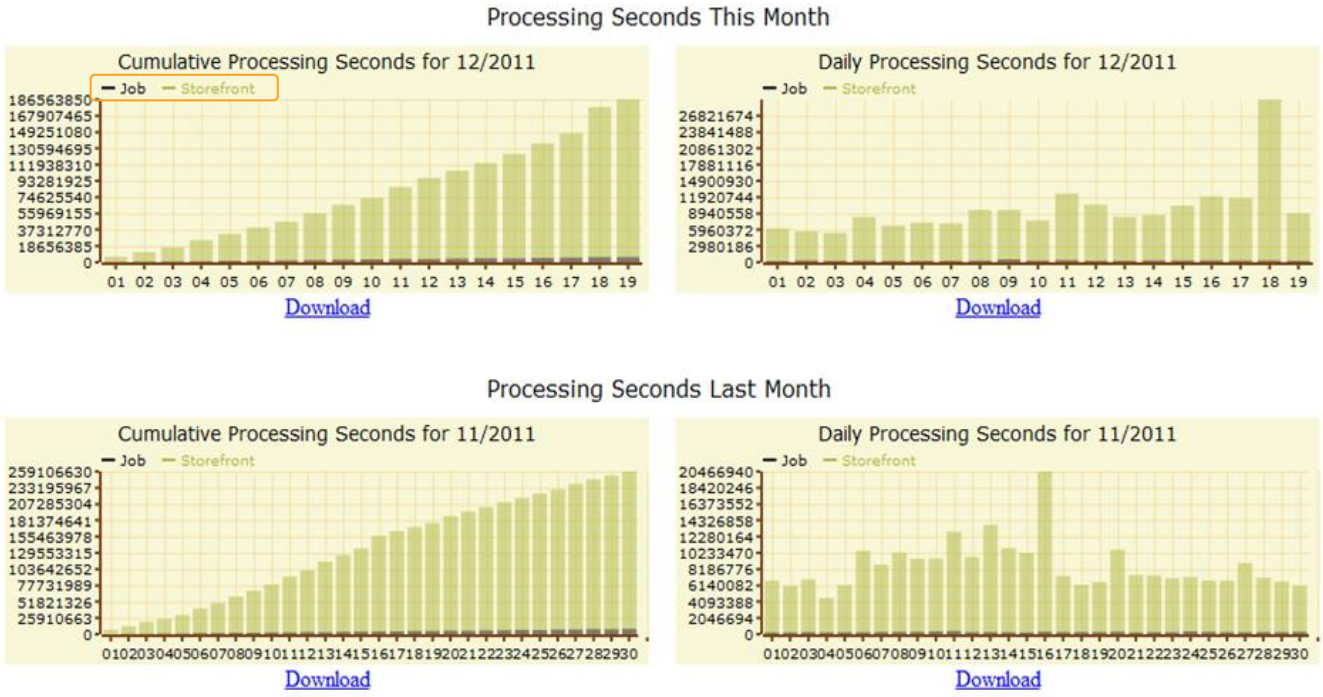
The Demandware Control Center provides a complete view of overall resource consumption for a site on the platform. Use the Control Center to verify storefront and job processing times. Ideally, resource consumption should not increase as the traffic increases.

A good strategy is to review the traffic patterns of the site using the Analytics Traffic Reports. Then, compare the traffic results to the overall resource consumption to validate the ability of the application to scale. Even if the CPU consumption scales with the traffic, there is room for improvement. If consumption grows even higher as the traffic increases, you need to take immediate action.

To view resource consumption details for your instances, you must have a Demandware Control Center account. Log in to the Demandware Control Center (https://controlcenter.podX.demandware.net). In the menu, click **Instance Management**. Expand to select an organization and instance.

This Control Center instance management report shows that the production, staging, and development instances are up:

To view resource consumption statistics for an instance, click **Statistics**. Scroll to the bottom to see usage statistics, for example, the following report compares processing seconds from last month versus this month:



The graphs show the cumulative and daily processing seconds for the current and last month. They distinguish between storefront and job processing.

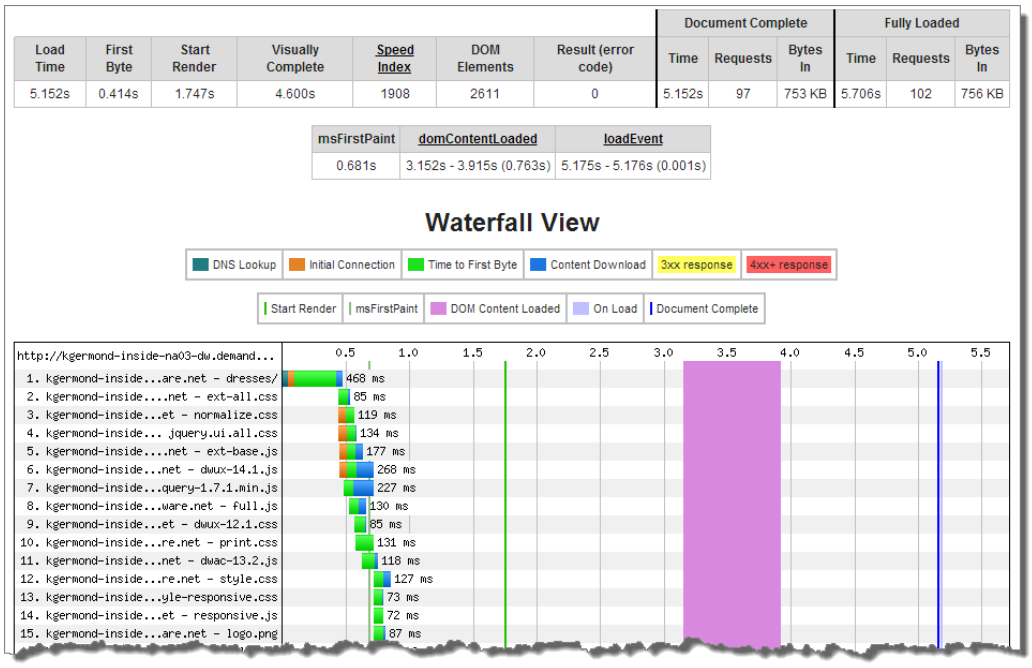
Click the **Download** link for each graph to save it as a CSV file.

Lesson 7.3: External Tools for Analyzing Performance

In addition to the utilities and reports Demandware provides, there are effective external tools to help you analyze performance. These free tools measure the performance of your pages and suggest ways to improve performance.

WebPagetest

You can use WebPagetest to test the performance of pages on your site (www.webpagetest.org). WebPagetest lets you select a browser and a location from which to run the test. The tool provides a waterfall view that can help pinpoint issues:



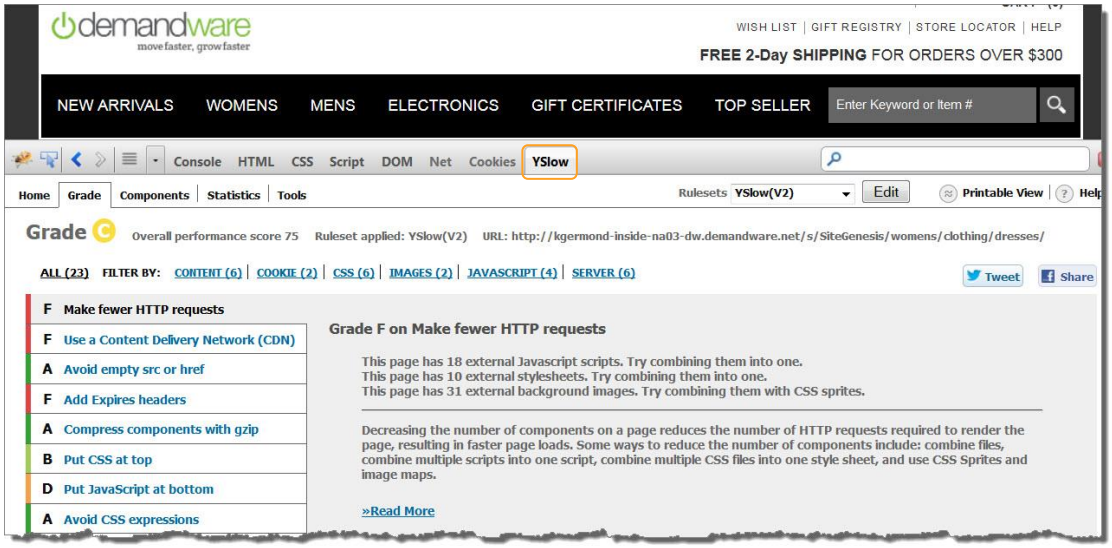
							Document Complete			Fully Loaded		
Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Result (error code)	Time	Requests	Bytes In	Time	Requests	Bytes In
5.152s	0.414s	1.747s	4.600s	1908	2611	0	5.152s	97	753 KB	5.706s	102	756 KB

msFirstPaint	domContentLoaded	loadEvent
0.681s	3.152s - 3.915s (0.763s)	5.175s - 5.176s (0.001s)

For details, see the WebPagetest Quick Start Guide: https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/quick-start-guide.

Firebug with YSlow

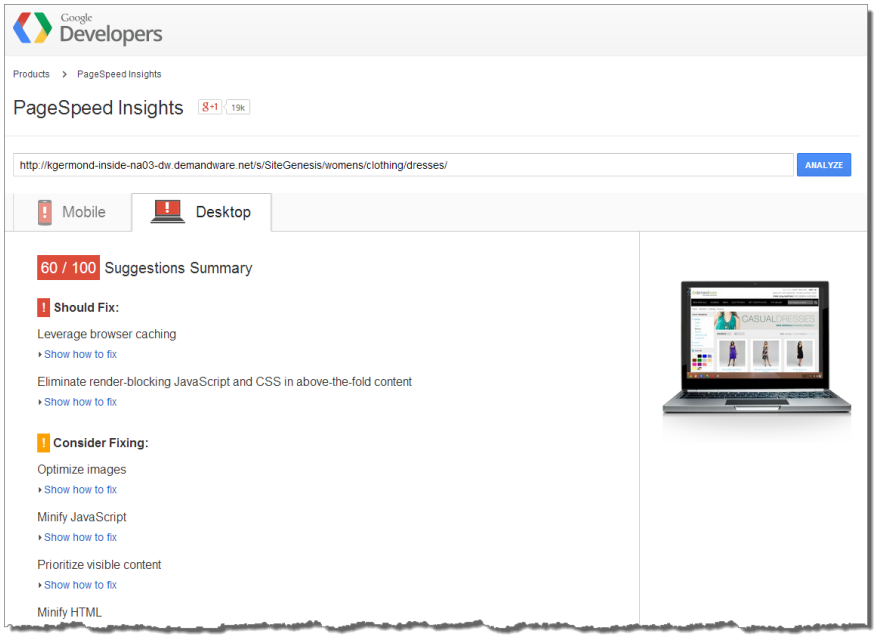
You can use the Firebug development tool to analyze performance issues, as well. Firebug's YSlow extension analyzes web page performance and provides suggestions to improve a page's performance:



To install Firebug, see https://getfirebug.com. For information on YSlow, see http://yslow.org/user-guide.

PageSpeed

PageSpeed is supported on the Google Developers site (https://developers.google.com/speed/pagespeed). PageSpeed provides suggestions for improving your site's performance – for either desktop or mobile usage:



This tool runs the site through the standard performance best practice checklist, and it also identifies exactly which files and pieces of code can be optimized. It provides tips and instructions on how to optimize them and rates the optimization's potential impact on performance.

Congratulations

Now that you've learned how to develop customizations and integrations using practices that ensure optimal performance and scalability, you can:

- Review your scripts to ensure that you are using the coding best practices you've learned in this course. Use the script data in the Pipeline Profiler to compare and optimize pipeline and script implementations. If possible, perform a code review and replace deprecated API calls with up-to-date API calls.
- Analyze your storefront site using the Storefront Toolkit Cache Information and the Pipeline Profiler to check that the site uses effective caching strategies.
- Use the Pipeline Profiler, Business Manager Analytics, and the Demandware Control Center to analyze and troubleshoot performance issues.

Be sure your team allocates time to analyze your site's performance continually and to implement ongoing optimizations.

If you have now completed the following courses, you can sign up for the Customization, Integration, and Performance certification exam:

- *Exploring SiteGenesis*
- *Working with the Demandware APIs*
- *Integrating with Demandware*
- *Developing for Improved Performance and Stability*