Magento
an ebay inc company

Magento Customer Segments Under the Hood

Magento Enterprise has a robust promotion engine that enables merchants to target products to specific audiences. Because targeting customer segments can be a labor-intensive process, automation is crucial to a promotion's success. This article focuses on the internal functionality of Magento's Customer Segment module to help understand its performance impact and the best ways to customize its implementation.

This article is based on Magento Enterprise Edition 1.13.0.2 and is intended for experienced Magento PHP developers who want to know more about customer segments. We will also take a look at some scenarios that illustrate the performance impact of using the Customer Segment features.

TABLE OF CONTENTS

1 Using Customer Segments

2 Under the Hood

- **2.1** Customer Segment Creation
- **2.2** Customer Login
- **2.3** Product Purchase

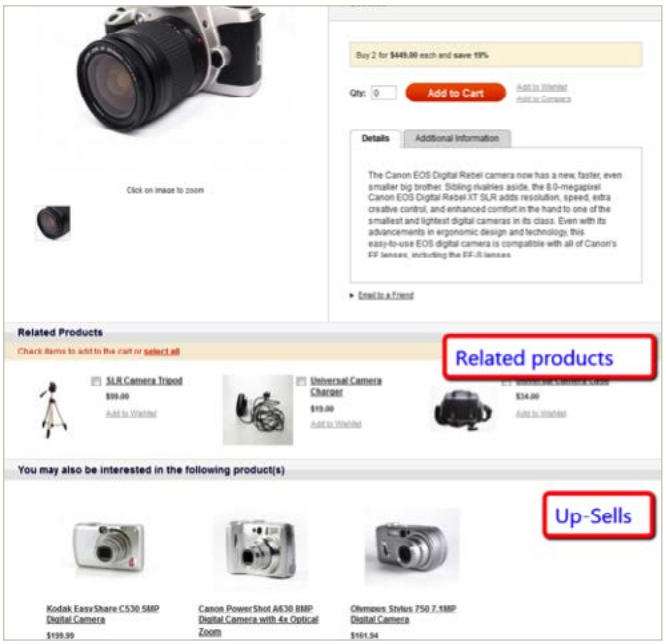
3 Performance Impact

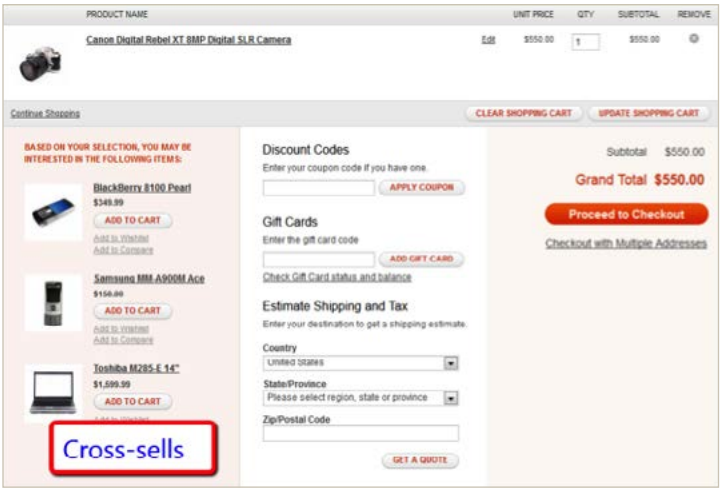
4 Guidelines

Magento
an ebay inc company

1 USING CUSTOMER SEGMENTS

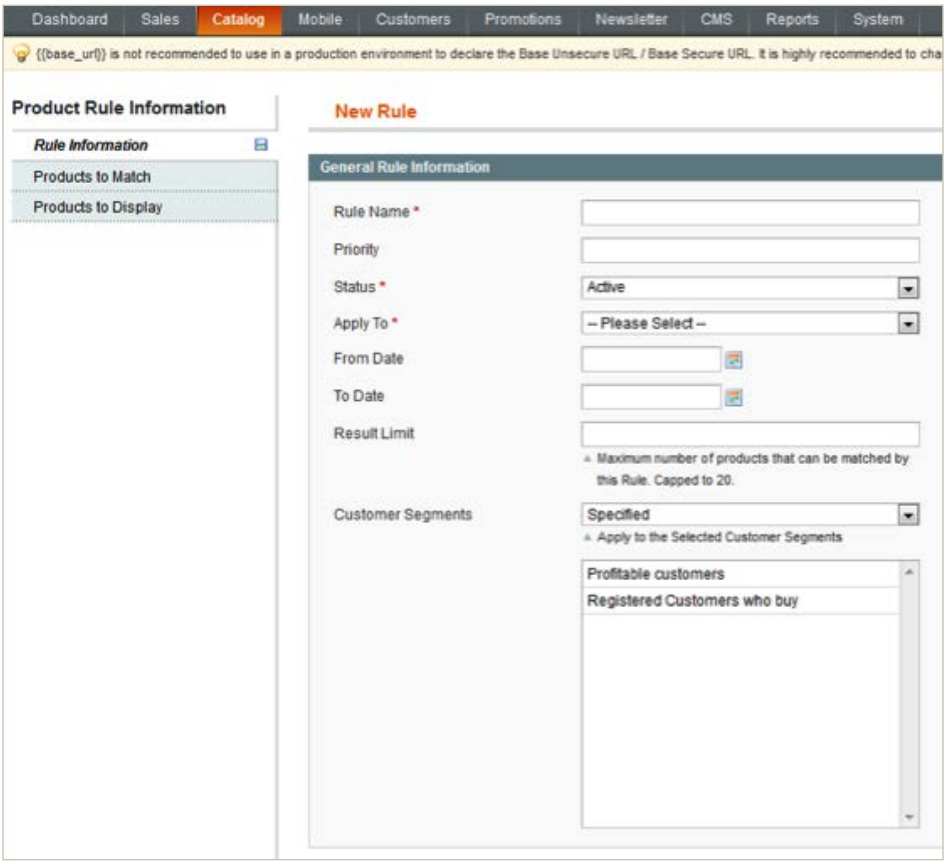
There are promotion features in Magento Enterprise Edition that are designed to assist in selling more products. These features include **related products, up-sells** and **cross-sells**. The **TargetRule** module, for example, helps automate the relationships between products so that you don't have to manually assign them.





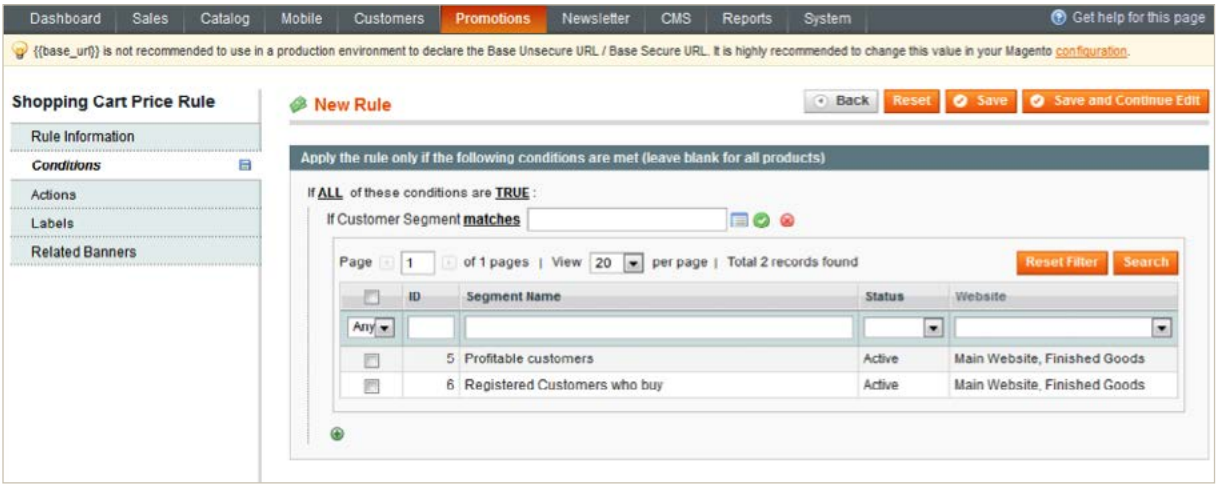
For an administrator to use this automated tool, go to the Admin Panel and click Catalog > Rule-Based Product Relations to add a new rule:

1 USING CUSTOMER SEGMENTS (CONT'D)

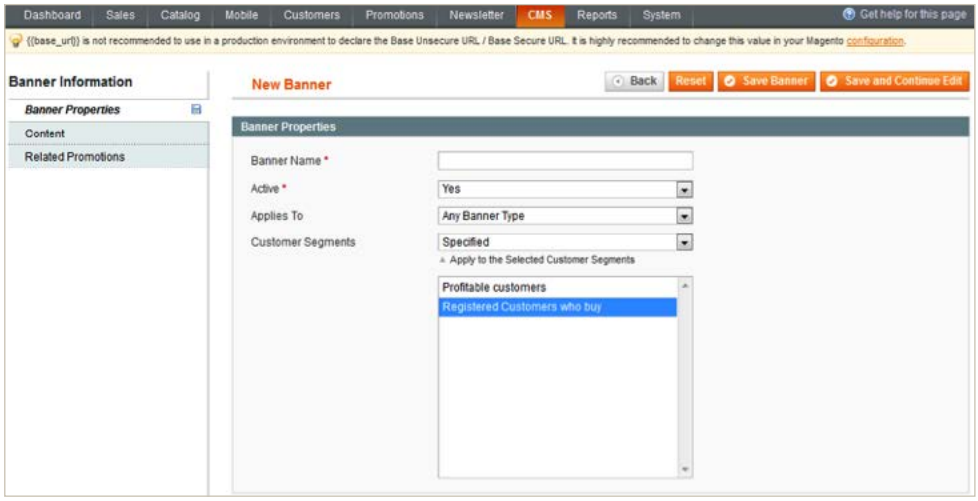


In one of the fields you can specify the customer segments to target in this rule. The screen shot shows two examples of segments: **Profitable customers** and **Registered Customers who buy.** These kinds of rule-based product relations allow you to specify which customer segments are eligible for up-sells, cross-sells or related products.

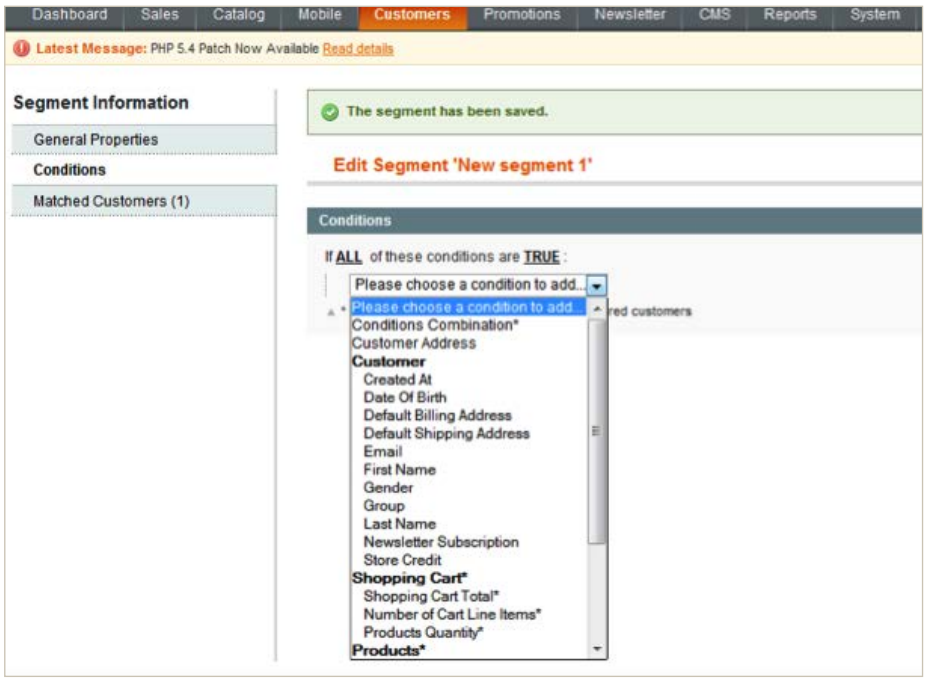
Two other types of promotions also use customer segments are the shopping cart price rule (SalesRule module) and Banners:



1 USING CUSTOMER SEGMENTS (CONT'D)



When you want to target a campaign to a specific audience, you may first think that the best way to do this is to assign customer groups to promotions. But this approach doesn't allow the same flexibility as customer segments.



The advantage of customer segments is that you can create a promotional rule for a customer segment once, and that segment of customers will automatically get updated. For example, when a Customer Segment uses a Sales Amount condition, Magento Enterprise Edition calculates the "Total Sales Amount" for every customer purchase and automatically and continuously moves qualifying customers (and visitors, if you set up the rule that way) into and out of the corresponding segment.

As appealing as this is, it comes with a price. A trade-off of system performance for the convenience of this rule may be negligible in the beginning but in time, page response times could become unacceptable. Before we discuss what options there are to deal with this, let's look at how this module works.

2 UNDER THE HOOD

The customer segment module uses four tables in the Magento database:

1. `enterprise_customersegment_customer`
2. `enterprise_customersegment_event`
3. `enterprise_customersegment_segment`
4. `enterprise_customersegment_website`

The `enterprise_customersegment_segment` table is used to store settings for all configured customer segments.

...	name	description	i...	conditions_serialized	pro...	condition_sql	apply_to *
5	Profitable customers	{null}	1	a:6:{s:4:"type";s:57:"enterprise_customersegment/segment_condition_combine_root";s:...	1	SELECT 1 FROM `...	0
6	Registered Customers who buy	{null}	1	a:7:{s:4:"type";s:57:"enterprise_customersegment/segment_condition_combine_root";s:...	1	SELECT 1 FROM `...	1

Additionally, the `enterprise_customersegment_website` stores the relation between a customer segment and the websites it relates to:

segment_id *	website_id *
5	1
6	1
7	1
5	2
6	2

The `enterprise_customersegment_customer` table stores IDs of customers who match the customer segment criteria, which updates after saving the Customer Segment:

segment_id *	customer_id *	added_date *	updated_date *	website_id *
5	1	7/3/2013 15:43:27 PM	7/3/2013 15:43:27 PM	1
5	2	7/12/2013 12:02:42 PM	7/12/2013 9:02:42 AM	1
5	3	7/3/2013 15:43:27 PM	7/3/2013 15:43:27 PM	1
6	2	7/12/2013 12:02:42 PM	7/12/2013 9:02:42 AM	1

The last table, `enterprise_customersegment_event`, is a little tricky. It stores events related to a segment_id based on conditions selected. This explains why different segments have a different set of events in this table. We will discuss this table later.

The most common scenarios involving the Customer Segment module are:

1. Customer segment creation
2. Customer login
3. Product purchasing

Let's look at them one by one.

segment_id *	event
5	customer_login
5	visitor_init
6	customer_login
6	sales_order_save_commit_after
6	visitor_init
6	customer_address_save_commit_after
6	customer_address_delete_commit_after
7	customer_login
7	sales_quote_save_commit_after
7	visitor_init

2.1 CUSTOMER SEGMENT CREATION

When an administrator creates a new customer, Magento Enterprise Edition saves data about the segment into an `enterprise_customersegment_segment` table.

```
INSERT INTO `enterprise_customersegment_segment` (`name`, `description`, `is_active`, `conditions
serialized`, `processing_frequency`, `condition_sql`, `apply_to`)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

The `conditions_serialized` and `condition_sql` fields contain the least restricted rules matching all the customers you have. For example:

`conditions_serialized`

```
Array
(
    [type] => enterprise_customersegment/segment_condition_combine_root
    [attribute] =>
    [operator] =>
    [value] => 1
    [is_value_processed] =>
    [aggregator] => all
)
```

`condition_sql`

```
SELECT 1 FROM `customer_entity` AS `root`
```

Magento creates two records in the `enterprise_customersegment_event` table with the basic events:

```
INSERT INTO `enterprise_customersegment_event` (`event`,`segment_id`) VALUES (?, ?), (?, ?) ON DUPLICATE KEY
UPDATE segment_id = VALUES(`segment_id`)
BIND: array (
  0 => ‘customer_login’,
  1 => 8,
  2 => ‘visitor_init’,
  3 => 8,
)
```

Magento saves website IDs that relate to this segment:

```
INSERT INTO `enterprise_customersegment_website` (`website_id`,`segment_id`) VALUES (?, ?) ON DUPLICATE KEY
UPDATE segment_id = VALUES(`segment_id`)
```

2.1 CUSTOMER SEGMENT CREATION (CONT'D)

Magento collects all matching customers and saves them in the `enterprise_customersegment_customer` table:

1. All possible records for this customer segment ID are cleared:

```
DELETE FROM `enterprise_customersegment_customer` WHERE (segment_id='8')
```

2. All customers for each website specified in the settings are selected:

```
SELECT `root`.`entity_id` FROM `customer_entity` AS `root` WHERE (website_id='2')
```

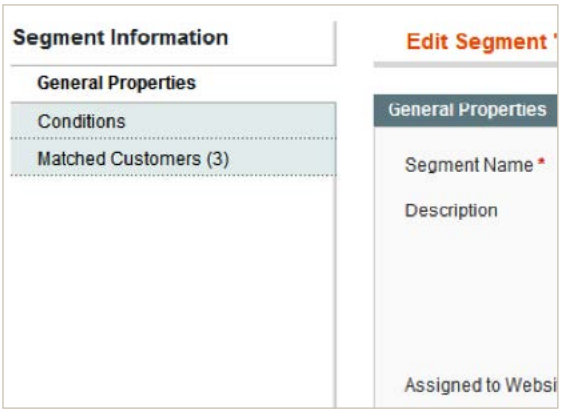
```
SELECT `root`.`entity_id` FROM `customer_entity` AS `root` WHERE (website_id='1')
```

3. The `aggregateMatchedCustomers` method of the Enterprise_CustomerSegment_Model_Resource_Segment class saves all matched customers into the `enterprise_customersegment_customer` table. It happens separately for each website and in groups of 1000. But if your segment was only created for visitors, this step doesn't occur. Also, the number of condition settings you can apply to a visitors-only customer segment will be less than for a segment that contains registered customers.

```
INSERT INTO `enterprise_customersegment_customer` (`segment_id`,`customer_id`,`website_id`,`added_
date`,`updated_date`)
VALUES (?, ?, ?, ?, ?), (?, ?, ?, ?, ?), (?, ?, ?, ?, ?)
```

2.1 CUSTOMER SEGMENT CREATION (CONT'D)

After this step, it becomes possible to create conditions and to restrict customers who belong to this segment. As shown in the following figure, two additional tab pages display: **Conditions** and **Matched Customers**.



Magento composes a SQL query to select all customers who match the conditions list, regardless of whether this segment is targeted for both customers and visitors or visitors only. This query is stored into the `condition_sql` field and may be quite long for a complex condition set. The following example is for a condition with three rules:


```
SELECT 1
  FROM `customer_entity` AS `root`
 WHERE (    (IF(
                (SELECT 1
                   FROM `sales_flat_quote` AS `quote`
                        INNER JOIN `core_store` AS `store`
                          ON quote.store_id = store.store_id
                  WHERE      (quote.is_active = 1)
                        AND (store.website_id = :website_id)
                        AND (quote.base_subtotal = '1000')
                        AND (    quote.customer_id = :customer_id
                                 OR quote.entity_id = :quote_id)
                  LIMIT 1),
                1,
                0) = 1)
        AND (IF(
                (SELECT 1
                   FROM `sales_flat_quote` AS `quote`
                        INNER JOIN `core_store` AS `store`
                          ON quote.store_id = store.store_id
                  WHERE      (quote.is_active = 1)
                        AND (store.website_id = :website_id)
                        AND (quote.items_qty = '2')
                        AND (    quote.customer_id = :customer_id
                                 OR quote.entity_id = :quote_id)
                  LIMIT 1),
                1,
                0) = 1)
        AND (IF(
                (SELECT 1
                   FROM `sales_flat_quote_item` AS `item`
                        INNER JOIN `sales_flat_quote` AS `list`
                          ON item.quote_id = list.entity_id
                        INNER JOIN `core_store` AS `store`
                          ON list.store_id = store.store_id
                  WHERE      (store.website_id IN (:website_id))
                        AND (list.is_active = 1)
                        AND (    list.customer_id = :customer_id
                                 OR list.entity_id = :quote_id)
                        AND (EXISTS
                                 (SELECT `main`.`entity_id`
                                    FROM `catalog_product_entity` AS `main`
                                   WHERE      (main.attribute_set_id = '4')
                                          AND (main.entity_id = item.product_id)))
                  LIMIT 1),
                1,
                0) = 1))
```

Let's look at a scenario when a customer logs in, to see when the system uses the value of the `condition_sql` field.

2.2 CUSTOMER LOGIN

The CustomerSegment module registers many listenable events. Let's look at one of them, the `customer_login` event.

```
233            <customer_login>
234                <observers>
235                    <enterprise_customersegment>
236                        <class>enterprise_customersegment/observer</class>
237                        <method>processEvent</method>
238                    </enterprise_customersegment>
239                </observers>
240            </customer_login>
```

When a customer logs in, the system triggers the `customer_login` event and runs the processEvent method of the customersegment observer. At this moment, the customer has already been verified and logged in with their customer ID. The observer in its turn, runs the processEvent method of `enterprise_customersegment/customer` model.

```
106         * @return  Enterprise_CustomerSegment_Model_Customer
107         */
108        public function processEvent($eventName, $customer, $website)
109        {
110            Varien_Profiler::start('__SEGMENTS_MATCHING__');
111            $website = Mage::app()->getWebsite($website);
112            $segments = $this->getActiveSegmentsForEvent($eventName, $website->getId());
113
114            $this->_processSegmentsValidation($customer, $website, $segments);
115
116            Varien_Profiler::stop('__SEGMENTS_MATCHING__');
117            return $this;
118        }
```

2.2 CUSTOMER LOGIN (CONT'D)

What happens next:

1. The getActiveSegmentsForEvent method collects all customer segments that have the `customer_login` event specified in the `enterprise_customersegment_event` table. Again, the list of events per segment in this table varies depending on segment conditions. The result is cached in a protected property. Here is the query which selects these events:

```
SELECT `main_table`.*
  FROM `enterprise_customersegment_segment` AS `main_table`
       INNER JOIN `enterprise_customersegment_event` AS `evt`
          ON main_table.segment_id = evt.segment_id
 WHERE     (evt.event = 'customer_login')
       AND (EXISTS
               (SELECT 1
                  FROM `enterprise_customersegment_website` AS `website`
                 WHERE     (website.website_id IN ('1'))
                       AND (main_table.segment_id = website.segment_id)))
       AND (is_active = '1')
```

2. The processEvent method also runs the `_processSegmentsValidation` method.

3. Magento checks for all segments registered for this event if their rules match the case for the current customer.

4. To do this, the validateCustomer method of the `Enterprise_CustomerSegment_Model_Segment` class takes the pre-composed SQL query of each segment saved in the `condition_sql` field of the `enterprise_customersegment_segment` table, adds the current customer ID to the condition, and executes the query.

5. If the segment's conditions match, its ID is stored in a local array.

6. Additionally, all segment IDs whose conditions did not match are also stored in another array.

7. The addCustomerToWebsiteSegments method inserts into the `enterprise_customersegment_customer` table all the segment IDs and the ID for a current customer (as well as the website ID, which keeps the flow in the scope of a website).

8. This method merges segment IDs that were already in that table with new ones.

9. Magento uses the array of non-matched segment IDs to remove them from the `enterprise_customersegment_customer` table and the resulting array of the matched segments, using the removeCustomerFromWebsiteSegments method.

   To add or remove customer segment IDs, the getCustomerSegmentIdsForWebsite method is used to retrieve active segments from the database.

2.2 CUSTOMER LOGIN (CONT'D)

As you can see, the system updates the `enterprise_customersegment_customer` table on the login step by adding and removing segment IDs for a current customer ID if customer segment condition rules are matched. This table is updated on the login step for each customer.

What is important to emphasize is that this scenario happens not only for a `customer_login` event, but for each and every event registered for the Customer Segment's processEvent method.

Here is a full list of events for the frontend:

1. `customer_login`
2. `sales_quote_save_commit_after`
3. `sales_order_save_commit_after`
4. `catalog_controller_product_view`
5. `checkout_cart_save_after`
6. `wishlist_items_renewed`
7. `newsletter_subscriber_save_commit_after`
8. `newsletter_subscriber_save_commit_after`
9. `visitor_init`

Here is a list of global events:

1. `customer_save_commit_after`
2. `customer_address_save_commit_after`
3. `customer_address_delete_commit_after`
4. `customer_balance_save_commit_after`

Some events can be fired on every, or almost every, page load and can trigger the preceding flow, resulting in performance loss without any benefit.

2.3 PRODUCT PURCHASE

When a product is added to a cart, at least two events are fired:

1. `sales_quote_save_commit_after`
2. `checkout_cart_save_after`

The sales_quote_save_commit_after event is called in every step of the checkout process, including placing an order.

On the place order step, at least four events are fired:

1. `customer_address_save_commit_after`
2. `customer_save_commit_after`
3. `sales_quote_save_commit_after`
4. `sales_order_save_commit_after`

As discussed earlier, the processEvent method runs for every event fired for a page. It collects all segments related to this event and executes each "condition SQL". After that, it updates the customer segment table by adding new and removing non-matched customers.

This scenario illustrates how a customer segment works in connection with the shopping cart price rule module (SalesRule). The CustomerSegment module was designed to facilitate promotions so that a merchant can target a dynamic group of customers.

Let's create a shopping cart price rule that provides a 10 percent discount for customers who have spent over $1,000.

First, create a customer segment as follows:

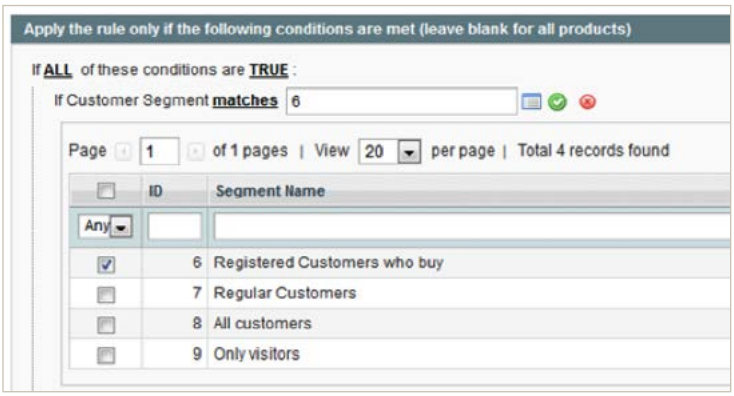
1. In the Magento Admin Panel, click **Customers** > **Customer Segments**.
2. In the upper right corner of the page, click **Add Segment**.
3. Follow the prompts on your screen to create the segment.
4. Click **Save and Continue Edit**.
5. From the left navigation bar, click **Conditions**.
6. Configure a condition as follows:

7. Save the customer segment and make a note of its ID.

2.3 PRODUCT PURCHASE (CONT'D)

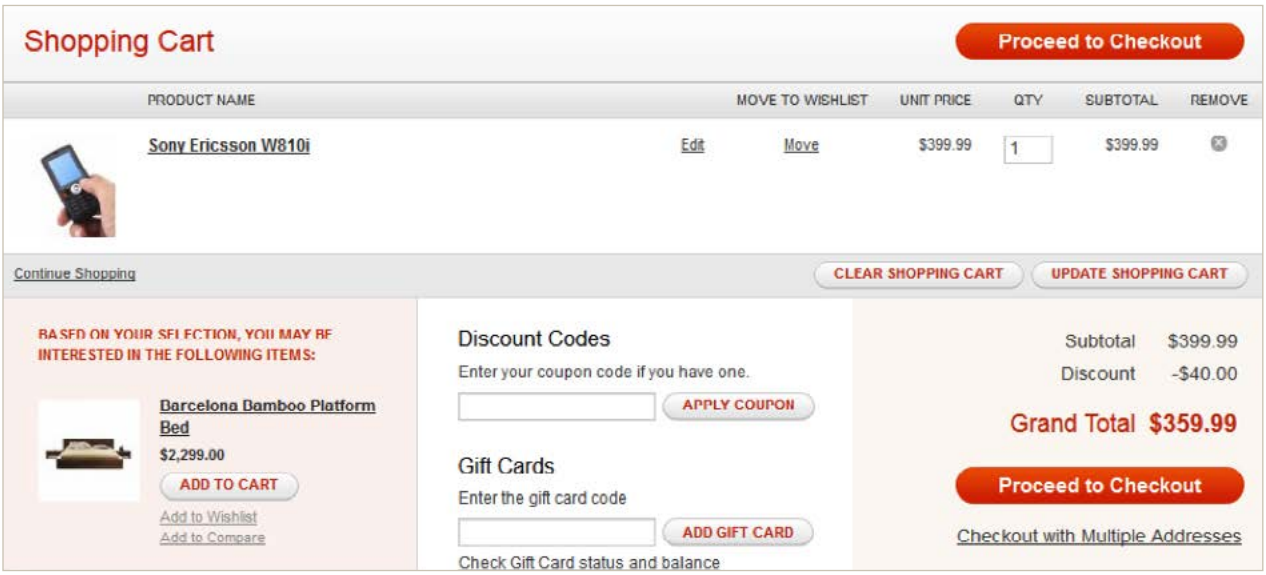
Next, create a shopping cart price rule that uses this customer segment:

1. In the Magento Admin Panel, click **Promotions > Shopping Cart Price Rules**.
2. In the upper right corner of the page, click **Add New Rule**.
3. Follow the prompts on your screen to enter the required information.
4. In the left navigation bar, click **Conditions**.
5. Enter conditions as follows:



6. Save the shopping cart price rule.

Any customer who is a member of the customer segment now gets a 10 percent discount, as follows:



This is how you can fully automate promotion behavior.

2.3 PRODUCT PURCHASE (CONT'D)

Looking at the Code

Now let's see how the SalesRule and CustomerSegment modules work at the code level.

When the system runs the collectTotals method of a quote object, it collects all registered collectors and runs their "collect" methods. The sequence is as follows:

1. `Mage_Sales_Model_Quote->collectTotals()`
2. `Mage_Sales_Model_Quote_Address->collectTotals()`
3. `Mage_SalesRule_Model_Quote_Freeshipping->collect()`

Here is a screenshot of the `Mage_Sales_Model_Quote_Address->collectTotals()`

```
1009    public function collectTotals()
1010    {
1011        Mage::dispatchEvent($this->_eventPrefix . '_collect_totals_before',
1012        foreach ($this->getTotalCollector()->getCollectors() as $model) {
1013            $model->collect($this);
1014        }
1015        Mage::dispatchEvent($this->_eventPrefix . '_collect_totals_after',
1016        return $this;
1017    }
```

The salesRule/freeshipping model is one of the registered collectors and comes up first.

```
132    <sales>
133        <quote>
134            <totals>
135                <freeshipping>
136                    <class>salesrule/quote_freeshipping</class>
137                    <after>subtotal</after>
138                    <before>tax_subtotal,shipping</before>
139                </freeshipping>
140                <discount>
141                    <class>salesrule/quote_discount</class>
142                    <after>subtotal,shipping</after>
143                    <before>grand_total</before>
144                </discount>
145            </totals>
```

Collectors are registered in config.xml files of different modules. Thereafter, the salesRule/freeshipping total collector is registered in the salesRule module's config.xml file.

2.3 PRODUCT PURCHASE (CONT'D)

As you step through `Mage_Sales_Model_Quote`->collectTotals() to `Mage_SalesRule_Model_Quote_Freeshipping`->collect(), five more steps happen in the SalesRule module classes:

4. `Mage_SalesRule_Model_Validator->processFreeShipping()`
5. `Mage_SalesRule_Model_Validator->_canProcessRule()`
6. `Mage_SalesRule_Model_Rule->validate()`
7. `Mage_SalesRule_Model_Rule_Condition_Combine->validate()`
8. `Enterprise_CustomerSegment_Model_Segment_Condition_Segment->validate()`

The last step invokes the Customer Segment module.

The system looks for segment IDs of a current customer by running the getCustomerSegmentIdsForWebsite method of the `Enterprise_CustomerSegment_Model_Customer` class.

```
148     public function validate(Varien_Object $object)
149     {
150         if (!Mage::helper('enterprise_customersegment')->isEnabled()) {...}
153         $customer = null;
154         if ($object->getQuote()) {...}
157         if (!$customer) {
158             return false;
159         }
160
161         $quoteWebsiteId = $object->getQuote()->getStore()->getWebsite()->getId();
162         if (!$customer->getId()) {...} else {
170             $segments = Mage::getSingleton('enterprise_customersegment/customer')->getCustomerSegmentIdsForWebsite(
171                 $customer->getId(),
172                 $quoteWebsiteId
173             );
174         }
175         return $this->validateAttribute($segments);
176     }
```

The `SalesRule_Model_Validator` in the canProcessRule method saves the result of validation in the setIsValidForAddress magic method.

```
174                 if ($coupon->getId()) {
175                     // check entire usage limit
176                     if ($coupon->getUsageLimit() && $coupon->getTimesUsed() >= $coupon->getUsageLimit()) {
177                         $rule->setIsValidForAddress($address, false);
178                         return false;
179                     }
180                     // check per customer usage limit
181                     $customerId = $address->getQuote()->getCustomerId();
182                     if ($customerId && $coupon->getUsagePerCustomer()) {
183                         $couponUsage = new Varien_Object();
184                         Mage::getResourceModel('salesrule/coupon_usage')->loadByCustomerCoupon(
185                             $couponUsage, $customerId, $coupon->getId());
186                         if ($couponUsage->getCouponId() &&
187                             $couponUsage->getTimesUsed() >= $coupon->getUsagePerCustomer()
188                         ) {
189                             $rule->setIsValidForAddress($address, false);
190                             return false;
191                         }
192                     }
193                 }
```

2.3 PRODUCT PURCHASE (CONT'D)

When you get to the model `Mage_SalesRule_Model_Quote_Discount`, which is actually used for the shopping cart price rule, the "process" method of the `Mage_SalesRule_Model_Validator` class executes.

```
97          if ($item->getHasChildren() && $item->isChildrenCalculated()) {
98              foreach ($item->getChildren() as $child) {
99                  $this->_calculator->process($child);
100                 $eventArgs['item'] = $child;
101                 Mage::dispatchEvent('sales_quote_address_discount_item', $eventArgs);
102
103                 $this->_aggregateItemDiscount($child);
104             }
105         } else {
106             $this->_calculator->process($item);
107             $this->_aggregateItemDiscount($item);
108         }
```

You're returned to the `_canProcessRule` method of that class, which is called in a loop, but will not check again whether a customer belongs to a customer segment because it has cached the result.

```
240     foreach ($this->_getRules() as $rule) {
241         /* @var $rule Mage_SalesRule_Model_Rule */
242         if (!$this->_canProcessRule($rule, $address)) {
243             continue;
244         }
245
246         if (!$rule->getActions()->validate($item)) {
247             continue;
248         }
```

This scenario illustrates touch points between promotion modules such as SalesRule and the CustomerSegment. To apply rules of the SalesRule module that are related to the CustomerSegment, Magento uses data of the `enterprise_customersegment_customer` table by calling the `Enterprise_CustomerSegment_Model_Segment_Condition_Segment`::validate() method. (See step 8 earlier.)

Note that this table was verified and updated twice in this flow, when the `sales_quote_save_commit_after` and `checkout_cart_save_after` events were fired.

3 PERFORMANCE IMPACT

The data that follows was tested on a desktop computer with the following configuration:

CPU Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 8Cores

RAM: 8 GB

OS: Debian GNU/Linux 7.1 (wheezy)

Web Server: nginx/1.4.2

PHP-FPM: PHP 5.4.17-1~dotdeb.1 with Zend OPcache v7.0.2

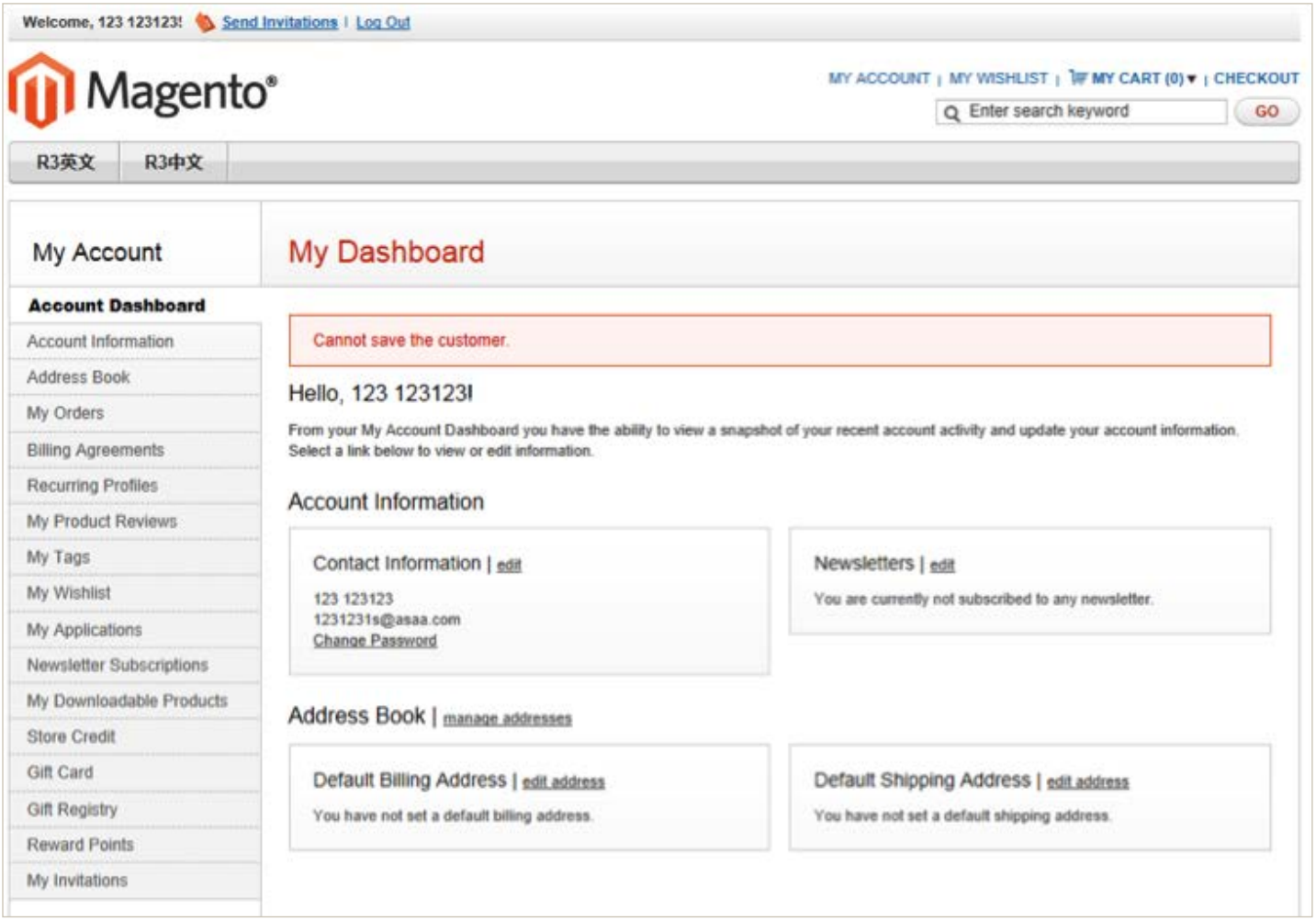
MySQL: Percona Server 5.6.12-rc60.4

Magento: EE-1.13.0.2

Impacts of New Segment Creation

For web stores with more than one million customers, the creation of a new customer segment can take a while. In our system, we had 12,300,004 customers distributed across 34 websites. On a particular website with 5,100,004 customers, **more than 10 minutes** were required to create an empty customer segment.

While it was being created, customers who tried to register saw the "Cannot save the customer" error. However, the customer was actually created and logged in as follows.



3 PERFORMANCE IMPACT (CONT'D)

The reason for the message is that the system was trying to insert a new record into a customer segment-related table, but timed out.

```
SQL: INSERT INTO `enterprise_customersegment_customer` (`segment_id`,`customer_id`,`added_date`,`updated_date`,`website_id`) VALUES (?, ?, ?, ?, ?) ON DUPLICATE KEY UPDATE updated_date = VALUES(`updated_date`)
BIND: array (
  0 => '4',
  1 => '12301104',
  2 => '2014-04-08 18:25:17',
  3 => '2014-04-08 18:25:17',
  4 => '1',
)
TIME: 51.0040
'PDOException' with message 'SQLSTATE[HY000]: General error: 1205 Lock wait timeout exceeded
```

Recommendation

It's better not to create new rules during peak business hours so that customers won't experience this error message. At the same time, a Magento administrator can experience problems updating an existing customer during the process of creating a customer segment under the same conditions.

The administrator might see the same error but the reasons are a little trickier because now it's the `customer_entity` table that is locked:

```
UPDATE `customer_entity` SET `entity_type_id` = ?, `attribute_set_id` = ?, `website_id` = ?, `email` = ?, `group_id` = ?, `increment_id` = ?, `store_id` = ?, `created_at` = '2013-09-20 16:26:21', `updated_at` = '2014-04-08 18:26:37', `disable_auto_group_change` = ? WHERE (`entity_id`=102)
BIND: array (
  0 => 1,
  1 => 0,
  2 => 1,
  3 => 'customer102@example.com',
  4 => 2,
  5 => NULL,
  6 => 0,
  7 => 0,
)
TIME: 51.0039
'PDOException' with message 'SQLSTATE[HY000]: General error: 1205 Lock wait timeout exceeded;
```

3 PERFORMANCE IMPACT (CONT'D)

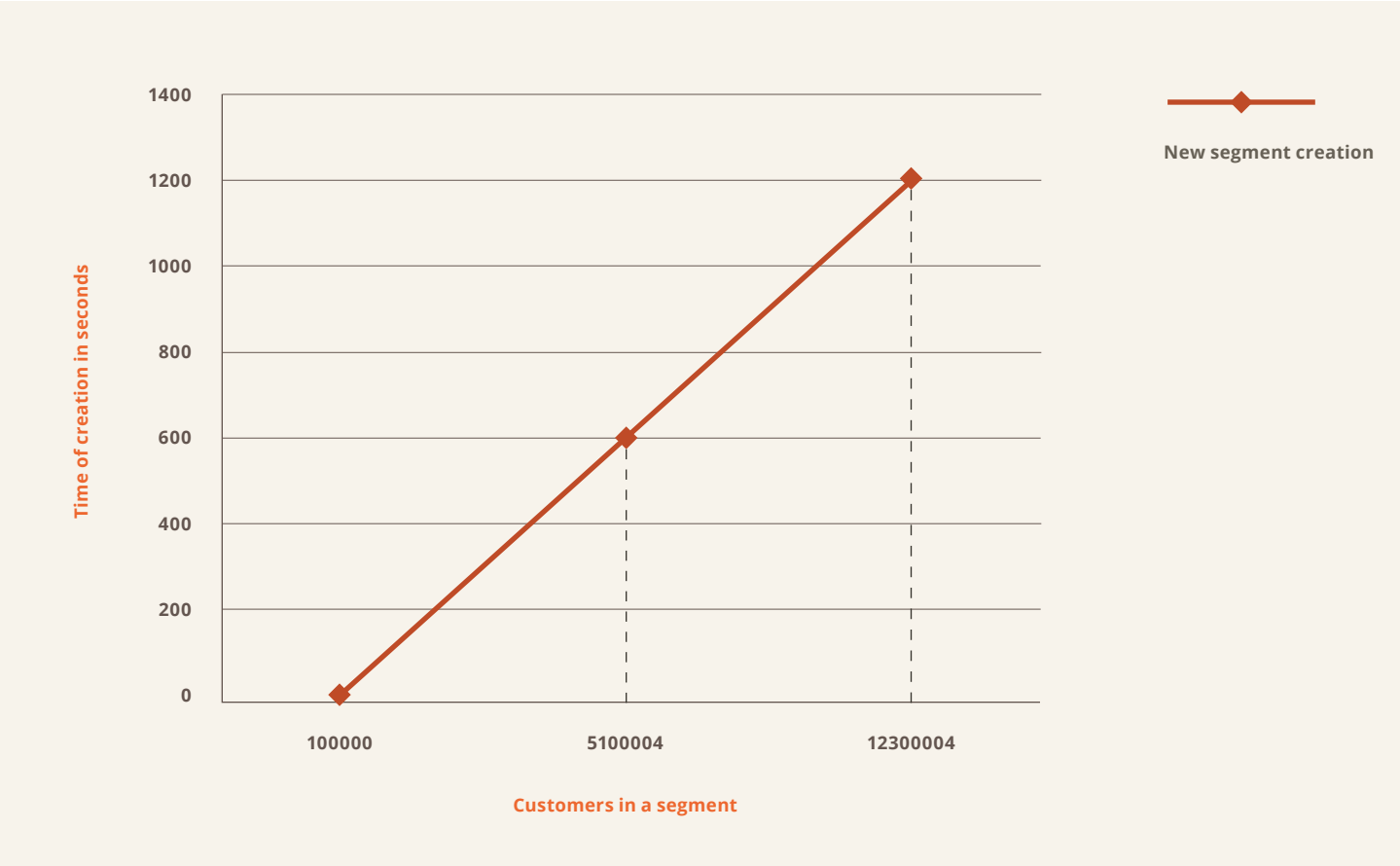
By contrast, we tested a website with 100,000 customers and found that it takes only a few seconds to complete. Long delays start when millions of customers are involved. Creating an empty segment for all 34 websites which involves all 12,300,004 customers took approximately 20 minutes. In that system, not only did errors display when creating or updating customers, but overall response time was slower.

To log in to the website, a customer must wait until after the timeout because Enterprise Edition tries to insert a new record into the `enterprise_customersegment_customer` table, which is locked. Eventually, a customer might be able to log in if the setting of MySQL `innodb_lock_wait_timeout` variable is short enough (default is 50 seconds), with no error message shown, but that exception still happens in the background:

```
SQL: INSERT INTO `enterprise_customersegment_customer` (`segment_id`,`customer_id`,`added_
date`,`updated_date`,`website_id`) VALUES (?, ?, ?, ?, ?) ON DUPLICATE KEY UPDATE updated_date
= VALUES(`updated_date`)

TIME: 51.0040

'PDOException' with message 'SQLSTATE[HY000]: General error: 1205 Lock wait timeout exceeded
```



3 PERFORMANCE IMPACT (CONT'D)

Impact of Customer Segments with Empty Rule Sets

To understand the performance impact of customer segments, we'll first look at seven time-consuming queries that are performed while a customer tries to log in. We created seven customer segments with empty rule sets.

```
## 2014-04-09 15:40:11
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2071
```

```
## 2014-04-09 15:40:14
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2044
```

```
## 2014-04-09 15:40:16
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2041
```

```
## 2014-04-09 15:40:18
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2039
```

```
## 2014-04-09 15:40:20
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2038
```

```
## 2014-04-09 15:40:23
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2212
```

```
## 2014-04-09 15:40:25
## 6145 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root`
AFF: 12300004
TIME: 2.2047
```

3 PERFORMANCE IMPACT (CONT'D)

The preceding queries select all 12 million customers each. In all, the queries take more than 15 seconds with no concurrency. Based on this, we strongly recommend you **do not create empty customer segments** on your production servers. When a simple rule is added to the customer segment, the queries run much faster:


```
## 2014-04-09 17:53:45
## 6460 ## QUERY
SQL: SELECT 1 FROM `customer_entity` AS `root` WHERE ((IF((SELECT 1 FROM `customer_entity_int` AS `main` WHERE
(main.entity_id = :customer_id) AND (main.attribute_id = '18') AND (`main`.`value` = '1') LIMIT 1), 1, 0) = 1))
BIND: array (
 ':customer_id' => '12301101',)
AFF: 0
TIME: 0.0004
```

4 GUIDELINES

If you have more than 100,000 customers in your production database, you must pay careful attention to how you use customer segments. As we discussed earlier, when customers log in to your storefront, it generates activity in the database related to applying promotional rules and assigning the customer to a customer segment. This fast and dynamic approach can lead to dramatic bottlenecks if you're not careful.

We recommend you observe the following guidelines:

1. The Customer Segment module uses dozens of events to update customer IDs in a corresponding table. To reduce system load, make sure your custom solution actually needs those Customer Segment events. For example, consider disabling the `wishlist_items_renewed` event if you're not using the wishlist on your site.

2. On projects with a large customer database, creating an empty segment produces high system load, without considering associated promotional price rules. When you create an empty customer segment, all customers in that website are added to it! Be careful and don't do this during peak traffic times.

3. A large number of customer segments can produce high system load and significant delay for actions with Customer Segment events, such as user login, logout, and checkout operations. Keep in mind that Magento collects all related segments and executes their `condition_sql` query, adding and deleting customer IDs in the Customer Segment customer table. Be careful with the number of segments on websites containing a large customer database.

This module can still be used with large quantities of data, but it may need to be customized in order to avoid performance issues. The first thing that can be done is to change its behavior to update segment content (the list of qualifying customers by rules) asynchronously. This could be performed by cron instead of using real-time events. It applies to both the front end (i.e. customer login, etc.) and the back end (i.e. new segment creation, etc.) activities.

Another method would be to change the flow of new segment creation so that it does not populate the segment with all of the available customers during its creation. Additional intelligent logic could be created to completely ignore segments that have no rules, so that they're not updated unnecessarily.

About Magento Expert Consulting Group (ECG)

Magento's Expert Consulting Group (ECG) helps Magento Enterprise Edition merchants and Solution Partners maximize their success. Our experts offer comprehensive analysis and best practice recommendations, from architecture planning through post-deployment.

For more expert articles from Magento ECG, visit:
magentocommerce.com/consulting/expert-articles