Product-Customer Matrix Prices

PDF Manual

MODULE

productcustomermatrix

LAST UPDATED

2026-03-29

Table of Contents

Product-Customer Matrix Prices · productcustomermatrix

Product-Customer Matrix Prices

Product-Customer Matrix Prices · /productcustomermatrix

Advanced pricing system with bidirectional product-customer matching, multi-attribute logic, and flexible assignment rules.

Overview

Product-Customer Matrix Prices enables sophisticated pricing scenarios where prices are determined by matching **products** against **customer attributes** through configurable matrices. Unlike simple pricelists, matrices support:

- **Bidirectional matching:** Products → Customer attributes OR Customer attributes → Products
- **Multi-attribute logic:** Combine multiple conditions with AND/OR operators
- **Flexible assignment:** Automatic attribute-based matching PLUS manual customer overrides
- **Priority resolution:** Conflict resolution when multiple matrices match
- **Quantity tiers:** Multiple price breakpoints per matrix
- **Time-based validity:** Dual date systems for matrix-level and customer-specific scheduling

Quick Start

1. **Install extension:** `composer require mageb2b/pricesystem-productcustomermatrix:*`
2. **Configure settings:** Stores > Configuration > Pricesystem > Product-Customer Matrix
3. **Create matrix:** Catalog > Product-Customer Matrices > Add New
4. **Set matching rules:** Define customer attributes (group, company, region, etc.)
5. **Add products:** Select products and set prices with quantity tiers
6. **Assign customers:** Automatic (via attributes) or manual assignment
7. **Activate matrix:** Set is_active = Yes

Key Features

Matrix Pricing System

Create pricing matrices that match products to customers based on flexible rules. Each matrix acts as a container with:

- Product selection (specific SKUs or attribute-based)
- Customer matching rules (group, company, location attributes)
- Priority for conflict resolution
- Date range validity
- Quantity-based pricing tiers

Learn more →

Multi-Attribute Matching

Combine multiple customer attributes with AND/OR logic for precise targeting:

- Customer Group + Region + Company (AND logic)

- Multiple Groups (OR logic within attribute)
- Complex scenarios: (Group A OR Group B) AND (Region X) AND (Company Y)
- Supports: group, company, tax/VAT, postcode, region, country

Learn more →

Priority-Based Resolution

When customers match multiple matrices, priority system determines which pricing applies:

- Priority field: 0-999 (higher wins)
- Merge option: Best price across all matrices OR highest priority only
- Clear hierarchy: Standard < Promotional < VIP < Contract
- Date-based priority shifts for campaigns

Learn more →

Common Use Cases

1. Contract Pricing

Scenario: ACME Corp negotiated specific pricing for 50 products, valid through 2025.

Setup:

- Matrix: "Contract - ACME Corp 2025"
- Company match: "ACME"
- Priority: 35 (contract level)
- Dates: 2025-01-01 to 2025-12-31
- Products: 50 contracted SKUs with negotiated prices

Result: All ACME employees get contract pricing automatically.

2. Regional Wholesale

Scenario: California wholesale customers get special pricing on California-specific products.

Setup:

- Matrix: "CA Wholesale 2025"
- Customer Group: Wholesale (ID 2)
- Region: "California"
- Country: "US"
- Products: California-themed product category
- Priority: 20

Result: Only wholesale customers in California see these prices.

3. VIP Customer Program

Scenario: Top customers (VIP group) get 15% discount on all products, plus extra discounts on selected products.

Setup:

- Matrix 1: "VIP Base Pricing" (Priority 30) - All products, 15% off
- Matrix 2: "VIP Extra Savings" (Priority 32) - Selected products, 25% off
- Customer Group: VIP (ID 4)
- Merge: Yes (best price wins)

Result: VIP customers get 15% off everything, 25% off selected items.

4. Seasonal Campaign with Exclusions

Scenario: Summer sale for all customers except wholesale (they have better pricing already).

Setup:

- Matrix: "Summer Sale 2025"
- Attributes: NOT Group = Wholesale (using exclusion logic)
- Dates: 2025-06-01 to 2025-08-31
- Priority: 22 (promotional)
- Products: Summer collection

Result: Retail customers get sale pricing, wholesale keeps standard (better) pricing.

5. Multi-Company B2B Pricing

Scenario: 5 partner companies each have different pricing for shared product catalog.

Setup:

- Matrix 1: "Partner - Company A" (Company = "A Corp", Priority 30)
- Matrix 2: "Partner - Company B" (Company = "B Corp", Priority 30)
- Matrix 3: "Partner - Company C" (Company = "C Corp", Priority 30)
- Same products, different prices per matrix
- Individual customer assignments for employees

Result: Each company's employees see their negotiated pricing automatically.

6. Quantity-Based Wholesale Tiers

Scenario: Wholesale customers get better pricing at higher quantities, retail customers don't.

Setup:

- Matrix: "Wholesale Bulk Discounts"
- Customer Group: Wholesale
- Products: All products with qty tiers:
    - Qty 1-9: $100
    - Qty 10-49: $95
    - Qty 50-99: $90
    - Qty 100+: $85

Result: Wholesale customers see graduated pricing; retail sees standard pricing.

Architecture

Database Tables

pricesystem_product_customer_matrix: Main matrix container with priority, dates, and matching configuration.

pricesystem_product_customer_matrix_attribute: Customer attribute matching rules with AND/OR logic support.

pricesystem_product_customer_matrix_customer: Manual customer assignments with optional date overrides.

pricesystem_product_customer_matrix_product: Product-price mappings with quantity tiers (inherited from pricesystem_pricelist_product).

Price Resolution Order

1. **Collect matching matrices:** Evaluate customer attributes against all active matrices
2. **Check date validity:** Matrix dates + customer-specific date overrides
3. **Apply priority:** Sort by priority (highest first)
4. **Merge or select:** Based on `merge_matrix_qtys` configuration
5. **Quantity matching:** Find best price for cart quantity
6. **Return price:** Final price for customer + product + quantity

Configuration

System Settings

Path: Stores > Configuration > Pricesystem > Product-Customer Matrix

Key Settings:

Enable Auto-Assign Customers:

- Evaluates customer attributes automatically on login
- Required for attribute-based matching
- Default: Yes

Match Exact Type:

- Loose: Partial string matching (e.g., "ACME" matches "ACME Corp")
- Exact: Strict equality (case-sensitive)
- Default: Loose

Merge Matrix Quantities:

- No: Use highest priority matrix only
- Yes: Combine qty tiers from all matching matrices, best price wins
- Default: No

Enable Matrix Validation:

- Prevents duplicate priorities
- Enforces attribute rule logic
- Default: Yes

Integration with Pricesystem

Product-Customer Matrix extends Magento 2 Pricesystem Core and integrates with:

- **Base Pricesystem:** Inherits core pricing architecture
- **Pricelists:** Matrices and pricelists can coexist (priority determines winner)
- **Category Prices:** Category-level pricing can combine with matrix pricing
- **Customer-Specific Prices:** Direct customer prices have highest precedence

Price Resolution Order (full system):

1. Shopping cart price rules (if enabled)
2. Customer-specific prices
3. Product-Customer Matrices (this extension)
4. Pricelists
5. Category prices
6. Catalog price

Performance Considerations

- **Indexing:** Matrices are evaluated on customer login and profile updates
- **Caching:** Matrix assignments cached per customer session
- **Scalability:** System handles thousands of matrices and millions of products
- **Optimization:** Use specific attributes (group + country) rather than company matching for large customer bases

Getting Started

Ready to create your first matrix?

1. Installation Guide - Install and configure the extension
2. Creating Matrices - Step-by-step matrix creation guide
3. Price Selection - How Pricesystem selects the final price when multiple price types match
4. My Prices - Customer account price overview + CSV download

Add-Ons

Extend functionality with these add-ons:

- **SOAP / REST API Add-On** - CRUD WebAPI endpoints
- **CSV Import/Export Add-On** - Bulk import/export via CSV
- **Advanced Config Add-On** - Price selection overrides, custom sort order, formula pricing
- **Sample Data** - Demo entities and example prices

Feature Documentation

- Matrix Pricing System - Core matrix architecture and workflow
- Multi-Attribute Matching - Customer attribute logic with AND/OR operators
- Priority Resolution - Conflict resolution and merge strategies

Installation & Configuration

Product-Customer Matrix Prices · /productcustomermatrix/getting-started/installation

Complete installation guide and configuration reference for Product-Customer Matrix Prices extension.

Requirements

System Requirements

- **Magento:** 2.4.x
- **PHP:** 8.1+
- **MySQL/MariaDB:** 5.7+ / 10.3+
- **Composer:** 2.x

Dependencies

Required:

- `mageb2b/pricesystem-core` - Base pricesystem architecture
- `magento/framework` - Magento core framework

Optional (Recommended):

- `mageb2b/pricesystem-pricelist` - Named pricelists (for integration scenarios)
- `mageb2b/pricesystem-categoryprice` - Category-based pricing

Installation

Step 1: Install via Composer

```
composer config bearer.repo.softwaresilo.io <token>
composer config repositories.softwaresilo composer
https://repo.softwaresilo.io/
composer require mageb2b/pricesystem-productcustomermatrix:*
```

Step 2: Enable Extension

```
php bin/magento module:enable MageB2B_PricesystemProductCustomerMatrix
```

Step 3: Run Setup

```
php bin/magento setup:upgrade
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy -f
```

Step 4: Clear Cache

```
php bin/magento cache:clean
php bin/magento cache:flush
```

Step 5: Verify Installation

Check that tables were created:

```
php bin/magento db:status
```

Expected tables:

- `pricesystem_product_customer_matrix`
- `pricesystem_product_customer_matrix_attribute`
- `pricesystem_product_customer_matrix_customer`

Admin menu should show: **Catalog > Product-Customer Matrices**

Configuration

Configuration Path

Admin Panel > Stores > Configuration > Pricesystem > Product-Customer Matrix

---

Core Settings

Enable Matrix Pricing

Path: `pricesystem/productcustomermatrix/enabled`

Options:

- **Yes (Default):** Matrix pricing active
- **No:** Matrix pricing disabled (fallback to other pricesystem modules)

Effect:

- When disabled, all matrices ignored regardless of active status
- Useful for testing or temporary disablement

Enable Auto-Assign Customers

Path: `pricesystem/productcustomermatrix/enable_auto_assign_customers`

Options:

- **Yes (Recommended):** Automatic customer assignment based on attributes
- **No:** Manual assignment only

How It Works:

When Enabled:

1. Customer logs in
2. System evaluates all active matrices
3. Matches customer attributes against matrix rules
4. Auto-assigns customer to matching matrices

When Disabled:

- Only manual assignments work (via Customers tab in matrix edit)
- Attribute fields on matrix are ignored

Use Case:

- Enable: Automatic, scalable pricing
- Disable: Full manual control (small customer bases)

Match Exact Type

Path: `pricesystem/productcustomermatrix/match_exact_type`

Options:

- **Loose (No - Default):** Partial string matching, case-insensitive
- **Exact (Yes):** Strict equality, case-sensitive

Examples:

Loose Matching:

```
Matrix Company: "ACME"
Customer Company: "ACME Corporation" → MATCH (contains "ACME")
Customer Company: "acme corp" → MATCH (case-insensitive)
```

Exact Matching:

```
Matrix Company: "ACME"
Customer Company: "ACME Corporation" → NO MATCH (not exact)
Customer Company: "ACME" → MATCH (exact)
Customer Company: "acme" → NO MATCH (case-sensitive)
```

Recommendation:

- Use **Loose** for flexible matching (handles typos, variations)
- Use **Exact** for security-sensitive scenarios (tax IDs, legal entity names)

Merge Matrix Quantities

Path: `pricesystem/productcustomermatrix/merge_matrix_qtys`

Options:

- **No (Default):** Use highest priority matrix only
- **Yes:** Merge quantity tiers from all matching matrices

Detailed Explanation:

No (Highest Priority Only)

When customer matches multiple matrices, use ONLY the highest priority matrix.

Example:

```
Customer matches:
- Matrix A (Priority 10): Product X @ qty=1 ($100), qty=10 ($95)
- Matrix B (Priority 20): Product X @ qty=1 ($98), qty=50 ($90)

Result: Use ONLY Matrix B (higher priority)
- qty=1: $98
- qty=50: $90
(Matrix A completely ignored)
```

Use Case: Simple, predictable pricing. One matrix wins.

Yes (Merge Tiers)

Combine quantity tiers from ALL matching matrices, taking best price at each tier.

Example:

```
Customer matches:
- Matrix A (Priority 10): Product X @ qty=1 ($100), qty=10 ($95)
- Matrix B (Priority 20): Product X @ qty=1 ($98), qty=50 ($90)

Result: MERGE tiers, best price wins:
- qty=1: $98 (from Matrix B, better than A's $100)
- qty=10: $95 (from Matrix A, B has no qty=10)
- qty=50: $90 (from Matrix B, A has no qty=50)

Customer gets: qty=1 ($98), qty=10 ($95), qty=50 ($90)
Best of both matrices!
```

Use Case: Complex pricing scenarios, best-of-all-matrices strategy.

Enable Matrix Validation

Path: `pricesystem/productcustomermatrix/enable_matrix_validation`

Options:

- **Yes (Default):** Strict validation on matrix save
- **No:** Allow potentially conflicting configurations

Validations:

1. **Duplicate priorities:** Warn when same priority exists (same website)
2. **Empty attributes:** Require at least one matching attribute OR manual customer assignment
3. **Date logic:** Ensure from_date <= to_date
4. **Attribute relation:** Validate attributes_relation field (AND/OR)

Recommendation: Keep enabled to prevent configuration errors.

Attribute Relation Default

Path: `pricesystem/productcustomermatrix/default_attributes_relation`

Options:

- **AND (Default):** All filled attributes must match
- **OR:** Any filled attribute matches

Example:

AND Logic:

```
Matrix:
- Customer Group: Wholesale (2)
- Country: US

Customer:
- Group: Wholesale ✓
- Country: US ✓
Result: MATCH (both match)

Customer:
- Group: Wholesale ✓
- Country: DE ✗
Result: NO MATCH (country doesn't match)
```

OR Logic:

```
Matrix:
- Customer Group: Wholesale (2)
- Country: US
```

```
Customer:
- Group: Retail (1)
- Country: US ✓
Result: MATCH (country matches, group doesn't matter)

Customer:
- Group: Wholesale ✓
- Country: DE
Result: MATCH (group matches, country doesn't matter)
```

Use Case:

- **AND:** Precise targeting (Wholesale + US + California)
- **OR:** Broad targeting (Wholesale OR VIP customers)

Advanced Settings

Cache Matrix Assignments

Path: `pricesystem/productcustomermatrix/cache_assignments`

Options:

- **Yes (Default):** Cache matrix-customer assignments per session
- **No:** Re-evaluate on every request

Performance Impact:

- Enabled: Faster, recommended for production
- Disabled: Slower, useful for testing/debugging

Priority Hierarchy

Path: `pricesystem/productcustomermatrix/priority_tiers`

Document your priority hierarchy for team reference.

Recommended Hierarchy:

```
0-9: Base/Fallback
10-19: Standard pricing
20-29: Promotional pricing
30-39: VIP/Contract pricing
40-49: Emergency overrides
50+: System overrides
```

Not a system setting, but document internally.

Database Schema

Table: pricesystem_product_customer_matrix

Main matrix container.

```
CREATE TABLE pricesystem_product_customer_matrix (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  is_active TINYINT DEFAULT 1,
  priority INT DEFAULT 0,
  from_date DATE DEFAULT NULL,
  to_date DATE DEFAULT NULL,
  website_id INT NOT NULL,
  attributes_relation VARCHAR(3) DEFAULT 'AND', -- 'AND' or 'OR'

  created_at TIMESTAMP,
  updated_at TIMESTAMP,

  FOREIGN KEY (website_id) REFERENCES store_website(website_id),
  INDEX idx_priority (priority),
  INDEX idx_active_dates (is_active, from_date, to_date)
);
```

Table: pricesystem_product_customer_matrix_attribute

Customer attribute matching rules.

```
CREATE TABLE pricesystem_product_customer_matrix_attribute (
  id INT PRIMARY KEY AUTO_INCREMENT,
  matrix_id INT NOT NULL,
  attribute_code VARCHAR(50) NOT NULL, -- 'group', 'company', 'tax',
'postcode', 'region', 'country'
  attribute_value VARCHAR(255) NOT NULL,

  FOREIGN KEY (matrix_id) REFERENCES
pricesystem_product_customer_matrix(id) ON DELETE CASCADE,
  INDEX idx_matrix_code (matrix_id, attribute_code)
);
```

Supported Attributes:

- `group` - Customer group ID
- `company` - Company name
- `tax` - Tax/VAT number
- `postcode` - ZIP/postal code
- `region` - State/province/region
- `country` - Country code (ISO 2-letter)

Table: pricesystem_product_customer_matrix_customer

Manual customer assignments with optional date overrides.

```
CREATE TABLE pricesystem_product_customer_matrix_customer (
  id INT PRIMARY KEY AUTO_INCREMENT,
  matrix_id INT NOT NULL,
  customer_id INT NOT NULL,
  from_date DATE DEFAULT NULL, -- Override matrix from_date
  to_date DATE DEFAULT NULL,   -- Override matrix to_date

  FOREIGN KEY (matrix_id) REFERENCES
pricesystem_product_customer_matrix(id) ON DELETE CASCADE,
  FOREIGN KEY (customer_id) REFERENCES customer_entity(entity_id) ON DELETE
CASCADE,
  UNIQUE KEY uk_matrix_customer (matrix_id, customer_id),
  INDEX idx_customer (customer_id)
);
```

Date Override Logic:

- If customer-specific dates set: Use those dates
- If not set: Use matrix-level dates
- Allows per-customer date customization within same matrix

Product-Price Relationship

Uses shared table from pricesystem core:

```
-- Inherited from pricesystem_pricelist_product
-- Matrix uses same structure via polymorphic relationship
CREATE TABLE pricesystem_pricelist_product (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pricelist_id INT NOT NULL, -- Maps to matrix_id for matrices
  product_id INT NOT NULL,
  qty DECIMAL(10,2) DEFAULT 1.00,
  price DECIMAL(20,4) NOT NULL,
  from_date DATE DEFAULT NULL,
  to_date DATE DEFAULT NULL,

  FOREIGN KEY (product_id) REFERENCES catalog_product_entity(entity_id) ON
DELETE CASCADE,
  INDEX idx_pricelist_product_qty (pricelist_id, product_id, qty)
);
```

Post-Installation Checklist

After installation, verify:

1. ✓ **Tables created:** Check database for 3 matrix tables
2. ✓ **Admin menu:** Catalog > Product-Customer Matrices visible
3. ✓ **Configuration:** All settings appear in Stores > Configuration
4. ✓ **Permissions:** Admin users can access matrix grid/edit
5. ✓ **Cache cleared:** Flush all caches
6. ✓ **Compilation:** DI compiled without errors

Troubleshooting

Extension Not Visible in Admin

Check:

1. Module enabled? `php bin/magento module:status`
2. Cache cleared? `php bin/magento cache:flush`
3. Admin permissions? Check role for matrix resources

Tables Not Created

Check:

1. Setup run? `php bin/magento setup:upgrade`
2. Database errors? Check `var/log/system.log`
3. Permissions? Database user needs CREATE TABLE rights

Configuration Not Saving

Check:

1. Config cache cleared? `php bin/magento cache:clean config`
2. Database writable? Check `core_config_data` table
3. Errors in logs? Check `var/log/exception.log`

Next Steps

- Creating Matrices - Step-by-step guide to creating your first matrix
- Matrix Pricing System - Understanding core matrix architecture

Creating Matrices

Product-Customer Matrix Prices · /productcustomermatrix/getting-started/creating-matrices

Step-by-step guide to creating product-customer pricing matrices with attribute matching and customer assignments.

Quick Start

1. **Catalog > Product-Customer Matrices** → Click "Add New"
2. Fill basic info (name, priority, dates, attributes relation)
3. Configure customer matching attributes
4. Save matrix
5. Add products with prices and quantity tiers
6. Assign customers (automatic via attributes OR manual)
7. Activate matrix

Creating a Matrix Container

Step 1: Navigate to Grid

Admin Panel > Catalog > Product-Customer Matrices

Grid shows existing matrices with:

- Name
- Active status
- Priority
- Attributes Relation (AND/OR)
- Date range
- Website
- Number of products
- Number of assigned customers
- Actions

Step 2: Add New Matrix

Click **Add New** button.

Step 3: Fill Basic Information

Name (Required)

Descriptive name for the matrix.

Examples:

- "Wholesale US 2025"
- "VIP Contract - ACME Corp"
- "California Regional Pricing"

- "Black Friday Campaign"
- "Partner Pricing - Company A"

Best Practices:

- Include year for time-bound pricing
- Include customer segment/tier
- Avoid generic names ("Test", "Matrix 1")

Is Active (Required)

Toggle matrix on/off.

- **Yes:** Matrix evaluated for customer matching
- **No:** Matrix ignored, even if customers match attributes

Use Case: Prepare matrices in advance, activate when ready.

Priority (Optional)

Numeric value (0-999) for conflict resolution.

- **Default:** 0
- **Higher = precedence**
- **Examples:**
  - 10: Standard pricing
  - 20: Promotional pricing
  - 30: VIP/contract pricing
  - 40: Emergency overrides

Leave gaps (10, 20, 30) not consecutive (10, 11, 12) to allow insertions later.

From Date (Optional)

Start date for matrix validity.

- **Format:** YYYY-MM-DD
- **NULL:** Active immediately
- **Example:** `2025-06-01` (Summer campaign start)

To Date (Optional)

End date for matrix validity.

- **Format:** YYYY-MM-DD
- **NULL:** No expiration
- **Must be >= From Date**
- **Example:** `2025-08-31` (Summer campaign end)

Website (Required)

Website scope for matrix.

- **Default:** Default website
- **Multi-website:** Select specific site
- **Effect:** Only customers on this website match

Attributes Relation (Required)

Logic for combining multiple attributes.

- **AND (Default):** ALL filled attributes must match
- **OR:** ANY filled attribute matches

AND Example:

```
Attributes:
- Customer Group: Wholesale (2)
- Country: US
- Region: California

Customer must be: Wholesale AND US AND California
```

OR Example:

```
Attributes:
- Customer Group: Wholesale (2)
- Customer Group: VIP (4)

Customer must be: Wholesale OR VIP
```

Best Practice: Use AND for precise targeting, OR for broad targeting.

Customer Matching Attributes

These fields enable **automatic customer assignment**. When customers match attributes, they're auto-assigned to matrix.

Important: Attributes work together based on `attributes_relation` field (AND/OR).

Customer Group

Match by customer group ID.

Field: Multiple select dropdown

Example:

- Select "Wholesale" (ID 2) and "VIP" (ID 4)
- **AND mode:** Customer must be in BOTH groups (impossible - customers belong to one group)
- **OR mode:** Customer can be Wholesale OR VIP (correct usage)

Use Case:

- OR mode: Multiple groups (Wholesale, VIP, Partner)
- AND mode: Single group + other attributes

Company

Match by company name from customer account.

Field: Text input

Example: "ACME Corporation"

Matching:

- **Loose mode:** "ACME" matches "ACME Corp", "acme corporation"
- **Exact mode:** Must be exactly "ACME Corporation"

Use Case: B2B contract pricing for specific companies

Tax/VAT Number

Match by tax identification number.

Field: Text input

Example: "DE123456789"

Use Case:

- EU VAT-registered businesses
- Tax-exempt organizations
- Verified business accounts

Notes:

- Usually exact match (even in loose mode)
- Format-sensitive (include dashes/spaces as stored)

Postcode

Match by billing or shipping postcode.

Field: Text input

Example: "90210"

Use Case:

- ZIP code-based regional pricing
- Local delivery zones

Notes:

- Checks both billing AND shipping addresses
- Match if EITHER address matches
- Loose mode: "9021" matches "90210"

Region

Match by state/province/region.

Field: Text input

Example: "California"

Use Case:

- State-specific pricing
- Regional campaigns
- Tax jurisdictions

Data Quality:

- Magento stores region as text (not ID)
- "California" vs "CA" might not match
- Standardize or use loose matching

Country

Match by country code.

Field: Text input (2-letter ISO code)

Example: "US"

Use Case:

- Country-specific pricing
- Currency localization
- Regulatory compliance

Notes:

- Always 2-letter ISO codes (US, DE, GB, FR, etc.)
- Exact match even in loose mode
- Most common matching attribute

Step 4: Save Matrix

Click **Save** button.

Matrix container created! Now add products.

Adding Products to Matrix

Method 1: From Matrix Edit Page

1. **Edit matrix** → Navigate to "Products" tab
2. **Click "Add Products"**
3. **Select products** from grid (checkboxes)
4. **Set prices for each product:**

- Quantity: Tier threshold (e.g., 1, 10, 50, 100)
- Price: Price value
- From Date: Start date (optional, overrides matrix date)
- To Date: End date (optional, overrides matrix date)

5. **Save**

Method 2: Bulk Import

Use the **CSV Import/Export Add-On** for bulk changes:

- Product-Customer Matrix CSV Import/Export Add-On
- Pricesystem CSV Import/Export (overview)

The Magento ImportExport entity **PricesystemProductCustomerMatrix** uses a structured CSV format (matrix rows + related attribute/customer rows).

Tip: Export first and use the exported CSV as your template (it matches your installed version/config).

Method 3: Product Attribute Rules

Configure automatic product selection based on attributes.

Example:

- Category: "Summer Collection"
- Attribute Set: "Apparel"
- Custom Attribute: "wholesale_eligible = Yes"

Result: All products matching attributes automatically included.

Creating Quantity Tiers

Add multiple price entries for same product with different qty values.

Example: Graduated Wholesale Pricing

Matrix: "Wholesale 2025" Product: "Widget Pro"

Qty Tier	Price	Description
1	$100	Standard price
10	$95	Bulk discount (5%)
50	$90	Volume discount (10%)
100	$85	Major volume (15%)

Customer orders 75 units: Gets $90/unit (Qty 50 tier applies)

How Qty Tiers Work:

- System finds largest qty tier <= cart quantity
- If customer orders 25 units: Uses qty=10 tier ($95)

- If customer orders 150 units: Uses qty=100 tier ($85)

Assigning Customers to Matrix

Method 1: Automatic Assignment (Recommended)

Configure matching attributes on matrix (see above).

Example:

```
Matrix: "Wholesale US 2025"
Attributes Relation: AND
Customer Group: Wholesale (2)
Country: US
Region: California
```

Result: All wholesale customers in California, USA auto-assigned.

Configuration Required:

- `enable_auto_assign_customers = Yes`

Trigger Points:

- Customer logs in
- Customer updates profile
- Customer group changes
- Admin manually triggers re-evaluation
- Cron job (if configured)

Method 2: Manual Customer Assignment

Matrix Edit Page > Customers Tab:

1. Click "Add Customers"
2. Search and select customers
3. **Optional:** Set customer-specific dates:
    - From Date: Override matrix from_date for this customer
    - To Date: Override matrix to_date for this customer
4. Save

Result: Only selected customers assigned.

Use Cases:

- Override automatic rules for specific customers
- Exception handling (VIP gets special matrix)
- Testing with test customers

Customer-Specific Date Example:

```
Matrix Dates: 2025-01-01 to 2025-12-31
Customer #123 Override: 2025-01-01 to 2025-06-30 (6-month contract)
Customer #456 Override: NULL (uses matrix dates)
```

Common Workflows

Workflow 1: Wholesale Pricing

Goal: Create wholesale pricing for US customers in 2025.

1. **Create matrix:**
    - Name: "Wholesale US 2025"
    - Active: Yes
    - Priority: 15
    - Attributes Relation: AND
    - From Date: 2025-01-01
    - To Date: 2025-12-31
    - Customer Group: Wholesale (2)
    - Country: US
2. **Add products:**
    - Import CSV with all products
    - Or add manually via Products tab
    - Set quantity tiers: qty=1, qty=10, qty=50, qty=100
3. **Test:**
    - Log in as wholesale customer in US
    - Verify prices display
    - Test quantity tier changes in cart

Workflow 2: Seasonal Campaign

Goal: Black Friday sale (Nov 29 - Dec 2, 2025) for all customers.

1. **Create matrix:**
    - Name: "Black Friday 2025"
    - Active: Yes
    - Priority: 25 (higher than standard pricing)
    - From Date: 2025-11-29
    - To Date: 2025-12-02
    - Customer Group: General (1) - all customers
    - No other attributes (all customers match)
2. **Add products:**

- Select sale products
- Set discounted prices (single qty=1 tier)

3. **Automatic:**
   - Activates Nov 29
   - Deactivates Dec 3
   - No manual intervention

Workflow 3: Multi-Company B2B Pricing

Goal: 3 partner companies with different pricing.

1. **Create matrices:**
   - Matrix A: "Partner - ACME Corp" (Company = "ACME", Priority 30)
   - Matrix B: "Partner - XYZ Inc" (Company = "XYZ", Priority 30)
   - Matrix C: "Partner - Global LLC" (Company = "Global", Priority 30)
2. **Add products:**
   - Same products across all matrices
   - Different prices per matrix (negotiated rates)
3. **Result:**
   - ACME employees see ACME pricing
   - XYZ employees see XYZ pricing
   - Global employees see Global pricing

Workflow 4: Regional + Group Targeting

Goal: California wholesale customers get special pricing on California-specific products.

1. **Create matrix:**
   - Name: "CA Wholesale 2025"
   - Active: Yes
   - Priority: 20
   - Attributes Relation: AND
   - Customer Group: Wholesale (2)
   - Country: US
   - Region: "California"
2. **Add products:**
   - Select California-themed products only
   - Set wholesale prices with qty tiers
3. **Result:**
   - California wholesale: See special prices
   - California retail: Don't see (not wholesale)
   - Texas wholesale: Don't see (not California)

Workflow 5: VIP + Extra Savings

Goal: VIP customers get 15% off everything, plus 25% off selected products.

1. **Create matrices:**
    - Matrix 1: "VIP Base Pricing" (Priority 30, all products, 15% discount)
    - Matrix 2: "VIP Extra Savings" (Priority 32, selected products, 25% discount)
    - Both: Customer Group = VIP (4)
2. **Configuration:**
    - `merge_matrix_qtys = Yes` (best price wins)
3. **Result:**
    - VIP customers get 15% off all products (Matrix 1)
    - VIP customers get 25% off selected products (Matrix 2 wins for those products)

Workflow 6: Contract with Customer Override

Goal: Annual contract with ACME Corp, but Customer #12345 has 6-month trial.

1. **Create matrix:**
    - Name: "Contract - ACME Corp 2025"
    - Active: Yes
    - Priority: 35
    - From Date: 2025-01-01
    - To Date: 2025-12-31
    - Company: "ACME"
2. **Add products:**
    - Contract products with negotiated prices
3. **Assign customers:**
    - Automatic: All ACME employees via company match
    - Manual: Customer #12345 with override dates: 2025-01-01 to 2025-06-30
4. **Result:**
    - Most ACME employees: 2025-01-01 to 2025-12-31
    - Customer #12345: 2025-01-01 to 2025-06-30 (6-month trial)

Multi-Attribute Matching Examples

Example 1: AND Logic (Precise Targeting)

Matrix Configuration:

```
Attributes Relation: AND
Customer Group: Wholesale (2)
Country: US
Region: California
```

Customer Matching:

```
Customer A:
- Group: Wholesale ✓
- Country: US ✓
- Region: California ✓
→ MATCH (all three attributes match)

Customer B:
- Group: Wholesale ✓
- Country: US ✓
- Region: Texas ✗
→ NO MATCH (region doesn't match)

Customer C:
- Group: Retail ✗
- Country: US ✓
- Region: California ✓
→ NO MATCH (group doesn't match)
```

Example 2: OR Logic (Broad Targeting)

Matrix Configuration:

```
Attributes Relation: OR
Customer Group: Wholesale (2), VIP (4)
```

Customer Matching:

```
Customer A:
- Group: Wholesale ✓
→ MATCH (one attribute matches)

Customer B:
- Group: VIP ✓
→ MATCH (one attribute matches)

Customer C:
- Group: Retail ✗
→ NO MATCH (no attributes match)
```

Example 3: Complex Scenario

Goal: Wholesale customers in California OR VIP customers anywhere.

Solution: Create TWO matrices:

```
Matrix 1: "CA Wholesale" (Priority 20)
- Attributes Relation: AND
- Group: Wholesale
- Country: US
- Region: California

Matrix 2: "VIP All Regions" (Priority 30)
- Attributes Relation: AND
- Group: VIP
- (no location attributes)
```

Result:

- California wholesale customers: Matrix 1
- VIP customers (any location): Matrix 2
- VIP customers in California: Matrix 2 wins (higher priority)

Testing Matrix Assignment

Test Plan

1. **Create test matrix:**
    - Name: "Test - Wholesale US"
    - Customer Group: Wholesale (2)
    - Country: US
    - Active: Yes
2. **Create test customer:**
    - Group: Wholesale
    - Country: US
    - Company: "Test Co"
3. **Test assignment:**
    - Log in as test customer
    - Check assigned matrices (admin: Customer Edit > Matrices tab)
    - Verify matrix appears
4. **Test exclusion:**
    - Change customer country to DE
    - Log out, log back in
    - Verify matrix NOT assigned
5. **Test re-assignment:**
    - Change customer back to US

- Log out, log back in
- Verify matrix re-assigned

Priority System in Action

Scenario: Customer Matches Multiple Matrices

Setup:

- Customer: John Doe (Wholesale group, California, ACME Corp)
- Matrix A: "Wholesale 2025" (Priority 15) - Matches group
- Matrix B: "California Special" (Priority 20) - Matches region
- Matrix C: "ACME Contract" (Priority 30) - Matches company

Configuration 1: merge_matrix_qtys = No

- **Result:** Only Matrix C applies (highest priority)
- Product X: Uses Matrix C prices only

Configuration 2: merge_matrix_qtys = Yes

- **Result:** Merge qty tiers from all three
  - Matrix A: qty=1 ($100), qty=10 ($95)
  - Matrix B: qty=25 ($92)
  - Matrix C: qty=50 ($88)
  - **Customer gets:** qty=1 ($100), qty=10 ($95), qty=25 ($92), qty=50 ($88)
  - **Best price from each matrix at each tier!**

Troubleshooting

Matrix Not Showing for Customer

Checklist:

1. ✓ Is Active = Yes?
2. ✓ Date range valid? (from_date <= today <= to_date)
3. ✓ Website matches?
4. ✓ Customer matches attributes (check attributes_relation: AND/OR)?
5. ✓ Auto-assignment enabled? (`enable_auto_assign_customers = Yes`)
6. ✓ Customer logged in recently? (triggers evaluation)
7. ✓ Cache cleared?

Debug SQL:

```
-- Check customer attributes
SELECT entity_id, group_id, email, company
FROM customer_entity
WHERE entity_id = 123;
```

```
-- Check matrix matching attributes
SELECT m.id, m.name, m.priority, m.attributes_relation, ma.attribute_code,
ma.attribute_value
FROM pricesystem_product_customer_matrix m
LEFT JOIN pricesystem_product_customer_matrix_attribute ma ON ma.matrix_id
= m.id
WHERE m.is_active = 1
ORDER BY m.priority DESC;

-- Check assignments
SELECT * FROM pricesystem_product_customer_matrix_customer
WHERE customer_id = 123;
```

Wrong Prices Displaying

Checklist:

1. ✓ Multiple matrices match? (check priority)
2. ✓ Merge config? (`merge_matrix_qtys` setting)
3. ✓ Product exists in matrix?
4. ✓ Quantity tier matches cart quantity?
5. ✓ Date ranges valid? (matrix + customer-specific)
6. ✓ Other pricesystem modules active? (check price resolution order)

Attributes Not Matching

Common Issues:

1. **AND vs OR:** Check `attributes_relation` field
2. **Data quality:** Company name mismatch ("ACME" vs "ACME Corp")
3. **Match mode:** Loose vs Exact (check `match_exact_type` config)
4. **Empty attributes:** Ensure at least one attribute filled

---

Best Practices

Naming Conventions

- Include year: "Wholesale 2025"
- Include segment: "VIP Contract - ACME"
- Include region: "California Regional"
- Avoid generics: "Test", "Matrix 1"

Priority Management

- Use tiers: 0, 10, 20, 30 (not 1, 2, 3, 4)
- Leave gaps: Insert later without renumbering
- Document hierarchy: Team reference

Attribute Selection

- Start simple: Group + Country
- Add complexity: Region, Company
- Test thoroughly: Edge cases
- Document logic: Why these attributes?

Performance

- Don't over-create: 100s OK, 1000s might slow
- Use specific attributes: Group faster than Company matching
- Cache enabled: Production setting
- Monitor: Regular audits

Next Steps

- Matrix Pricing System - Deep dive into core architecture
- Multi-Attribute Matching - Advanced attribute logic
- Priority Resolution - Conflict resolution strategies

Matrix Pricing System

Product-Customer Matrix Prices · /productcustomermatrix/features/matrix-pricing

Deep dive into the core matrix pricing architecture, workflow, and integration with Magento 2 Pricesystem.

Overview

Matrix Pricing enables sophisticated product-customer price matching through configurable matrices that act as logical pricing containers. Unlike traditional pricelists where you assign customers to lists, matrices use **bidirectional matching**: products can be matched to customer attributes, or customer attributes can select products.

Key Concepts

Matrix Container: Logical unit that groups:

- Product selection rules
- Customer matching rules
- Priority for conflict resolution
- Date range validity
- Quantity-based pricing tiers

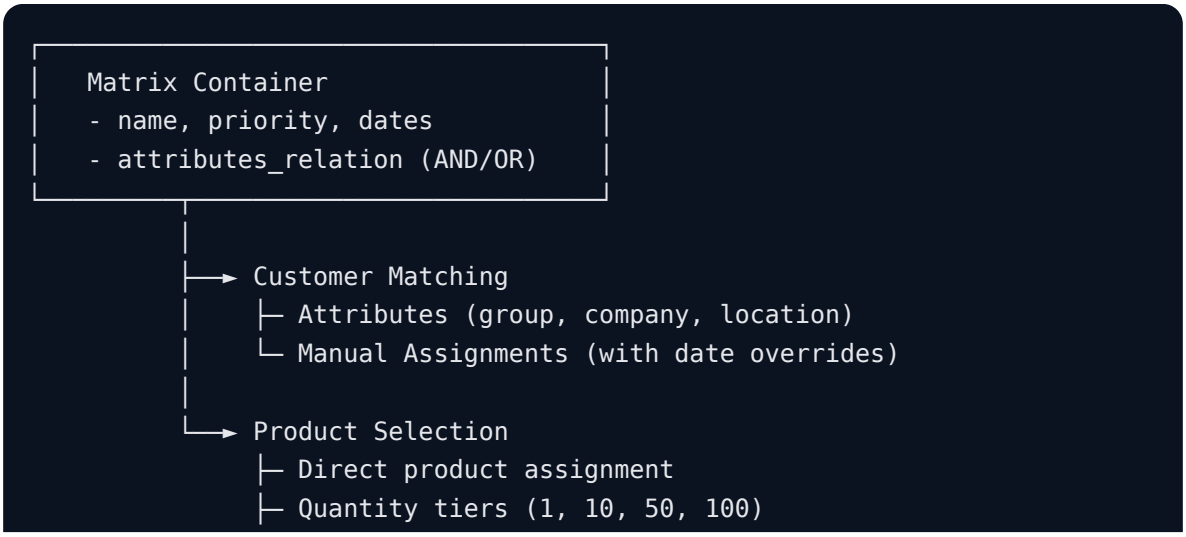
Bidirectional Matching:

- **Traditional:** Customer → Pricelist → Products
- **Matrix:** Customer ←→ Attributes ←→ Products

Attribute-Based Logic: Customers matched via attributes (group, company, location) rather than manual assignment.

Matrix Architecture

Component Diagram



```
                └─ Product-level date ranges
```

Data Flow

```
Customer Login
    ↓
Evaluate Active Matrices
    ↓
Check Attributes Match (AND/OR logic)
    ↓
Filter by Date Validity (matrix + customer dates)
    ↓
Assign Matching Matrices
    ↓
Cache Assignment (session)
    ↓
Product View/Cart
    ↓
Resolve Price (priority + merge logic)
    ↓
Display Final Price
```

Matrix Lifecycle

1. Creation Phase

Admin Creates Matrix:

- Set basic info (name, priority, dates)
- Configure attributes (group, company, location)
- Choose attributes relation (AND/OR)
- Save matrix container

Status: Inactive (default) or Active

2. Product Assignment

Add Products:

- Select products from catalog
- Set prices for each product
- Add quantity tiers (optional)
- Set product-level date ranges (optional)

Example:

```
Product: Widget Pro (SKU-123)
Tiers:
  - qty=1, price=$100, dates=NULL
```

```
- qty=10, price=$95, dates=NULL
- qty=50, price=$90, dates=2025-06-01 to 2025-08-31 (summer special)
```

3. Customer Assignment

Automatic (Attribute-Based):

- System evaluates customer attributes on login
- Matches against matrix attribute rules
- Auto-assigns if all conditions met

Manual:

- Admin explicitly assigns customer to matrix
- Optional: Set customer-specific date overrides
- Bypasses attribute matching

4. Activation

Set Matrix Active:

- `is_active = Yes`
- Matrix immediately available for matching

Date-Based Activation:

- `from_date = 2025-06-01`
- Automatically activates June 1st
- No manual intervention needed

5. Price Resolution

Customer Views Product:

1. System finds all assigned matrices
2. Filters by date validity
3. Checks product exists in matrix
4. Sorts by priority (highest first)
5. Applies merge logic (if enabled)
6. Returns final price for current quantity

6. Deactivation/Expiration

Manual Deactivation:

- Set `is_active = No`
- Immediately stops price evaluation

Automatic Expiration:

- `to_date = 2025-12-31`
- Expires January 1, 2026
- No cleanup needed

Price Resolution Algorithm

Step-by-Step Resolution

Input:

- Customer ID: 123
- Product ID: 456
- Quantity: 25

Process:

Step 1: Find Matching Matrices

```
SELECT m.*
FROM pricesystem_product_customer_matrix m
WHERE m.is_active = 1
  AND m.website_id = 1
  AND (m.from_date IS NULL OR m.from_date <= CURDATE())
  AND (m.to_date IS NULL OR m.to_date >= CURDATE())
  AND m.id IN (
    -- Matrices customer is assigned to
    SELECT matrix_id FROM pricesystem_product_customer_matrix_customer
    WHERE customer_id = 123
  )
ORDER BY m.priority DESC;
```

Result: 3 matrices (A=Priority 15, B=Priority 20, C=Priority 30)

Step 2: Check Product Availability

```
SELECT p.*
FROM pricesystem_pricelist_product p
WHERE p.pricelist_id IN (matrixA, matrixB, matrixC)
  AND p.product_id = 456
  AND (p.from_date IS NULL OR p.from_date <= CURDATE())
  AND (p.to_date IS NULL OR p.to_date >= CURDATE());
```

Result: Product exists in all 3 matrices

Step 3: Find Quantity Tiers

```
SELECT p.qty, p.price, p.pricelist_id
FROM pricesystem_pricelist_product p
WHERE p.pricelist_id IN (matrixA, matrixB, matrixC)
  AND p.product_id = 456
  AND p.qty <= 25  -- Customer quantity
ORDER BY p.pricelist_id, p.qty DESC;
```

Result:

```
Matrix A (Priority 15):
  - qty=1: $100
  - qty=10: $95
  - qty=25: $92  ← Best tier for qty=25

Matrix B (Priority 20):
  - qty=1: $98
  - qty=10: $93
  (no qty=25 tier, use qty=10)

Matrix C (Priority 30):
  - qty=1: $96
  - qty=50: $88
  (no qty=25 tier, use qty=1)
```

Step 4: Apply Merge Logic

If merge_matrix_qtys = No (Highest Priority Only):

```
Use Matrix C only (Priority 30)
Product qty=25 → Use Matrix C qty=1 tier: $96
Result: $96/unit
```

If merge_matrix_qtys = Yes (Best Price Wins):

```
Compare best price at qty=25 across all matrices:
- Matrix A qty=25: $92  ← WINNER
- Matrix B qty=10: $93
- Matrix C qty=1: $96

Result: $92/unit (from Matrix A)
```

Step 5: Return Final Price

Output:

- merge=No: $96/unit × 25 = $2,400 total
- merge=Yes: $92/unit × 25 = $2,300 total (customer saves $100!)

Integration with Pricesystem Core

Price Resolution Order (Full System)

Product-Customer Matrix integrates into Magento 2 Pricesystem price resolution:

```
1. Shopping Cart Price Rules (if enabled)
   ↓
2. Customer-Specific Prices (direct assignments)
   ↓
3. Product-Customer Matrices ← THIS EXTENSION
   ↓
4. Named Pricelists (pricesystem-pricelist)
   ↓
5. Category Prices (pricesystem-categoryprice)
   ↓
6. Catalog Price (fallback)
```

Example Scenario:

```
Product: Widget Pro
Catalog Price: $150

Customer has:
- Matrix pricing: $100 (Priority 20)
- Pricelist pricing: $110 (Priority 15)
- Category pricing: $120

Resolution:
Step 1: No cart rules
Step 2: No customer-specific price
Step 3: Matrix price found: $100 ← WINNER
(Pricelists and category prices never evaluated)

Final Price: $100
```

Cross-Module Conflicts

Matrix vs Pricelist (both active):

- Matrices evaluated BEFORE pricelists
- Matrix price found → Pricelist ignored
- No matrix price → Fallback to pricelist

Matrix vs Category Price:

- Matrices evaluated BEFORE category prices
- Same logic as pricelists

Priority Within Matrices:

- Multiple matrices resolved via priority field
- NOT via module loading order

Advanced Scenarios

Scenario 1: Overlapping Date Ranges

Setup:

```
Matrix A: "Standard Wholesale" (Priority 15)
- Dates: 2025-01-01 to 2025-12-31 (permanent)
- Product X: $100

Matrix B: "Black Friday Special" (Priority 25)
- Dates: 2025-11-29 to 2025-12-02 (4 days)
- Product X: $75
```

Timeline:

- **Nov 28:** Only Matrix A active → $100
- **Nov 29-Dec 2:** Both active, Matrix B wins (Priority 25) → $75
- **Dec 3+:** Only Matrix A active → $100

Key Point: Temporary high-priority matrices override standard pricing.

Scenario 2: Customer-Specific Date Override

Setup:

```
Matrix: "Annual Contract - ACME Corp" (Priority 35)
- Dates: 2025-01-01 to 2025-12-31
- Company: "ACME"
- Product X: $90

Customer #123 (ACME employee):
- Manual assignment to matrix
- Override dates: 2025-01-01 to 2025-06-30 (6-month trial)

Customer #456 (ACME employee):
- Automatic assignment via company match
- No override dates (uses matrix dates)
```

Result:

- Customer #123: Sees $90 from Jan 1 to Jun 30, then loses access
- Customer #456: Sees $90 from Jan 1 to Dec 31 (full year)

Use Case: Trial periods, temporary access, phased rollouts.

Scenario 3: Product-Level Date Override

Setup:

```
Matrix: "Seasonal Pricing 2025" (Priority 20)
- Dates: 2025-01-01 to 2025-12-31 (permanent)
- Customer Group: Wholesale

Product X:
- qty=1: $100, dates=NULL (year-round)
- qty=10: $95, dates=NULL (year-round)
- qty=50: $85, dates=2025-06-01 to 2025-08-31 (summer special)
```

Result:

- Jan-May: qty tiers: 1=$100, 10=$95 (qty=50 tier not available)
- Jun-Aug: qty tiers: 1=$100, 10=$95, 50=$85 (summer tier active)
- Sep-Dec: qty tiers: 1=$100, 10=$95 (qty=50 tier expired)

Use Case: Seasonal quantity discounts, limited-time bulk pricing.

Scenario 4: Multi-Product Matrix with Partial Overlap

Setup:

```
Matrix A (Priority 15): Products X, Y, Z
Matrix B (Priority 20): Products X, Y (Z missing)

Customer matches both matrices.
merge_matrix_qtys = No (highest priority only)
```

Result:

- Product X: Uses Matrix B (Priority 20)
- Product Y: Uses Matrix B (Priority 20)
- Product Z: **NO MATRIX PRICE** (Matrix B doesn't have Z, Matrix A ignored)
    - Fallback to next pricesystem module (pricelist, category, catalog)

Lesson: High-priority matrices with incomplete product sets can cause fallback.

Solution: Use `merge_matrix_qtys = Yes` to include Matrix A prices for Product Z.

Scenario 5: Three-Way Priority Resolution

Setup:

```
Customer: John Doe
```

```
Matches three matrices:

Matrix A: "Wholesale 2025" (Priority 15)
- Product X: qty=1 ($100), qty=10 ($95), qty=50 ($90)

Matrix B: "California Regional" (Priority 20)
- Product X: qty=1 ($98), qty=25 ($92)

Matrix C: "ACME Contract" (Priority 30)
- Product X: qty=1 ($96), qty=100 ($88)
```

Customer orders 30 units of Product X.

merge_matrix_qtys = No:

```
Use Matrix C only (Priority 30)
Best tier for qty=30: qty=1 ($96)
Result: $96/unit × 30 = $2,880
```

merge_matrix_qtys = Yes:

```
Merge all tiers:
- qty=1: Best of ($100, $98, $96) = $96 (Matrix C)
- qty=10: Best of ($95, $98, $96) = $95 (Matrix A)
- qty=25: Best of ($90, $92, $96) = $90 (Matrix A)
- qty=50: Best of ($90, $92, $96) = $90 (Matrix A)
- qty=100: $88 (Matrix C)

Customer orders 30 units → Uses qty=25 tier: $90
Result: $90/unit × 30 = $2,700

Savings: $180 by using merge!
```

Database Operations

Creating a Matrix

```
INSERT INTO pricesystem_product_customer_matrix
(name, is_active, priority, from_date, to_date, website_id,
attributes_relation, created_at, updated_at)
VALUES
('Wholesale US 2025', 1, 15, '2025-01-01', '2025-12-31', 1, 'AND', NOW(),
NOW());

SET @matrix_id = LAST_INSERT_ID();
```

Adding Attributes

```
INSERT INTO pricesystem_product_customer_matrix_attribute
(matrix_id, attribute_code, attribute_value)
VALUES
(@matrix_id, 'group', '2'),  -- Wholesale group
(@matrix_id, 'country', 'US');  -- United States
```

Adding Products with Qty Tiers

```
INSERT INTO pricesystem_pricelist_product
(pricelist_id, product_id, qty, price, from_date, to_date)
VALUES
(@matrix_id, 123, 1, 100.00, NULL, NULL),
(@matrix_id, 123, 10, 95.00, NULL, NULL),
(@matrix_id, 123, 50, 90.00, NULL, NULL),
(@matrix_id, 123, 100, 85.00, NULL, NULL);
```

Manual Customer Assignment

```
INSERT INTO pricesystem_product_customer_matrix_customer
(matrix_id, customer_id, from_date, to_date)
VALUES
(@matrix_id, 456, '2025-01-01', '2025-06-30');  -- 6-month trial
```

Finding Customer's Matrices

```
SELECT
  m.id,
  m.name,
  m.priority,
  m.from_date,
  m.to_date,
  CASE
    WHEN mc.from_date IS NOT NULL THEN mc.from_date
    ELSE m.from_date
  END AS effective_from_date,
  CASE
    WHEN mc.to_date IS NOT NULL THEN mc.to_date
    ELSE m.to_date
  END AS effective_to_date
FROM pricesystem_product_customer_matrix m
JOIN pricesystem_product_customer_matrix_customer mc ON mc.matrix_id = m.id
WHERE mc.customer_id = 123
  AND m.is_active = 1
ORDER BY m.priority DESC;
```

Performance Optimization

Caching Strategy

Matrix Assignment Cache:

- Cached per customer session
- Cache key: `matrix_assignment_customer_{id}`
- TTL: Session lifetime
- Invalidated on: Profile update, group change, logout

Price Cache:

- Cached per customer + product + qty
- Cache key: `matrix_price_{customer_id}_{product_id}_{qty}`
- TTL: Configurable (default: 1 hour)
- Invalidated on: Matrix update, product update, price change

Indexing

Critical Indexes:

```
-- Matrix priority + active status
CREATE INDEX idx_matrix_active_priority ON
pricesystem_product_customer_matrix(is_active, priority);

-- Matrix dates
CREATE INDEX idx_matrix_dates ON
pricesystem_product_customer_matrix(from_date, to_date);

-- Customer assignments
CREATE INDEX idx_matrix_customer ON
pricesystem_product_customer_matrix_customer(customer_id);

-- Product prices
CREATE INDEX idx_product_matrix_qty ON
pricesystem_pricelist_product(pricelist_id, product_id, qty);
```

Query Optimization

Avoid:

```
-- BAD: Scans all matrices for each customer
SELECT * FROM pricesystem_product_customer_matrix WHERE is_active = 1;
```

Prefer:

```
-- GOOD: Uses customer assignment index
```

```
SELECT m.*
FROM pricesystem_product_customer_matrix m
JOIN pricesystem_product_customer_matrix_customer mc ON mc.matrix_id = m.id
WHERE mc.customer_id = 123
  AND m.is_active = 1;
```

Best Practices

Matrix Design

1. **Use specific attributes:** Group + Country faster than Company matching
2. **Leave gaps in priority:** 10, 20, 30 (not 10, 11, 12) allows insertions
3. **Document hierarchy:** Team reference for priority tiers
4. **Test edge cases:** Overlapping dates, missing products, same priority

Quantity Tiers

1. **Consistent tiers:** Use same breakpoints across products (1, 10, 50, 100)
2. **Logical progression:** Each tier should offer meaningful discount
3. **Avoid too many tiers:** 4-6 tiers maximum (performance + UX)
4. **Document strategy:** Why these breakpoints?

Date Management

1. **Include grace periods:** Contract renewals need overlap
2. **Test transitions:** Verify date changes in staging
3. **Use future dates:** Schedule campaigns in advance
4. **Monitor expirations:** Alert before critical matrices expire

Assignment Strategy

1. **Prefer automatic:** Attribute-based assignment scales better
2. **Manual for exceptions:** Override automatic rules sparingly
3. **Document overrides:** Why this customer is special?
4. **Regular audits:** Review manual assignments quarterly

Troubleshooting

Debug Checklist

Matrix not applying:

1. ✓ Matrix active? (`is_active = 1`)
2. ✓ Dates valid? (matrix + customer override)
3. ✓ Customer assigned? (check both automatic + manual)
4. ✓ Product exists in matrix?
5. ✓ Quantity tier exists for cart quantity?

6. ✓ Priority conflicts? (other matrix winning)
7. ✓ Cache cleared?

Wrong price displaying:

1. ✓ Check merge config (`merge_matrix_qtys`)
2. ✓ Verify priority values (highest should win)
3. ✓ Check quantity tier (correct tier for cart quantity?)
4. ✓ Review other pricesystem modules (higher precedence?)
5. ✓ Inspect cache (stale price cached?)

Related Documentation

- Creating Matrices - Step-by-step creation guide
- Multi-Attribute Matching - Attribute logic with AND/OR
- Priority Resolution - Conflict resolution strategies

Multi-Attribute Matching

Product-Customer Matrix Prices · /productcustomermatrix/features/multi-attribute-matching

Advanced customer attribute matching with configurable AND/OR logic for precise targeting.

Overview

Multi-Attribute Matching enables sophisticated customer targeting by combining multiple customer attributes with flexible logic operators. Unlike simple pricelists where attributes use fixed AND logic, Product-Customer Matrix allows you to configure **AND** or **OR** relationships between attributes for each matrix.

Key Advantages

- **Flexible Logic:** Choose AND (all must match) or OR (any can match) per matrix
- **Multiple Values:** Assign multiple values per attribute (e.g., multiple groups)
- **Complex Scenarios:** (Group A OR Group B) AND (Region X) patterns
- **Attribute Table:** Dedicated table for scalable attribute storage
- **Real-Time Evaluation:** Attributes evaluated on login and profile updates

---

How It Works

Attributes Relation Field

Each matrix has an `attributes_relation` field that controls how multiple attributes combine.

Options:

- **AND (Default):** ALL filled attributes must match
- **OR:** ANY filled attribute can match

Attribute Storage

Attributes stored in dedicated table: `pricesystem_product_customer_matrix_attribute`

Structure:

```
CREATE TABLE pricesystem_product_customer_matrix_attribute (
  id INT PRIMARY KEY AUTO_INCREMENT,
  matrix_id INT NOT NULL,
  attribute_code VARCHAR(50) NOT NULL,  -- 'group', 'company', 'tax', etc.
  attribute_value VARCHAR(255) NOT NULL,

  FOREIGN KEY (matrix_id) REFERENCES
pricesystem_product_customer_matrix(id) ON DELETE CASCADE
);
```

Example Data:

```
matrix_id | attribute_code | attribute_value
----------|----------------|----------------
1         | group          | 2               (Wholesale)
1         | group          | 4               (VIP)
1         | country        | US
```

This means: (Group = 2 OR Group = 4) AND (Country = US)

Evaluation Process

Step-by-Step

Input:

- Customer ID: 123
- Customer attributes: Group=2 (Wholesale), Country=US, Region=California, Company="ACME Corp"

Process:

Step 1: Get Active Matrices

```
SELECT * FROM pricesystem_product_customer_matrix
WHERE is_active = 1
  AND website_id = 1
  AND (from_date IS NULL OR from_date <= CURDATE())
  AND (to_date IS NULL OR to_date >= CURDATE());
```

Result: 3 active matrices

Step 2: Load Attributes for Each Matrix

```
SELECT matrix_id, attribute_code, attribute_value
FROM pricesystem_product_customer_matrix_attribute
WHERE matrix_id IN (1, 2, 3);
```

Result:

```
Matrix 1 (attributes_relation='AND'):
- group: 2
- country: US

Matrix 2 (attributes_relation='OR'):
- group: 2
- group: 4
- region: California

Matrix 3 (attributes_relation='AND'):
```

```
- company: ACME
- country: US
- region: California
```

Step 3: Evaluate Each Matrix

Matrix 1 (AND logic):

```
Check: group=2? YES (customer group = 2) ✓
Check: country=US? YES (customer country = US) ✓
Result: MATCH (all attributes matched)
```

Matrix 2 (OR logic):

```
Check: group=2? YES (customer group = 2) ✓
(OR logic: stop here, already matched)
Result: MATCH (at least one attribute matched)
```

Matrix 3 (AND logic):

```
Check: company=ACME? YES (customer company contains "ACME") ✓
Check: country=US? YES ✓
Check: region=California? YES ✓
Result: MATCH (all three attributes matched)
```

Step 4: Assign Matching Matrices

Customer assigned to: Matrix 1, Matrix 2, Matrix 3

Supported Attributes

Customer Group

Attribute Code: `group`

Source: Customer entity → `group_id` field

Example:

```
Attribute: group = 2 (Wholesale)
Customer: group_id = 2
Match: YES
```

Multiple Values:

```
Attributes:
- group = 2 (Wholesale)
- group = 4 (VIP)
```

```
Logic: Customer matches if group=2 OR group=4
```

Use Case:

- Target multiple customer segments
- VIP + Partner programs
- Wholesale + Retail (different tiers)

Company Name

Attribute Code: `company`

Source: Customer entity → `company` attribute

Example:

```
Attribute: company = ACME
Customer: company = "ACME Corporation"
Match: YES (loose mode contains "ACME")
```

Matching Mode:

- **Loose:** Partial string match, case-insensitive
- **Exact:** Strict equality, case-sensitive

Configuration: `pricesystem/productcustomermatrix/match_exact_type`

Multiple Values:

```
Attributes:
- company = ACME
- company = XYZ
- company = Global

Logic: Customer matches if company contains any of these strings
```

Use Case:

- Multi-company B2B contracts
- Partner programs with multiple companies
- Corporate group pricing

Tax/VAT Number

Attribute Code: `tax`

Source: Customer entity → `taxvat` attribute

Example:

```
Attribute: tax = DE123456789
Customer: taxvat = DE123456789
Match: YES
```

Use Case:

- EU VAT-registered businesses
- Tax-exempt organizations
- B2B verified accounts

Notes:

- Usually exact match (even in loose mode)
- Format-sensitive (include dashes, spaces as stored)
- Verify data quality

Postcode

Attribute Code: `postcode`

Source: Customer address → `postcode`

Example:

```
Attribute: postcode = 90210
Customer billing: postcode = 90210
Match: YES
```

Address Checking:

- Checks BOTH billing AND shipping addresses
- Match if EITHER address matches

Loose Matching:

```
Attribute: postcode = 9021
Customer: postcode = 90210
Match: YES (contains "9021")
```

Multiple Values:

```
Attributes:
- postcode = 90210
- postcode = 90211
- postcode = 90212

Logic: Customer matches if any address has any of these postcodes
```

Use Case:

- ZIP code-based regional pricing
- Local delivery zones
- Metropolitan area targeting

Region

Attribute Code: `region`

Source: Customer address → `region` text field

Example:

```
Attribute: region = California
Customer billing: region = California
Match: YES
```

Data Quality:

- Magento stores region as text (not ID)
- "California" vs "CA" might not match
- Standardize or use loose matching

Multiple Values:

```
Attributes:
- region = California
- region = Nevada
- region = Arizona

Logic: Customer matches if any address in any of these regions
```

Use Case:

- State/province-specific pricing
- Regional campaigns
- Tax jurisdiction targeting

Country

Attribute Code: `country`

Source: Customer address → `country_id` (ISO 2-letter code)

Example:

```
Attribute: country = US
Customer billing: country_id = US
Match: YES
```

Format:

- Always 2-letter ISO codes: US, DE, GB, FR, IT, ES, etc.
- Exact match even in loose mode

Multiple Values:

```
Attributes:
- country = US
- country = CA
- country = MX

Logic: North American customers (USMCA)
```

Use Case:

- Country-specific pricing
- Currency localization
- Regulatory compliance
- Continental targeting (EU, APAC, etc.)

AND Logic Patterns

Pattern 1: Precise Targeting

Goal: California wholesale customers only.

Matrix Configuration:

```
attributes_relation: AND

Attributes:
- group = 2 (Wholesale)
- country = US
- region = California
```

Matching:

```
Customer A: Wholesale, US, California → MATCH ✓
Customer B: Wholesale, US, Texas → NO MATCH (wrong region)
Customer C: Retail, US, California → NO MATCH (wrong group)
Customer D: Wholesale, DE, California → NO MATCH (wrong country)
```

Use Case: Very specific regional + segment targeting.

Pattern 2: B2B Company + Location

Goal: ACME Corp employees in US only.

Matrix Configuration:

```
attributes_relation: AND

Attributes:
- company = ACME
- country = US
```

Matching:

```
Customer A: ACME Corp, US → MATCH ✓
Customer B: ACME Corp, DE → NO MATCH (wrong country)
Customer C: XYZ Inc, US → NO MATCH (wrong company)
```

Use Case: Multi-national B2B contracts with regional restrictions.

Pattern 3: Triple Attribute Precision

Goal: Wholesale California customers from ACME Corp.

Matrix Configuration:

```
attributes_relation: AND

Attributes:
- group = 2 (Wholesale)
- company = ACME
- region = California
```

Matching:

```
Customer A: Wholesale, ACME, California → MATCH ✓
Customer B: Wholesale, ACME, Texas → NO MATCH (wrong region)
Customer C: VIP, ACME, California → NO MATCH (wrong group)
```

Use Case: Highly specific targeting with multiple conditions.

OR Logic Patterns

Pattern 1: Multiple Customer Groups

Goal: Wholesale OR VIP customers.

Matrix Configuration:

```
attributes_relation: OR

Attributes:
- group = 2 (Wholesale)
- group = 4 (VIP)
```

Matching:

```
Customer A: Wholesale → MATCH ✓
Customer B: VIP → MATCH ✓
Customer C: Retail → NO MATCH
```

Use Case: Target multiple customer segments with same pricing.

Pattern 2: Multi-Region Campaign

Goal: Customers in California, Nevada, or Arizona.

Matrix Configuration:

```
attributes_relation: OR

Attributes:
- region = California
- region = Nevada
- region = Arizona
```

Matching:

```
Customer A: California → MATCH ✓
Customer B: Nevada → MATCH ✓
Customer C: Texas → NO MATCH
```

Use Case: Regional campaigns covering multiple states.

Pattern 3: Multi-Country Pricing

Goal: North American customers (US, Canada, Mexico).

Matrix Configuration:

```
attributes_relation: OR

Attributes:
- country = US
- country = CA
- country = MX
```

Matching:

```
Customer A: US → MATCH ✓
Customer B: CA → MATCH ✓
Customer C: DE → NO MATCH
```

Use Case: Continental/regional pricing zones.

Complex Scenarios

Scenario 1: (Group A OR Group B) AND (Region X)

Goal: Wholesale OR VIP customers in California.

Challenge: Need OR for groups, AND for region.

Solution: Create TWO matrices:

Matrix 1: Wholesale + California (Priority 20)

```
attributes_relation: AND
- group = 2 (Wholesale)
- region = California
```

Matrix 2: VIP + California (Priority 20)

```
attributes_relation: AND
- group = 4 (VIP)
- region = California
```

Result:

- Wholesale California customers: Matrix 1
- VIP California customers: Matrix 2
- Same priority, same products/prices

Scenario 2: (Company A OR Company B) AND (Country US)

Goal: ACME Corp OR XYZ Inc, but only in US.

Solution: Create TWO matrices:

Matrix 1: ACME + US

```
attributes_relation: AND
- company = ACME
- country = US
```

Matrix 2: XYZ + US

```
attributes_relation: AND
- company = XYZ
- country = US
```

Alternative (if same pricing): Use single matrix with manual customer assignments instead of attributes.

Scenario 3: Exclusion Logic (NOT)

Goal: All customers EXCEPT wholesale.

Challenge: System doesn't support NOT operator.

Workaround 1: Use OR with all other groups:

```
attributes_relation: OR
- group = 1 (General)
- group = 4 (VIP)
- group = 5 (Partner)
(exclude group = 2 Wholesale by not listing it)
```

Workaround 2: Create separate matrices with higher priority for excluded segment.

Attribute Data Quality

Best Practices

1. Standardize Data Entry:

```
Good: Company = "ACME Corporation"
Bad: Company = "acme corp.", "ACME Corp", "Acme Corporation"
```

2. Validate on Registration:

- Require company name for B2B customers
- Validate tax IDs (format + checksum)
- Use dropdown for regions (not free text)
- Use ISO country codes

3. Data Cleanup:

```
-- Find company name variations
SELECT DISTINCT company FROM customer_entity WHERE company LIKE '%ACME%';

-- Standardize company names
UPDATE customer_entity SET company = 'ACME Corporation' WHERE company LIKE
'%ACME%';
```

4. Use Loose Matching:

- Forgives typos and variations
- "ACME" matches "ACME Corp", "acme corporation"
- Trade-off: Less precise

Testing Multi-Attribute Logic

Test Case 1: AND Logic

Setup:

```
Matrix: "Test AND Logic"
attributes_relation: AND
Attributes:
- group = 2 (Wholesale)
- country = US
```

Test Customers:

```
Customer A: Wholesale, US → Expected: MATCH ✓
Customer B: Wholesale, DE → Expected: NO MATCH
Customer C: Retail, US → Expected: NO MATCH
Customer D: Retail, DE → Expected: NO MATCH
```

Verify:

1. Log in as each customer
2. Check assigned matrices (admin or API)

3. Verify expectations

Test Case 2: OR Logic

Setup:

```
Matrix: "Test OR Logic"
attributes_relation: OR
Attributes:
- group = 2 (Wholesale)
- group = 4 (VIP)
```

Test Customers:

```
Customer A: Wholesale → Expected: MATCH ✓
Customer B: VIP → Expected: MATCH ✓
Customer C: Retail → Expected: NO MATCH
```

Verify: Same as Test Case 1.

Test Case 3: Multiple Values per Attribute

Setup:

```
Matrix: "Multi-Country"
attributes_relation: OR
Attributes:
- country = US
- country = CA
- country = MX
```

Test Customers:

```
Customer A: US → Expected: MATCH ✓
Customer B: CA → Expected: MATCH ✓
Customer C: MX → Expected: MATCH ✓
Customer D: DE → Expected: NO MATCH
```

Database Queries

View Matrix Attributes

```
SELECT
  m.id,
  m.name,
  m.attributes_relation,
  ma.attribute_code,
  ma.attribute_value
FROM pricesystem_product_customer_matrix m
LEFT JOIN pricesystem_product_customer_matrix_attribute ma ON ma.matrix_id
= m.id
WHERE m.id = 1
ORDER BY ma.attribute_code, ma.id;
```

Example Output:

```
id | name               | attributes_relation | attribute_code |
attribute_value
---|--------------------|---------------------|----------------|------------
----
1  | Wholesale US 2025  | AND                 | country        | US
1  | Wholesale US 2025  | AND                 | group          | 2
```

Find Customers Matching Attributes

Complex Query (AND Logic Example):

```
SELECT DISTINCT ce.entity_id, ce.email, ce.group_id, ad.country_id
FROM customer_entity ce
JOIN customer_address_entity ad ON ad.parent_id = ce.entity_id AND
ad.is_default_billing = 1
WHERE ce.group_id = 2  -- Wholesale
  AND ad.country_id = 'US'
  AND ce.website_id = 1;
```

Result: All wholesale customers in US (matches AND logic).

Count Potential Matches

```
-- How many customers would match this matrix?
SELECT COUNT(DISTINCT ce.entity_id) AS potential_matches
FROM customer_entity ce
JOIN customer_address_entity ad ON ad.parent_id = ce.entity_id
WHERE ce.group_id = 2  -- Wholesale
  AND ad.country_id = 'US';
```

Troubleshooting

Attributes Not Matching

Checklist:

1. ✓ Correct `attributes_relation` (AND/OR)?
2. ✓ Attribute codes correct? (group, company, tax, postcode, region, country)
3. ✓ Attribute values exact? (check customer data)
4. ✓ Matching mode? (loose vs exact)
5. ✓ Data quality? (typos, case differences)
6. ✓ Address vs customer? (some attributes check addresses)

Debug SQL:

```
-- Check customer attributes
SELECT ce.entity_id, ce.email, ce.group_id, ce.company, ce.taxvat
FROM customer_entity ce
WHERE ce.entity_id = 123;

-- Check customer addresses
SELECT ad.postcode, ad.region, ad.country_id
FROM customer_address_entity ad
WHERE ad.parent_id = 123;

-- Check matrix attributes
SELECT ma.attribute_code, ma.attribute_value
FROM pricesystem_product_customer_matrix_attribute ma
WHERE ma.matrix_id = 1;
```

OR Logic Not Working

Common Issue: Mixing AND and OR expectations.

Example:

```
Matrix (attributes_relation=OR):
- group = 2
- country = US

Customer: group=1 (Retail), country=US

Expected (incorrect): NO MATCH (thinking AND logic)
Actual: MATCH (OR logic: country matches)
```

Fix: Review attributes_relation field. If AND needed, create separate matrices or change to AND.

Too Many/Too Few Matches

Issue: Attribute logic too broad or too narrow.

Debug:

1. Check how many customers have each attribute value
2. Test with known customers
3. Review attributes_relation logic
4. Consider splitting into multiple matrices

Best Practices

Attribute Selection

1. Start simple: Single attribute (group)
2. Add complexity: Group + Country
3. Test thoroughly: Edge cases
4. Document logic: Why these attributes?

Logic Choice

- **AND:** Use for precise targeting (3-4 attributes max)
- **OR:** Use for broad targeting (multiple groups/regions)
- **Complex:** Split into multiple matrices

Data Quality

- Standardize company names
- Validate tax IDs
- Use consistent region naming
- Encourage complete profiles

Performance

- Fewer attributes = faster evaluation
- Group + Country very fast (indexed)
- Company matching slower (text search)
- Regular data cleanup

Related Documentation

- Creating Matrices - Step-by-step matrix creation
- Matrix Pricing System - Core architecture and workflow
- Priority Resolution - Handling multiple matches

Priority-Based Resolution

Product-Customer Matrix Prices · /productcustomermatrix/features/priority-resolution

Handle scenarios where customers match multiple matrices using numeric priority values and merge strategies.

Overview

Customers can match multiple matrices through:

- Different matching attributes (group + region + company)
- Manual matrix assignments
- Overlapping date ranges
- Broad attribute rules (OR logic)

The Priority System resolves these conflicts predictably and transparently.

Priority Field

Every matrix has a `priority` field:

- **Type:** Integer
- **Range:** 0-999
- **Default:** 0
- **Rule:** Higher value = higher precedence

Example:

- Matrix A: Priority 10
- Matrix B: Priority 20
- **Winner:** Matrix B

Configuration

Merge Matrix Quantities

Path: Stores > Configuration > Pricesystem > Product-Customer Matrix > Merge Matrix Qtys

Setting: `pricesystem/productcustomermatrix/merge_matrix_qtys`

Values:

No (Default): Select Highest Priority

When customer matches multiple matrices, use ONLY the highest priority matrix.

Example:

```
Customer matches:
```

```
- Matrix A (Priority 10): Product X @ qty=1 ($100), qty=10 ($95)
- Matrix B (Priority 20): Product X @ qty=1 ($98), qty=50 ($90)

Result: Use ONLY Matrix B (higher priority)
- qty=1: $98
- qty=50: $90
(Matrix A ignored completely)
```

Use Case: Simple, predictable pricing. One matrix wins.

Yes: Merge Quantity Tiers

Combine quantity tiers from ALL matching matrices, taking best price at each tier.

Example:

```
Customer matches:
- Matrix A (Priority 10): Product X @ qty=1 ($100), qty=10 ($95)
- Matrix B (Priority 20): Product X @ qty=1 ($98), qty=50 ($90)

Result: MERGE tiers, best price wins:
- qty=1: $98 (from Matrix B, better than A's $100)
- qty=10: $95 (from Matrix A, B has no qty=10)
- qty=50: $90 (from Matrix B, A has no qty=50)

Customer gets: qty=1 ($98), qty=10 ($95), qty=50 ($90)
Best of both matrices!
```

Use Case: Complex pricing scenarios, best-of-all-matrices strategy.

Priority Hierarchy Recommendations

Standard Hierarchy

0-9: Base/Fallback

- Priority 0: Base matrix (website default)
- Priority 5: Legacy/migration pricing

10-19: Standard Pricing

- Priority 10: Standard retail
- Priority 12: Standard wholesale
- Priority 15: Regional standard

20-29: Promotional Pricing

- Priority 20: Standard promotions
- Priority 22: Seasonal sales
- Priority 25: Flash sales

30-39: VIP/Contract

- Priority 30: VIP tier pricing
- Priority 32: Annual contracts
- Priority 35: Enterprise agreements

40-49: Emergency Overrides

- Priority 40: Temporary fixes
- Priority 45: Urgent campaigns

50+: Reserved

- Priority 50+: System overrides
- Priority 99: Testing
- Priority 999: Absolute override (avoid)

Resolution Scenarios

Scenario 1: Same Priority

Setup:

```
Matrix A: Priority 10
Matrix B: Priority 10
```

Merge = No:

- **Undefined behavior!** (might be ID order, creation order, or random)
- **Solution:** Assign unique priorities (10 vs 11)

Merge = Yes:

- Merges tiers from both
- Best price wins at each tier
- Defined behavior

Best Practice: Avoid same-priority conflicts by using unique values.

Scenario 2: Clear Winner

Setup:

```
Matrix A: Priority 10 (Standard wholesale)
Matrix B: Priority 30 (VIP contract)
```

Merge = No:

- Use Matrix B only (Priority 30 > 10)

Merge = Yes:

- Merge tiers from both
- VIP contract prices likely win at each tier (better pricing)
- But if wholesale has unique tiers, customer gets those too

Scenario 3: Three-Way Conflict

Setup:

```
Customer matches three matrices:
- Matrix A: Priority 15 (Wholesale)
- Matrix B: Priority 20 (California Regional)
- Matrix C: Priority 30 (ACME Contract)

Product X prices:
- A: qty=1 ($100), qty=10 ($95)
- B: qty=1 ($98), qty=25 ($92)
- C: qty=1 ($96), qty=50 ($88)
```

Merge = No:

- Use ONLY Matrix C (Priority 30)
- Customer gets: qty=1 ($96), qty=50 ($88)

Merge = Yes:

- Merge all three matrices
- Best price at each tier:
  - qty=1: $96 (from C)
  - qty=10: $95 (from A)
  - qty=25: $92 (from B)
  - qty=50: $88 (from C)
- Customer gets: qty=1 ($96), qty=10 ($95), qty=25 ($92), qty=50 ($88)

Analysis: Merge gives best pricing across all tiers.

Scenario 4: Date-Based Priority Shift

Setup:

```
Standard pricing:
- Matrix: "Wholesale 2025"
- Priority: 15
- Dates: 2025-01-01 to 2025-12-31
- Product X: qty=1 ($100)
```

```
Campaign:
- Matrix: "Black Friday 2025"
- Priority: 25
- Dates: 2025-11-29 to 2025-12-02
- Product X: qty=1 ($75)
```

Timeline:

Nov 28:

- Only Wholesale active: $100

Nov 29-30 (Black Friday):

- Both active, campaign wins (Priority 25 > 15): $75

Dec 3+:

- Only Wholesale active again: $100

Key Point: Temporary high-priority campaigns override standard pricing.

Scenario 5: Product Not in High-Priority Matrix

Setup:

```
Matrix A (Priority 10): Products X, Y, Z
Matrix B (Priority 20): Products X, Y (Z missing)
```

Customer views Product Z:

Merge = No:

- Use Matrix B (higher priority)
- Product Z not in Matrix B
- **Result:** No matrix price (fallback to next pricesystem module)

Merge = Yes:

- Check all matrices
- Product Z found in Matrix A
- **Result:** Uses Matrix A price for Product Z

Lesson: Merge mode more forgiving when high-priority matrices have incomplete product sets.

Scenario 6: Customer-Specific Date Override

Setup:

```
Matrix: "Annual Contract - ACME" (Priority 35)
- Matrix Dates: 2025-01-01 to 2025-12-31
- Company: ACME

Customer #123:
- Automatic assignment via company match
- Manual override dates: 2025-01-01 to 2025-06-30 (6-month trial)

Customer #456:
- Automatic assignment via company match
- No override dates (uses matrix dates)
```

Result:

- Customer #123: Matrix active Jan 1 - Jun 30 only
- Customer #456: Matrix active Jan 1 - Dec 31 (full year)

Key Point: Customer-specific dates override matrix dates, allowing per-customer scheduling within same matrix.

Scenario 7: Priority + Merge with Overlapping Tiers

Setup:

```
Matrix A (Priority 10):
- Product X: qty=1 ($100), qty=10 ($90), qty=50 ($80)

Matrix B (Priority 20):
- Product X: qty=1 ($95), qty=25 ($85), qty=100 ($75)

Matrix C (Priority 30):
- Product X: qty=1 ($98), qty=50 ($78)
```

Customer orders 40 units.

Merge = No (Highest Priority Only):

```
Use Matrix C only (Priority 30)
Best tier for qty=40: qty=1 ($98)
Result: $98/unit × 40 = $3,920
```

Merge = Yes (Best Price Wins):

```
Merge all tiers:
- qty=1: Best of ($100, $95, $98) = $95 (Matrix B)
- qty=10: Best of ($90, $95, $98) = $90 (Matrix A)
- qty=25: Best of ($80, $85, $98) = $80 (Matrix A)
- qty=50: Best of ($80, $75, $78) = $75 (Matrix B)
```

```
- qty=100: $75 (Matrix B)

Customer orders 40 units → Uses qty=25 tier: $80
Result: $80/unit × 40 = $3,200

Savings: $720 by using merge!
```

Analysis: Merge can significantly benefit customers by combining best prices from multiple matrices.

Priority Management

Setting Priorities

At Creation:

1. Identify purpose (standard/promo/VIP/contract)
2. Check existing matrices in same tier
3. Assign priority within tier
4. Document reasoning

Example:

```
Creating "Summer Sale 2025"
Purpose: Seasonal promotion
Tier: 20-29 (promotional)
Existing: Spring Sale (22)
Assign: 23 (next in sequence)
Document: "Summer campaign, overrides standard wholesale (15)"
```

Changing Priorities

Caution: Affects all customers assigned to matrix.

Process:

1. Understand current behavior
2. Test priority change in staging
3. Verify customer impact
4. Update during low-traffic period
5. Monitor after change
6. Document change reason

Bulk Priority Updates

Scenario: Increase all VIP priorities by 10.

SQL:

```
UPDATE pricesystem_product_customer_matrix
```

```
SET priority = priority + 10
WHERE name LIKE 'VIP%'
OR id IN (
  SELECT matrix_id FROM pricesystem_product_customer_matrix_attribute
  WHERE attribute_code = 'group' AND attribute_value = '4' -- VIP group
);
```

Caution: Test first! Affects all customers.

Testing Priority Logic

Test Case 1: Basic Priority

Setup:

1. Create Matrix A (Priority 10) with Product X = $100
2. Create Matrix B (Priority 20) with Product X = $90
3. Assign test customer to both matrices (manually)

Test Merge = No:

- Expected: $90 (Matrix B wins)
- Verify: Log in, check Product X price

Test Merge = Yes:

- Expected: $90 (still $90, both have same tier)
- Verify: Same result

Test Case 2: Merge Benefit

Setup:

1. Matrix A (Priority 10): Product X @ qty=1 ($100), qty=10 ($95)
2. Matrix B (Priority 20): Product X @ qty=1 ($98), qty=50 ($90)
3. Assign test customer to both

Test Merge = No:

- Expected: Only qty=1 ($98), qty=50 ($90) from Matrix B
- Verify: qty=10 tier NOT available

Test Merge = Yes:

- Expected: qty=1 ($98), qty=10 ($95), qty=50 ($90)
- Verify: qty=10 tier available (from Matrix A)

Test Case 3: Date-Based Priority Shift

Setup:

1. Matrix A: Standard pricing (Priority 15, permanent)

2. Matrix B: Campaign pricing (Priority 25, 7 days only)
3. Assign customer to both

Test Before Campaign:

- Expected: Matrix A price
- Verify: Check product price

Test During Campaign:

- Expected: Matrix B price (higher priority)
- Verify: Price changed

Test After Campaign:

- Expected: Matrix A price again
- Verify: Price reverted

Best Practices

1. Use Tiers, Not Random Values

Good:

- 0, 10, 20, 30, 40 (clear tiers)
- 10, 15, 20, 25, 30 (with gaps)

Bad:

- 7, 13, 21, 29, 33 (no pattern)
- 1, 2, 3, 4, 5 (no gaps)

2. Leave Gaps

Allows inserting priorities later without renumbering.

Example:

- Current: 10, 20, 30
- Need to insert between 10 and 20: Use 15
- With consecutive (10, 11, 12): Must renumber everything

3. Document Hierarchy

Create internal documentation:

```
Priority Hierarchy - MageB2B Product-Customer Matrices

0-9: Base matrices
  0 = Website default pricing

10-19: Standard pricing
```

```
  10 = Standard retail
  12 = Standard wholesale
  15 = Regional standard

20-29: Promotional pricing
  20 = Standard promotions
  25 = Flash sales/urgent campaigns

30-39: VIP/Contract pricing
  30 = VIP tier
  35 = Enterprise contracts

40+: Emergency/testing
```

4. Test Merge Configuration

Before enabling merge:

1. Test with 2-3 matrices
2. Verify expected behavior
3. Check edge cases (missing products, same tiers)
4. Monitor after enabling

5. Avoid Priority 999

Reserve for absolute emergencies only. Creates maintenance issues.

6. Audit Regularly

Quarterly review:

- List matrices by priority
- Find duplicates (same priority)
- Find gaps (orphaned tiers)
- Verify hierarchy still makes sense

Troubleshooting

Wrong Price Displaying

Checklist:

1. ✓ Which matrices does customer match?
2. ✓ What are their priorities?
3. ✓ Is merge enabled?
4. ✓ Does product exist in high-priority matrix?
5. ✓ Are dates valid for all matrices?
6. ✓ Cache cleared?

Debug SQL:

```
-- Find customer's matrices
SELECT
  m.id,
  m.name,
  m.priority,
  m.is_active,
  m.from_date,
  m.to_date,
  CASE
    WHEN mc.from_date IS NOT NULL THEN mc.from_date
    ELSE m.from_date
  END AS effective_from_date,
  CASE
    WHEN mc.to_date IS NOT NULL THEN mc.to_date
    ELSE m.to_date
  END AS effective_to_date
FROM pricesystem_product_customer_matrix m
JOIN pricesystem_product_customer_matrix_customer mc ON mc.matrix_id = m.id
WHERE mc.customer_id = 123
  AND m.is_active = 1
ORDER BY m.priority DESC;

-- Check product prices in those matrices
SELECT
  m.name,
  m.priority,
  p.qty,
  p.price
FROM pricesystem_pricelist_product p
JOIN pricesystem_product_customer_matrix m ON m.id = p.pricelist_id
WHERE p.product_id = 456
  AND p.pricelist_id IN (
    SELECT matrix_id FROM pricesystem_product_customer_matrix_customer
    WHERE customer_id = 123
  )
ORDER BY m.priority DESC, p.qty ASC;
```

Merge Not Working

Symptoms: Customer only sees one matrix's prices, not merged.

Checks:

1. ✓ `merge_matrix_qtys = Yes` in config?
2. ✓ Config cache cleared?
3. ✓ Customer actually assigned to multiple matrices?
4. ✓ Matrices have different qty tiers to merge?

Test SQL:

```
-- Check merge config
SELECT * FROM core_config_data
```

```
WHERE path = 'pricesystem/productcustomermatrix/merge_matrix_qtys';

-- Check customer's matrices count
SELECT COUNT(*) FROM pricesystem_product_customer_matrix_customer
WHERE customer_id = 123;
```

Same Priority Issues

Symptom: Unpredictable pricing behavior.

Cause: Multiple matrices with same priority.

Resolution:

1. Find duplicates:

```
SELECT priority, COUNT(*) AS cnt
FROM pricesystem_product_customer_matrix
WHERE is_active = 1
GROUP BY priority
HAVING cnt > 1;
```

2. Assign unique priorities
3. Test again

Priority Not Respected

Symptom: Lower priority matrix wins.

Possible Causes:

1. High-priority matrix missing product
2. High-priority matrix expired (date range)
3. Customer-specific date override on lower-priority matrix
4. Cache issue

Debug:

1. Check product exists in all matrices
2. Verify date ranges (matrix + customer)
3. Clear cache
4. Test with fresh customer session

Advanced: Manual Priority Override

For specific customers, override priority system:

Method 1: Highest Priority Matrix

Create customer-specific matrix with priority 50+.

Example:

- Standard matrices: 10-30
- Customer exception: Priority 60
- Always wins for that customer

Method 2: Date Range Stacking

Layer matrices with different date ranges but same customer.

Example:

```
Base: Priority 10, permanent
Q1 Special: Priority 20, Jan-Mar
Q2 Special: Priority 20, Apr-Jun
VIP Override: Priority 30, permanent
```

Customer gets VIP pricing except during Q1/Q2 when specials might offer better prices (if merge enabled).

Method 3: Customer-Specific Date Restriction

Use customer-specific dates to limit when high-priority matrix applies.

Example:

```
Matrix: VIP Pricing (Priority 30, permanent at matrix level)

Customer #123:
- Override dates: 2025-01-01 to 2025-06-30 (6-month trial)
- After Jun 30: Loses VIP pricing, falls back to lower-priority matrices

Customer #456:
- No override dates
- Gets VIP pricing permanently (or until matrix expires)
```

Integration with Other Pricesystem Modules

Matrix vs Pricelist Priority

Price Resolution Order:

1. Shopping cart price rules
2. Customer-specific prices
3. **Product-Customer Matrices** ← Higher in chain
4. Named Pricelists
5. Category prices
6. Catalog price

Key Point: Matrices evaluated BEFORE pricelists. If matrix price found, pricelist never checked.

Cross-Module Conflicts

Scenario: Customer has both matrix and pricelist pricing.

Resolution:

- Matrix price found → Use it (pricelist ignored)
- Matrix price NOT found → Fallback to pricelist

Recommendation: Use matrices OR pricelists, not both, to avoid confusion.

Performance Considerations

Priority Sorting

Indexed:

```
CREATE INDEX idx_matrix_priority ON
pricesystem_product_customer_matrix(priority);
```

Query Optimization:

- Matrices sorted by priority DESC
- Highest priority evaluated first
- If merge=No, stop after first match (fast)
- If merge=Yes, evaluate all matching matrices (slower)

Merge Impact

Merge = No:

- Fast: First matching matrix wins
- Single database query per product

Merge = Yes:

- Slower: All matrices must be checked
- Multiple queries to find best price at each tier
- Consider for VIP customers only

Related Documentation

- Creating Matrices - Step-by-step matrix creation
- Matrix Pricing System - Core architecture and workflow
- Multi-Attribute Matching - Attribute logic with AND/OR

Advanced Config

Product-Customer Matrix Prices · /productcustomermatrix/addons/advanced-config

The Advanced Config add-on extends Pricesystem with additional configuration and override options that affect how **matrix prices** are selected and applied.

Package

- Composer package: `mageb2b/pricesystem-advancedconfig`
- Magento module: `MageB2B_PricesystemCoreAdvancedConfig`

Related

- Price Selection
- Pricesystem Advanced Config

CSV Import/Export

Product-Customer Matrix Prices · /productcustomermatrix/addons/csv-import-export

Bulk import and export **product-customer matrices** via CSV.

Package

- Composer package: `mageb2b/pricesystem-productcustomermatrix-importexport`
- Magento module: `MageB2B_PricesystemProductCustomerMatrixImpExp`

Installation

```
composer config bearer.repo.softwaresilo.io <token>
composer config repositories.softwaresilo composer
https://repo.softwaresilo.io/
composer require mageb2b/pricesystem-productcustomermatrix-importexport:*

php bin/magento setup:upgrade
php bin/magento cache:flush
```

Admin Import/Export (Magento ImportExport)

1. Go to **System > Data Transfer > Import**
2. Entity type: **PricesystemProductCustomerMatrix**

Tip: Export first and use the exported CSV as your template.

CLI Commands

```
php bin/magento pricesystem:import-productcustomermatrix
/path/to/productcustomermatrix.csv
php bin/magento pricesystem:export-productcustomermatrix
```

Related

- Price Selection
- Pricesystem CSV Import/Export

SOAP / REST API

Product-Customer Matrix Prices · /productcustomermatrix/addons/rest-api

Adds CRUD WebAPI endpoints for **product-customer matrices** (matrices, matrix attributes, and matrix customer assignments).

Package

- Composer package: `mageb2b/pricesystem-productcustomermatrix-api`
- Magento module: `MageB2B_PricesystemProductCustomerMatrixApi`

Installation

```
composer config bearer.repo.softwaresilo.io <token>
composer config repositories.softwaresilo composer
https://repo.softwaresilo.io/
composer require mageb2b/pricesystem-productcustomermatrix-api:*

php bin/magento setup:upgrade
php bin/magento cache:flush
```

Endpoints (Overview)

Matrix endpoints:

Method	URL
GET	`/V1/pricesystem/productcustomermatrix/:id`
GET	`/V1/pricesystem/productcustomermatrix/search`
POST	`/V1/pricesystem/productcustomermatrix`
PUT	`/V1/pricesystem/productcustomermatrix/:id`
DELETE	`/V1/pricesystem/productcustomermatrix/:id`

Matrix attribute endpoints:

Method	URL
GET	`/V1/pricesystem/productcustomermatrixattribute/:id`
GET	`/V1/pricesystem/productcustomermatrixattribute/search`
POST	`/V1/pricesystem/productcustomermatrixattribute`
PUT	`/V1/pricesystem/productcustomermatrixattribute/:id`
DELETE	`/V1/pricesystem/productcustomermatrixattribute/:id`

Matrix customer assignment endpoints:

Method	URL
GET	`/V1/pricesystem/productcustomermatrixcustomer/:id`
GET	`/V1/pricesystem/productcustomermatrixcustomer/search`
POST	`/V1/pricesystem/productcustomermatrixcustomer`
PUT	`/V1/pricesystem/productcustomermatrixcustomer/:id`
DELETE	`/V1/pricesystem/productcustomermatrixcustomer/:id`

For the full list, see: Pricesystem SOAP / REST API

Sample Data

Product-Customer Matrix Prices · /productcustomermatrix/addons/sample-data

Installs demo entities and example **product-customer matrix** records.

Packages

- Core sample data: `mageb2b/pricesystem-sample-data` (`MageB2B_PricesystemCoreSampleData`)
- Product Customer Matrix sample data: `mageb2b/pricesystem-productcustomermatrix-sample-data` (`MageB2B_PricesystemProductCustomerMatrixSampleData`)

Installation

```
composer config bearer.repo.softwaresilo.io <token>
composer config repositories.softwaresilo composer
https://repo.softwaresilo.io/
composer require mageb2b/pricesystem-sample-data:*
composer require mageb2b/pricesystem-productcustomermatrix-sample-data:*

php bin/magento setup:upgrade
php bin/magento cache:flush
```

Related

- Pricesystem Sample Data

Price Selection

Product-Customer Matrix Prices · /productcustomermatrix/price-selection

Product Customer Matrix is a Pricesystem price type (`product_customer_matrix`).

Pricesystem selects the final purchasable price using a global strategy (Lowest / Highest / Sort order / Formula), and the Matrix module also has internal selection logic (priority, date ranges, matching rules).

Quick Links

- System Configuration (Global)
- Sort Order Strategy (Priority List)
- Customer Group Overrides
- Customer Overrides + Fallback
- Formula Pricing

Matrix In The Strategy

- Price code: `product_customer_matrix`

If you want matrix prices to win, put `product_customer_matrix` near the top of your sort order list.

Customer Group Overrides

Product-Customer Matrix Prices · /productcustomermatrix/price-selection/customer-group-overrides

Customer/group override strategies require:

- `mageb2b/pricesystem-advancedconfig` (`MageB2B_PricesystemCoreAdvancedConfig`)

When Group Overrides Apply

- For **guests**: group-level selection applies (NOT_LOGGED_IN group).
- For **logged-in customers**: group-level selection applies only if the customer-level rule is set to **Fallback**.

Group Custom Sort Order

If the group-level rule is set to "Custom sort order":

- Pricesystem uses the group-specific priority list stored in `pricesystem_customer_group_sort_position`
- If the group has no custom list configured, it falls back to the global `price_sort_order`

Customer Overrides + Fallback

Product-Customer Matrix Prices · /productcustomermatrix/price-selection/customer-overrides

Customer overrides require:

- `mageb2b/pricesystem-advancedconfig` (`MageB2B_PricesystemCoreAdvancedConfig`)

System Config vs Fallback

- "System Config": uses the global strategy and does not fall back to group settings.
- "Fallback": uses group settings first (if configured), otherwise system config.

Recommendation:

- Set `pricesystem/advanced/default_priceselect` to **Fallback** so group-level rules apply by default.

Formula Pricing

Product-Customer Matrix Prices · /productcustomermatrix/price-selection/formula-pricing

Formula pricing requires:

- `mageb2b/pricesystem-advancedconfig` (`MageB2B_PricesystemCoreAdvancedConfig`)

To reference matrix pricing in formulas, use:

- `product_customer_matrix`

Formula inputs and fallback behavior are controlled by:

- `pricesystem/price_select_rule/formula_missing_input_behavior`
- `pricesystem/price_select_rule/formula_fallback_strategy`

Sort Order Strategy

Product-Customer Matrix Prices · /productcustomermatrix/price-selection/sort-order-strategy

When `pricesystem/price_select_rule/priceselect = Sort order`, Pricesystem selects the first available price from your configured priority list.

To prioritize matrix pricing, include `product_customer_matrix` near the top.

System Configuration (Global)

Product-Customer Matrix Prices · /productcustomermatrix/price-selection/system-configuration

Global selection strategy is configured in Pricesystem core:

- Admin -> Stores -> Configuration -> MageB2B -> Pricesystem

Global Strategy: Price Select Rule

- `pricesystem/price_select_rule/priceselect` (Lowest / Highest / Sort order; Formula requires Advanced Config Add-On)

Product Customer Matrix Settings

Matrix adds settings under Pricesystem:

- Admin -> Stores -> Configuration -> MageB2B -> Pricesystem -> Product Customer Matrix

Important keys:

- Download link: `pricesystem/productcustomermatrix/enable_download`
- CSV delimiter: `pricesystem/productcustomermatrix/csv_delimeter`
- CSV encloser: `pricesystem/productcustomermatrix/csv_encloser`
- Additional customer attribute: `pricesystem/productcustomermatrix/customer_attribute`
- Custom matrix validation: `pricesystem/productcustomermatrix/custom_matrix_validation`