**Custom Indexer**

**Concepts in indexing**

* Indexing is used to optimize the retrieval of data.
* For example, when products are added or updated, the related data in the database can be restructured or computed for quicker access, enhancing overall performance.

**Indexing Modes**:

Magento supports different indexing modes:

* **Update on Save**: Indexes are updated immediately after a data change.

**Example:**

If you change the price of a product in the admin panel from $100 to $120, Magento will immediately update the product price index to reflect this change without needing to run any additional commands.

* **Update on Schedule**: Indexes are updated on a schedule using a cron job.

**Example:**

Let’s say you change the product price from $100 to $120. This change won't immediately reflect on the frontend because the index will only update when the cron job runs, for example, every hour or day depending on your configuration.

* If a customer visits the product page before the cron job runs, they may still see the $100 price.
* **Update on Demand**: Indexes can be triggered manually through the command line.

**Example:**

You update multiple products’ categories at once, but the changes won’t reflect on the frontend until you manually run a command to reindex:

php bin/magento indexer:reindex catalog\_category\_product

**Types of Indexing**:

* **Full Index**: Processes all records in the index.

**Example:**

You’ve imported 100 new products in bulk. When performing a full index, Magento will process all products (new and old) to ensure everything is up to date.

php bin/magento indexer:reindex

* **Partial Index**: Updates only the affected records based on changes made.

**Example:**

If you change the price for 5 products out of 1000, only those 5 products will be reindexed instead of the entire catalog. This saves time and resources. With partial index (usually happens automatically based on the indexing mode), only changed records are updated.

* **Reindex**: Sometimes you may need to reindex all data after a bulk update or significant change in the store.

**Example:**

You’ve imported a large dataset of products, or you’ve made bulk updates to product attributes. You’ll need to run the following command to reindex all data

This command ensures all indexes (such as prices, stock, categories) are up to date after the bulk change.  
  
php bin/magento indexer:reindex

**Command Line Tools**:

Magento provides command-line tools to manage indexers, such as:

* + php bin/magento indexer:reindex - to reindex all indexers.
  + php bin/magento indexer:info - to view information about each indexer.
  + php bin/magento indexer:set-mode - to change the indexing mode.

**Indexing in Magento 2**

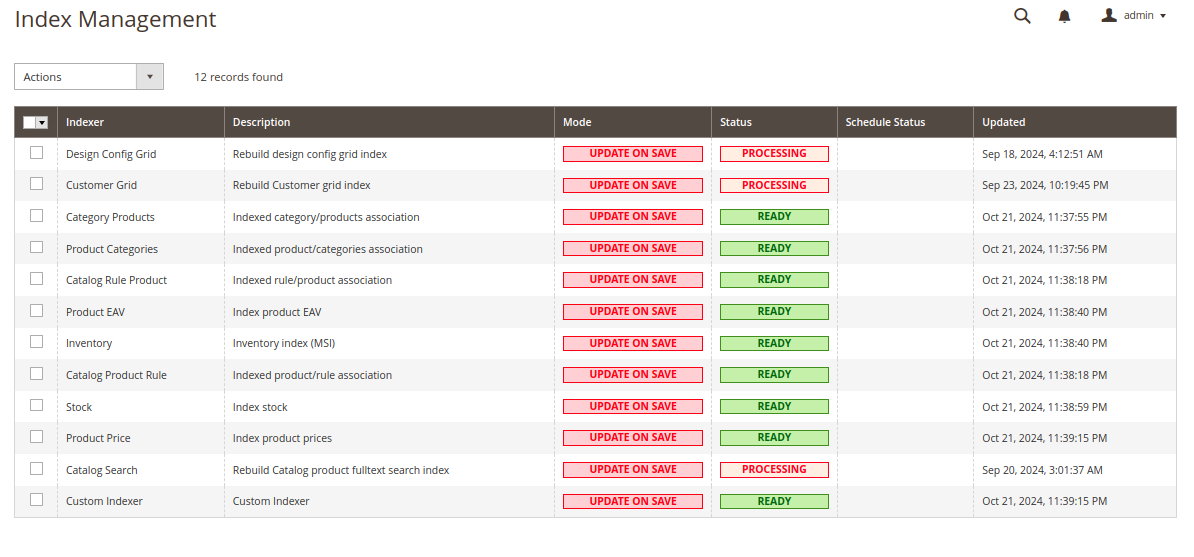
* Indexing is the process that Magento uses to improve performance by pre-calculating and storing data.
* Instead of querying the database every time a page loads, Magento builds and maintains various indexes to speed up the retrieval of data like product prices, inventory levels, and customer information.

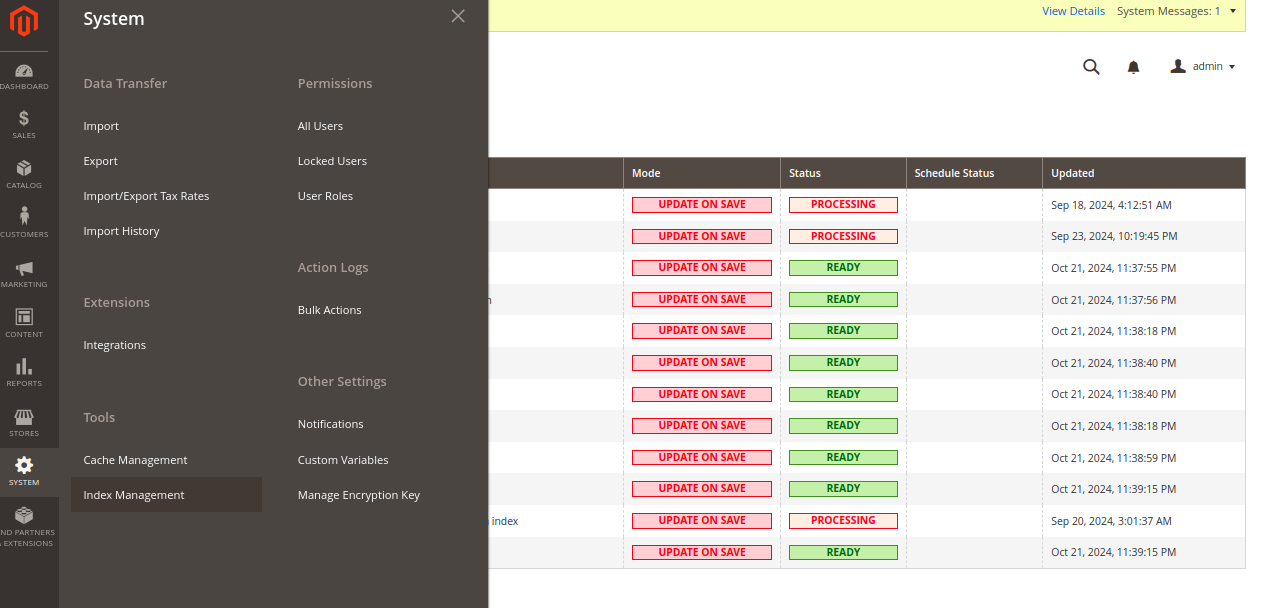
**Need for custom indexer**

* You have custom data that needs to be indexed.
* You want to improve the performance of your store by optimizing data retrieval.
* You need to keep your custom data in sync with changes in your store.

**Create a custom Indexer**

* Indexer is a process that creates and maintains an optimized data structure to improve the performance of the store.
* Magento 2 provides several default indexers, but you can also create custom indexers to add your own business logic to the indexing process.
* For creating your custom indexer use the steps below:
* Create etc/mview.xml file.
* Create etc/indexer.xml file.
* Create indexer class
* MView => The mview.xml file is used to track database changes for a certain entity.
* Indexer => Declare a new indexer process in the etc/indexer.xml file
* This module will respond to changes in the catalog\_product\_entity table





**Advantages**

1. **Performance Boost**: Speeds up data retrieval and enhances page load times.
2. **Tailored Logic**: Allows custom business logic for indexing based on specific needs.
3. **Data Management**: Improves organization and handling of large datasets.
4. **Reduced Server Load**: Minimizes load during peak traffic by optimizing indexing.
5. **Specific Optimization**: Adapts indexing processes for unique products or categories.

**Disadvantages**

1. **Increased Complexity**: Adds layers of complexity to the Magento setup.
2. **Development Time**: Requires additional time and resources to create and maintain.
3. **Bug Risks**: Introduces potential bugs that can be hard to troubleshoot.
4. **Maintenance Dependence**: Requires ongoing updates with the Magento system.
5. **Performance Trade-offs**: May perform worse than built-in indexers if not optimized.

Search Indexing

* Indexing is a process in Magento where certain data (like product attributes) is processed and stored in special database tables to make it easier and faster to retrieve later. For product searches, Magento uses an **indexer** to gather relevant product data (like names, descriptions, prices, etc.) and stores it in an optimized format.
* Magento typically uses either:
* **MySQL** (default) for smaller stores.
* **Elasticsearch** (recommended for larger stores) to manage search indexes efficiently.
* **Gathering Product Data: Magento** collects the necessary product data (like product name, SKU, price, description, etc.) for all products.
* **Processing the Data:** The data is processed, transforming it into a format optimized for search queries. For example, Magento will: Extract keywords from product names or descriptions.

Organize data into categories (like price, product type, etc.).

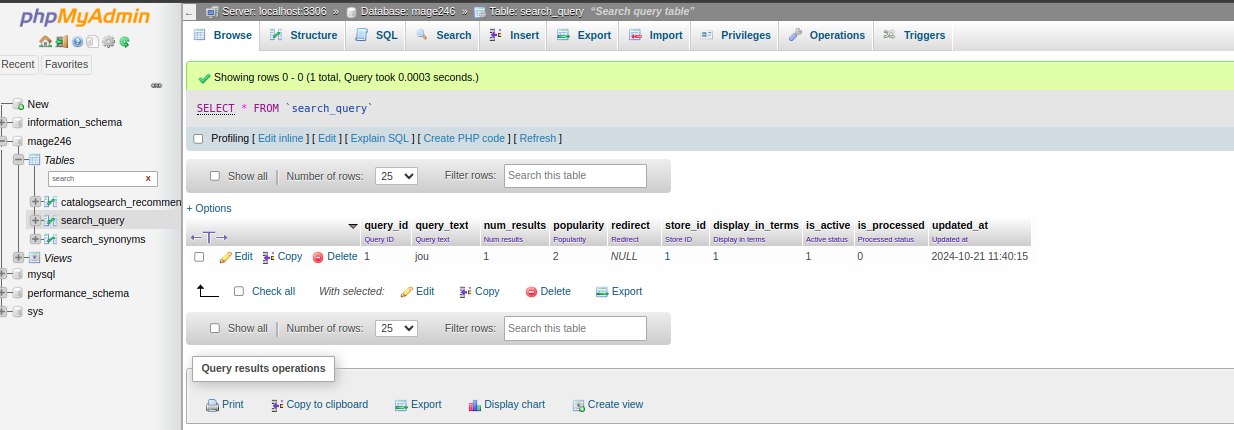
* **Storing the Index :** Magento stores this processed data in index tables in the database (if using MySQL) or in Elasticsearch indexes.

**Retrieving data on demand**

When a user searches for a product, Magento uses the stored index to quickly look up relevant products. The search query is run against the index, and results are retrieved faster because the data is already optimized for search.

**Indexing Process:**

* Suppose you have a product called "Joust Duffle Bag"
* Magento's indexer collects the product name " Joust Duffle Bag." along with other attributes (price, SKU, etc.).
* The indexer breaks down the name " Joust Duffle Bag" into keywords: "Joust" and "Duffle Bag"
* These keywords are stored in the search\_query table (or Elasticsearch index).



**Retrieving Process**

1. A customer searches for the keyword "Joust."
2. Magento looks up this keyword in the index (search\_query table).
3. Magento finds that "Joust" is associated with product ID 1 (Joust Duffle Bag).
4. The product data for "Joust Duffle Bag" is retrieved and shown to the customer.