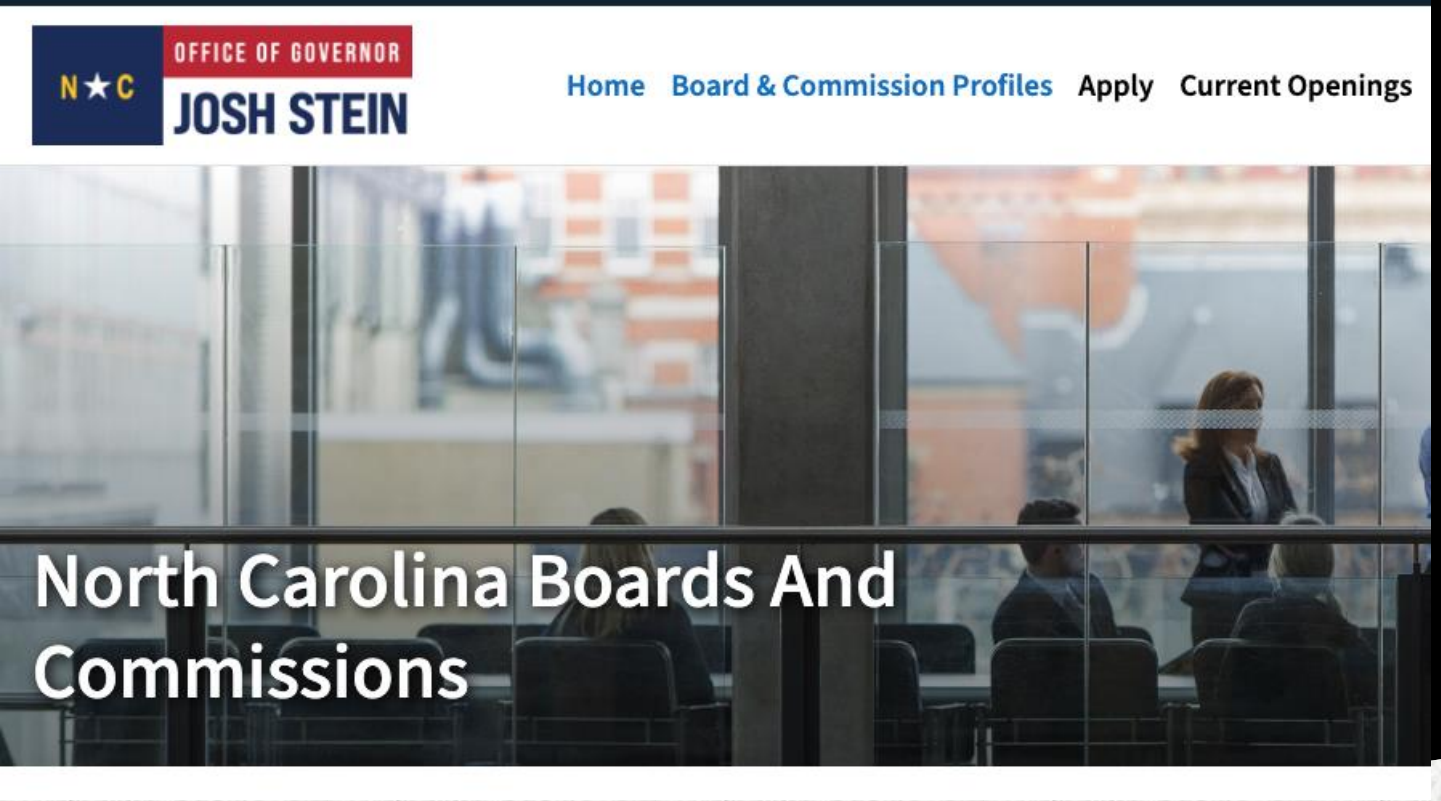


DRUPAL CAMP ASHEVILLE

Leveraging Custom Views Plugins to Display Remote API Data in a Drupal View

Mark Colebank - NCDIT Senior Drupal Application Specialist

Custom Views Plugins

01	Why a Views Query Plugin?
02	Module Scaffolding
03	The API Service Layer — ApiTools.php
04	The Query Plugin — DynamicsApi.php
05	Exposing Fields — hook_views_data()
06	Custom Filter Plugins
07	Caching Strategy
08	Wiring It Together & Demo
Q&A	Questions & Discussion

NCDIT Digital Solutions

- Web Developer since 1996
- Federal Government Contractor
  - NIEHS mobile redesign 2013
  - NTP mobile redesign in 2017
- Working for NCDIT since 2019

- Developed AWS hosted Drupal 10 multi site with minimal dependence on vendors

- 2014 - Digital Solutions Drupal 7 platform
- 2018-2022 Migrated 67 sites to Drupal 8-9
- 2022 NC.Gov Citizen portal redesign

Using AI – Github Copilot

The Approach

- Emphasis on writing prompts to have AI generate the code
- AI (Claude) has become very good in the last year at creating plugins and modules
- Developers understanding the structure of Drupal guide AI to generate code in hours that used to take days

Models Primarily Used

Claude Sonnet
Claude Opus
Copilot CLI (Agentic)

Segment 1 — Why a Views Query Plugin?

The Problem

- Data in an external system (CRM, REST API)
- You want site editors to use the familiar Drupal Views UI
- Sorting, filtering, paging, and theming — all out of the box
- You DON'T want to replicate data into Drupal nodes
- Stale data, migration overhead, sync complexity

Real-World Example

NC Boards & Commissions

300+ records pulled live from Microsoft Dynamics 365 CRM

Displayed as a filterable, searchable table on bc.governor.nc.gov/BoardsAndCommissions

Zero database rows.
Zero content migration.

The Solution: QueryPluginBase

Drupal's Views system is built around query plugins.

Default: SQL

Drupal\views\Plugin\views\query\Sql
Generates SQL against Drupal's DB
Powers every standard View

Your Plugin

Extends QueryPluginBase
Swap in ANY data source
Results appear in standard Views UI

Full Views Power

Filters · Sorting · Paging
Theming · Exposed forms
Caching · REST export

Drupal 10 & 11: Plugin APIs are stable across both versions · core_version_requirement: ^10 || ^11

Segment 2 — Module Scaffolding

Required File Structure

```
nc_views_query/
├── nc_views_query.info.yml
├── nc_views_query.services.yml
├── nc_views_query.module
└── src/
    ├── Service/
    │   └── ApiTools.php
    └── Plugin/
        └── views/
            ├── query/
            │   └── DynamicsApi.php
            └── filter/
                ├── DepartmentFilter.php
                └── CategoryFilter.php
```

Each File's Role

`info.yml`	Module metadata, version requirements, dependencies
`services.yml`	Register ApiTools in Drupal's DI container
`.module`	hook_views_data() — virtual table definition
`ApiTools.php`	OAuth2 token + HTTP request + caching service
`DynamicsApi.php`	The query plugin — execute() maps API → ResultRow[]
`*Filter.php`	Custom filter plugins — dropdown from live API data

Module Scaffolding — info.yml & services.yml

nc_views_query.info.yml

```
name: 'NC Views Query'
type: module
description: 'Views Query plugin for
  displaying Dynamics API data.'
package: North Carolina
core_version_requirement: ^10 || ^11
dependencies:
  - drupal:views
```

nc_views_query.services.yml

```
services:
  nc_views_query.api_tools:
    class: 'Drupal\nc_views_query\
            Service\ApiTools'
```

Key Takeaways

`core_version_requirement: ^10 || ^11`
Supports Drupal 10 and 11 with one module

`dependencies: drupal:views`
Declares Views as a required module

`services.yml registers ApiTools`
Makes it injectable via \Drupal::service()

`D10→D11 Migration Note:`
Annotations still work in D11 but PHP Attributes are the forward-compatible approach

Segment 3 — The API Service Layer: ApiTools.php

A reusable Drupal service that handles three responsibilities:

1. OAuth 2.0

Client Credentials flow
Fetches bearer token
Caches token 55 min
Handles refresh

2. HTTP Request

Builds OData URL
Passes FetchXML
Bearer auth header
guzzle via httpClient()

3. Two-Layer Cache

Token → default bin
API data → data bin
MD5 cache keys
TTL configurable

Usage: \Drupal::service('nc_views_query.api_tools') — called from query plugin and filter plugins

ApiTools.php — getAccessToken()

```
public function getAccessToken()
{
  // Check cache first
  $cached = \Drupal::cache()->get('_bearer_token');
  $access_token = $cached ? $cached->data : null;

  if (!$access_token) {
    $secrets = \Drupal::config('_settings');
    $key = $secrets->get('_secret_key');

    $client = new Client();
    $response = $client->post(
      'https://login.microsoftonline.com/{tenant}/
       oauth2/v2.0/token',
      ['form_params' => [
        'client_id'     => '{client-id}',
        'client_secret' => $key,
        'scope'         => $this->api_url . '.default',
        'grant_type'    => 'client_credentials',
      ]]
    );
    $data = json_decode($response->getBody(), true);
    // Cache for 55 min (token TTL is 60 min)
    \Drupal::cache()->set(
      '_bearer_token', $data['access_token'],
      time() + 3300
    );
    $access_token = $data['access_token'];
  }
```

Key Points

- Store secrets in Drupal config or AWS SSM — never hardcode
- Cache tokens to avoid OAuth rate-limit hits
- Token TTL = 60 min → cache for 55 min
- Throw RuntimeException on auth failure
- OAuth Client Credentials = server-to-server (no user login)

ApiTools.php — getApiData()

```
public function getApiData(
  $xmlfetch, $entity, $headers, int $ttl = 300
) {
  $url = $this->api_url . 'api/data/v9.2/'
       . $entity . '?fetchXml=' . urlencode($xmlfetch);

  // Unique cache key per URL+headers combo
  $cid = 'nc_views_query:api:'
       . md5($url . serialize($headers));

  // Return cached data if available
  if ($ttl > 0 && ($cached = \Drupal::cache('data')->get($cid))) {
    return $cached->data;
  }

  $token = $this->getAccessToken();
  $response = \Drupal::httpClient()->get($url, [
    'headers' => [
      'Authorization' => 'Bearer ' . $token,
      'Accept'        => 'application/json',
    ]
  ]);
  $data = json_decode($response->getBody(), true);

  if ($ttl > 0) {
    \Drupal::cache('data')->set($cid, $data, time() + $ttl);
  }
  return $data;
}
```

Key Points

- Use \Drupal::httpClient()

not new GuzzleHttp\Client()

- MD5 cache key = different

filter combos → separate entries

- 'data' bin = Redis in

production (not 'default')

- Always validate response:

return ['value' => []] on failure

- Pass explicit $ttl so the

query plugin controls freshness

ApiTools.php — Building FetchXML Queries

```
public function getFetchXML(
  $department = '', $category = ''
) {
  $xmlfetch = <<<XML
    <fetch>
      <entity name="bc…">
        <attribute name="bc…"
                   alias="BoardName" />
        <filter type="and">...</filter>
        <order attribute="bc…" />
        ...
    XML;

  // Conditionally inject a filter
  if (!empty($department)) {
    $esc = htmlspecialchars(
      $department,
      ENT_QUOTES | ENT_XML1,
      'UTF-8'
    );
    $xmlfetch .= <<<XML
      <filter>
        <condition attribute="bc…"
          operator="eq"
          value="$esc" />
      </filter>
    XML;
  }
  // ... close XML, return
```

Key Points

- Always htmlspecialchars() user-supplied values into XML
- ENT_QUOTES | ENT_XML1 prevents XML injection attacks
- PHP heredoc (<<<XML) keeps complex XML readable
- Pattern adapts to any API: OData, GraphQL, REST $filter
- Dynamic filter = only one HTTP endpoint needed for all combos

Segment 4 — The Query Plugin: DynamicsApi.php

Drupal 10 — Annotation

```
/**
 * @ViewsQuery(
 *   id = "dynamics_api",
 *   title = @Translation("Dynamics API"),
 *   help = @Translation("Query from Dynamics.")
 * )
 */
class DynamicsApi extends QueryPluginBase
```

Drupal 11 — PHP Attribute (also valid in D10.3+)

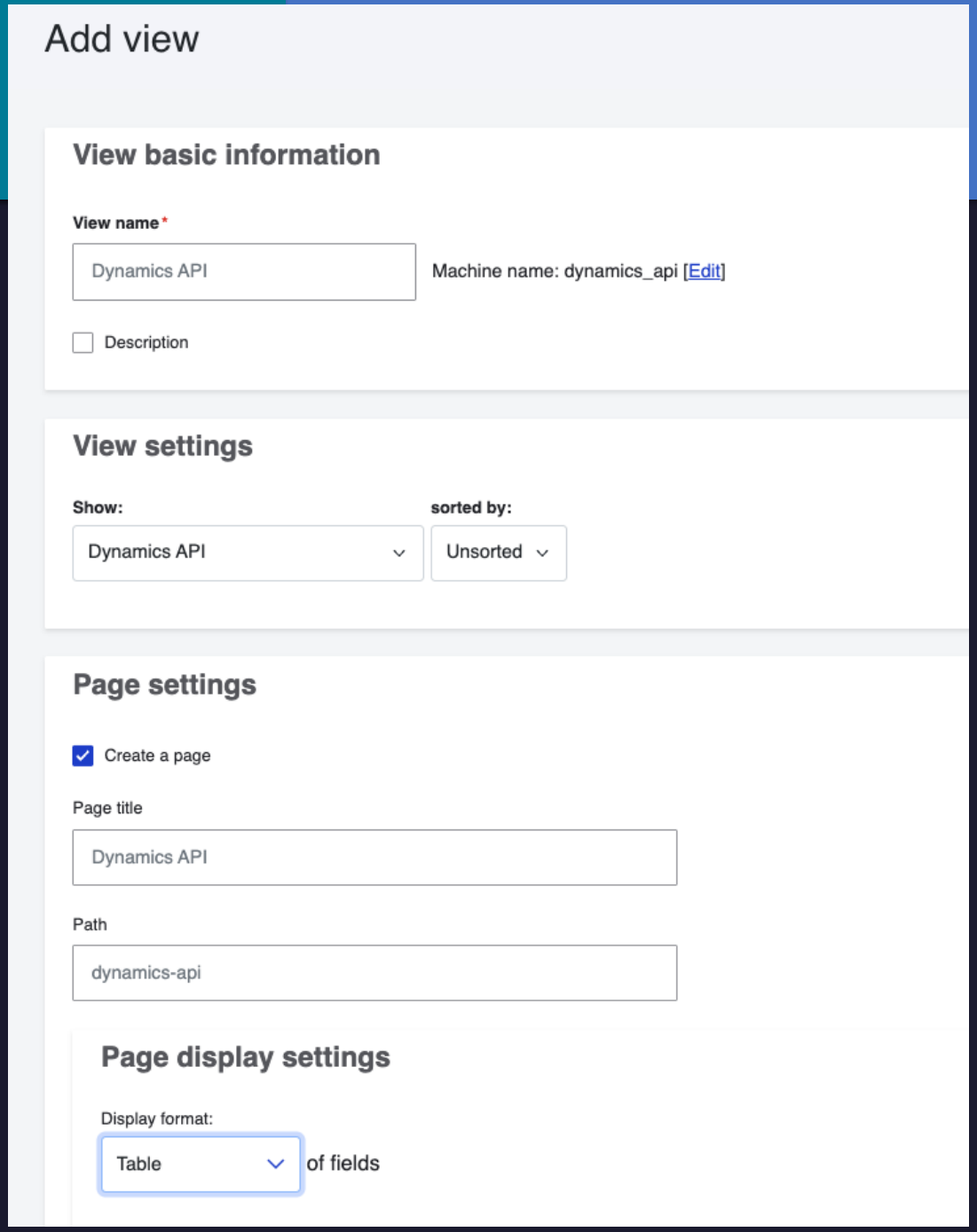
```
#[ViewsQuery(
    id: 'dynamics_api',
    title: new TranslatableMarkup('Dynamics API'),
    help: new TranslatableMarkup('Query from Dynamics.')
)]
class DynamicsApi extends QueryPluginBase
```

Migration Guide: D10 → D11

- Annotations still work in D11 but are soft-deprecated
- PHP Attributes are the forward-compatible approach
- id: 'dynamics_api' must match query_id in hook_views_data()
- Both styles register the same plugin — pick one per file
- Namespace: Drupal\views\Attribute\ViewsQuery (D10.3+ / D11)

Adding View

- In Admin > Structure > Views > Add View
- View Settings pick the Title in Query Plugin Annotation
- Pick page or Block and pick Display Format (ex. Table)



Query Plugin — addWhere(), ensureTable(), addField()

```
class DynamicsApi extends QueryPluginBase
{
    protected array $where   = [];  // collects filter conditions
    protected array $results = [];  // stores query results

    // Views calls this for each active filter
    public function addWhere($group, $field,
                             $value = null, $operator = null)
    {
        // Instead of appending SQL, we stash the value
        // to use when building the API request in execute()
        $this->where[$field] = [
            'value'    => $value,
            'operator' => $operator,
        ];
    }

    // For non-SQL sources, no-op implementations:
    public function ensureTable($table, $relationship = null)
    {
        return $table;          // No SQL JOINs needed
    }

    public function addField($table, $field, $alias = '', $params = [])
    {
        return $alias ?: $field; // Just return field identifier
    }
```

Query Plugin — execute(): The Heart of the Plugin

```
public function execute(ViewExecutable $view)
{
    // 1. Read filter values stored by addWhere()
    $dept = !empty(
      $this->where['dynamics_api.department']['value'])
      ? trim($this->where['dynamics_api.department']['value'])
      : '';
    $cat  = !empty(
      $this->where['dynamics_api.category']['value'])
      ? trim($this->where['dynamics_api.category']['value'])
      : '';

    // 2. Call the API service
    $apiTools = \Drupal::service('nc_views_query.api_tools');
    $xml      = $apiTools->getFetchXML($dept, $cat);
    $data     = $apiTools->getApiData(
      $xml, 'bc_boardcommissions', null, 300
    );

    // 3. Map API rows → ResultRow objects
    $view->result = [];
    foreach ($data['value'] as $item) {
        $row             = new ResultRow();
        $row->id         = $item['bc_boardcommissionid'] ?? '';
        $row->boardname  = trim($item['BoardShortName'] ?? '');
        $row->department = $item['DeptName'] ?? '';
        $row->category   = $item['CatName']  ?? '';
        $view->result[]  = $row;
    }
```

Critical Rules

- ResultRow property names MUST match hook_views_data() field keys
- $view->result = array of ResultRow objects — Views iterates this during render
- Handle deduplication here (merge duplicate board rows before creating ResultRow)
- Always check !empty() and isset() — API can return nulls or missing keys
- Service can be injected via create() factory for better testability

Segment 5 — Exposing Fields: hook_views_data()

The Virtual Table Definition

```
// nc_views_query.module
function nc_views_query_views_data()
{
    $data = [];

    $data['dynamics_api'] = [
        'table' => [
            'group' => t('Dynamics API'),
            'base'  => [
                'field'    => 'id',
                'title'    => t('Dynamics API'),
                'help'     => t('From Dynamics.'),
                // Critical: ties to your plugin id
                'query_id' => 'dynamics_api',
            ],
        ],
    ];
}
```

query_id is the magic connector →

Views UI: 'Show: Dynamics API' uses your plugin

Registering Fields

```
// Field keys MUST match ResultRow
// property names from execute()
'boardname' => [
    'title'  => t('Board Full Name'),
    'field'  => ['id' => 'standard'],
    'filter' => ['id' => 'string'],
    'sort'   => ['id' => 'standard'],
],
'department' => [
    'title'  => t('Department'),
    'field'  => ['id' => 'standard'],
    // Custom filter plugin:
    'filter' => [
      'id' => 'dynamics_api_department_filter'
    ],
    'sort'   => ['id' => 'standard'],
],
'category' => [
    // ... same pattern with
    // dynamics_api_category_filter
],
```

Segment 6 — Custom Filter Plugins

DepartmentFilter.php

```
// D10 annotation / D11 PHP attribute:
#[ViewsFilter("dynamics_api_department_filter")]
class DepartmentFilter extends FilterPluginBase
{
    // 1. Build form element — options from API
    protected function valueForm(
      &$form, FormStateInterface $form_state
    ) {
        $api  = \Drupal::service('nc_views_query.api_tools');
        $data = $api->getApiData(
          $api->getFetchXML(), 'bc_boardcommissions', null
        );
        $options = [];
        foreach ($data['value'] as $item) {
            if (isset($item['DeptName'])) {
                $options[$item['DeptName']] =
                        $item['DeptName'];
            }
        }
        asort($options);
        $form['value'] = [
          '#type'    => 'select',
          '#title'   => $this->t('Department'),
          '#options' => $options,
        ];
    }

    // 2. Pass value into query plugin
    public function query() {
```

The Two-Step Pattern

- valueForm() = builds the exposed filter form element (any Drupal #type works)
- query() = calls addWhere() which routes to your DynamicsApi::addWhere()
- CategoryFilter is identical — just change field key & options source (CatName vs DeptName)
- Options are API-cached so the dropdown doesn't make an uncached HTTP call
- ViewsFilter attribute id must match 'filter' => ['id' => ...] in hook_views_data()

Segment 7 — Caching Strategy

Layer	Bin	TTL	What it stores
OAuth Token	default	55 min	Bearer token string
API Data	data	5 min	Full JSON response array
Render Cache	render	5 min	Views HTML output (CloudFront max-age)

getCacheMaxAge() & getCacheTags() — in DynamicsApi.php

```
// Bubbles to CloudFront:
// Cache-Control: max-age=300, public
public function getCacheMaxAge(): int
{
    return 300; // 5 minutes
}

// Evict render cache when webhook fires:
// Cache::invalidateTags(['nc_views_query:_']);
public function getCacheTags(): array
{
    return ['nc_views_query:bc_boardcommissions'];
```

Cache Bins to Use

`\Drupal::cache()`
'default' — tokens, short metadata

`\Drupal::cache('data')`
'data' — API payloads (Redis in prod)

`Cache::invalidateTags()`
Evicts all layers tagged together

Segment 8 — Wiring It Together: End-to-End Data Flow

Browser Request

User visits View page / applies filter

Views executes View

Calls DynamicsApi::execute()

execute()

Reads $this->where[], calls ApiTools

ApiTools::getApiData()

Checks 'data' cache bin first

Cache miss →

getAccessToken() → checks default cache

OAuth POST

Microsoft token endpoint (cached 55 min)

GET Dynamics OData

API response cached 5 min in 'data' bin

Map → ResultRow[]

$view->result[] = new ResultRow(...)

Views renders rows

Twig templates → HTML

CloudFront

Cache-Control: max-age=300, public

Checklist — From Code to Working View

1. **Enable your module**
   drush en nc_views_query
2. **Clear caches**
   drush cr
3. **Create new View in UI**
   Show: Dynamics API (your base table)
4. **Add Fields**
   Board Name, Department, Category
5. **Add Filters (exposed)**
   Department (select), Category (select)
6. **Save & visit**
   Filter dropdowns populated from live API

Common Q&A

Q: Can I use DI instead of \Drupal::service()?
→ Yes — inject via create() factory method

Q: How do I add paging?
→ Override initQuery(), use getItemsPerPage()

Q: How do I add sorting?
→ Override addOrderBy(), pass to API query

Q: How do I export the View to config?
→ drush cex → set base_table: dynamics_api
and query.type: dynamics_api in YAML

Key Takeaways

QueryPluginBase	The single extension point. Override execute(), addWhere(), ensureTable(), and addField().
hook_views_data()	Creates the virtual table and connects it to your plugin via query_id. Field keys must match ResultRow properties.
FilterPluginBase	Subclass for custom filters. valueForm() builds the widget; query() feeds selected values into addWhere().
Cache in Two Places	Raw API data in the 'data' bin. Render output via getCacheMaxAge() & getCacheTags().
D10 vs D11	Annotations → PHP Attributes is the main migration concern. All plugin APIs are stable across both versions.

Questions?

Drupal Docs

Creating a custom Views query plugin · drupal.org/docs/8/api/views-api

Contact Info

COLEBANK.COM

MARK@COLEBANK.COM