

Queries

TABLE OF CONTENTS

- [entityQuery](#)
 - [Find matching nodes - example 1](#)
 - [Find matching nodes - example 2](#)
 - [Find matching article nodes – example 3](#)
 - [Find nodes that match a taxonomy term](#)
 - [Find 5 nodes that have a matching taxonomy term](#)
 - [Find matching nodes and delete them](#)
 - [Slice up entityQuery results into batches of 100 nodes](#)
 - [Query the creation date \(among other things\) using entityQuery](#)
 - [entityQuery frequently used conditions](#)
 - [Update menu items programatically](#)
 - [Query multi-value fields](#)
 - [Query entity reference fields if they have a value or no value](#)
 - [Query entity reference fields](#)
- [Static and dynamic Queries](#)
 - [Static Queries](#)
 - [Get a connection object](#)
 - [SQL select example](#)
 - [Find the biggest value in a field](#)
 - [SQL update query - example 1](#)
 - [SQL update query - example 2](#)
 - [SQL update query - example 3](#)
 - [SQL insert](#)
 - [SQL Insert Query](#)
 - [SQL Delete query](#)
 - [Paragraph query](#)
 - [Create a custom table for your module](#)
- [Reference](#)

views 151

For most work, I use entityQueries. There are a few circumstances where I've needed to get into the SQL which meant using static or dynamic queries. There are examples of these different techniques below.

entityQuery

Find matching nodes - example 1

In this EntityQuery example we search for nodes of content type (bundle) ws_product and match field_product_sku with the \$sku variable.

```
use Drupal\node\Entity\Node;
```

```
function getProductId($sku) {
    $productId = false;
    $query = \Drupal::entityQuery('node')
        ->condition('type', 'ws_product')
        ->condition('field_product_sku', $sku);

    $nids = $query->execute();
    if ($nids) {
        $nid = array_values($nids);
        $node = Node::load($nid[0]);
        $productId = $node->get('field_product_id')->value;
    }
    return $productId;
}
```

Find matching nodes - example 2

In this entityQuery we search for published nodes of type contract with field_contract_status having the value “Active”. This puts the resulting nids and node titles in a render array for display.

This is a simple query which outputs a bunch of nids and titles

```
public function loadRawSalesforceData() {
    $node_storage = \Drupal::entityTypeManager()->getStorage('node');
    $query = \Drupal::entityQuery('node')
        ->condition('type', 'contract')
        ->condition('status', 1)
        ->condition('field_contract_status', 'Active')
        ->sort('title', 'DESC');

    $nids = $query->execute();
    if ($nids) {
        $nodes = $node_storage->loadMultiple($nids);
        foreach ($nodes as $node) {
            $nid = $node->id();
            $titles[] = [
                '#type' => 'markup',
                '#markup' => "<p>" . "nid=$nid " . "Title=" . $node->getTitle() . "</p>",
            ];
        }
        return $titles;
    }
    return [
        '#markup' => $this->t('nothing, nada, not a sausage'),
    ];
}
```

Find matching article nodes – example 3

This example looks for an entity of type 'article' with the name \$name

```

public function entityExists() {

    $name = 'hello';

    // See if the article named hello exists.

    $query = \Drupal::entityQuery('node')
        ->condition('type', 'article')
        ->condition('title', $name)
        ->count();

    $count_nodes = $query->execute();

    if ($count_nodes == 0) {
        $str = "Found no articles";
    }
    elseif ($count_nodes > 0) {
        $str = "Found $count_nodes articles";
    }

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

Find nodes that match a taxonomy term

Find all nodes that match a term_id and retrieve the first 5 nodes sorted by title. This code also puts them into a render array for display.

```

protected function loadFirstOpinion($term_id) {

    $storage = \Drupal::entityTypeManager()->getStorage('node');
    $query = \Drupal::entityQuery('node')
        ->condition('status', 1)
        ->condition('type', 'opinion')
        ->condition('field_category', $term_id, '=')
        ->sort('title', 'ASC') //or DESC
        ->range(0, 5);

    $nids = $query->execute();
    $nodes = $storage->loadMultiple($nids);

    $render_array = [];
    foreach ($nodes as $node) {
        $render_array[] = [
            '#type' => 'markup',
            '#markup' => '<p>' . $node->getTitle(),
        ];
    }

    return $render_array;
}

```

Find 5 nodes that have a matching taxonomy term

We look for published nodes of node type opinion that have a term in the category field, sorted by title ascending, starting with the first result and giving us 5 results. The resulting titles are put into a render array.

```
protected function loadFirstOpinion($term_id) {
    $storage = \Drupal::entityTypeManager()->getStorage('node');
    $query = \Drupal::entityQuery('node')
        ->condition('status', 1)
        ->condition('type', 'opinion')
        ->condition('field_category', $term_id, '=')
        ->sort('title', 'ASC') //or DESC
        ->range(0, 5);
    $nids = $query->execute();
    $nodes = $storage->loadMultiple($nids);

    $render_array = [];
    foreach ($nodes as $node) {
        $render_array[] = [
            '#type' => 'markup',
            '#markup' => '<p>' . $node->getTitle(),
        ];
    }
    return $render_array;
}
```

Find matching nodes and delete them

```
public function deleteQuery1() {
    $results = \Drupal::entityQuery('node')
        ->condition('type', 'event')
        ->range(0, 10)
        ->execute();

    if ($results) {
        foreach ($results as $result) {
            $node = Node::load($result);
            $node->delete();
        }
    }

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => t("10 nodes deleted."),
    ];

    return $render_array;
}
```

Slice up entityQuery results into batches of 100 nodes

This is often used for batch API operations.

```
$query = \Drupal::entityQuery('node')
->condition('type', 'contract')
->condition('status', 1)
->sort('title', 'ASC');

$nids = $query->execute();
$nid_count = count($nids);

// Grab 100 nids at a time to batch process.
$batches = [];
for ($i=0;$i<=$nid_count;$i+=100) {
    $batches[] = array_slice($nids, $i, 100);
}
```

Query the creation date (among other things) using entityQuery

Note. The created (and changed) field uses a unix timestamp. This is an int 11 field in the db with a value like 1525302749 If you add a Drupal date field, its data looks like 2019-05-15T21:32:00 (varchar 20)

If you want to query a date field in a content type, you will have to fiddle around with the setTimezone stuff that is commented out below. The date field referenced below (field_date) is a standard Drupal date field.

More at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update> and <https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

```

protected function loadOpinionForAYear($year, $term_id) {
    $storage = \Drupal::entityTypeManager()->getStorage('node');

    // Get a date string suitable for use with entity query.
    // $date = new DrupalDateTime(); // now

    $format = 'Y-m-d H:i';
    $start_date = DrupalDateTime::createFromFormat($format, $year . "-01-01 00:00");
    $end_date = DrupalDateTime::createFromFormat($format, $year . "-12-31 23:59");

    $start_date = $start_date->getTimestamp();
    $end_date = $end_date->getTimestamp();

    // $start_date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
    // $end_date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
    // $start_date = $start_date->format(DateTimeItemInterface::DATETIME_STORAGE_FORMAT);
    // $end_date = $end_date->format(DateTimeItemInterface::DATETIME_STORAGE_FORMAT);

    // Set the condition.
    // $query->condition('field_date.value', $start_date, '>=');
    // $query->condition('field_date.value', $end_date, '<=');

    $query = \Drupal::entityQuery('node')
        ->condition('status', 1)
        ->condition('type', 'opinion')
        ->condition('field_category', $term_id, '=')
        ->condition('created', $start_date, '>=')
        ->condition('created', $end_date, '<=')
        ->sort('title', 'DESC');
    $nids = $query->execute();
    $titles = [];
    if ($nids) {
        $nodes = $storage->loadMultiple($nids);
        foreach ($nodes as $node) {
            $titles[] = $node->getTitle();
        }
    }
    return $titles;
}

```

entityQuery frequently used conditions

- Published: ->condition('status', 1)
- Text field not empty: ->condition('field_source_url', "", '<>')
- Field value > 14: ->condition('field_some_field', 14, '>')
- Reference field empty: ->notExists('field_sf_account_ref');
- Null: ->condition(\ \$field, NULL, \IS NULL\);
- Not Null: ->condition(\ \$field, NULL, \IS NOT NULL\);

Lots more at <https://www.drupal.org/docs/8/api/database-api/dynamic-queries/conditions>

Update menu items programatically

To update several items in a menu use hook_update.

```
function park_academy_update_8002() {

  $mids = \Drupal::entityQuery('menu_link_content')
    ->condition('menu_name', 'park-wide-utility')
    ->execute();

  foreach($mids as $mid) {

    $menu_link = \Drupal::entityTypeManager()->getStorage('menu_link_content')->load($mid);

    $title = $menu_link->getTitle();
    if ($title === 'Support') {
      $menu_link->set('weight',2);
      $menu_link->set('expanded', TRUE);
      // $menu_link->set('title','yomama');
      $menu_link->set('link', 'https://www.google.com');
      $menu_link->save();

    }
  }
}
```

Query multi-value fields

When querying multivalue fields, you need to use `%delta` to specify the position (or delta) for the value you are looking for. You also have to identify to the query which position (or delta) you want to query. In the example below, we specify `field_srp_voting_status.%delta` as 1 - indicating the second position (0 based always) and `field_srp_voting_status.%delta.value` for the actual value we are looking for (either accepted, rejected or incomplete):

```
$vote_number = 1;
$query = \Drupal::entityQuery('node')
  ->condition('type', 'correlation', '=')
  ->accessCheck(FALSE)
  ->condition('field_program', $this->programNid, '=')
  ->condition('field_voting_status.%delta', $vote_number, '=')
  ->condition('field_voting_status.%delta.value', [
    'accepted',
    'rejected',
    'incomplete'
  ], 'IN');
$correlation_nids = $query->execute();
$correlation_nids = array_values($correlation_nids);
return $correlation_nids;
```

Query entity reference fields if they have a value or no value

To check if there is a value in an entity reference fields, use the following code

```

$query = \Drupal::entityQuery('node')
->condition('type', 'srp_voting_record')
->accessCheck(FALSE);
if ($vote_type == 'citation') {
    // Check for empty entity reference field.
    $query->notExists('field_ref_error_feedback');
}
if ($vote_type == 'feedback_error'){
    // Check for filled entity reference field.
    $query->exists('field_ref_error_feedback');
}

```

Query entity reference fields

In the following query, we check for a value in the entity that is referenced in the entity reference field? For example, if you have an entity reference field which references node (entity) 27. This query can look in node 27 and check a field value in that node. Here we check in field_first_name for the the value Fred:

```

->condition('field_tks_pub_expectation.entity.field_first_name', 'Fred', '=')

```

For querying for a user id, we query the field_voter.entity:user.uid value. See code below:

```

protected function loadErrorFeedbackVotingRecordNode(int $user_id, int $error_feedback_nid, int $vote_number) {
    $node = [];
    $query = \Drupal::entityQuery('node')
->condition('type', 'srp_voting_record')
->condition('field_voter.entity:user.uid', $user_id)
->condition('field_ref_error_feedback', $error_feedback_nid)
->condition('field_srp_vote_number', $vote_number)
->accessCheck(FALSE);
    $nids = $query->execute();
    if (!empty($nids)) {
        $nid = reset($nids);
        $node = Node::load($nid);
    }
    return $node;
}

```

Static and dynamic Queries

Sometimes you will use static or dynamic queries rather than entityQueries. These use actual SQL versus the entityQuery approach where you build the various parts of the query.

Static Queries

See <https://www.drupal.org/docs/drupal-apis/database-api/static-queries>

Get a connection object

There are two ways to get a connection object:


```

/** @var \Drupal\Core\Database\Connection $connection */
$connection = Database::getConnection();

//OR

/** @var \Drupal\Core\Database\Connection $connection */
$connection = \Drupal::service('database');

```

SQL select example

Static query example from a controller. This loads some fields from the donors table and returns a render array with a count of how many results it found.

```

public function queryBuild1() {
    $database = \Drupal::database();
    $query = $database->query("SELECT id, name, amount FROM {donors}");
    $results = $query->fetchAll();

    $result_count = count($results);

    $str = "Results from db query";
    $str .= "<br/> Result count = $result_count";

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

Find the biggest value in a field

We make a quick query and retrieve the result. In this case we are finding the highest value for the id column.

```

public function highestId() {
    $database = \Drupal::database();
    $connection = Database::getConnection();

    $query = $connection->select('donors', 'n');
    $query->addExpression('MAX(id)', 'id');
    $result = $query->execute();
    $highest_id = intval($result->fetchField());

    $str = "Highest id = $highest_id";
    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

SQL update query - example 1

This shows how to update a status field to the new value \$status when the uuid matches, the event is either update or add and the status is new.

```

public function setStatus(string $uuid, string $status) {
    $db_connection = \Drupal::database();
    $result = $db_connection->update('nocs_info')
        ->fields(['status' => $status])
        ->condition('uuid', $uuid)
        ->condition('event', ['UPDATE', 'ADD'], 'IN')
        ->condition('status', 'new')
        ->execute();

    return $result;
}

```

SQL update query - example 2

```

/**
 * Converts imported eventlog item/s from UPDATE to ADD by uuid.
 *
 * @param string $uuid
 *   UUID for events to convert from UPDATE to ADD.
 *
 * @return mixed
 *   Results of update query.
 */
public function convertUpdateToAddEvent(string $uuid) {
    $update_connection = \Drupal::database();
    $result = $update_connection->update('nocs_connect')
        ->fields([
            'event' => 'ADD',
        ])
        ->condition('uuid', $uuid)
        ->condition('event', 'UPDATE')
        ->condition('status', 'new')
        ->execute();

    return $result;
}

```

SQL update query - example 3

This will update values in a table and return the number of rows updated.

```

//use Drupal\Core\Database\Database;

public function updateQuery1() {
    $database = \Drupal::database();
    $query_string = "Update {donors} set amount=amount+1 where id<=10 ";
    $affectedRows = $database->query($query_string,[],
        ['return' => Database::RETURN_AFFECTED]);
    $str = "Affected rows = $affectedRows";
    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

Note. This will be deprecated in Drupal 11. See

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Database%21Statement.php/function/Statement%3A%3ArowCount/9.3.x>

and

<https://git.drupalcode.org/project/drupal/-/blob/9.5.x/core/lib/Drupal/Core/Database/Connection.php#L968>

SQL insert

From <https://www.drupal.org/docs/drupal-apis/database-api/insert-queries>

Which to use? `$connection->insert()` or `$connection->query()` or what are the difference between `insert()` and `query()`?

- `insert()` has each column specified as a separate entry in the fields array and the code can clean each column value. `query()` has an SQL string with no way of checking individual columns.
- If you use `query()` with placeholders, the code can check the column values but placeholders are just an option, there is no way to ensure your SQL does not contain values not passed through placeholders.
- `insert()` passes the request through a set of hooks to let other modules check and modify your requests. This is the right way to work with other modules.
- `query()` is slightly faster because `query()` does not pass the request through the hooks. You might save processing time but your code will not let other modules help your code.
- `insert()` is more likely to work with other databases and future versions of Drupal.

SQL Insert Query

```

/**
 * @throws \Exception
 */

public function insert() {

    /** @var \Drupal\Core\Database\Connection $connection */
    $connection = \Drupal::service('database');

    // $query = $connection->insert('donors', $options);

    // single insert.
    $result = $connection->insert('donors')
        ->fields([
            'name' => 'Singleton',
            'amount' => 1,
        ])
        ->execute();

    // Note, there is an auto-increment field so insert() returns the value
    // for the new row in $result.
    $str = "Single insert returned auto-increment value of $result";

    // Multi-insert1.
    $result = $connection->insert('donors')
        ->fields(['name', 'amount',])
        ->values(['name' => 'Multiton1', 'amount' => 11,])
        ->values(['name' => 'Multiton1', 'amount' => 22,])
        ->execute();

    $str .= "<br/>Multi-insert1 added 2 rows";

    // Multi-insert2.
    $values = [
        ['name' => 'Multiton2', 'amount' => 111,],
        ['name' => 'Multiton2', 'amount' => 222,],
        ['name' => 'Multiton2', 'amount' => 333,],
        ['name' => 'Multiton2', 'amount' => 444,],
        ['name' => 'Multiton2', 'amount' => 555,],
    ];
    $query = $connection->insert('donors')
        ->fields(['name', 'amount',]);
    foreach ($values as $record) {
        $query->values($record);
    }
    $result = $query->execute();
    $str .= "<br/>Multi-insert2 added 5 rows";

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];

    return $render_array;
}

```

More at <https://www.drupal.org/docs/drupal-apis/database-api/insert-queries>

SQL Delete query

This will return the number of rows affected by the SQL delete query.

```

use Drupal\Core\Database\Database;

public function deleteQuery2() {

  $database = \Drupal::database();
  $query_string = "Delete FROM {donors} where id>10 ";
  $affectedRows = $database->query($query_string,[],['return' => Database::RETURN_AFFECTED]);

  $str = "Affected rows = $affectedRows";
  $render_array['content'] = [
    '#type' => 'item',
    '#markup' => $str,
  ];

  return $render_array;
}

```

Note. This will be deprecated in Drupal 11. See

<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Database%21Statement.php/function/Statement%3A%3ArowCount/9.3.x>
 also <https://git.drupalcode.org/project/drupal/-/blob/9.5.x/core/lib/Drupal/Core/Database/Connection.php#L968>.

Paragraph query

In the txg.theme file this code digs into a table for a paragraph field and grabbing the delta field value.

```

function txg_preprocess_paragraph__simple_card(&$variables) {
  $card_parent = $variables['paragraph']->getParentEntity();
  if ($card_parent->bundle() == 'home_aof_card') {
    $variables['parent_delta'] = 0;
    $database = Database::getConnection();
    $result = $database->query("
      Select delta from node__field_para_aofs n where n.bundle = 'home_page' AND n.field_para_aofs_target_id = :target_id", [':target_id' => $card_parent->id()]);
  };
  if ($result) {
    while ($row = $result->fetchAssoc()) {
      $variables['parent_delta'] = $row['delta'];
    }
  }
  $parent_label = $card_parent->field_ref_aof->entity->label();
  $parent_path = $card_parent->field_ref_aof->entity->toUrl()->toString();
  $variables['parent_label'] = $parent_label;
  $variables['parent_path'] = $parent_path;
}
}

```

Create a custom table for your module

If you need a custom database table (or two) for use in a custom module, you can use hook_schema in your module.install file. This will cause the table(s) to be created at module install time and **removed** at module uninstall time.

```

function nocs_connect_schema() {
  $schema['noccs_connect'] = [
    'description' => 'Stores data from event log used to import/update content to site.',
    'fields' => [
      'id' => [
        'type' => 'int',
        'not null' => TRUE,
        'description' => 'Primary Key: Unique ID of event log event.',
      ],
    ],
  ];
}

```

```

        'uuid' => [
            'type' => 'varchar',
            'length' => 255,
            'not null' => TRUE,
            'description' => "Unique ID for content created or updated.",
        ],
        'nid' => [
            'type' => 'varchar',
            'length' => 255,
            'not null' => FALSE,
            'description' => "Unique ID for content created or updated.",
        ],
        'event' => [
            'type' => 'varchar',
            'length' => 255,
            'not null' => TRUE,
            'description' => 'Type of event e.g. UPDATE or ADD.',
        ],
        'created' => [
            'type' => 'int',
            'not null' => TRUE,
            'description' => 'Content creation date.',
        ],
        'updated' => [
            'type' => 'int',
            'not null' => TRUE,
            'description' => 'Content update date.',
        ],
        'type' => [
            'type' => 'varchar',
            'length' => 255,
            'not null' => TRUE,
            'description' => 'Type of content created or updated.',
        ],
        'version' => [
            'type' => 'int',
            'not null' => TRUE,
            'description' => 'Content version number.',
        ],
        'status' => [
            'type' => 'varchar',
            'length' => 255,
            'not null' => TRUE,
            'default' => 'NEW',
            'description' => 'Status of event log row - NEW, PROCESSING, COMPLETE, or ERROR.',
        ],
    ],
    'primary key' => ['id'],
    'indexes' => [
        'uuid' => ['uuid'],
        'event' => ['event'],
        'type' => ['type'],
    ],
];


return $schema;
}

```

Reference

- [API documentation for query conditions](#)
 - [Entity query cheat sheet](#)
 - [Static queries](#)
 - [Dynamic Queries](#)
 - [Insert Queries](#)
 - [Querying date fields from 2018](#)
 - [Querying date fields](#)
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) is licensed under [CC BY 4.0](#) 

Page last modified: Jul 23 2023.

[Edit this page on GitHub](#)

This site uses [Just the Docs](#), a documentation theme for Jekyll.