Composite Primary Keys

This guide is an introduction to composite primary keys for database tables.

After reading this guide you will be able to:

- Create a table with a composite primary key
- Query a model with a composite primary key
- Enable your model to use a composite primary key for queries and associations
- Create forms for models that use composite primary keys
- Extract composite primary keys from controller parameters
- Use database fixtures for tables with composite primary keys

Chapters

1 What are Composite Primary Keys?

Sometimes a single column's value isn't enough to uniquely identify every row of a table, and a combination of two or more columns is required. This can be the case when using a legacy database schema without a single `id` column as a primary key, or when altering schemas for sharding or multitenancy.

Composite primary keys increase complexity and can be slower than a single primary key column. Ensure your use-case requires a composite primary key before using one.

2 Composite Primary Key Migrations

You can create a table with a composite primary key by passing the `:primary_key` option to `create_table` with an array value:

```
class CreateProducts < ActiveRecord::Migration[8.0]
  def change
    create_table :products, primary_key: [:store_id, :sku] do |t|
      t.integer :store_id
      t.string :sku
      t.text :description
    end
  end
end
```

3 Querying Models

3.1 Using `#find`

If your table uses a composite primary key, you'll need to pass an array when using `#find` to locate a record:

```
# Find the product with store_id 3 and sku "XYZ12345"
irb> product = Product.find([3, "XYZ12345"])
=> #<Product store_id: 3, sku: "XYZ12345", description: "Yellow socks">
```

The SQL equivalent of the above is:

```
SELECT * FROM products WHERE store_id = 3 AND sku = "XYZ12345"
```

To find multiple records with composite IDs, pass an array of arrays to `#find`:

```
# Find the products with primary keys [1, "ABC98765"] and [7, "ZZZ11111"]
irb> products = Product.find([[1, "ABC98765"], [7, "ZZZ11111"]])
=> [
  #<Product store_id: 1, sku: "ABC98765", description: "Red Hat">,
  #<Product store_id: 7, sku: "ZZZ11111", description: "Green Pants">
]
```

The SQL equivalent of the above is:

```
SELECT * FROM products WHERE (store_id = 1 AND sku = 'ABC98765' OR store_id = 7
AND sku = 'ZZZ11111')
```

Models with composite primary keys will also use the full composite primary key when ordering:

```
irb> product = Product.first
=> #<Product store_id: 1, sku: "ABC98765", description: "Red Hat">
```

The SQL equivalent of the above is:

```
SELECT * FROM products ORDER BY products.store_id ASC, products.sku ASC LIMIT
```

3.2 Using `#where`

Hash conditions for `#where` may be specified in a tuple-like syntax. This can be useful for querying composite primary key relations:

```
Product.where(Product.primary_key => [[1, "ABC98765"], [7, "ZZZ11111"]])
```

3.2.1 Conditions with `:id`

When specifying conditions on methods like `find_by` and `where`, the use of `id` will match against an `:id` attribute on the model. This is different from `find`, where the ID passed in should be a primary key value.

Take caution when using `find_by(id:)` on models where `:id` is not the primary key, such as composite primary key models. See the Active Record Querying guide to learn more.

4 Associations between Models with Composite Primary Keys

Rails can often infer the primary key-foreign key relationships between associated models. However, when dealing with composite primary keys, Rails typically defaults to using only part of the composite key, usually the `id` column, unless explicitly instructed otherwise. This default behavior only works if the model's composite primary key contains the `:id` column, *and* the column is unique for all records.

Consider the following example:

```
class Order < ApplicationRecord
  self.primary_key = [:shop_id, :id]
  has_many :books
end

class Book < ApplicationRecord
  belongs_to :order
end
```

In this setup, `Order` has a composite primary key consisting of `[:shop_id, :id]`, and `Book` belongs to `Order`. Rails will assume that the `:id` column should be used as the primary key for the association between an order and its books. It will infer that the foreign key column on the books table is `:order_id`.

Below we create an `Order` and a `Book` associated with it:

```
order = Order.create!(id: [1, 2], status: "pending")
book = order.books.create!(title: "A Cool Book")
```

To access the book's order, we reload the association:

```
book.reload.order
```

When doing so, Rails will generate the following SQL to access the order:

```
SELECT * FROM orders WHERE id = 2
```

You can see that Rails uses the order's `id` in its query, rather than both the `shop_id` and the `id`. In this case, the `id` is sufficient because the model's composite primary key does in fact contain the `:id` column, *and* the column is unique for all records.

However, if the above requirements are not met or you would like to use the full composite primary key in associations, you can set the `foreign_key:` option on the association. This option specifies a composite foreign key on the association; all columns in the foreign key will be used when querying the associated record(s). For example:

```
class Author < ApplicationRecord
  self.primary_key = [:first_name, :last_name]
  has_many :books, foreign_key: [:first_name, :last_name]
end

class Book < ApplicationRecord
  belongs_to :author, foreign_key: [:author_first_name, :author_last_name]
end
```

In this setup, `Author` has a composite primary key consisting of `[:first_name, :last_name]`, and `Book` belongs to `Author` with a composite foreign key `[:author_first_name, :author_last_name]`.

Create an `Author` and a `Book` associated with it:

```
author = Author.create!(first_name: "Jane", last_name: "Doe")
book = author.books.create!(title: "A Cool Book", author_first_name: "Jane",
author_last_name: "Doe")
```

To access the book's author, we reload the association:

```
book.reload.author
```

Rails will now use the `:first_name` *and* `:last_name` from the composite primary key in the SQL query:

```
SELECT * FROM authors WHERE first_name = 'Jane' AND last_name = 'Doe'
```

5 Forms for Composite Primary Key Models

Forms may also be built for composite primary key models. See the Form Helpers guide for more information on the form builder syntax.

Given a `@book` model object with a composite key `[:author_id, :id]`:

```
@book = Book.find([2, 25])
# => #<Book id: 25, title: "Some book", author_id: 2>
```

The following form:

```
<%= form_with model: @book do |form| %>
  <%= form.text_field :title %>
  <%= form.submit %>
<% end %>
```

Outputs:

```
<form action="/books/2_25" method="post" accept-charset="UTF-8" >
  <input name="authenticity_token" type="hidden" value="..." />
  <input type="text" name="book[title]" id="book_title" value="My book" />
  <input type="submit" name="commit" value="Update Book" data-disable-
with="Update Book">
</form>
```

Note the generated URL contains the `author_id` and `id` delimited by an underscore. Once submitted, the controller can extract primary key values from the parameters and update the record. See the next section for more details.

6 Composite Key Parameters

Composite key parameters contain multiple values in one parameter. For this reason, we need to be able to extract each value and pass them to Active Record. We can leverage the `extract_value` method for this use-case.

Given the following controller:

```
class BooksController < ApplicationController
  def show
    # Extract the composite ID value from URL parameters.
    id = params.extract_value(:id)
    # Find the book using the composite ID.
    @book = Book.find(id)
    # use the default rendering behaviour to render the show view.
  end
end
```

And the following route:

```
get "/books/:id", to: "books#show"
```

When a user opens the URL `/books/4_2`, the controller will extract the composite key value `["4", "2"]` and pass it to `Book.find` to render the right record in the view. The `extract_value` method may be used to extract arrays out of any delimited parameters.

7 Composite Primary Key Fixtures

Fixtures for composite primary key tables are fairly similar to normal tables. When using an id column, the column may be omitted as usual:

```
class Book < ApplicationRecord
  self.primary_key = [:author_id, :id]
  belongs_to :author
end
```

```
# books.yml
alices_adventure_in_wonderland:
  author_id: <%= ActiveRecord::FixtureSet.identify(:lewis_carroll) %>
  title: "Alice's Adventures in Wonderland"
```

However, in order to support composite primary key relationships, you must use the `composite_identify` method:

```
class BookOrder < ApplicationRecord
  self.primary_key = [:shop_id, :id]
  belongs_to :order, foreign_key: [:shop_id, :order_id]
  belongs_to :book, foreign_key: [:author_id, :book_id]
end
```

```
# book_orders.yml
alices_adventure_in_wonderland_in_books:
  author: lewis_carroll
  book_id: <%= ActiveRecord::FixtureSet.composite_identify(
              :alices_adventure_in_wonderland, Book.primary_key)[:id] %>
  shop: book_store
  order_id: <%= ActiveRecord::FixtureSet.composite_identify(
              :books, Order.primary_key)[:id] %>
```

Feedback

You're encouraged to help improve the quality of this guide.

Please contribute if you see any typos or factual errors. To get started, you can read our documentation contributions section.

You may also find incomplete content or stuff that is not up to date. Please do add any missing documentation for main. Make sure to check Edge Guides first to verify if the issues are already fixed or not on the main branch. Check the Ruby on Rails Guides Guidelines for style and conventions.

If for whatever reason you spot something to fix but cannot patch it yourself, please open an issue.

And last but not least, any kind of discussion regarding Ruby on Rails documentation is very welcome on the official Ruby on Rails Forum.