DO NOT READ THIS FILE ON GITHUB, GUIDES ARE PUBLISHED ON https://guides.rubyonrails.org.

Active Record and PostgreSQL

This guide covers PostgreSQL specific usage of Active Record.

After reading this guide, you will know:

- How to use PostgreSQL's datatypes.
- How to use UUID primary keys.
- How to implement full text search with PostgreSQL.
- How to back your Active Record models with database views.

In order to use the PostgreSQL adapter you need to have at least version 9.3 installed. Older versions are not supported.

To get started with PostgreSQL have a look at the configuring Rails guide. It describes how to properly set up Active Record for PostgreSQL.

Datatypes

PostgreSQL offers a number of specific datatypes. Following is a list of types, that are supported by the PostgreSQL adapter.

Bytea

- type definition
- functions and operators

```
# db/migrate/20140207133952_create_documents.rb
create_table :documents do |t|
  t.binary 'payload'
end
```

```
# app/models/document.rb
class Document < ApplicationRecord
end
```

```
# Usage
data = File.read(Rails.root + "tmp/output.pdf")
Document.create payload: data
```

Array

- type definition
- functions and operators

```
# db/migrate/20140207133952_create_books.rb
create_table :books do |t|
```

```
  t.string 'title'
  t.string 'tags', array: true
  t.integer 'ratings', array: true
end
add_index :books, :tags, using: 'gin'
add_index :books, :ratings, using: 'gin'
```

```
# app/models/book.rb
class Book < ApplicationRecord
end
```

```
# Usage
Book.create title: "Brave New World",
            tags: ["fantasy", "fiction"],
            ratings: [4, 5]

## Books for a single tag
Book.where("'fantasy' = ANY (tags)")

## Books for multiple tags
Book.where("tags @> ARRAY[?]::varchar[]", ["fantasy", "fiction"])

## Books with 3 or more ratings
Book.where("array_length(ratings, 1) >= 3")
```

Hstore

- type definition
- functions and operators

NOTE: You need to enable the `hstore` extension to use hstore.

```
# db/migrate/20131009135255_create_profiles.rb
class CreateProfiles < ActiveRecord::Migration[7.0]
  enable_extension 'hstore' unless extension_enabled?('hstore')
  create_table :profiles do |t|
    t.hstore 'settings'
  end
end
```

```
# app/models/profile.rb
class Profile < ApplicationRecord
end
```

```
irb> Profile.create(settings: { "color" => "blue", "resolution" => "800x600" })

irb> profile = Profile.first
irb> profile.settings
```

```
=> {"color"=>"blue", "resolution"=>"800x600"}

irb> profile.settings = {"color" => "yellow", "resolution" => "1280x1024"}
irb> profile.save!

irb> Profile.where("settings->'color' = ?", "yellow")
=> #<ActiveRecord::Relation [#<Profile id: 1, settings: {"color"=>"yellow",
"resolution"=>"1280x1024"}>]>
```

JSON and JSONB

- type definition
- functions and operators

```
# db/migrate/20131220144913_create_events.rb
# ... for json datatype:
create_table :events do |t|
  t.json 'payload'
end
# ... or for jsonb datatype:
create_table :events do |t|
  t.jsonb 'payload'
end
```

```
# app/models/event.rb
class Event < ApplicationRecord
end
```

```
irb> Event.create(payload: { kind: "user_renamed", change: ["jack", "john"]})

irb> event = Event.first
irb> event.payload
=> {"kind"=>"user_renamed", "change"=>["jack", "john"]}

## Query based on JSON document
# The -> operator returns the original JSON type (which might be an object), whereas
->> returns text
irb> Event.where("payload->>'kind' = ?", "user_renamed")
```

Range Types

- type definition
- functions and operators

This type is mapped to Ruby `Range` objects.

```
# db/migrate/20130923065404_create_events.rb
create_table :events do |t|
  t.daterange 'duration'
end
```

```
# app/models/event.rb
class Event < ApplicationRecord
end
```

```
irb> Event.create(duration: Date.new(2014, 2, 11)..Date.new(2014, 2, 12))

irb> event = Event.first
irb> event.duration
=> Tue, 11 Feb 2014...Thu, 13 Feb 2014

## All Events on a given date
irb> Event.where("duration @> ?::date", Date.new(2014, 2, 12))

## Working with range bounds
irb> event = Event.select("lower(duration) AS starts_at").select("upper(duration) AS
ends_at").first

irb> event.starts_at
=> Tue, 11 Feb 2014
irb> event.ends_at
=> Thu, 13 Feb 2014
```

Composite Types

- type definition

Currently there is no special support for composite types. They are mapped to normal text columns:

```
CREATE TYPE full_address AS
(
  city VARCHAR(90),
  street VARCHAR(90)
);
```

```
# db/migrate/20140207133952_create_contacts.rb
execute <<-SQL
  CREATE TYPE full_address AS
  (
    city VARCHAR(90),
    street VARCHAR(90)
  );
SQL
create_table :contacts do |t|
  t.column :address, :full_address
end
```

```
# app/models/contact.rb
class Contact < ApplicationRecord
end
```

```
irb> Contact.create address: "(Paris,Champs-Élysées)"
irb> contact = Contact.first
irb> contact.address
=> "(Paris,Champs-Élysées)"
irb> contact.address = "(Paris,Rue Basse)"
irb> contact.save!
```

Enumerated Types

- type definition

The type can be mapped as a normal text column, or to an `ActiveRecord::Enum`.

```
# db/migrate/20131220144913_create_articles.rb
def up
  create_enum :article_status, ["draft", "published"]

  create_table :articles do |t|
    t.enum :status, enum_type: :article_status, default: "draft", null: false
  end
end

# There's no built in support for dropping enums, but you can do it manually.
# You should first drop any table that depends on them.
def down
  drop_table :articles

  execute <<-SQL
    DROP TYPE article_status;
  SQL
end
```

```
# app/models/article.rb
class Article < ApplicationRecord
  enum status: {
    draft: "draft", published: "published"
  }, _prefix: true
end
```

```
irb> Article.create status: "draft"
irb> article = Article.first
irb> article.status_draft!
irb> article.status
=> "draft"
```

```
irb> article.status_published?
=> false
```

To add a new value before/after existing one you should use ALTER TYPE:

```
# db/migrate/20150720144913_add_new_state_to_articles.rb
# NOTE: ALTER TYPE ... ADD VALUE cannot be executed inside of a transaction block so
here we are using disable_ddl_transaction!
disable_ddl_transaction!

def up
  execute <<-SQL
    ALTER TYPE article_status ADD VALUE IF NOT EXISTS 'archived' AFTER 'published';
  SQL
end
```

NOTE: Enum values can't be dropped. You can read why here.

Hint: to show all the values of the all enums you have, you should call this query in `bin/rails db` or `psql` console:

```
SELECT n.nspname AS enum_schema,
       t.typname AS enum_name,
       e.enumlabel AS enum_value
  FROM pg_type t
      JOIN pg_enum e ON t.oid = e.enumtypid
      JOIN pg_catalog.pg_namespace n ON n.oid = t.typnamespace
```

UUID

- type definition
- pgcrypto generator function
- uuid-ossp generator functions

NOTE: You need to enable the `pgcrypto` (only PostgreSQL >= 9.4) or `uuid-ossp` extension to use uuid.

```
# db/migrate/20131220144913_create_revisions.rb
create_table :revisions do |t|
  t.uuid :identifier
end
```

```
# app/models/revision.rb
class Revision < ApplicationRecord
end
```

```
irb> Revision.create identifier: "A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11"

irb> revision = Revision.first
```

```
irb> revision.identifier
=> "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11"
```

You can use `uuid` type to define references in migrations:

```
# db/migrate/20150418012400_create_blog.rb
enable_extension 'pgcrypto' unless extension_enabled?('pgcrypto')
create_table :posts, id: :uuid

create_table :comments, id: :uuid do |t|
  # t.belongs_to :post, type: :uuid
  t.references :post, type: :uuid
end
```

```
# app/models/post.rb
class Post < ApplicationRecord
  has_many :comments
end
```

```
# app/models/comment.rb
class Comment < ApplicationRecord
  belongs_to :post
end
```

See this section for more details on using UUIDs as primary key.

Bit String Types

- type definition
- functions and operators

```
# db/migrate/20131220144913_create_users.rb
create_table :users, force: true do |t|
  t.column :settings, "bit(8)"
end
```

```
# app/models/user.rb
class User < ApplicationRecord
end
```

```
irb> User.create settings: "01010011"
irb> user = User.first
irb> user.settings
=> "01010011"
irb> user.settings = "0xAF"
irb> user.settings
```

```
=> "10101111"
irb> user.save!
```

Network Address Types

- type definition

The types `inet` and `cidr` are mapped to Ruby `IPAddr` objects. The `macaddr` type is mapped to normal text.

```
# db/migrate/20140508144913_create_devices.rb
create_table(:devices, force: true) do |t|
  t.inet 'ip'
  t.cidr 'network'
  t.macaddr 'address'
end
```

```
# app/models/device.rb
class Device < ApplicationRecord
end
```

```
irb> macbook = Device.create(ip: "192.168.1.12", network: "192.168.2.0/24", address:
"32:01:16:6d:05:ef")

irb> macbook.ip
=> #<IPAddr: IPv4:192.168.1.12/255.255.255.255>

irb> macbook.network
=> #<IPAddr: IPv4:192.168.2.0/255.255.255.0>

irb> macbook.address
=> "32:01:16:6d:05:ef"
```

Geometric Types

- type definition

All geometric types, with the exception of `points` are mapped to normal text. A point is casted to an array containing `x` and `y` coordinates.

Interval

- type definition
- functions and operators

This type is mapped to `ActiveSupport::Duration` objects.

```
# db/migrate/20200120000000_create_events.rb
create_table :events do |t|
  t.interval 'duration'
end
```

```ruby
# app/models/event.rb
class Event < ApplicationRecord
end
```

```ruby
irb> Event.create(duration: 2.days)

irb> event = Event.first
irb> event.duration
=> 2 days
```

UUID Primary Keys

NOTE: You need to enable the `pgcrypto` (only PostgreSQL >= 9.4) or `uuid-ossp` extension to generate random UUIDs.

```ruby
# db/migrate/20131220144913_create_devices.rb
enable_extension 'pgcrypto' unless extension_enabled?('pgcrypto')
create_table :devices, id: :uuid do |t|
  t.string :kind
end
```

```ruby
# app/models/device.rb
class Device < ApplicationRecord
end
```

```ruby
irb> device = Device.create
irb> device.id
=> "814865cd-5a1d-4771-9306-4268f188fe9e"
```

NOTE: `gen_random_uuid()` (from `pgcrypto`) is assumed if no `:default` option was passed to `create_table`.

Generated Columns

NOTE: Generated columns are supported since version 12.0 of PostgreSQL.

```ruby
# db/migrate/20131220144913_create_users.rb
create_table :users do |t|
  t.string :name
  t.virtual :name_upcased, type: :string, as: 'upper(name)', stored: true
end

# app/models/user.rb
class User < ApplicationRecord
end
```

```
# Usage
user = User.create(name: 'John')
User.last.name_upcased # => "JOHN"
```

Full Text Search

```
# db/migrate/20131220144913_create_documents.rb
create_table :documents do |t|
  t.string :title
  t.string :body
end

add_index :documents, "to_tsvector('english', title || ' ' || body)", using: :gin,
name: 'documents_idx'
```

```
# app/models/document.rb
class Document < ApplicationRecord
end
```

```
# Usage
Document.create(title: "Cats and Dogs", body: "are nice!")

## all documents matching 'cat & dog'
Document.where("to_tsvector('english', title || ' ' || body) @@ to_tsquery(?)",
                "cat & dog")
```

Optionally, you can store the vector as automatically generated column (from PostgreSQL 12.0):

```
# db/migrate/20131220144913_create_documents.rb
create_table :documents do |t|
  t.string :title
  t.string :body

  t.virtual :textsearchable_index_col,
            type: :tsvector, as: "to_tsvector('english', title || ' ' || body)",
stored: true
end

add_index :documents, :textsearchable_index_col, using: :gin, name: 'documents_idx'

# Usage
Document.create(title: "Cats and Dogs", body: "are nice!")

## all documents matching 'cat & dog'
Document.where("textsearchable_index_col @@ to_tsquery(?)", "cat & dog")
```

Database Views

- view creation

Imagine you need to work with a legacy database containing the following table:

```
rails_pg_guide=# \d "TBL_ART"
                                        Table "public.TBL_ART"
   Column   |            Type             |                          Modifiers
------------+-----------------------------+-------------------------------------------------
 INT_ID     | integer                     | not null default
nextval('"TBL_ART_INT_ID_seq"'::regclass)
 STR_TITLE  | character varying           |
 STR_STAT   | character varying           | default 'draft'::character varying
 DT_PUBL_AT | timestamp without time zone |
 BL_ARCH    | boolean                     | default false
Indexes:
    "TBL_ART_pkey" PRIMARY KEY, btree ("INT_ID")
```

This table does not follow the Rails conventions at all. Because simple PostgreSQL views are updateable by default, we can wrap it as follows:

```
# db/migrate/20131220144913_create_articles_view.rb
execute <<-SQL
CREATE VIEW articles AS
  SELECT "INT_ID" AS id,
         "STR_TITLE" AS title,
         "STR_STAT" AS status,
         "DT_PUBL_AT" AS published_at,
         "BL_ARCH" AS archived
  FROM "TBL_ART"
  WHERE "BL_ARCH" = 'f'
  SQL
```

```
# app/models/article.rb
class Article < ApplicationRecord
  self.primary_key = "id"
  def archive!
    update_attribute :archived, true
  end
end
```

```
irb> first = Article.create! title: "Winter is coming", status: "published",
published_at: 1.year.ago
irb> second = Article.create! title: "Brace yourself", status: "draft",
published_at: 1.month.ago

irb> Article.count
=> 2
irb> first.archive!
```

```
irb> Article.count
=> 1
```

NOTE: This application only cares about non-archived `Articles`. A view also allows for conditions so we can exclude the archived `Articles` directly.

Structure dumps

If your `config.active_record.schema_format` is `:sql`, Rails will call `pg_dump` to generate a structure dump.

You can use `ActiveRecord::Tasks::DatabaseTasks.structure_dump_flags` to configure `pg_dump`. For example, to exclude comments from your structure dump, add this to an initializer:

```
ActiveRecord::Tasks::DatabaseTasks.structure_dump_flags = ['--no-comments']
```