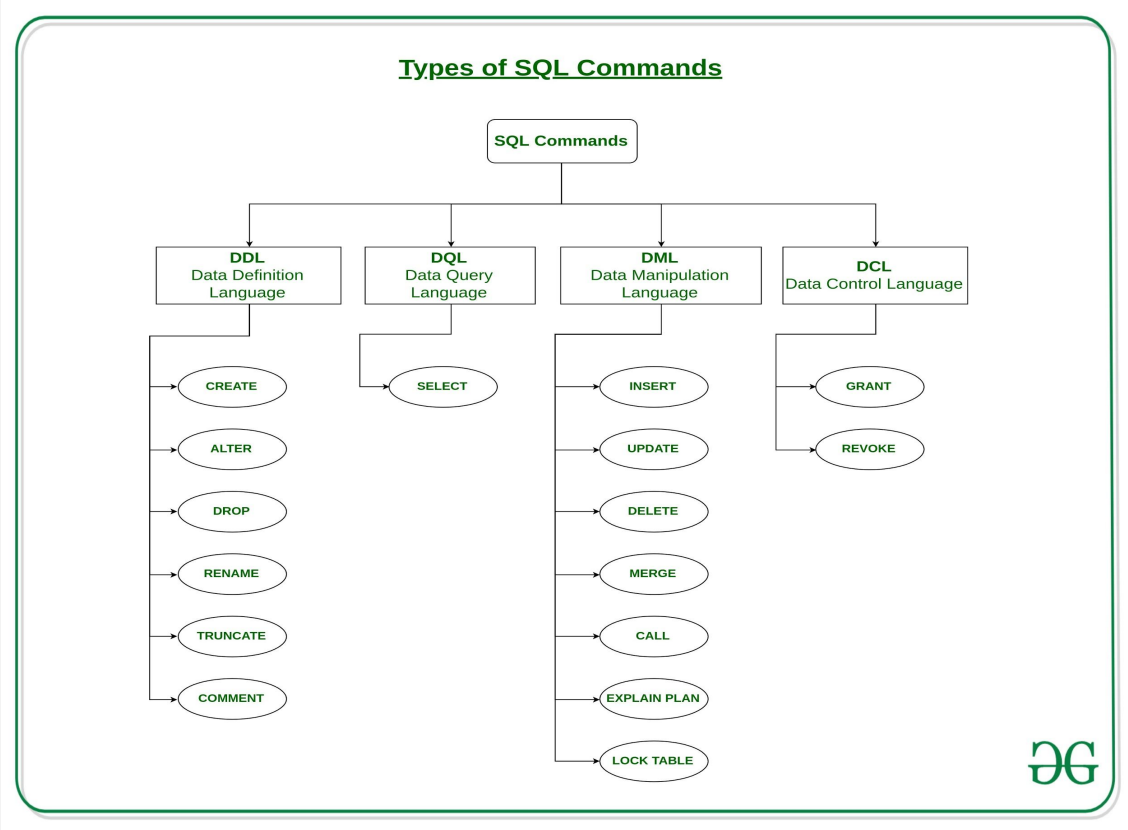
Yalantis

Лекция 13: ActiveRecord

Курс лекций по основами web-разработки на языке программирования Ruby

Yalantis

SQL | DDL, DQL, DML, DCL and TCL Commands



Yalantis

1. **DDL(Data Definition Language)**
   1.1. CREATE - используется для создания базы данных или ее объектов (таких как таблица, индекс, функция, представления, процедура хранения и триггеры).
   1.2. DROP - используется для удаления объектов из базы данных.
   1.3. ALTER - используется для изменения структуры базы данных.
   1.4. TRUNCATE - используется для удаления всех записей из таблицы, включая все места, выделенные для записей.
   1.5. COMMENT - используется для добавления комментариев в словарь данных.
   1.6. RENAME - используется для переименования объекта, существующего в базе данных.
2. **DQL (Data Query Language)**
   2.1. SELECT - используется для извлечения данных из базы данных.
3. **DML(Data Manipulation Language)**
   3.1. INSERT – используется для вставки данных в таблицу
   3.2. UPDATE – используется для обновления данных в таблице.
   3.3. DELETE – используется для удаления записей.
4. **DCL(Data Control Language)**
   4.1. GRANT-дает пользователю права доступа к базе данных.
   4.2. REVOKE - отозвать права доступа пользователя, предоставленные с помощью команды GRANT.
5. **TCL (Transaction control language)**
   5.1. COMMIT– совершает транзакцию.
   5.2. ROLLBACK - откат транзакции в случае возникновения ошибки.
   5.3. SAVEPOINT - устанавливает точку сохранения в транзакции.
   5.4. SET TRANSACTION - указать характеристики для транзакции.

Yalantis

SQL constraints

Constraints SQL используются для указания правил для данных в таблице.

- NOT NULL - гарантирует, что столбец не может иметь значение NULL
- UNIQUE - гарантирует, что все значения в столбце разные
- ПЕРВИЧНЫЙ КЛЮЧ - комбинация NOT NULL и UNIQUE. Уникально идентифицирует каждую строку в таблице
- FOREIGN KEY - уникально идентифицирует строку / запись в другой таблице
- CHECK - Гарантирует, что все значения в столбце удовлетворяют определенному условию
- DEFAULT - устанавливает значение по умолчанию для столбца, если значение не указано
- INDEX - используется для очень быстрого извлечения данных из базы данных

Yalantis

Миграции

Миграции - это простой механизм управления структурой базы данных и данными в ней.

```
class AddSystemSettings < ActiveRecord::Migration[5.0]
  def up
    create_table :system_settings do |t|
      t.string  :name
      t.string  :label
      t.text    :value
      t.string  :type
      t.integer :position
    end

    SystemSetting.create  name:  'notice',
                          label: 'Use notice?',
                          value: 1
  end

  def down
    drop_table :system_settings
  end
end
```

Yalantis

Именование миграций

По умолчанию Rails генерирует миграции, которые выглядят так:

```
20080717013526_your_migration_name.rb
```

Префикс является меткой времени генерации (в UTC).

Если вы предпочитаете использовать числовые префиксы, вы можете отключить метки времени, установив:

```
config.active_record.timestamped_migrations = false
```

В файле `application.rb`

Yalantis

Миграции: создание

```
rails generate migration CreateProducts name:string part_number:string
      invoke  active_record
      create    db/migrate/20200509161755_create_products.rb
```

```
class CreateProducts < ActiveRecord::Migration[6.0]
  def change
    create_table :products do |t|
      t.string :name
      t.string :part_number
    end
  end
end
```

```
CREATE TABLE public.products (
    id bigint NOT NULL,
    name character varying,
    part_number character varying
);
```

Yalantis

Запуск и отмена миграций

```
rails db:migrate
```

```
rails db:rollback VERSION=X
```

```
rails db:rollback STEP=2
```

Yalantis

Доступные преобразования: создание

- create_join_table(table_1, table_2, options): Создает таблицу соединений, имя которой соответствует лексическому порядку первых двух аргументов.
- create_table(name, options): Создает таблицу с именем name и делает объект таблицы доступным для блока, который затем может добавить в него столбцы, следуя тому же формату, что и add_column.
- add_column(table_name, column_name, type, options): Добавляет новый столбец в таблицу с именем table_name с именем column_name, указанным для одного из следующих типов: :string, :text, :integer, :float, :decimal, :datetime, :timestamp, :time, :date, :binary, :boolean.
- add_foreign_key(from_table, to_table, options): Добавляет вторичный ключ. from_table - это таблица с ключевым столбцом, to_table содержит ссылочный первичный ключ.
- add_index(table_name, column_names, options): Добавляет новый индекс с именем столбца. Другие опции включают :name, :unique (пример: { name: 'users_name_index', unique: true }) и :order (пример: { order: { name: :desc } }).
- add_reference(:table_name, :reference_name): Добавляет новый столбец reference_name_id по умолчанию integer.
- add_timestamps(table_name, options): Добавляет timestamps (created_at and updated_at) колонки к table_name.

Yalantis

Доступные преобразования: модификация

- change_column(table_name, column_name, type, options): Изменяет тип колонки на другой в таблице table_name
- change_column_default(table_name, column_name, default_or_changes): Устанавливает значение по умолчанию для column_name, определяемое default_or_changes для table_name. Передача хеша, содержащего: from и: to как default_or_changes, сделает это изменение обратимым в процессе миграции.
- change_column_null(table_name, column_name, null, default = nil): Устанавливает или удаляет constraint + NOT NULL + для column_name. Пустой флаг указывает, может ли значение быть NULL.
- change_table(name, options): Позволяет вносить изменения в столбец таблицы с именем name. Это делает объект таблицы доступным в блоке, где затем можно добавлять / удалять столбцы, индексы или внешние ключи к нему.
- rename_column(table_name, column_name, new_column_name): Переименовывает столбец, но сохраняет тип и содержание.
- rename_index(table_name, old_name, new_name): Переименовывает индекс.
- rename_table(old_name, new_name): Переименовывает таблицу.

Yalantis

Доступные преобразования: удаление

- drop_table (name): удаляет таблицу с именем name.
- drop_join_table (table_1, table_2, options): удаляет таблицу соединений, заданную данными аргументами.
- remove_column (table_name, column_name, type, options): удаляет столбец с именем column_name из таблицы с именем table_name.
- remove_columns (table_name, * column_names): удаляет указанные столбцы из определения таблицы.
- remove_foreign_key (from_table, to_table = nil, ** options): удаляет указанный foreign_key из таблицы с именем table_name.
- remove_index (table_name, column: column_names): удаляет index, указанный в column_names.
- remove_index (table_name, name: index_name): удаляет index, указанный в index_name.
- remove_reference (table_name, ref_name, options): удаляет ссылку (и) на table_name, указанное в ref_name.
- remove_timestamps (table_name, options): удаляет столбцы timestamps (created_at и udpated_at) из table_name.

Yalantis

Обратимые миграции

Обратимые миграции - это миграции, которые знают, как выполнить «down». Вы просто предоставляете логику «up», и система миграции выясняет, как выполнять команды «down».

```
class TenderloveMigration < ActiveRecord::Migration[5.0]
  def change
    create_table(:horses) do |t|
      t.column :content, :text
      t.column :remind_at, :datetime
    end
  end
end
```

Для получения списка команд, которые являются обратимыми, см. ActiveRecord::Migration::CommandRecorder.

Yalantis

Использование модели после обновления ее таблицы

Иногда нужно добавить столбец в миграцию и заполнить его сразу после. В этом случае нужно будет вызвать Base # reset_column_information, чтобы убедиться, что модель содержит самые последние данные столбца после добавления нового столбца.

```
class AddPeopleSalary < ActiveRecord::Migration[5.0]
  def up
    add_column :people, :salary, :integer
    Person.reset_column_information
    Person.all.each do |p|
      p.update_attribute :salary, SalaryCalculator.compute(p)
    end
  end
end
```

Yalantis

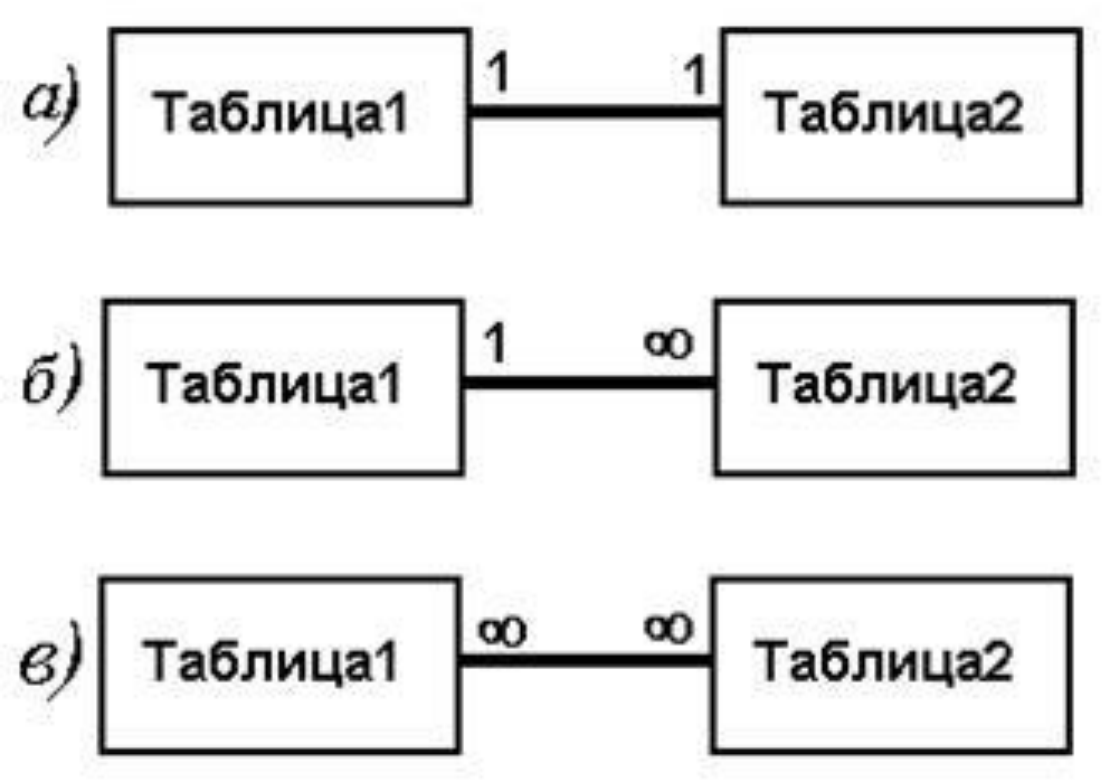
Reversible

Используется для указания операции, которую можно запустить в том или ином направлении

```
class ReplaceFirstNameAndLAstNameWIthName < ActiveRecord::Migration
  def change
    add_column :users, :name, :string

    reversible do |direction|
      direction.up do
       say_with_time ‘Generating names’ do
         execute "UPDATE users SET name = concat(first_name, ' ', last_name)"
       end
      end

      direction.down do
        User.reset_column_information
        User.all.each do |user|
          user.first_name, user.last_name = user.name.split(' ')
          user.save
        end
      end
    end

    revert do
      add_column :users, :first_name, :string
      add_column :users, :last_name, :string
    end
  end
end
```

Yalantis

Отношения между таблицами



Yalantis

Ассоциации

```
class Project < ActiveRecord::Base
  belongs_to                    :portfolio
  has_one                       :project_manager
  has_many                      :milestones
  has_and_belongs_to_many :categories
  has_many :categories, through: :category_projects
end
```

Класс проекта теперь имеет следующие методы (и более), чтобы облегчить обход и манипулирование его отношениями:

```
Project#portfolio, Project#portfolio=(portfolio), Project#reload_portfolio

Project#project_manager, Project#project_manager=(project_manager), Project#reload_project_manager

Project#milestones.empty?, Project#milestones.size, Project#milestones, Project#milestones<<(milestone),
Project#milestones.delete(milestone), Project#milestones.destroy(milestone), Project#milestones.find(milestone_id),
Project#milestones.build, Project#milestones.create

Project#categories.empty?, Project#categories.size, Project#categories, Project#categories<<(category1),
Project#categories.delete(category1), Project#categories.destroy(category1)
```

Yalantis

Belongs_to

```
belongs_to :firm, foreign_key: "client_of"
belongs_to :person, primary_key: "name", foreign_key: "person_name"
belongs_to :author, class_name: "Person", foreign_key: "author_id"
belongs_to :valid_coupon, ->(o) { where "discounts > ?", o.payments_count },
                          class_name: "Coupon", foreign_key: "coupon_id"
belongs_to :attachable, polymorphic: true
belongs_to :project, -> { readonly }
belongs_to :post, counter_cache: true
belongs_to :comment, touch: true
belongs_to :company, touch: :employees_last_updated_at
belongs_to :user, optional: true
belongs_to :account, default: -> { company.account }
```

Yalantis

Has_one

```
has_one :credit_card, dependent: :destroy  # destroys the associated credit card
has_one :credit_card, dependent: :nullify  # updates the associated records foreign
                                           # key value to NULL rather than destroying it
has_one :last_comment, -> { order('posted_on') }, class_name: "Comment"
has_one :project_manager, -> { where(role: 'project_manager') }, class_name: "Person"
has_one :attachment, as: :attachable
has_one :boss, -> { readonly }
has_one :club, through: :membership
has_one :primary_address, -> { where(primary: true) }, through: :addressables, source: :addressable
has_one :credit_card, required: true
```

Yalantis

has_many

```
has_many :comments, -> { order("posted_on") }
has_many :comments, -> { includes(:author) }
has_many :people, -> { where(deleted: false).order("name") }, class_name: "Person"
has_many :tracks, -> { order("position") }, dependent: :destroy
has_many :comments, dependent: :nullify
has_many :tags, as: :taggable
has_many :reports, -> { readonly }
has_many :subscribers, through: :subscriptions, source: :user
```

Yalantis

has_and_belongs_to_many

```
has_and_belongs_to_many :projects
has_and_belongs_to_many :projects, -> { includes(:milestones, :manager) }
has_and_belongs_to_many :nations, class_name: "Country"
has_and_belongs_to_many :categories, join_table: "prods_cats"
has_and_belongs_to_many :categories, -> { readonly }
```

Yalantis

nested_attributes

```
class Member < ActiveRecord::Base
  has_one :avatar
  accepts_nested_attributes_for :avatar
end
```

```
params = { member: { name: 'Jack', avatar_attributes: { icon: 'smiling' } } }
member = Member.create(params[:member])
member.avatar.id # => 2
member.avatar.icon
```

Yalantis

Валидации: пример использования

```
class Person < ApplicationRecord
  validates :name, presence: true
end

>> p = Person.new
# => #<Person id: nil, name: nil>
>> p.errors.messages
# => {}

>> p.valid?
# => false
>> p.errors.messages
# => {name:["can't be blank"]}

>> p = Person.create
# => #<Person id: nil, name: nil>
>> p.errors.messages
# => {name:["can't be blank"]}

>> p.save
# => false

>> p.save!
# => ActiveRecord::RecordInvalid: Validation failed: Name can't be blank

>> Person.create!
# => ActiveRecord::RecordInvalid: Validation failed: Name can't be blank
```

Yalantis

Примеры валидаций

```
validates :terms_of_service, acceptance: true

has_many :books
validates_associated :books

validates :email, confirmation: true

validates :subdomain, exclusion: { in: %w(www us ca jp),
    message: "%{value} is reserved." }

validates :size, inclusion: { in: %w(small medium large),
    message: "%{value} is not a valid size" }

validates :legacy_code, format: { with: /\A[a-zA-Z]+\z/,
    message: "only allows letters" }

validates :name, length: { minimum: 2 }
validates :bio, length: { maximum: 500 }
validates :password, length: { in: 6..20 }
validates :registration_number, length: { is: 6 }
```

```
validates :points, numericality: true
validates :games_played, numericality: { only_integer: true }

validates :name, :login, :email, presence: true

validates :email, uniqueness: true

class GoodnessValidator < ActiveModel::Validator
  def validate(record)
    if record.first_name == "Evil"
      record.errors[:base] << "This person is evil"
    end
  end
end

class Person < ApplicationRecord
  validates_with GoodnessValidator
end
```

Yalantis

Примеры валидаций

```
class NewUserContract < Dry::Validation::Contract
  params do
    required(:email).filled(:string)
    required(:age).value(:integer)
  end

  rule(:email) do
    unless /\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i.match?(value)
      key.failure('has invalid format')
    end
  end

  rule(:age) do
    key.failure('must be greater than 18') if value <= 18
  end
end

contract = NewUserContract.new

contract.call(email: 'jane@doe.org', age: '17')
# #<Dry::Validation::Result{:email=>"jane@doe.org", :age=>17} errors={:age=>["must be greater than 18"]}>
```

Yalantis

Колбеки

```
CALLBACKS = [ :after_initialize, :after_find, :after_touch, :before_validation, :after_validation,
:before_save, :around_save, :after_save, :before_create, :around_create, :after_create, :before_update,
:around_update, :after_update, :before_destroy, :around_destroy, :after_destroy, :after_commit,
:after_rollback ]
```

Пример инициированных обратных вызовов, когда происходит ActiveRecord :: Base # save для новой записи:

```
(-) save
(-) valid
(1) before_validation
(-) validate
(2) after_validation
(3) before_save
(4) before_create
(-) create
(5) after_create
(6) after_save
(7) after_commit
```

Yalantis

Получение объектов из базы данных

annotate
find
create_with
distinct
eager_load
extending
extract_associated
from
group
having
includes
joins
left_outer_joins
limit

lock
none
offset
optimizer_hints
order
preload
readonly
references
reorder
reselect
reverse_order
select
where

Yalantis

find, find_by, find_by!, where

```
# Find the client with primary key (id) 10.
client = Client.find(10)

# => #<Client id: 10, first_name: "Ryan">

# raise an ActiveRecord::RecordNotFound exception if no
matching record is found.
----------------------------------------------------------------------

# find_by method finds the first record matching some
conditions. For example:

Client.find_by first_name: 'Lifo'
# => #<Client id: 1, first_name: "Lifo">

Client.find_by first_name: 'Jon'
# => nil
```

```
# the find_by! method behaves exactly like find_by, except
that it will raise ActiveRecord::RecordNotFound if no
matching record is found. For example:

Client.find_by! first_name: 'does not exist'
# => ActiveRecord::RecordNotFound

----------------------------------------------------------------------

Client.where(first_name: 'Lifo').first

# is equivalent to Client.find_by first_name: 'Lifo'

SQL:
SELECT * FROM clients WHERE
(clients.first_name = 'Lifo') LIMIT 1
```

Yalantis

find_each

```
# This may consume too much memory if the table is big.
User.all.each do |user|
  NewsMailer.weekly(user).deliver_now
end
```

```
User.find_each do |user|
  NewsMailer.weekly(user).deliver_now
end
```

```
User.where(weekly_subscriber: true).find_each do |user|
  NewsMailer.weekly(user).deliver_now
end
```

```
User.find_each(batch_size: 5000) do |user|
  NewsMailer.weekly(user).deliver_now
end
```

Yalantis

where

Pure String Conditions

```
Client.where("orders_count = #{params[:orders]}") # bad
```

Array Conditions

```
Client.where("orders_count = ?", params[:orders]) # good
```

Hash Conditions

```
Client.where(orders_count: params[:orders]) # good
```

Placeholder Conditions

```
Client.where("created_at >= :start_date AND created_at <=
:end_date",
  { start_date: params[:start_date], end_date: params[:end_date] }) #
good
```

Range Conditions

```
Client.where(created_at: (Time.now.midnight -
1.day)..Time.now.midnight)
```

```
SELECT * FROM clients WHERE (clients.created_at
BETWEEN '2008-12-21 00:00:00' AND '2008-12-22 00:00:00')
```

Subset Conditions

```
Client.where(orders_count: [1,3,5])
```

```
SELECT * FROM clients WHERE (clients.orders_count IN (1,3,5))
```

OR Conditions

```
Client.where(locked: true).or(Client.where(orders_count: [1,3,5]))
```

```
SELECT * FROM clients WHERE (clients.locked = 1 OR
clients.orders_count IN (1,3,5))
```

NOT Conditions

```
Client.where.not(locked: true)
```

```
SELECT * FROM clients WHERE (clients.locked != 1)
```

Yalantis

order, select

```
Client.order(:created_at)
# OR
Client.order("created_at")
You could specify ASC or DESC as well:

Client.order(created_at: :desc)
# OR
Client.order(created_at: :asc)
# OR
Client.order("created_at DESC")
# OR
Client.order("created_at ASC")
Or ordering by multiple fields:

Client.order(orders_count: :asc, created_at: :desc)
# OR
Client.order(:orders_count, created_at: :desc)
# OR
Client.order("orders_count ASC, created_at DESC")
# OR
Client.order("orders_count ASC", "created_at DESC")

Client.order("orders_count ASC").order("created_at DESC")
# SELECT * FROM clients ORDER BY orders_count ASC, created_at
DESC
```

```
Client.select(:viewable_by, :locked)
# OR
Client.select("viewable_by, locked")

SELECT viewable_by, locked FROM clients
-------------------------------------------------------------------------------
Client.select(:name).distinct

SELECT DISTINCT name FROM clients
-------------------------------------------------------------------------------
query = Client.select(:name).distinct

query.distinct(false)
# => Returns all names, even if there are duplicates
```

Yalantis

limit and offset

```
Client.limit(5)

SELECT * FROM clients LIMIT 5
--------------------------------------------------------------------------------

Client.limit(5).offset(30)

SELECT * FROM clients LIMIT 5 OFFSET 30
```

Yalantis

group and having

```
Order.select("date(created_at) as ordered_date, sum(price) as
total_price").group("date(created_at)")


SELECT date(created_at) as ordered_date, sum(price) as total_price
FROM orders
GROUP BY date(created_at)
--------------------------------------------------------------------------------
Order.group(:status).count
# => { 'awaiting_approval' => 7, 'paid' => 12 }

SELECT COUNT (*) AS count_all, status AS status
FROM "orders"
GROUP BY status
--------------------------------------------------------------------------------
```

aggregate functions: COUNT, MAX, MIN, SUM, AVG

```
Order.select("date(created_at) as ordered_date, sum(price) as
total_price").
  group("date(created_at)").having("sum(price) > ?", 100)

SELECT date(created_at) as ordered_date, sum(price) as total_price
FROM orders
GROUP BY date(created_at)
HAVING sum(price) > 100
```

Yalantis

joining tables

```
Author.joins("INNER JOIN posts ON posts.author_id = authors.id
AND posts.published = 't'")


SELECT authors.* FROM authors INNER JOIN posts ON
posts.author_id = authors.id AND posts.published = 't'
--------------------------------------------------------------------------------
class Article < ApplicationRecord
  belongs_to :category
  has_many :comments
end

Article.joins(:category, :comments)

SELECT articles.* FROM articles
  INNER JOIN categories ON categories.id = articles.category_id
  INNER JOIN comments ON comments.article_id = articles.id
--------------------------------------------------------------------------------
```

```
Author.left_outer_joins(:posts).distinct.select('authors.*,
COUNT(posts.*) AS posts_count').group('authors.id')
Which produces:

SELECT DISTINCT authors.*, COUNT(posts.*) AS posts_count
FROM "authors"
LEFT OUTER JOIN posts ON posts.author_id = authors.id GROUP
BY authors.id
```

Yalantis

eager loading associations: includes, preload, eager_load

```
# N + 1 problem

clients = Client.limit(10)

clients.each do |client|
  puts client.address.postcode
end

--------------------------------------------------------------------------------
Preload loads the association data in a separate query.

User.preload(:posts)

# =>
SELECT "users".* FROM "users"
SELECT "posts".* FROM "posts"  WHERE "posts"."user_id" IN (1)

User.preload(:posts).where("posts.desc='ruby is awesome'")

# =>
SQLite3::SQLException: no such column: posts.desc:
SELECT "users".* FROM "users"  WHERE (posts.desc='ruby is
awesome')
```

```
Includes loads the association data in a separate query just like
preload. However it is smarter than preload.

User.includes(:posts).where('posts.desc = "ruby is awesome"').to_a

# =>
SELECT "users"."id" AS t0_r0, "users"."name" AS t0_r1, "posts"."id"
AS t1_r0,
       "posts"."title" AS t1_r1,
       "posts"."user_id" AS t1_r2, "posts"."desc" AS t1_r3
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user_id" =
"users"."id"
WHERE (posts.desc = "ruby is awesome")
--------------------------------------------------------------------------------
eager loading loads all association in a single query using LEFT
OUTER JOIN.

User.eager_load(:posts)

# =>
SELECT "users"."id" AS t0_r0, "users"."name" AS t0_r1, "posts"."id"
AS t1_r0,
       "posts"."title" AS t1_r1, "posts"."user_id" AS t1_r2, "posts"."desc"
AS t1_r3
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user_id" =
"users"."id"
```

Yalantis

scopes

```
class Article < ApplicationRecord
  scope :created_before, ->(time) { where("created_at < ?", time) }
end

Article.created_before(Time.zone.now)

class Article < ApplicationRecord
  def self.created_before(time)
    where("created_at < ?", time)
  end
End
```

Conditionals:

```
class Article < ApplicationRecord
  scope :created_before, ->(time) { where("created_at < ?", time) if time.present? }
end

class Article < ApplicationRecord
  def self.created_before(time)
    where("created_at < ?", time) if time.present?
  end
End
```

A scope will always return an ActiveRecord::Relation object, even if the conditional evaluates to false, whereas a class method, will return nil. This can cause NoMethodError when chaining class methods with conditionals, if any of the conditionals return false.

Yalantis

Transactions

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end

---------------------------------------------------------------------------
Nested transactions

User.transaction do
  User.create(username: 'Kotori')
  User.transaction do
    User.create(username: 'Nemu')
    raise ActiveRecord::Rollback
  end
end
---------------------------------------------------------------------------
```

Yalantis

Не ActiveRecord единым

1. ROM (Ruby object mapper) - https://rom-rb.org/
2. Sequel - https://github.com/jeremyevans/sequel

Yalantis

Что почитать?

1. https://api.rubyonrails.org/classes/ActiveRecord/Migration.html
2. https://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html
3. https://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html
4. https://api.rubyonrails.org/classes/ActiveModel/Validator.html
5. https://dry-rb.org/gems/dry-validation/1.5/
6. https://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html
7. https://guides.rubyonrails.org/active_record_querying.html

Yalantis

Thanks!

Any questions? Feel free to contact us hello@yalantis.com