## 3 ways to work with Time in Postgres (& ActiveRecord) Cheatsheet

Let's say you're developing a project management app with users and projects and the product manager asks you:

1. How much time does it take a given user to create their first project?
2. Give me a list of users who took longer than 1 month to create their first project.
3. What is the average amount of time that it takes for a user to create their first project?
4. What is the average amount of time per user between project creation?

Our models look like this:

```
class User < ActiveRecord::Base
  has_many :projects
end

class Project < ActiveRecord::Base
  belongs_to :user
end
```

# A brief primer on the `SELECT` function in SQL

A `SELECT` statement retrieves zero or more rows from one or more database tables or database views.

So something like:

```
SELECT * FROM users
```

will retrieve all available rows from the `users` table.

```
SELECT MIN(created_at) FROM users
```

will return the earliest `created_at` (which is a timestamp) in the users table.

---

# The Difference Operator

In Postgres, we can use the arithmetic difference operator `-` to calculate the difference between two timestamps or dates. The data type matters - if we calculate the difference between two timestamps, the return value is an "interval", and if we calculate the difference between two dates, the return value is an integer representing the number of days between the two dates.

To get the time it takes each user to create their first project:

```
    User.joins(:projects)
        .where('projects.created_at = (SELECT
MIN(projects.created_at) FROM projects WHERE projects.user_id =
users.id)')
        .select("users.email, (projects.created_at -
users.created_at) as time_to_first_project")
```

Intervals in Postgres are the largest datatype available for storing time and consequently contain a lot of detail. If you inspect `time_to_first_project` for a few records, you'll notice that they look something like: `"00:18:43.082321"` or `"9 days 12:48:48.220725"`, which means you might have to do a bit of parsing and/or decorating before you present the information to the user.

Postgres also makes available the `AGE(..)` function, to which you can pass in 2 timestamps and get an interval. If you pass in one timestamp to `AGE()`, you'll get back the difference between the current date (at midnight) and the passed-in timestamp.

If you have two timestamps and you want find the number of days between them, then you can use the `CAST` function to take advantage of the fact that when you subtract two dates you get the days between them.

---

# Comparison and Filtering

Postgres supports the comparison of dates and times to each other. We have to make sure that we compare apples to apples. In other words, if our left hand side is an interval, then the right hand side needs to be one as well.

```
    User.joins(:projects)
        .where('projects.created_at = (SELECT
MIN(projects.created_at) FROM projects WHERE projects.user_id =
users.id)')
        .where("(projects.created_at - users.created_at) > INTERVAL
'1 month'")
        .select(...)
```

As you can see, we can pass in a human readable string like "1 month" to `INTERVAL`, which is nice.

---

# Aggregations

To calculate the average amount of time it takes to create a project:

```
  User.joins(:projects)
      .where('...')
      .select("AVG(projects.created_at - users.created_at) as
average_time_to_first_project")
```

`ActiveRecord` also has available the `average` function, which we can call like this:

```
  User.joins(...)
      .where(...)
      .average('projects.created_at - users.created_at)
```

This query's return value will be a `BigDecimal`, and because of this we might lose some information. For example, if the true average is `INTERVAL "1 day 23:00:01.2234"`, the `BigDecimal` value will be `1`.

Finally, what if we want to calculate the average time between project creation for each user?

Our query will look like this:

```
  User.joins(:projects)
      .group('users.email')
      .having('COUNT(projects) > 1')
      .select("(CAST(MAX(projects.created_at) as date) -
CAST(MIN(projects.created_at) as date))/(COUNT(projects) - 1) as
avg_time_between_projects, users.email as email")
```