GraphQL in the Real World

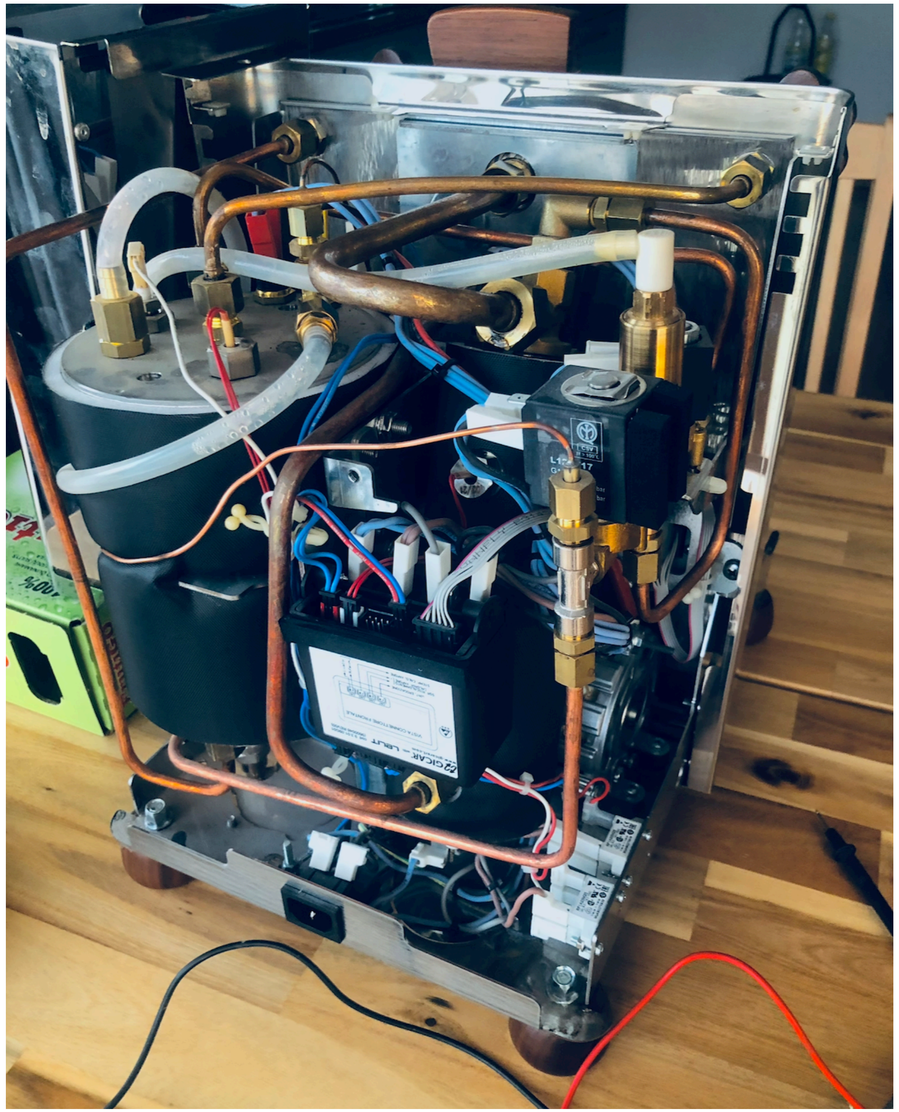
Artūrs Mekšs

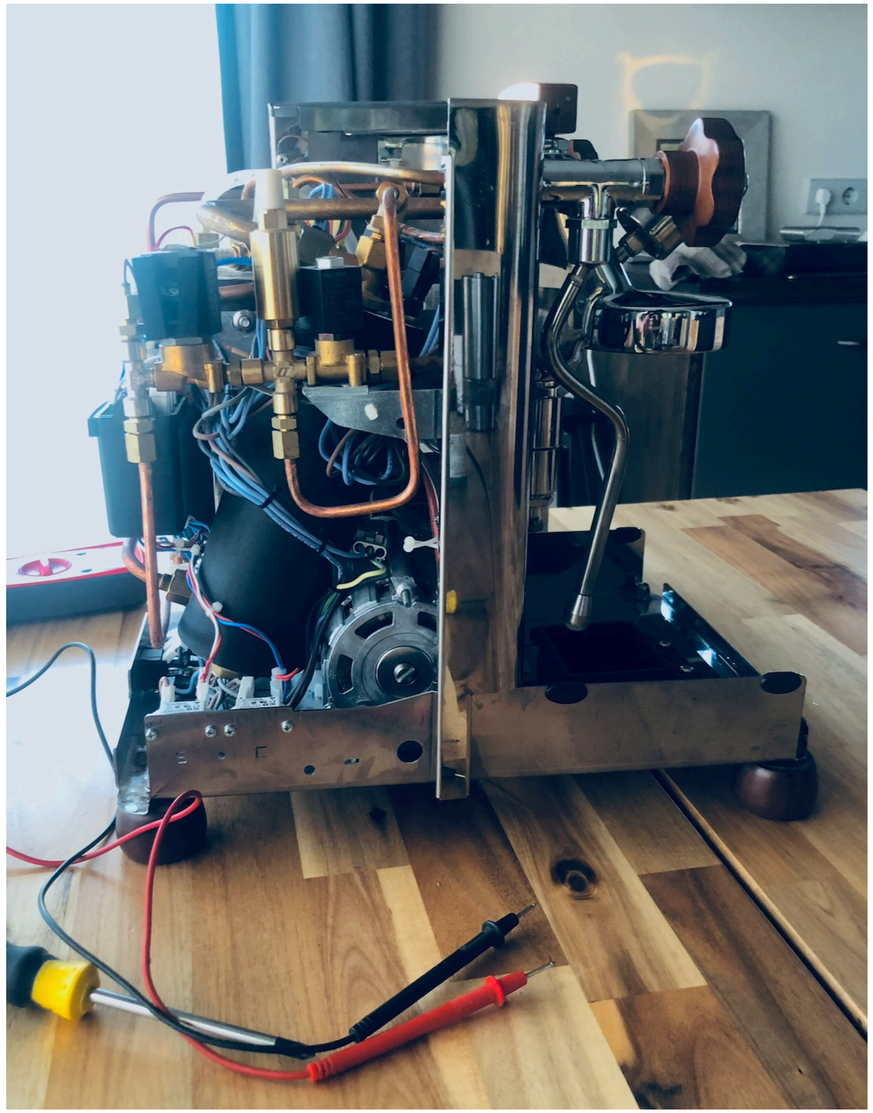
2026

About Me

- Hi, I'm **Artūrs**
- Working **at Roger** (goroger.com) our mission is to eliminate fax machines from German dental practices
- Been running **GraphQL in production** — Rails backend, Flutter frontend — since 2021
- Coffee enthusiast and **home barista**

Me fixing my Coffee





Before we start

All code examples
and this presentation
are available at

github.com/AMekss/buzzer

What is GraphQL?

GraphQL 101

- **Query language for APIs** developed by Facebook in 2012, open-sourced in 2015
- Clients request **exactly the data they need** - no more, no less
- **Strongly typed schema** defines all available data and operations
- Three operation types: **Query** (read), **Mutation** (write), **Subscription** (real-time)
- **Language-agnostic** - implementations exist for Ruby, JavaScript, Python, Java, Go, etc.
- Works over HTTP (typically **single POST /graphql**) instead of multiple REST endpoints
- Is **not tied to HTTP** - can run over WebSockets, gRPC, etc.
- Not a database query language - it’s an API layer that **can connect to any data source**

Queries, Fields and Types

SCHEMA

```
type Query {
  conversation(title: String!): Conversation
}

type Conversation {
  title: String!
  description: String
  participants: [Participant!]!
  messages: [Message]
}

type Participant {
  fullName: String!
  messageCount: Int!
  conversations: [Conversation]
}

type Message {
  from: Participant!
  text: String!
  isUrgent: Boolean!
}
```

QUERY

```
{
  conversation(
    title: "Ruby community chat"
  ) {
    messages {
      from {
        fullName
      }
      text
    }
  }
}
```

RESPONSE

```
{
  "data": {
    "conversation": {
      "messages": [
        {
          "from": {
            "fullName": "Matz"
          },
          "text": "Welcome everyone!"
        },
        {
          "from": {
            "fullName": "DHH"
          },
          "text": "Rails 8 is out!"
        }
      ]
    }
  }
}
```

Meet ConnectionType

SCHEMA

```
type Query {
  conversation(title: String!): Conversation
}

type Conversation {
  title: String!
  description: String
  participants: [Participant!]!
  messagesConnection(
    after: String
    before: String
    first: Int
    last: Int
  ): MessagesConnection!
}

type Participant {
  fullName: String!
  messageCount: Int!
  conversations: [Conversation]
}

type Message {
  from: Participant!
  text: String!
  isUrgent: Boolean!
}
```

CONNECTION TYPES SCHEMA

```
type MessagesConnection {
  edges: [MessageEdge!]!
  nodes: [Message!]!
  pageInfo: PageInfo!

  totalCount: Int!
  hasUrgentMessages: Boolean!
}

type MessageEdge {
  cursor: String!
  node: Message!
}

type PageInfo {
  endCursor: String
  hasNextPage: Boolean!
  hasPreviousPage: Boolean!
  startCursor: String
}
```

Using Nodes with Cursors

QUERY

```
{
  conversation(title: "Ruby community chat") {
    messagesConnection(first: 2) {
      nodes {
        from {
          fullName
        }
        text
      }
      pageInfo {
        endCursor
        hasNextPage
        hasPreviousPage
        startCursor
      }
    }
  }
}
```

RESPONSE

```
{
  "data": {
    "conversation": {
      "messagesConnection": {
        "nodes": [
          {
            "from": { "fullName": "Matz" },
            "text": "Welcome everyone!"
          },
          {
            "from": { "fullName": "DHH" },
            "text": "Rails 8 is out!"
          }
        ],
        "pageInfo": {
          "endCursor": "YXJyYXljb25uZWN0aW9uOjE=",
          "hasNextPage": true,
          "hasPreviousPage": false,
          "startCursor": "YXJyYXljb25uZWN0aW9uOjA="
        }
      }
    }
  }
}
```

Using Edges with Cursors

QUERY

```
{
  conversation(title: "Ruby community chat") {
    messagesConnection(first: 2) {
      edges {
        cursor
        node {
          from {
            fullName
          }
          text
        }
      }
      pageInfo {
        endCursor
        hasNextPage
        hasPreviousPage
        startCursor
      }
    }
  }
}
```

RESPONSE

```
{
  "data": {
    "conversation": {
      "messagesConnection": {
        "edges": [
          {
            "cursor": "YXJyYXljb25uZWN0aW9uOjA=",
            "node": {
              "from": { "fullName": "Matz" },
              "text": "Welcome everyone!"
            }
          },
          {
            "cursor": "YXJyYXljb25uZWN0aW9uOjE=",
            "node": {
              "from": { "fullName": "DHH" },
              "text": "Rails 8 is out!"
            }
          }
        ],
        "pageInfo": {
          "endCursor": "YXJyYXljb25uZWN0aW9uOjE=",
          "hasNextPage": true,
          "hasPreviousPage": false,
          "startCursor": "YXJyYXljb25uZWN0aW9uOjA="
        }
      }
    }
  }
}
```

Key Takeaways

- Don't reinvent the ~~wheel~~ **pagination**
- Use **ConnectionType** for potentially large or growing lists
- Use **ConnectionType** when you need custom attributes on the connection (e.g., *totalCount*, *hasUrgentMessages*)
- Use simple **ListType** when the list is fixed and small (e.g., enum values, user roles)
- Always enforce a server-side limit on **ListType** fields
- Use the **Connection suffix** to distinguish paginated fields (*messages* vs *messagesConnection*)

GraphQL Mutations

SCHEMA

```
type Mutation {
  writeMessage(input: MessageInput!): Message!
}

input MessageInput {
  conversationTitle: String!
  text: String!
  isUrgent: Boolean
}
```

OPERATION

```
mutation WriteMessage($input: MessageInput!) {
  writeMessage(input: $input) {
    from {
      fullName
    }
    text
  }
}
```

VARIABLES

```
{
  "conversationTitle": "Ruby community chat",
  "text": "Hey there!"
}
```

RESPONSE

```
{
  "data": {
    "writeMessagingMessage": {
      "__typename": "MessagingMessage",
      "from": {
        "fullName": "Bob"
      },
      "text": "Hey There!"
    }
  }
}
```

GraphQL Subscriptions

SCHEMA

```
type Subscription {
  messageReceived(
    conversationTitle: String!
  ): Message!
}
```

OPERATION

```
subscription($conversationTitle: String!) {
  messageReceived(
    conversationTitle: $conversationTitle
  ) {
    from {
      fullName
    }
    text
  }
}
```

VARIABLES

```
{
  "conversationTitle": "Ruby community chat"
}
```

RECEIVED UPDATE

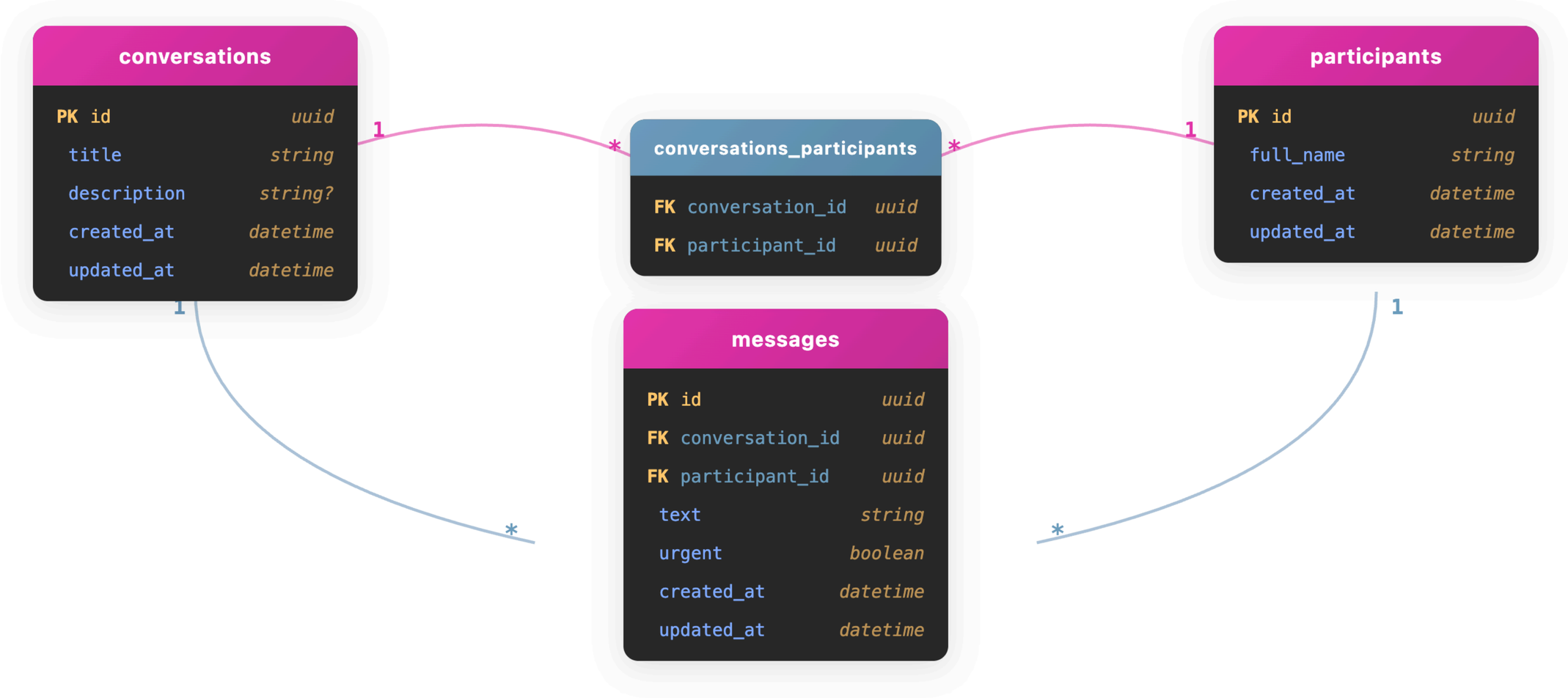
```
{
  "data": {
    "messageReceived": {
      "from": {
        "fullName": "Bob"
      },
      "text": "Hey there!"
    }
  }
}
```

Key Takeaways

- Typically GraphQL servers provide **query** and **mutation** operations
- GraphQL doesn't specify what transport protocol to use for **subscription** operations
- Most often choices are **WebSockets** or **Server-Sent Events**
- Introduction of subscription operations will require **more complicated** architecture and implementation
- Subscription operations are well suited for delivering **incremental** updates as close to **real-time** as possible

Implementation details!

Buzzer's data model



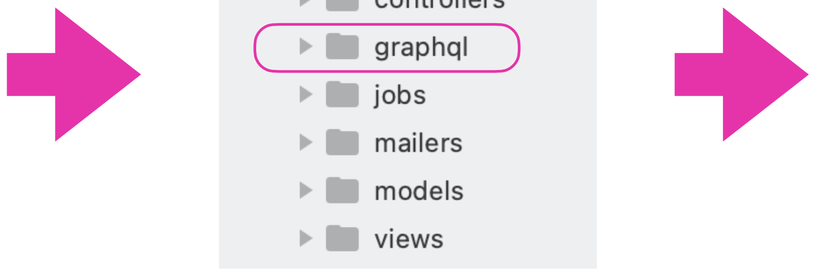
GraphQL on Rails

Add it to the Gemfile of your Rails App

```
# Gemfile
gem "graphql"
```

And then

```
$ bundle install
$ rails g graphql:install
```



GraphQL file Structure

Entry Point

ROUTES `routes.rb`

```ruby
Rails.application.routes.draw do
  post "/graphql", to: "graphql#execute"

  # Define your application routes per the DSL in
  # https://guides.rubyonrails.org/routing.html

  # Reveal health status on /up that returns 200
  # if the app boots with no exceptions, otherwise 500.
  get "up" => "rails/health#show",
      as: :rails_health_check
end
```

CONTROLLER `graphql_controller.rb`

```ruby
class GraphqlController < ApplicationController
  def execute
    variables = prepare_variables(params[:variables])
    query = params[:query]
    operation_name = params[:operationName]
    context = {
      # Query context goes here, for example:
      # current_user: current_user,
    }
    result = BuzzerSchema.execute(
      query,
      variables: variables,
      context: context,
      operation_name: operation_name
    )
    render json: result
  rescue StandardError => e
    raise e unless Rails.env.development?
    handle_error_in_development(e)
  end
end
```

Doors into the Rabbit hole

SCHEMA

`buzzer_schema.rb`

```ruby
class BuzzerSchema < GraphQL::Schema
  mutation(Types::MutationType)
  query(Types::QueryType)
end
```

MUTATIONS

`types/mutation_type.rb`

```ruby
module Types
  class MutationType < Types::Base::Object
    # a place for mutation definitions
    has_no_fields(true)
  end
end
```

QUERIES

`types/query_type.rb`

```ruby
module Types
  class QueryType < Types::Base::Object
    # a place for root query field definitions
    has_no_fields(true)
  end
end
```

Let's implement a Query

query_type.rb

```rb
module Types
  class QueryType < Types::Base::Object
    field :conversation, ConversationType, null: true do
      argument :title, String, required: true
    end

    def conversation(title:)
      Conversation.find_by(title: title)
    end
  end
end
```

conversation_type.rb

```rb
module Types
  class ConversationType < Types::Base::Object
    field :title, String, null: false
    field :description, String, null: true
    field :participants, [ParticipantType], null: false
    field :messages_connection,
          resolver: Resolvers::MessagesConnection
  end
end
```

messages_connection.rb

```rb
module Resolvers
  class MessagesConnection < BaseResolver
    type Types::MessageType.connection_type, null: false

    def resolve
      object.messages
    end
  end
end
```

Rest of the Types

message_type.rb

```ruby
module Types
  class MessageType < Types::Base::Object
    field :text, String, null: false
    field :is_urgent, Boolean, null: false, method: :urgent?
    field :from, ParticipantType, null: false, method: :participant
  end
end
```

participant_type.rb

```ruby
module Types
  class ParticipantType < Types::Base::Object
    field :full_name, String, null: false
    field :message_count, Integer, null: false
    field :conversations, [ConversationType], null: false

    def message_count
      object.messages.count
    end
  end
end
```

Schema generation

Gemfile

```
gem "graphql-schema_comparator"
```

lib/tasks/graphql.rake

```
GRAPHQL_SCHEMA = "BuzzerSchema"
GRAPHQL_EXPORT_PATH = "./app/graphql/schema.graphql"

namespace :graphql do
  desc "Export #{GRAPHQL_SCHEMA} to app/graphql/schema.graphql"
  task export: :environment do
    exported = File.read(GRAPHQL_EXPORT_PATH)
    in_code = GraphQL::Schema::Printer.print_schema(GRAPHQL_SCHEMA.constantize)

    res = GraphQL::SchemaComparator.compare(exported, in_code)
    if res.breaking?
      puts "\n🔥  Breaking changes detected:\n\n"
      res.breaking_changes.each do |change|
        puts "    -> #{change.path} - #{change.message}"
      end
      puts "\n"
    end

    File.write(GRAPHQL_EXPORT_PATH, in_code)
  end

  task verify: :environment do
    exported = File.read(GRAPHQL_EXPORT_PATH)
    in_code = GraphQL::Schema::Printer.print_schema(GRAPHQL_SCHEMA.constantize)

    res = GraphQL::SchemaComparator.compare(exported, in_code)
    unless res.identical?
      puts "\n🙀  Exported schema is not identical to in-code schema"
      puts "➡️   Run rake graphql:export to update the exported schema\n\n"
      exit 1
    end
  end
end
```

Code-driven schema

app/graphql/types/message_type.rb

```
@@ -1,7 +1,6 @@
module Types
  class MessageType < Types::Base::Object
-   field :text, String, null: false
-   field :is_urgent, Boolean, null: false, method: :mandatory?
+   field :text, String, null: true
    field :from, ParticipantType, null: false, method: :participant
  end
end
```

```
root@d5b6455bf5e1:/rails# rake graphql:verify

🙀  Exported schema is not identical to in-code schema
➡️   Run rake graphql:export to update the exported schema

root@d5b6455bf5e1:/rails# rake graphql:export

🔥  Breaking changes detected:

    -> Message.text - Field `Message.text` changed type from `String!` to `String`
    -> Message.isUrgent - Field `isUrgent` was removed from object type `Message`
```

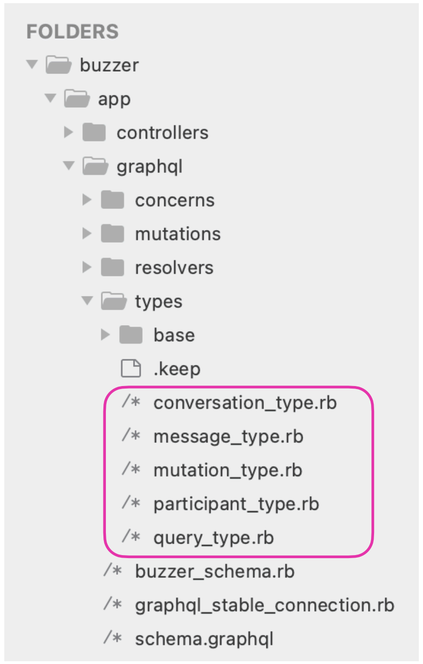
app/graphql/schema.graphql

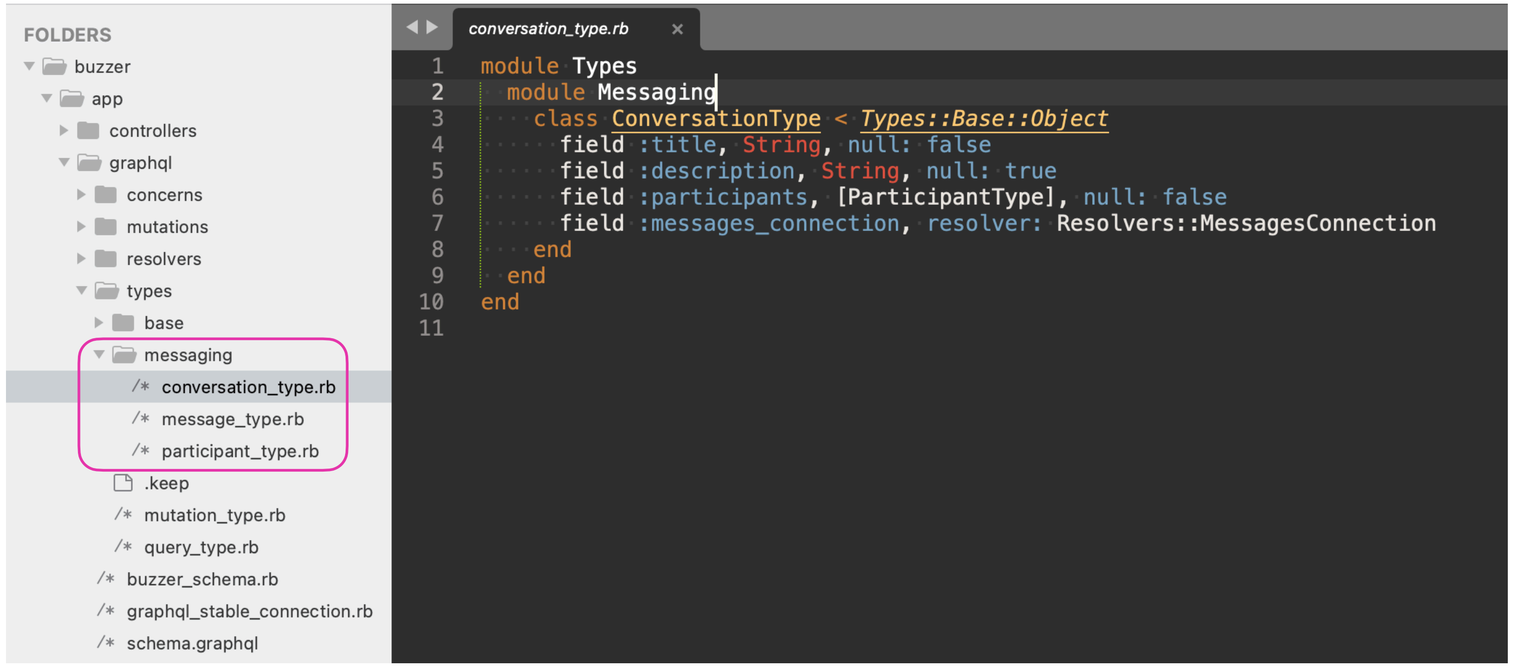
```
@@ -27,8 +27,7 @@ type Conversation {

type Message {
  from: Participant!
-  isUrgent: Boolean!
-  text: String!
+  text: String
}
```

Pretty cool! BUT

Default GraphQL Type
name resolution

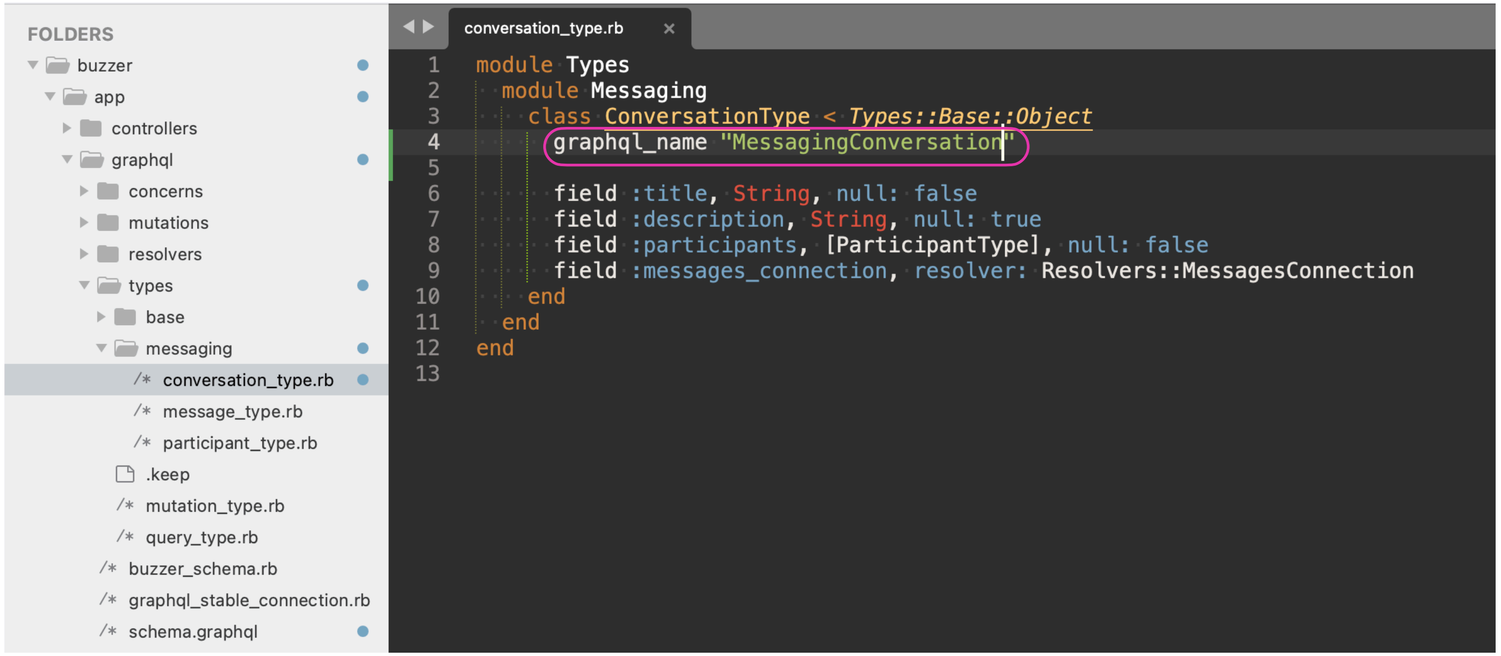
Namespaces





That doesn’t impact schema!

Manual name override



```
root@d5b6455bf5e1:/rails# rake graphql:export

🔥  Breaking changes detected:

    -> Query.conversation - Field `Query.conversation` changed type from `Conversation` to `MessagingConversation`
    -> Participant.connections - Field `Participant.connections` changed type from `[Conversation!]!` to `[MessagingConversation!]!`
    -> Conversation - Type `Conversation` was removed
```

We can do better!

FOLDERS

- buzzer
  - app
    - controllers
    - graphql
      - concerns
        - graphql_namespaced_na
      - mutations
        - .keep
        - base_mutation.rb
      - resolvers
        - messaging
          - messages_connection.rb
        - base_resolver.rb
      - types
        - base
          - argument.rb
          - connection.rb
          - edge.rb
          - enum.rb
          - field.rb
          - input_object.rb
          - interface.rb
          - object.rb
          - scalar.rb
          - union.rb
        - messaging
          - conversation_type.rb
          - message_type.rb
          - participant_type.rb
        - .keep
        - mutation_type.rb
        - query_type.rb
      - buzzer_schema.rb
      - graphql_stable_connection.rb
      - schema.graphql

graphql_namespaced_name_resolver.rb

```ruby
module GraphqlNamespacedNameResolver
  extend ActiveSupport::Concern

  included do
    # in case we need the old behaviour let's alias it to `unspaced_graphql_name`
    singleton_class.send(:alias_method, :unspaced_graphql_name, :default_graphql_name)

    # override `default_graphql_name` with new namespaced behaviour, which makes it default,
    # unless subclass adds `graphql_name unspaced_graphql_name`
    singleton_class.send(:alias_method, :default_graphql_name, :namespaced_graphql_name)
  end

  class_methods do
    # Similar to what original #default_graphql_name does
    # but this one maintains namespace hierarchy, so we can organize types in modules
    def namespaced_graphql_name
      @structural_graphql_name ||= begin
        raise GraphQL::RequiredImplementationMissingError, "Anonymous class should declare a `graphql_name`" if name.nil?

        name.gsub(/(\ATypes|::|Type\z)/, "")
      end
    end
  end
end

```

It’s all automated now!

app/graphql/schema.graphql

```
@@ -1,4 +1,4 @@
-type Conversation {
+type MessagingConversation {
   description: String
   messagesConnection(
     """
@@ -20,30 +20,30 @@ type Conversation {
     Returns the last _n_ elements from the list.
     """
     last: Int
-  ): MessageConnection!
-  participants: [Participant!]!
+  ): MessagingMessageConnection!
+  participants: [MessagingParticipant!]!
   title: String!
 }

-type Message {
-  from: Participant!
+type MessagingMessage {
+  from: MessagingParticipant!
   isUrgent: Boolean!
   text: String!
 }

 """
-The connection type for Message.
+The connection type for MessagingMessage.
 """
-type MessageConnection {
+type MessagingMessageConnection {
   """
   A list of edges.
   """
-  edges: [MessageEdge]
+  edges: [MessagingMessageEdge]
```

Key Takeaways

- For **simple data** resolutions use methods or delegations
- For **more complex** resolutions with many arguments use resolvers
- Keep your resolvers lean, they are **“controllers” of the GraphQL**
- **Organize** GraphQL types **into modules**
- Use Code-driven schema as your **API contract**

Let's give it a go!

Please Mister Postman

Buzzer API / **conversation**

Save | Share

POST http://127.0.0.1:3000/graphql **Send**

Docs | Params | Authorization | Headers (8) | Body | Scripts | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL **Auto Fetch** | Schema Fetched

QUERY

```
query conversation($title: String!) {
    conversation(title: $title) {
        title
        messagesConnection(first: 10) {
            nodes {
                from {
                    fullName
                }
                text
            }
        }
    }
}
```

GRAPHQL VARIABLES

```
{
    "title": "Ruby community chat"
}
```

Body | Cookies | Headers (13) | Test Results — 200 OK • 134 ms • 1.38 KB | Save Response

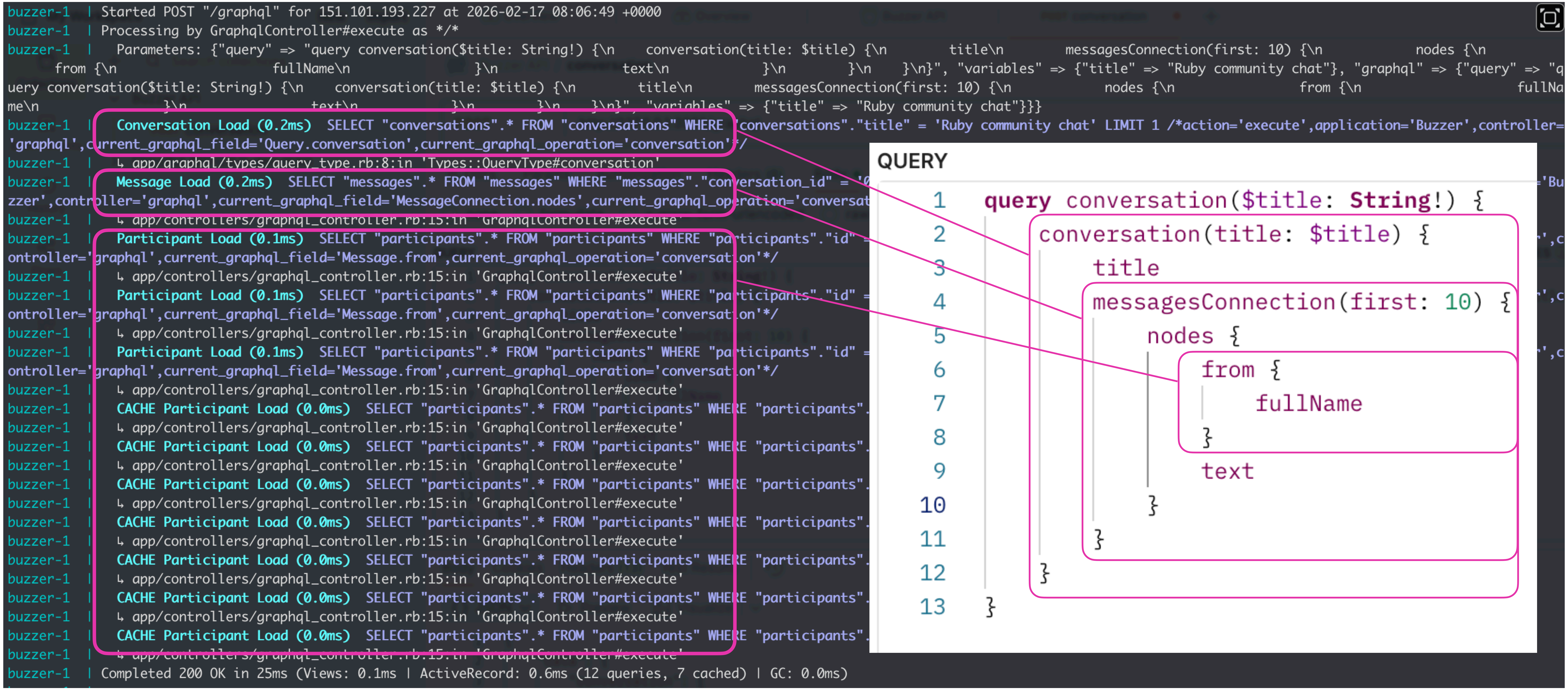
JSON | Preview | Visualize

```
{
    "data": {
        "conversation": {
            "title": "Ruby community chat",
            "messagesConnection": {
                "nodes": [
                    {
                        "from": {
                            "fullName": "Matz"
                        },
                        "text": "Welcome everyone!"
                    },
                    {
                        "from": {
                            "fullName": "DHH"
```

N+1 monster

```
buzzer-1  | Started POST "/graphql" for 151.101.193.227 at 2026-02-17 08:06:49 +0000
buzzer-1  | Processing by GraphqlController#execute as */*
buzzer-1  |   Parameters: {"query" => "query conversation($title: String!) {\n    conversation(title: $title) {\n        title\n        messagesConnection(first: 10) {\n            nodes {\n
      from {\n                fullName\n            }\n            text\n            }\n        }\n    }\n}", "variables" => {"title" => "Ruby community chat"}, "graphql" => {"query" => "q
uery conversation($title: String!) {\n    conversation(title: $title) {\n        title\n        messagesConnection(first: 10) {\n            nodes {\n                from {\n                fullNa
me\n            }\n            text\n            }\n        }\n    }\n}", "variables" => {"title" => "Ruby community chat"}}}
buzzer-1  |   Conversation Load (0.2ms)  SELECT "conversations".* FROM "conversations" WHERE "conversations"."title" = 'Ruby community chat' LIMIT 1 /*action='execute',application='Buzzer',controller=
'graphql',current_graphql_field='Query.conversation',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/graphql/types/query_type.rb:8:in 'Types::QueryType#conversation'
buzzer-1  |   Message Load (0.2ms)  SELECT "messages".* FROM "messages" WHERE "messages"."conversation_id" = '019c6a6b-a09c-7a7b-94ef-978babb8fd27' LIMIT 10 OFFSET 0 /*action='execute',application='Bu
zzer',controller='graphql',current_graphql_field='MessageConnection.nodes',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   Participant Load (0.1ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b3-77be-bb90-902021e34d29' LIMIT 1 /*action='execute',application='Buzzer',c
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   Participant Load (0.1ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b0-70f7-a991-b304da762228' LIMIT 1 /*action='execute',application='Buzzer',c
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   Participant Load (0.1ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b6-7836-a3bd-735a4ccdb6fc' LIMIT 1 /*action='execute',application='Buzzer',c
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b3-77be-bb90-902021e34d29' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b0-70f7-a991-b304da762228' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b6-7836-a3bd-735a4ccdb6fc' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b3-77be-bb90-902021e34d29' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b0-70f7-a991-b304da762228' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b6-7836-a3bd-735a4ccdb6fc' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id" = '019c6a6b-a0b3-77be-bb90-902021e34d29' LIMIT 1
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  | Completed 200 OK in 25ms (Views: 0.1ms | ActiveRecord: 0.6ms (12 queries, 7 cached) | GC: 0.0ms)
```

N+1 monster


buzzer-1 | Started POST "/graphql" for 151.101.193.227 at 2026-02-17 08:06:49 +0000
buzzer-1 | Processing by GraphqlController#execute as */*
buzzer-1 | Parameters: {"query" => "query conversation($title: String!) {\n conversation(title: $title) {\n title\n messagesConnection(first: 10) {\n nodes {\n from {\n fullName\n }\n text\n }\n }\n }\n}", "variables" => {"title" => "Ruby community chat"}, "graphql" => {"query" => "query conversation($title: String!) {\n conversation(title: $title) {\n title\n messagesConnection(first: 10) {\n nodes {\n from {\n fullName\n }\n text\n }\n }\n }\n}", "variables" => {"title" => "Ruby community chat"}}}
buzzer-1 | Conversation Load (0.2ms) SELECT "conversations".* FROM "conversations" WHERE "conversations"."title" = 'Ruby community chat' LIMIT 1 /*action='execute',application='Buzzer',controller='graphql',current_graphql_field='Query.conversation',current_graphql_operation='conversation'*/
buzzer-1 | ↳ app/graphql/types/query_type.rb:8:in 'Types::QueryType#conversation'
buzzer-1 | Message Load (0.2ms) SELECT "messages".* FROM "messages" WHERE "messages"."conversation_id" =
zzer',controller='graphql',current_graphql_field='MessageConnection.nodes',current_graphql_operation='conversat
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | Participant Load (0.1ms) SELECT "participants".* FROM "participants" WHERE "participants"."id" =
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | Participant Load (0.1ms) SELECT "participants".* FROM "participants" WHERE "participants"."id" =
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | Participant Load (0.1ms) SELECT "participants".* FROM "participants" WHERE "participants"."id" =
ontroller='graphql',current_graphql_field='Message.from',current_graphql_operation='conversation'*/
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | CACHE Participant Load (0.0ms) SELECT "participants".* FROM "participants" WHERE "participants".
buzzer-1 | ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1 | Completed 200 OK in 25ms (Views: 0.1ms | ActiveRecord: 0.6ms (12 queries, 7 cached) | GC: 0.0ms)
QUERY
query conversation($title: String!) {
conversation(title: $title) {
title
messagesConnection(first: 10) {
nodes {
from {
fullName
}
text
}
}
}
}

Queries are dynamic

```
query conversation($title: String!) {
  conversation(title: $title) {
    title
    messagesConnection(first: 10) {
      nodes { text }
    }
  }
}
```

Is actually alright

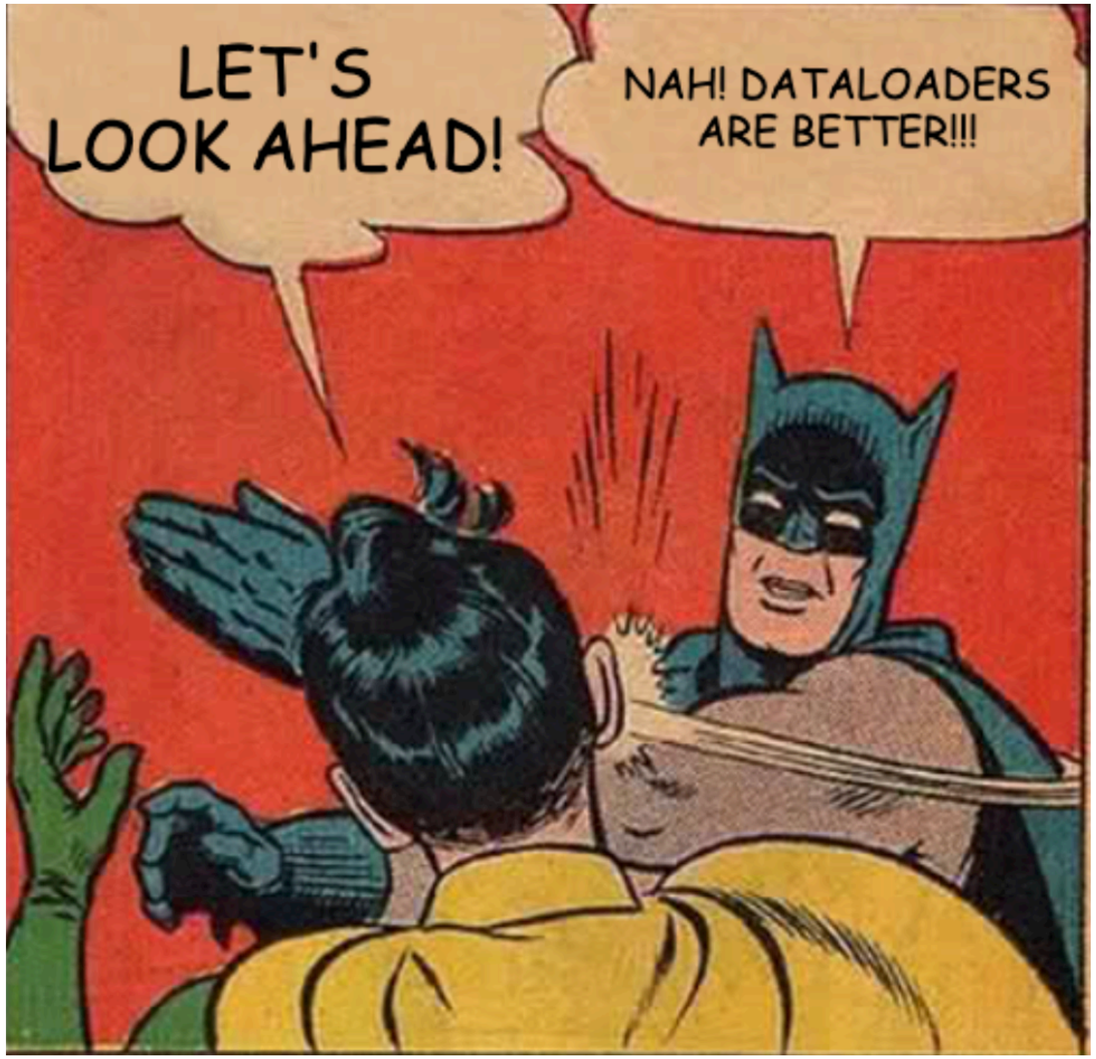
```
query conversation($title: String!) {
  conversation(title: $title) {
    title
    messagesConnection(first: 10) {
      nodes {
        from { fullName }
        text
      }
    }
  }
}
```

Causes N+1

```
query conversation($title: String!) {
  conversation(title: $title) {
    title
    participants {
      fullName
      messageCount
    }
    messagesConnection(first: 10) {
      nodes {
        from { fullName, messageCount }
        text
      }
    }
  }
}
```

Makes it even worse

The Heroes of the Day



Lookaheads and Dataloaders

Lookaheads

query_type.rb

```rb
module Types
  class QueryType < Types::Base::Object
    field :conversation, Types::Messaging::ConversationType, null: true, extras: [:lookahead] do
      argument :title, String, required: true
    end

    def conversation(title:, lookahead:)
      rel = Conversation.all
      if lookahead.selection(:messages_connection).selection(:nodes).selects?(:from)
        rel = rel.includes(messages: :participant)
      end

      rel.find_by(title: title)
    end
  end
end
```

Lookaheads refactored

`query_type.rb`

```ruby
module Types
  class QueryType < Types::Base::Object
    field :conversation, resolver: Resolvers::Messaging::Conversation
  end
end
```

`graphql/resolvers/messaging/conversation.rb`

```ruby
module Resolvers
  module Messaging
    class Conversation < BaseResolver
      type Types::Messaging::ConversationType, null: true

      argument :title, String, required: true

      extras [:lookahead]

      def resolve(title:, lookahead:)
        rel = ::Conversation.all
        if lookahead.selection(:messages_connection).selection(:nodes).selects?(:from)
          rel = rel.includes(messages: :participant)
        end

        rel.find_by(title: title)
      end
    end
  end
end
```

Lookaheads saves the Day

QUERY

```
query conversation($title: String!) {
    conversation(title: $title) {
        title
        messagesConnection(first: 10) {
            nodes {
                from {
                    fullName
                }
                text
            }
        }
    }
}
```

```
buzzer-1  | Started POST "/graphql" for 151.101.193.227 at 2026-02-20 17:57:12 +0000
buzzer-1  | Processing by GraphqlController#execute as */*
buzzer-1  |   Parameters: {"query" => "query conversation($title: String!) {\n    conversation(title: $title
      text\n            }\n        }\n    }\n}", "variables" => {"title" => "Ruby community chat"}, "graphq
es {\n                from {\n                    fullName\n                }\n                text\n
buzzer-1  |   Conversation Load (0.1ms)  SELECT "conversations".* FROM "conversations" WHERE "conversations"
ration='conversation'*/
buzzer-1  |   ↳ app/graphql/types/query_type.rb:13:in 'Types::QueryType#conversation'
buzzer-1  |   Message Load (0.5ms)  SELECT "messages".* FROM "messages" WHERE "messages"."conversation_id" =
ation='conversation'*/
buzzer-1  |   ↳ app/graphql/types/query_type.rb:13:in 'Types::QueryType#conversation'
buzzer-1  |   Participant Load (0.1ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id
',controller='graphql',current_graphql_field='Query.conversation',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/graphql/types/query_type.rb:13:in 'Types::QueryType#conversation'
buzzer-1  | Completed 200 OK in 9ms (Views: 0.1ms | ActiveRecord: 0.7ms (3 queries, 0 cached) | GC: 0.2ms)
buzzer-1  |
```

QUERY

```
query conversation($title: String!) {
    conversation(title: $title) {
        title
        messagesConnection(first: 10) {
            nodes {
                text
            }
        }
    }
}
```

```
buzzer-1  | Started POST "/graphql" for 151.101.193.227 at 2026-02-20 17:51:06 +0000
buzzer-1  | Processing by GraphqlController#execute as */*
buzzer-1  |   Parameters: {"query" => "query conversation($title: String!) {\n    conversation(title: $title
 => "Ruby community chat"}, "graphql" => {"query" => "query conversation($title: String!) {\n    conversatio
es" => {"title" => "Ruby community chat"}}}
buzzer-1  |   Conversation Load (0.2ms)  SELECT "conversations".* FROM "conversations" WHERE "conversations"
ration='conversation'*/
buzzer-1  |   ↳ app/graphql/types/query_type.rb:13:in 'Types::QueryType#conversation'
buzzer-1  |   Message Load (0.1ms)  SELECT "messages".* FROM "messages" WHERE "messages"."conversation_id" =
tion.nodes',current_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  | Completed 200 OK in 6ms (Views: 0.1ms | ActiveRecord: 0.2ms (2 queries, 0 cached) | GC: 0.6ms)
buzzer-1  |
buzzer-1  |
```

Lookaheads **are fragile**

QUERY

```graphql
query conversation($title: String!) {
  conversation(title: $title) {
    title
    messagesConnection(first: 10) {
      edges {
        node {
          from {
            fullName
          }
          text
        }
      }
    }
  }
}
```

```
buzzer-1  |   Participant Load (0.1ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id"
t_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id"
t_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants"."id"
t_graphql_operation='conversation'*/
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  |   CACHE Participant Load (0.0ms)  SELECT "participants".* FROM "participants" WHERE "participants
buzzer-1  |   ↳ app/controllers/graphql_controller.rb:15:in 'GraphqlController#execute'
buzzer-1  | Completed 200 OK in 24ms (Views: 0.1ms | ActiveRecord: 0.5ms (12 queries, 7 cached) | GC: 1.7ms)
```

Lookahead Helper

lib/lookahead.rb

```ruby
class Lookahead
  def initialize(root_lookahead)
    @root_lookahead = root_lookahead
  end

  def selects?(*field_names)
    lookaheads.any? { |lookahead| deep_match?(lookahead, field_names) }
  end

  private
  attr_reader :root_lookahead

  def deep_match?(lookahead, field_name)
    case field_name
    when Symbol then lookahead.selects?(field_name)
    when Array  then field_name.any? { |val| deep_match?(lookahead, val) }
    when Hash   then field_name.any? { |key, val| deep_match?(lookahead.selection(key), val) }
    else false
    end
  end

  def lookaheads
    [
      root_lookahead,                                  # Base lookahead
      root_lookahead.selection(:nodes),                # nodes { ... }
      root_lookahead.selection(:edges).selection(:node)  # edges { node { ... } }
    ]
  end
end
```

Bullet proof Lookahead

graphql/resolvers/messaging/conversation.rb

```ruby
module Resolvers
  module Messaging
    class Conversation < BaseResolver
      type Types::Messaging::ConversationType, null: true

      argument :title, String, required: true

      extras [:lookahead]

      def resolve(title:, lookahead:)
        rel = ::Conversation.all
        if Lookahead.new(lookahead.selection(:messages_connection)).selects?(:from)
          rel = rel.includes(messages: :participant)
        end

        rel.find_by(title: title)
      end
    end
  end
end
```

Potential failure points

SCHEMA

```
type Query {
  conversation(title: String!): Conversation
}

type Conversation {
  title: String!
  description: String
  participants: [Participant!]!
  messages: [Message]
}

type Participant {
  fullName: String!
  messageCount: Int!
  conversations: [Conversation]
}

type Message {
  from: Participant!
  text: String!
  isUrgent: Boolean!
}
```

Dataloaders

message_type.rb

```ruby
module Types
  module Messaging
    class MessageType < Types::Base::Object
      field :text, String, null: false
      field :is_urgent, Boolean, null: false, method: :mandatory?
      field :from, ParticipantType, null: false

      def from
        dataloader.with(::Sources::ActiveRecord::BelongsTo, ::Participant).load(object.participant_id)
      end
    end
  end
end
```

graphql/sources/active_record/belongs_to.rb

```ruby
class Sources::ActiveRecord::BelongsTo < GraphQL::Dataloader::Source
  attr_reader :model

  def initialize(model)
    @model = model
  end

  def fetch(ids)
    records = model.where(id: ids).index_by(&:id)
    ids.map { |id| records.fetch(id, nil) }
  end
end
```

Dataloaders caveat

message_type.rb

```ruby
module Types
  module Messaging
    class MessageType < Types::Base::Object
      field :text, String, null: false
      field :is_urgent, Boolean, null: false, method: :mandatory?
      field :from, ParticipantType, null: false

      def from
        dataloader
          .with(::Sources::ActiveRecord::BelongsTo, ::Participant.active)
          .load(object.participant_id)
      end
    end
  end
end
```

Dataloaders root cause

graphql-ruby/lib/graphql/dataloader/source.rb

```ruby
# These arguments are given to `dataloader.with(source_class, ...)`.
# The object returned from this method is used to de-duplicate batch
# loads under the hood by using it as a Hash key.
#
# By default, the arguments are all put in an Array. To customize how
# this source's batches are merged, override this method to return
# something else.
#
# For example, if you pass `ActiveRecord::Relation`s to `.with(...)`
# you could override this method to call `.to_sql` on them, thus
# merging `.load(...)` calls when they apply to equivalent relations.
#
# @param batch_args [Array<Object>]
# @param batch_kwargs [Hash]
# @return [Object]
def self.batch_key_for(*batch_args, **batch_kwargs)
  [*batch_args, **batch_kwargs]
end
```

Dataloaders fixed

graphql/sources/base.rb

```ruby
# It is important to have it here, solves issues when we passing in
# not "scalar" objects into data loader `.with(...)` method
def batch_key_for(*batch_args, **batch_kwargs)
  normalize([*batch_args, **batch_kwargs]).compact
end

private

def normalize(value)
  if Array === value
    value.flat_map { |v| normalize(v) }
  elsif Hash === value
    Array(value).flat_map { |v| normalize(v) }
  elsif ::GraphQL::Execution::Lookahead === value
    # Lookahead should not change the result set
    nil
  elsif ::ActiveRecord::Relation === value
    value.to_sql
  elsif ::Class === value
    value.name
  elsif value.respond_to?(:cache_key_with_version)
    value.cache_key_with_version
  elsif SCALARS.include?(value.class)
    value
  else
    raise ArgumentError, "don't know how to normalize #{value.inspect}"
  end
end
```

Flexibility comes with a cost

app/controllers/api/v1/graphql_controller.rb +1 -1 Viewed

```
@@ -27,7 +27,7 @@ def execute
27      end                                                    27      end
28                                                             28
29      ActiveSupport::Notifications.instrument("render.json") do  29      ActiveSupport::Notifications.instrument("render.json") do
30  -     render json: result                                  30  +     fast_json_render result
31      end                                                    31      end
32    end                                                      32    end
33                                                             33
```

```ruby
def fast_json_render(response_body)
  _set_vary_header
  self.content_type = Mime[:json]
  self.response_body = response_body.is_a?(String) ?
    response_body : Oj.dump(response_body, mode: :rails)
end
```

Fragment Caching

- We’re using **gem “graphql-fragment_cache”**
- It’s **simple to install**, **reliable and flexible**
- It works **very similar** to Rails fragment caching
- **Recommend!**

Stable Cursor Pagination

CONNECTION `graphql_stable_connection.rb`

```ruby
class GraphqlStableConnection < GraphQL::Pagination::Connection
  # Available instance variables
  # @items - ActiveRecord Relation
  # @first - Int value of forward pagination limit
  # @after - String value of forward pagination cursor
  # @last - Int value of backward pagination limit
  # @before - String value of backward cursor

  # returns paginated slice of @items
  def nodes
  end

  # true if items exist after #nodes
  def has_next_page
  end

  # true if items exist before #nodes
  def has_previous_page
  end

  # returns cursor String for item
  def cursor_for(node)
  end
end
```

SCHEMA `buzzer_schema.rb`

```ruby
class BuzzerSchema < GraphQL::Schema
  mutation(Types::MutationType)
  query(Types::QueryType)

  connections.add(
    ActiveRecord::Relation,
    GraphqlStableConnection
  )
  # or
  # GraphQL::Pro::PostgresStableRelationConnection
  # GraphQL::Pro::MySQLStableRelationConnection
  # GraphQL::Pro::SqliteStableRelationConnection
end
```

Key Takeaways

- Dataloaders **are more robust** approach to fix N+1 issues
- Sometimes Lookaheads are **the only option**
- Good thing they **don’t exclude** each other
- Use fragment caching for **expensive computations** or static parts
- Measure and optimize **JSON rendering**

Where do I send my message?

Hi Bob!

graphql_controller.rb

```ruby
class GraphqlController < ApplicationController
  def execute
    variables = prepare_variables(params[:variables])
    query = params[:query]
    operation_name = params[:operationName]
    context = {
      current_user: current_user
    }
    result = BuzzerSchema.execute(
      query, variables: variables, context: context, operation_name: operation_name
    )
    render json: result
  rescue StandardError => e
    raise e unless Rails.env.development?
    handle_error_in_development(e)
  end

  private

  # IRL would be set from the auth token
  def current_user
    @current_user ||= Participant.find_by!(full_name: "Bob")
  end
end
```

Let’s implement a Mutation

mutation_type.rb

```ruby
module Types
  class MutationType < Types::Base::Object
    field :write_messaging_message, mutation: ::Mutations::Messaging::WriteMessage
  end
end
```

graphql/mutations/messaging/write_message.rb

```ruby
module Mutations
  module Messaging
    class WriteMessage < Mutations::BaseMutation
      type ::Types::Messaging::MessageType

      argument :conversation_title, String, required: false
      argument :text, String, required: true

      def resolve(conversation_title:, text:)
        conversation = ::Conversation.find_by!(title: conversation_title)
        conversation.messages.create!(text: text, participant: context[:current_user])
      end
    end
  end
end
```

And it Works!

QUERY

```graphql
mutation WriteMessage($input: MutationsMessagingWriteMessageInput!){
    writeMessagingMessage(input: $input) {
        __typename
        from {
            fullName
        }
        text
    }
}
```

GRAPHQL VARIABLES

```json
{
    "input": {
        "conversationTitle": "Ruby community chat",
        "text": "Hey, whazzzzup!!!1!"
    }
}
```

Body Cookies Headers (13) Test Results

200 OK • 40 ms • 815 B | Save Response

{ } JSON | Preview | Visualize

```json
{
    "data": {
        "writeMessagingMessage": {
            "__typename": "MessagingMessage",
            "from": {
                "fullName": "Bob"
            },
            "text": "Hey, whazzzzup!!!1!"
        }
    }
}
```

What when it does not?

QUERY

```
mutation WriteMessage($input: MutationsMessagingWriteMessageInput!){
    writeMessagingMessage(input: $input) {
        __typename
        from {
            fullName
        }
        text
    }
}
```

GRAPHQL VARIABLES

```
{
    "input": {
        "conversationTitle": "nonexistent one",
        "text": "Hey, whazzzzup!!!1!"
    }
}
```

Body Cookies Headers (4) Test Results 404 Not Found 52 ms 158 B Save Response

HTML Preview Debug with AI

```
1
```

Error handling

- The GraphQL specification allows three top-level keys in a response: **data**, **errors**, and **extensions**
- At **least one** of **data or errors** will be present on the response (when it contains both it is a “partial response”)
- The data key contains the **result of** the executed **operation**
- Information about **raised errors** is included in the errors key of the response
- GraphQL implementations may include additional **arbitrary information** about the response in the **extensions** key

Step 1 - Hide everything

buzzer_schema.rb

```ruby
class BuzzerSchema < GraphQL::Schema
  mutation(Types::MutationType)
  query(Types::QueryType)
  use GraphQL::Dataloader
  connections.add(ActiveRecord::Relation, GraphqlStableConnection)

  rescue_from(StandardError) do |err, obj, args, ctx, field|
    raise GraphqlError.build(Rails.env.development?, err, obj, args, ctx, field)
  end
end
```

graphql_error.rb

```ruby
class GraphqlError
  class DevelopmentMode < StandardError; end

  def self.build(*) = new(*).error

  def initialize(development_mode, err, obj, args, ctx, field)
    @ref = SecureRandom.uuid  # Connect FE errors with BE errors
    @development_mode = development_mode
    @err, @obj, @args, @ctx, @field = err, obj, args, ctx, field
  end

  def error
    ::ErrorReporter.report(err, **context)  # Report backend error
    return DevelopmentMode if development_mode

    # Return graphql-friendly error with reference to backend error
    ::GraphQL::ExecutionError.new(
      "Something went wrong",
      extensions: {error_code: "UNEXPECTED_ERROR", error_ref: ref}
    )
  end
end
```

Step 1 - In Action

QUERY

```
mutation WriteMessage($input: MutationsMessagingWriteMessageInput!){
    writeMessagingMessage(input: $input) {
        __typename
        from {
            fullName
        }
        text
    }
}
```

GRAPHQL VARIABLES

```
{
    "input": {
        "conversationTitle": "nonexistent one",
        "text": "Hey, whazzzzup!!!1!"
    }
}
```

Body Cookies Headers (12) Test Results 200 OK • 29 ms • 696 B Save Response

JSON Preview Visualize

```
{
    "errors": [
        {
            "message": "Something went wrong",
            "locations": [
                {
                    "line": 2,
                    "column": 5
                }
            ],
            "path": [
                "writeMessagingMessage"
            ],
            "extensions": {
                "error_code": "UNEXPECTED_ERROR",
                "error_ref": "a2f4a17a-2dca-4421-a973-90d146824ef9"
            }
        }
    ],
    "data": {
        "writeMessagingMessage": null
    }
}
```

```
buzzer-1  | [39b28071-97c9-48fc-9280-7350cdfa4017] {error: "ActiveRecord::RecordNotFound: Co
E \"conversations\".\"title\" = ?]", context: {ref: "a2f4a17a-2dca-4421-a973-90d146824ef9",
2637b55c294", field: "writeMessagingMessage", query: "{\"input\":{\"conversation_title\":\"n
azzzzup!!!1!\"}}"}, backtrace: ["/usr/local/bundle/gems/activerecord-8.1.2/lib/active_record
'ActiveRecord::FinderMethods#raise_record_not_found_exception!'", "/usr/local/bundle/gems/ac
core.rb:330:in 'ActiveRecord::Core::ClassMethods#find_by!'", "/rails/app/graphql/mutations/m
tations::Messaging::WriteMessage#resolve'", "/usr/local/bundle/gems/graphql-2.5.19/lib/graph
l#public_send'", "/usr/local/bundle/gems/graphql-2.5.19/lib/graphql/schema/resolver.rb:118:i
esolve'", "/usr/local/bundle/gems/graphql-2.5.19/lib/graphql/schema/mutation.rb:69:in 'Graph
, "/usr/local/bundle/gems/graphql-2.5.19/lib/graphql/schema/resolver.rb:105:in 'block (3 lev
esolve_with_support'", "/usr/local/bundle/gems/graphql-2.5.19/lib/graphql/execution/interpre
cution::Interpreter::Runtime#minimal_after_lazy'", "/usr/local/bundle/gems/graphql-2.5.19/li
Query::Runnable#after_lazy'", "/usr/local/bundle/gems/graphql-2.5.19/lib/graphql/schema/reso
GraphQL::Schema::Resolver#resolve_with_support'"]}
```

Step 2 - Show something

graphql_user_error.rb

```ruby
class GraphqlUserError
  ErrorDetail = Struct.new(:code, :message)

  GENERIC_ERROR_DETAIL = ErrorDetail.new("GENERIC_ERROR", "Generic error")
  HANDLED_ERROR_DETAILS = {
    not_authorized:   ErrorDetail.new("NOT_AUTHORIZED", "Not authorized"),
    record_not_found: ErrorDetail.new("RECORD_NOT_FOUND", "Record not found"),
    record_invalid:   ErrorDetail.new("RECORD_INVALID", "Record can not be saved")
  }

  def self.build(code, context = {})
    error_detail = HANDLED_ERROR_DETAILS.fetch(code, GENERIC_ERROR_DETAIL)
    ::GraphQL::ExecutionError.new(
      error_detail.message,
      extensions: {err_code: error_detail.code}.merge(context)
    )
  end

  def self.maybe(err)
    case err
    when ActiveRecord::RecordNotFound
      build(:record_not_found, {model: err&.model})
    when ActiveRecord::RecordInvalid
      build(:record_invalid, {errors: err&.record.errors.full_messages})
    else
      err
    end
  end
end
```

Step 2 - Code adjustments

graphql/mutations/messaging/write_message.rb

```ruby
module Mutations::Messaging
  class WriteMessage < Mutations::BaseMutation
    type ::Types::Messaging::MessageType

    argument :conversation_title, String, required: false
    argument :text, String, required: true
    argument :is_urgent, Boolean, required: false

    def resolve(conversation_title:, text:, is_urgent: false)
      raise GraphqlUserError.build(:not_authorized) if is_urgent && !context[:current_user].admin?

      conversation = ::Conversation.find_by!(title: conversation_title)
      conversation.messages.create!(
        text: text, participant: context[:current_user], urgent: is_urgent
      )
    rescue => err
      GraphqlUserError.maybe(err)
    end
  end
end
```

Step 2 - In Action

```
{
    "input": {
        "conversationTitle": "nonexistent one",
        "text": "Hey there!"
    }
}
```

```
"errors": [
    {
        "message": "Record not found",
        "locations": [
            {
                "line": 2,
                "column": 5
            }
        ],
        "path": [
            "writeMessagingMessage"
        ],
        "extensions": {
            "err_code": "RECORD_NOT_FOUND",
            "model": "Conversation"
        }
    }
],
```

Step 2 - In Action

```
{
    "input": {
        "conversationTitle": "Ruby community chat",
        "text": ""
    }
}
```

```
"errors": [
    {
        "message": "Record can not be saved",
        "locations": [
            {
                "line": 2,
                "column": 5
            }
        ],
        "path": [
            "writeMessagingMessage"
        ],
        "extensions": {
            "err_code": "RECORD_INVALID",
            "errors": [
                "Text can't be blank"
            ]
        }
    }
],
```

Step 2 - In Action

```
{
    "input": {
        "conversationTitle": "Ruby community chat",
        "text": "Hey There!",
        "isUrgent": true
    }
}
```

```
"errors": [
    {
        "message": "Not authorized",
        "locations": [
            {
                "line": 2,
                "column": 5
            }
        ],
        "path": [
            "writeMessagingMessage"
        ],
        "extensions": {
            "err_code": "NOT_AUTHORIZED"
        }
    }
],
```

Key Takeaways

- Mutations **are resolvers** of a special type
- Make **errors** part of the response
- Use **GraphQL::ExecutionError** to expose errors
- **Be careful** not to leak internal details

Testing Queries

spec/graphql/queries/conversation_spec.rb

```ruby
RSpec.describe "query conversation" do
  subject(:data) { root_result.fetch("data").fetch("conversation") }
  let(:root_result) { BuzzerSchema.execute(query).to_h }
  let(:query) do
    <<-GRAPHQL
    {
      conversation(title: "#{title}") {
        __typename
        title
      }
    }
    GRAPHQL
  end
  let(:title) { "Test-communication" }
  let!(:conversation) { ::Conversation.create(title: title) }

  it { expect(root_result).to_not include("errors") }
  it { is_expected.to include("__typename" => "MessagingConversation") }
  it { is_expected.to include("title" => title) }
end
```

Testing Mutations

spec/graphql/mutations/write_message_spec.rb

```ruby
RSpec.describe "mutation writeMessagingMessage" do
  subject(:data) { root_result.fetch("data").fetch("writeMessagingMessage") }
  let(:root_result) { BuzzerSchema.execute(query, context: context, variables: variables).to_h }
  let(:query) do
    <<-GRAPHQL
    mutation ($input: MutationsMessagingWriteMessageInput!){
        writeMessagingMessage(input: $input) {
            __typename
            from {
              __typename
              fullName
            }
            text
        }
    }
    GRAPHQL
  end
  let(:context) { {current_user: current_user} }
  let(:variables) do
    {
      input: {
        conversationTitle: title,
        text: "testing"
      }
    }
  end
  let(:title) { "Test-communication" }
  let!(:conversation) { ::Conversation.create(title: title) }
  let!(:current_user) { ::Participant.create(full_name: "Bob") }

  it { expect { root_result }.to change { Message.count }.by(1) }
  it { expect(root_result).to_not include("errors") }
  it { is_expected.to include("__typename" => "MessagingMessage") }
  it { is_expected.to include("text" => "testing") }
  it { is_expected.to include("from" => {"__typename" => "MessagingParticipant", "fullName" => "Bob"}) }
end
```

Testing Types

spec/graphql/types/messaging/message_spec.rb

```ruby
RSpec.describe "type MessagingMessage" do
  subject(:result) { schema.execute(query).to_h.fetch("data").fetch("__test") }

  let(:schema) do
    TestSchemaBuilder.build_with_query do
      field :__test, ::Types::Messaging::MessageType, null: false do
        argument :id, String, required: true
      end

      def __test(id:)
        ::Message.find(id)
      end
    end
  end

  let(:query) do
    <<-GRAPHQL
    {
      __test(id: "#{message.id}") {
        __typename
        text
        isUrgent
        from {
          __typename
        }
      }
    }
    GRAPHQL
  end

  let(:message) { ::Message.create!(text: "testing", urgent: true, participant: participant, conversation: conversation) }
  let(:conversation) { ::Conversation.create!(title: "Test-comm") }
  let(:participant) { ::Participant.create!(full_name: "Bob") }

  it { is_expected.to include("__typename" => "MessagingMessage") }
  it { is_expected.to include("text" => "testing") }
  it { is_expected.to include("isUrgent" => true) }
  it { is_expected.to include("from" => {"__typename" => "MessagingParticipant"}) }
end
```

Key Takeaways

- Test fields through **isolated type tests**
- Test only direct fields of the type, **don’t go deeper**
- Use **built-in “__typename”** field to assert on returned type in the other tests
- Watch out for key casing - Ruby uses **snake_case** but GraphQL JSON input/output uses **camelCase**
- Include a few full request tests for **end-to-end integrity**

Before we end

All code examples
and this presentation
are available at

github.com/AMekss/buzzer

Connection closed!

Thanks for listening