Call Me Back, Postgres

Introduction and application of Postgres
Triggers, Listen, and Notify

PGCONF NYC 2023

ifat@launchpadlab.com

@i_ribon9

inveterateliterate

Agenda

01 Key Concepts

02 Practical Applications

03 Case Study

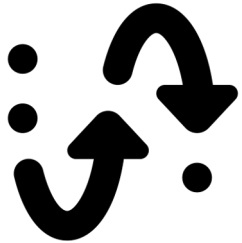
04 Resources

01

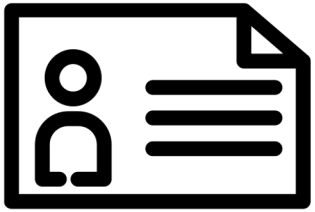
Key Concepts

PG Triggers

Callback at the database level that executes a defined function *before*, *after*, or *instead* of identified operations (i.e., `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`)



Surfaces **metadata** of the event, such as the operation (event name), schema, table name, and `OLD` and `NEW` versions of the record's attributes



Define a Function

```
CREATE OR REPLACE FUNCTION process_record() RETURNS TRIGGER AS $$
  BEGIN
    IF (TG_OP = 'DELETE') THEN
      INSERT INTO user_log(user_id, user_last_name, deleted_at)
      VALUES(OLD.id, OLD.last_name, NOW())
      RETURN OLD;
    ELSE
      INSERT INTO user_log(user_id, user_last_name, updated_at)
      VALUES(NEW.id, NEW.last_name, NOW())
      RETURN NEW;
    END IF;
   END;
$$ LANGUAGE plpgsql;
```

Apply Trigger Function

```
CREATE TRIGGER process_user_record
AFTER INSERT OR UPDATE OR DELETE ON users FOR EACH ROW
EXECUTE PROCEDURE process_record();
```

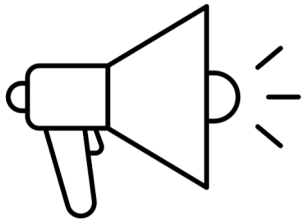
List Triggers

```
SELECT event_object_table AS table_name ,trigger_name
FROM information_schema.triggers
GROUP BY table_name , trigger_name
ORDER BY table_name ,trigger_name;
```

```
 table_name |    trigger_name
------------+---------------------
 users      | process_user_record
```

PG Notify and Listen

NOTIFY sends a notification event with a defined payload string through defined channels



LISTEN establishes sessions on defined channels to capture notifications sent on those channels

Broadcast a Notification

```
pg_notify(
  'create_or_update_record',
  '{table_name: users, crud_method: create_or_update, record_id: 2}'
);
```

Listen to a Channel

```
LISTEN create_or_update_record;
```

Disconnect

```
UNLISTEN *;
```

Perform pg_notify in a Trigger

```
CREATE OR REPLACE FUNCTION process_record() RETURNS TRIGGER AS $$
  BEGIN
    IF (TG_OP = 'DELETE') THEN
      PERFORM pg_notify('delete_record', '{klass_name: "' || TG_TABLE_NAME
    || '", crud_method: delete, record_id: "' || COALESCE(OLD.id, 0) ||
    '"}');
      RETURN OLD;
    ELSE
          PERFORM pg_notify('create_or_update_record', '{klass_name: "' ||
TG_TABLE_NAME || '", crud_method: create_or_update, record_id: "' ||
COALESCE(NEW.id, 0) || '"}');
      RETURN NEW;
    END IF;
   END;
$$ LANGUAGE plpgsql;
```

```
ifat:pg_triggers_demo/ (main*) $                                    [21:48:23]
psql -d pg_triggers_demo_dev
pg_triggers_demo_dev=#
```

02

Practical Applications

Listen for Notifications

```ruby
class DatabaseListener
  def listen
    ActiveRecord::Base.connection_pool.with_connection do |connection|
      conn = connection.instance_variable_get(:@connection)
      begin
        channels.each { |channel| conn.async_exec "LISTEN #{channel}" }
        loop do
          conn.wait_for_notify do |chan, _pid, payload|
            puts "Received a NOTIFY on #{chan} with payload: #{payload}"
            process_payload(args)
          end
        end
      ensure
        conn.async_exec 'UNLISTEN *'
      end
    end
  end
end
```

```
ifat:pg_triggers_demo/ (main*) $                    [22:01:27]
rails c
irb(main):003>
```

```
ifat:pg_triggers_demo/ (main*) $                    [22:03:48]
```

Process Payload in Ruby

```ruby
def process_payload
  NewUserMailer.deliver_later(user: payload[:user])
end

def process_payload
  AssignToAgent.call(ticket: payload[:ticket])
end

def process_payload
  ArchiveCourses.call(curriculum: payload[:curriculum])
end
```

03

Case Study: Integrating Salesforce with a Rails App

Main Players

Core Legacy App

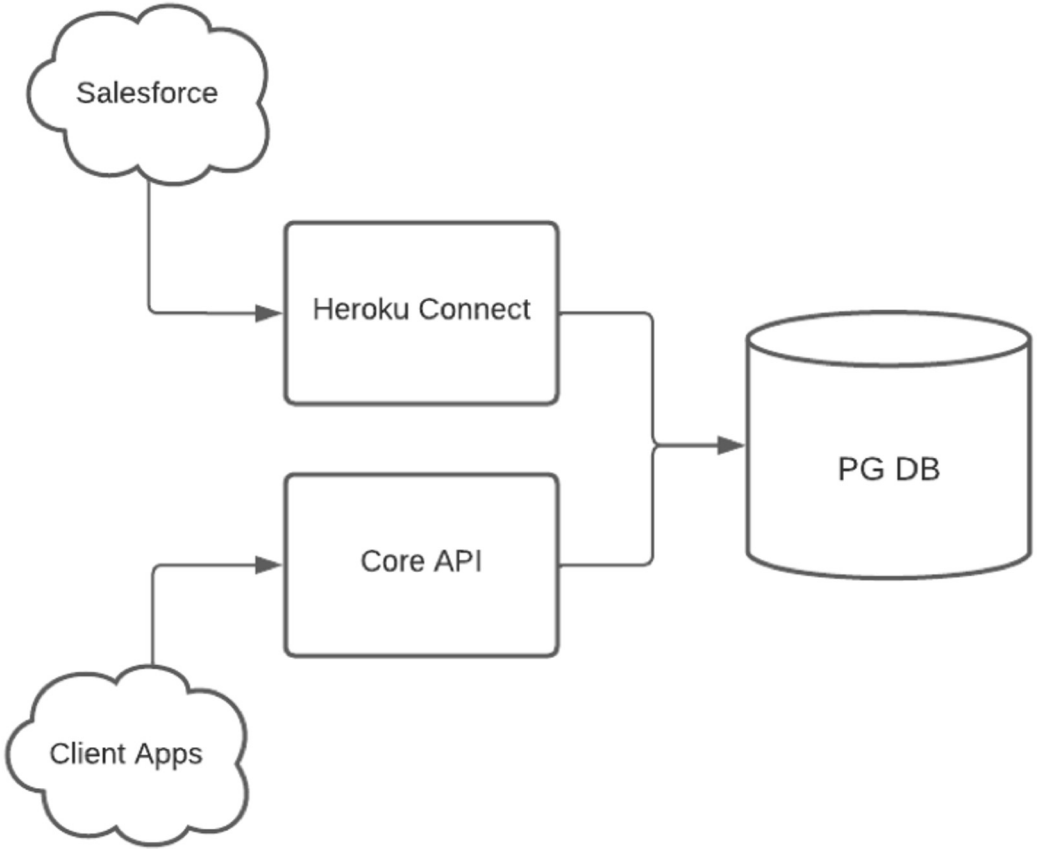
Salesforce

Heroku Connect

Core Postgres Database

Core Rails API

Key Connections



Rails Callbacks

- **What**: Rails ActiveRecord object life cycle hooks such as `after_create`, `after_update`, `after_delete`
- **Why**: Use Rails DSL and conventions to trigger additional logic based on a change to an object
- **Why not**: Direct consumers of the app database will not reach the Rails callbacks

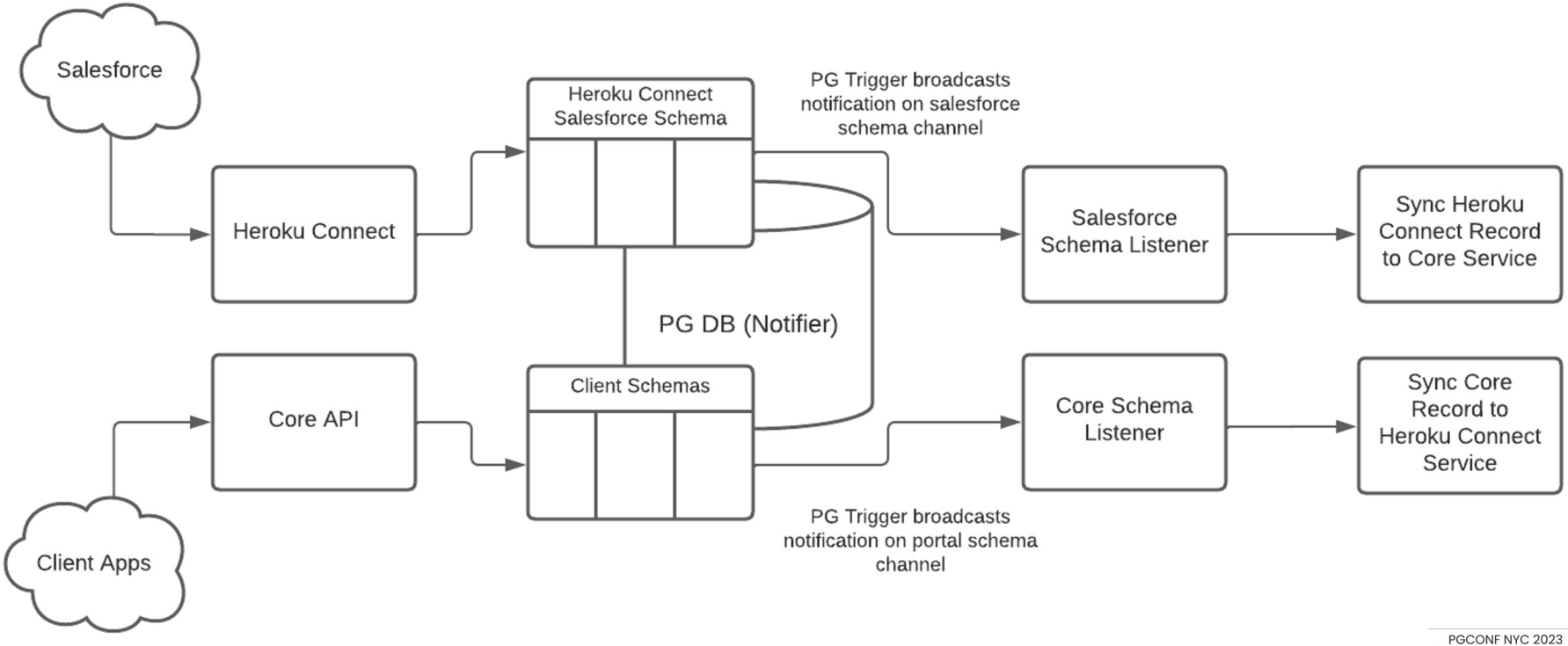
Polling

- **What:** Regular job to look at data in the database and identify changes such as newly created, updated, or deleted records
- **Why:** Accessible to all consumers of the database, can control the timing and considerations of the data syncing process
- **Why not:** Expensive to frequently query the database, lower reliability writing the checks and queries by hand

PG Triggers with Listen / Notify

- **What:** Database-level callbacks that tap into a pub/sub scheme
- **Why:** Any event from any client connected to the database will be captured
- **Why Not:** Requires Postgres-specific knowledge and requires writing explicit SQL code, may be less discoverable or obvious in a conventional Rails application

Syncing Process Design



Logging Triggers and Events

```
CREATE OR REPLACE FUNCTION process_record() RETURNS TRIGGER AS $$
  DECLARE
    payload jsonb;
    database_trigger_log_id integer;
  BEGIN
      payload = json_build_object('operation', TG_OP, 'klass_name',
TG_TABLE_NAME, 'crud_method', 'create_or_update', 'record_id', NEW.id);
      INSERT INTO database_trigger_logs(payload, schema, klass_name, created_at,
updated_at) VALUES(payload, TG_TABLE_SCHEMA,TG_TABLE_NAME,  NOW(), NOW())
RETURNING id INTO database_trigger_log_id;
      PERFORM pg_notify('create_or_update_record', '{klass_name: "' ||
TG_TABLE_NAME || '", crud_method: create_or_update, record_id: "' ||
COALESCE(NEW.id, 0) || '", database_trigger_log_id: "' || database_trigger_log_id
|| '"}');
      RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;
```

Avoid Continuous Trigger Loops

```
def commit_transaction_without_trigger(&block)
  ActiveRecord::Base.transaction do
    ActiveRecord::Base.connection_pool.with_connection do
|connection|
      connection.execute "ALTER TABLE users DISABLE TRIGGER
process_record"
      yield.tap do |return_value|
        connection.execute "ALTER TABLE users ENABLE TRIGGER
    process_record”
        update_database_trigger_log
      end
    end
  end
end
```

Write Automated Tests!

Integration Tests

```
RSpec.describe 'Process User - Integrated Scenario', type: :scenario
do
  describe 'user created' do
    before do
      @subscription_count = Subscription.count
      User.create(first_name: 'New', last_name: 'User', email:
'user1@example.com')
      sleep(1.0)
    end

    it 'creates a subscription' do
      expect(Subscription.count).to eq(@subscription_count + 1)
    end
  end
end
```


ifat:pg_triggers_demo/ (main*) $
[23:16:56]
bundle exec rspec -t type:scenario

05

Resources

Resources

- PG Trigger Documentation
- PG Trigger Variables
- PG Notify Documentation
- PG Listen Documentation
- Heroku Connect
- PostgreSQL Listen / Notify

Thank You!

ifat@launchpadlab.com

github.com/inveterateliterate