Chrono - Bug #55393

IntegrityError: duplicate key value violates unique constraint "agendas_event_agenda_id_slug_9ab8727a_uniq"

05 juillet 2021 10:28 - Sentry Io

Statut:	Fermé	**Début:**	05 juillet 2021
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	Non

Description

https://sentry.entrouvert.org/entrouvert/publik/issues/49312/

```
Event.DoesNotExist: Event matching query does not exist.
  File "chrono/agendas/models.py", line 1494, in get_or_create_event_recurrence
    return Event.objects.get(agenda=self.agenda, slug=event.slug)
  File "django/db/models/manager.py", line 82, in manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "django/db/models/query.py", line 408, in get
    self.model._meta.object_name

IntegrityError: duplicate key value violates unique constraint "agendas_event_agenda_id_slug_9ab87
27a_uniq"
DETAIL:  Key (agenda_id, slug)=(235, vdm-participer-aux-vaccinations-a-saint-symphorien-infirmiers
-event-5--2021-08-07-0800) already exists.

  File "django/db/backends/utils.py", line 84, in _execute
    return self.cursor.execute(sql, params)

IntegrityError: duplicate key value violates unique constraint "agendas_event_agenda_id_slug_9ab87
27a_uniq"
DETAIL:  Key (agenda_id, slug)=(235, vdm-participer-aux-vaccinations-a-saint-symphorien-infirmiers
-event-5--2021-08-07-0800) already exists.

(21 additional frame(s) were not displayed)
...
  File "django/db/backends/utils.py", line 67, in execute
    return self._execute_with_wrappers(sql, params, many=False, executor=self._execute)
  File "django/db/backends/utils.py", line 76, in _execute_with_wrappers
    return executor(sql, params, many, context)
  File "django/db/backends/utils.py", line 84, in _execute
    return self.cursor.execute(sql, params)
  File "django/db/utils.py", line 89, in __exit__
    raise dj_exc_value.with_traceback(traceback) from exc_value
  File "django/db/backends/utils.py", line 84, in _execute
    return self.cursor.execute(sql, params)
```

Révisions associées

Révision bd51c513 - 09 juillet 2021 09:53 - Paul Marillonnet

mitigate race condition while get-or-creating reccurent event (#55393)

```
inspired from django's behavior on get_or_create, e.g.
    https://github.com/django/django/blob/stable/-
    2.2.x/django/db/models/query.py#L567
```

Historique

#1 - 05 juillet 2021 10:29 - Lauréline Guérin

- Projet changé de Suivi des traces à Chrono

#2 - 05 juillet 2021 14:01 - Valentin Deniaud

J'ai regardé sans succès, le code

```
        with transaction.atomic():
            try:
                return Event.objects.get(agenda=self.agenda, slug=event.slug)
            except Event.DoesNotExist:
                event.save()
                return event
```

lève l'exception sur la clé (agenda, slug) et je vois pas comment c'est possible.

#3 - 07 juillet 2021 18:31 - Paul Marillonnet

- Fichier 0001-agendas-don-t-mix-db-operation-and-exception-handlin.patch ajouté

- Statut changé de Nouveau à Solution proposée

- Patch proposed changé de Non à Oui

Oui c'est ce bout de code qui pose problème, c'est un comportement non déterministe de django, i.e. écrire en base lors de la capture d'une exception à l'intérieur d'un bloc atomic.
Je crois qu'il vaut mieux faire quelque chose comme ça.

#4 - 08 juillet 2021 09:18 - Lauréline Guérin

Ca marchera mieux avec ta proposition, mais alors on commencera toujours par faire un create, puis un get, à partir du moment où l'event existera en DB.
C'est dommage de devoir faire 2 requêtes alors qu'une suffirait dans tous les cas sauf la première fois :)

#5 - 08 juillet 2021 09:59 - Paul Marillonnet

Lauréline Guerin a écrit :

> Ca marchera mieux avec ta proposition, mais alors on commencera toujours par faire un create, puis un get, à partir du moment où l'event existera en DB.
> C'est dommage de devoir faire 2 requêtes alors qu'une suffirait dans tous les cas sauf la première fois :)

Ah oui, mauvaise connaissance de ce code de ma part, je n'avais pas réalisé que c'est la lecture en base d'une occurrence existante qui aura lieu plus souvent que l'écriture d'une nouvelle occurrence.
Je crois qu'avec une solution comprenant le get d'abord, il faut un checkpoint intermédiaire. Je vais voir si ça tient la route.

#6 - 08 juillet 2021 10:03 - Lauréline Guérin

Et sinon, en transformant le code existant pour avoir un get_or_create ? Django gère très bien ce cas :) (et il commence par tenter un get)

#7 - 08 juillet 2021 10:12 - Valentin Deniaud

Le code a cette tête parce que je pensais qu'en mettant le try/except dans un transaction.atomic, un objet ne pouvait pas être créé entre le get et le create, mais en fait ça n'est pas du tout le cas, donc le code est nul avant même de considérer que c'est mal de rattraper une exception à l'intérieur du bloc.

Ce que je voulais faire c'est bien un get_or_create à la Django, comme ça
https://github.com/django/django/blob/main/django/db/models/query.py#L573.

#8 - 08 juillet 2021 10:20 - Paul Marillonnet

Valentin Deniaud a écrit :

> Le code a cette tête parce que je pensais qu'en mettant le try/except dans un transaction.atomic, un objet ne pouvait pas être créé entre le get et le create, mais en fait ça n'est pas du tout le cas, donc le code est nul avant même de considérer que c'est mal de rattraper une exception à l'intérieur du bloc.
>
> Ce que je voulais faire c'est bien un get_or_create à la Django, comme ça
> https://github.com/django/django/blob/main/django/db/models/query.py#L573.

J'étais en train d'écrire une réponse en référençant précisément cette ligne (mais dans la version 2.2 de django, qui fait apparaître une méthode _create_object_from_params[1]) :)
En effet on peut s'inspirer de ça, i.e. (a) commencer par essayer un get, et sur inexistance de l'objet, (b) basculer sur une transaction (telle que proposée dans le précédent patch) dont l'échec par une erreur d'intégrité provoque (c) un dernier get.

1. https://github.com/django/django/blob/stable/2.2.x/django/db/models/query.py#L567

#9 - 08 juillet 2021 10:21 - Paul Marillonnet

- Statut changé de Solution proposée à En cours

#10 - 08 juillet 2021 10:26 - Paul Marillonnet

Petite question au passage : quitte à reproduire le comportement django-esque, est-ce qu'on en profite pour s'aligner sur ce que django retourne sur get_or_create (non pas l'objet seul mais le tuple (object, created)) ? Est-ce que ça aurait un intérêt ici ?

#11 - 08 juillet 2021 10:37 - Valentin Deniaud

Paul Marillonnet a écrit :

> Est-ce que ça aurait un intérêt ici ?

Nop, faire au plus simple.

#12 - 08 juillet 2021 10:40 - Paul Marillonnet

- Fichier 0001-mitigate-race-condition-while-get-or-creating-reccur.patch ajouté

- Statut changé de En cours à Solution proposée

Ok. Une version simple qui reprend l'esprit du get_or_create de django. Sans test, je suis à court d'idée sur comment simuler l'accès concurrent en échec :)

#13 - 08 juillet 2021 11:10 - Valentin Deniaud

- Statut changé de Solution proposée à Solution validée

#14 - 09 juillet 2021 09:55 - Paul Marillonnet

- Statut changé de Solution validée à Résolu (à déployer)

```
commit bd51c5138a6da218013999d191479d8adb131cda
Author: Paul Marillonnet <pmarillonnet@entrouvert.com>
Date:   Wed Jul 7 17:49:30 2021 +0200

    mitigate race condition while get-or-creating reccurent event (#55393)

        inspired from django's behavior on get_or_create, e.g.
        https://github.com/django/django/blob/stable/-
        2.2.x/django/db/models/query.py#L567
```

#15 - 13 juillet 2021 00:17 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

Fichiers

0001-agendas-don-t-mix-db-operation-and-exception-handlin.patch	1,97 ko	07 juillet 2021	Paul Marillonnet
0001-mitigate-race-condition-while-get-or-creating-reccur.patch	2,26 ko	08 juillet 2021	Paul Marillonnet