



PostgreSQL sans schéma avec Python



Martin Kirchgessner / CTO Luxia / [@martin_kirch](https://twitter.com/martin_kirch)

Meetup Python Grenoble - 27 Nov. 2019



Il était une fois...

Un blog

```
from uuid import uuid4
from sqlalchemy import Column, String, TIMESTAMP
from sqlalchemy.dialects.postgresql import UUID
from sa_metadata import Base

class Article(Base):
    __tablename__ = 'articles'

    uuid = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    date = Column(TIMESTAMP(timezone=True), nullable=False)
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
```



On peut écrire

```
article = Article(  
    date=datetime.now(),  
    title="Super Python Blog #1",  
    content="Hello World")
```

```
session.add(article)  
session.commit()
```



Et rechercher

```
articles = (session.query(Article)
            .order_by(Article.date.desc())
            .limit(5)
            .all()
)
log.info("Latest 5 articles in blog:")

for article in articles:
    log.info("%r", article)
```



« et si on ajoutait »

- L'auteur
- Des auteurs
- Un slug
- Plusieurs slugs
- Des tags
- Une miniature
- ...



JSONB



- Un type de colonne
- ⚠ JSON != JSONB ⚠
- Syntaxe un peu lourde en SQL

Par exemple `ma_colonne->'level' = '1'::jsonb`

mais merci `jsonb_pretty(ma_colonne_jsonb)` !



La classe

```
from sqlalchemy.ext.mutable import MutableDict
from sqlalchemy.dialects.postgresql import JSONB

class Article(Base):
    __tablename__ = 'articles_with_extra'

    uuid = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    date = Column(TIMESTAMP(timezone=True), nullable=False)
    title = Column(String, nullable=False)
    content = Column(String, nullable=False)
    extra = Column(MutableDict.as_mutable(JSONB), nullable=False, default={})
```



Enregistrons un slug

```
article = Article(  
    date=datetime.now(),  
    title="Super Python Blog #3",  
    content="Hello World of URI slugs",  
    extra={  
        'slug': 'super-python-blog-3'  
    })
```

```
article = (session  
    .query(Article)  
    .filter(Article.extra['slug'].astext == 'super-python-blog-3')  
    .first()  
)
```




Enregistrons des tags

```
article = Article(  
    date=datetime.now(),  
    title="Super Python Blog #4",  
    content="Now with tags",  
    extra={  
        'tags': ["JSONB", "PostgreSQL", "Python"]  
    })
```

```
by_tag = (session  
    .query(Article)  
    .filter(Article.extra.contains(  
        {'tags': ["PostgreSQL"]}  
    ))  
    .all()  
)
```

Mise à jour dangereuse

```
article = (session
    .query(Article)
    .filter(Article.extra.contains(
        {'tags': ["PostgreSQL"]})))
    .first()
)
```

```
article_uuid = article.uuid
article.extra['tags'][0] = "JSON"
session.commit()
```

```
# ... plus tard, dans un session lointaine
```

```
article = session.query(Article).get(article_uuid)
```

```
# article.extra['tags'][0] ==??
```



Solution

```
article = (session
    .query(Article)
    .filter(Article.extra.contains(
        {'tags': ["PostgreSQL"]})))
    .first()
)
article.extra['tags'][0] = "JSON"
article.extra.changed() # 👁 👁 👁 👁 👁
session.commit()
```

... plus tard, dans un session lointaine

```
article = session.query(Article).get(article_uuid)
```

```
# article.extra['tags'][0] == "JSON" \o/
```



**Voilà votre table est prête
pour les articles les plus
tordus**



Quand soudain



- Déjà 1 million d'articles !
- La recherche est leeeente



Recherche par date

```
by_date = (session
            .query(Article)
            .filter(Article.date == datetime(2019, 11, 27))
            .all()
            )
```



Recherche par date

```
date = Column(TIMESTAMP(timezone=True), nullable=False, index=True)
```





Recherche par slug

```
extra = Column(  
    MutableDict.as_mutable(JSONB),  
    nullable=False,  
    default={}  
)  
  
__table_args__ = (  
    Index(  
        'ix_articles_polymorphic_extra_slug',  
        extra['slug'].astext),  
    )  
)
```




Rechercher par tag

```
extra = Column(  
    MutableDict.as_mutable(JSONB),  
    nullable=False,  
    default={}  
)  
  
__table_args__ = (  
    Index(  
        'ix_articles_polymorphic_extra_tag',  
        extra['tags'],  
        postgresql_using='gin'  
    )  
)
```

GIN=General Inverted Index



et ça repart



... mais c'est un
schéma caché, non ?



L'autre schéma caché

Le code devient envahi de tests comme

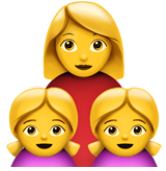
```
if 'tags' in extra
...
if isinstance(extra['tags'], list)
```



Le polymorphisme

SQLAlchemy va instancier une classe différente en fonction d'un critère donné

```
class Article(Base):  
    # ... pareil ... plus une colonne :  
    article_type = Column(Integer)  
  
    __mapper_args__ = {  
        'polymorphic_on': article_type,  
        'polymorphic_identity': None,  
    }
```



Deux sous-classes

```
'polymorphic_on': article_type,
```

```
...
```

```
class TaggedArticle(Article):
```

```
    __mapper_args__ = {
```

```
        'polymorphic_identity':
```

```
            ArticleType.TAGGED.value}
```

```
class ImageArticle(Article):
```

```
    __mapper_args__ = {
```

```
        'polymorphic_identity':
```

```
            ArticleType.IMAGE.value}
```

```
class ArticleType(Enum):
```

```
    TAGGED = 1
```

```
    IMAGE = 2
```

Examples 🙄

```
tagged = TaggedArticle(  
    date=datetime.now(),  
    title="My first tagged article",  
    content="Bla bla bla",  
    extra={'tags': ['blog']},  
)
```

```
image = ImageArticle(  
    date=datetime.now(),  
    title="My first picture",  
    content="https://example.com/favicon.ico",  
)
```

Exemples 🙄

```
articles = (session
            .query(Article)
            .order_by(Article.date.desc())
            .limit(5)
            .all()
            )
```

➡ **Mélange de TaggedArticle et ImageArticle**

Examples 🙄🙄

```
count = (session
        .query(TaggedArticle)
        .filter(Article.extra.contains(
            { 'tags': ["article"] }
        )) .count()
```



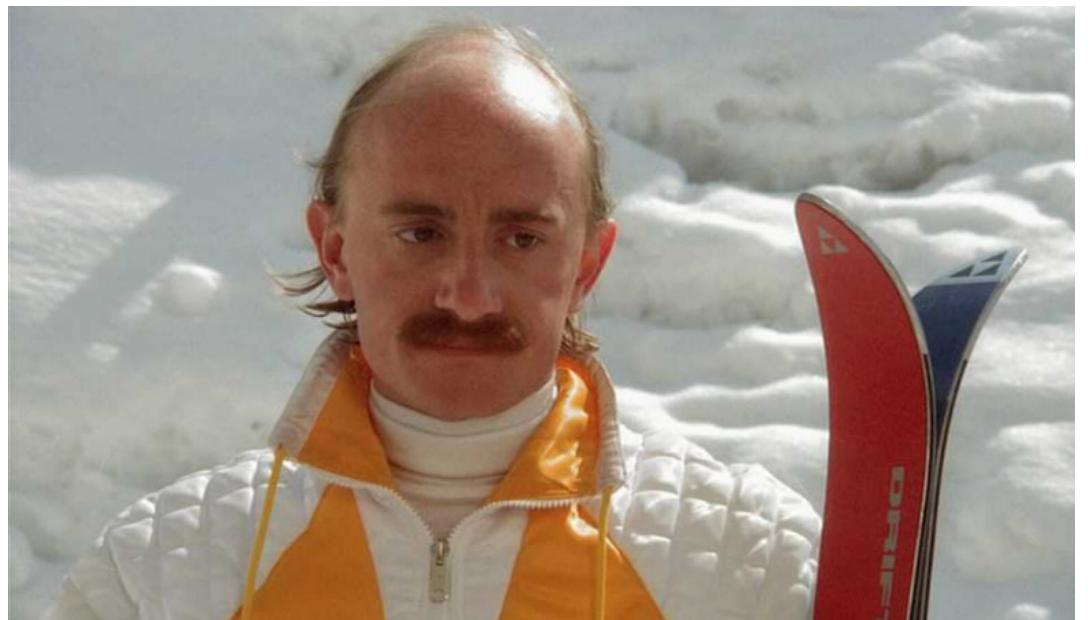

Magie de calendriers

- On peut fournir nos (dé)sérialiseurs JSON (lors de la création de la connexion)
- Mais, en SQL, '2019-11-27' < '2019-11-27 00:00:00'

```
filter(Article  
      .extra['first_publication_date']  
      .astext.cast(Date) >= my_datetime)
```

Utilisation chez LUXIA

- Télécharge/traite/indexe des documents juridiques
- Tout ça dans une table Documents
- Bientôt 100 sous-classes de Document
- Avec de l'héritage multiple, des mixins
- On met des **extra** partout
- Au fait : on recrute !



Pour conclure



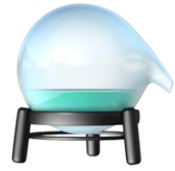
Postgres c'est chouette

- Sharding horizontal
- Réplication logique
- Communauté importante
 - On devrait faire un meetup qui s'appelle PostGRE
- Fiable et difficile à tuer
- Pensez-y en même temps que Mongo/Couch/Elastic/...
 - Sauf si vous écrivez *beaucoup*



Existe en tableau

```
sa.Column(  
    MutableList.as_mutable(  
        postgresql.ARRAY(sa.BigInteger))
```



Alembic

- Versionne le schéma à partir des classes SQLAlchemy
- Mais pas les index /JSON

Tout cela est dans un projet-demo :

<https://github.com/martinkirch/jsonb-sqlalchemy-demo>

?