Implementing

a Cross-DB

`JSONField`

background


background


{
"name": "Sage M. Abdullah",
"background": "Computer Science student at Universitas Indonesia
"roles": {
"django": {"user": true, "contributor": true}
},
"experiences": [
{
"title": "Google Summer of Code",
"year": 2019,
"organization": "Django Software Foundation",
"project": "Cross-DB JSONField"
}
],
"username": "laymonage"
}


JASONField

JSONField

`JSONField`

“A field for storing JSON-encoded data.”

JSONField

“A field for storing JSON-encoded data.”

In Python, represented as:

- **dict**
- **list**
- str
- int
- float
- bool
- None

JSON-encoded data

JSON data

```json
{
  "name": "Sage",
  "active": true,
  "age": 21,
  "height": 170.0,
  "interests": [
    {"hobbies": ["reading", "coding"]},
    {"others": ["cats", 42]}
  ]
}
```

JSON data

```json
{
  "name": "Sage",
  "active": true,
  "age": 21,
  "height": 170.0,
  "interests": [
    {"hobbies": ["reading", "coding"]},
    {"others": ["cats", 42]}
  ],
  "partner": null
}
```

JSON-encoded data

```
# This is in Python
data = '''{
  "name": "Sage",
  "active": true,
  "age": 21,
  "height": 170.0,
  "interests": [
    {"hobbies": ["reading", "coding"]},
    {"others": ["cats", 42]}
  ],
  "partner": null
}'''
```

Databases

Databases

```
class Profile(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.CharField(max_length=255)
    last_sync = models.DateTimeField(auto_now=True)
```

Databases

```python
class Profile(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.CharField(max_length=255)
    last_sync = models.DateTimeField(auto_now=True)
```

myapp_profile

user_id	status	last_sync
32	Happy!	2020-08-17T19:45:05.48151
97	Bored...	2020-08-15T12:34:56.12345

Databases

myapp_profile

user_id	status	last_sync
32	Happy!	2020-08-17T19:45:05.481516
97	Bored...	2020-08-15T12:34:56.123456

User Config in Databases

myapp_profile

user_id	status	last_sync	dark_mode
32	Happy!	2020-08-17T19:45:05.481516	1
97	Bored...	2020-08-15T12:34:56.123456	0

User Config in Databases

myapp_profile

user_id	status	last_sync	dark_mode	font_size
32	Happy!	2020-08-17T19:45:05.481516	1	1
97	Bored...	2020-08-15T12:34:56.123456	0	3

User Config in Databases

myapp_profile

user_id	status	last_sync	dark_mode	font_size	color_accent
32	Happy!	2020-08-17T19:45:05.481516	1	1	blue
97	Bored...	2020-08-15T12:34:56.123456	0	3	red

User Config in Databases

myapp_profile

user_id	status	last_sync	config
32	Happy!	2020-08-17T19:45:05.481516	1;1;blue
97	Bored...	2020-08-15T12:34:56.123456	0;3;red

JSON in (SQL) Databases

myapp_profile

user_id	status	last_sync	config

32	Happy!	2020-08-17T19:45:05.481516	<pre>{
			  "dark_mode": true,
			  "font_size": 1,
			  "color_scheme": "blue"
			}</pre>

97	Bored...	2020-08-15T12:34:56.123456	<pre>{
			  "dark_mode": false,
			  "font_size": 3,
			  "color_scheme": "red"
			}</pre>

JSON in (SQL) Databases

myapp_profile

user_id	status	last_sync	config
32	Happy!	2020-08-17T19:45:05.481516	{"dark_mode": true, "font_size": 1, "color_scheme": "blue"}
97	Bored...	2020-08-15T12:34:56.123456	{"dark_mode": false, "font_size": 3, "color_scheme": "red"}

JSONField

JSONField

How does it work?

JSONField

How does it work?

```
1 class Profile(models.Model):
2     ...
3     config = models.JSONField()
```

JSONField

How does it work?

```
1 class Profile(models.Model):
2     ...
3     config = models.JSONField()
```

```
1 >>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
2 >>> profile = Profile.objects.create(config=config)
3 >>> # Some time later...
4 >>> saved_profile = Profile.objects.get(id=profile.id)
5 >>> saved_profile.config == config
6 True
```

JSONField

How does it work?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile.objects.create(config=config)
>>> # Some time later...
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config == config
True
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> saved_profile.config['font_size'] = 3
>>> saved_profile.save()
>>> Profile.objects.get(id=saved_profile.id).config['font_size']
3
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile.objects.create(config=config)
>>> # Some time later...
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config == config
True
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> saved_profile.config['font_size'] = 3
>>> saved_profile.save()
>>> Profile.objects.get(id=saved_profile.id).config['font_size']
3
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile.objects.create(config=config)
>>> # Some time later...
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile(config=config)
>>> profile.save()
>>> # Some time later...
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile(config=config)
>>> profile.save()
>>> # Turn it into a JSON-encoded data!
>>> '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile(config=config)
>>> profile.save()
>>> # Turn it into a JSON-encoded data!
>>> '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> # Eventually, it will be:
>>> """
INSERT INTO myapp_profile
VALUES (42, '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}')
"""
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile(config=config)
>>> profile.save()
>>> # Turn it into a JSON-encoded data!
>>> '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> # Eventually, it will be:
>>> """
INSERT INTO myapp_profile
VALUES (42, '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}')
"""
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> """SELECT id, config FROM myapp_profile WHERE id = 42"""
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

JSONField

How does it work... in the background?

```
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> profile = Profile(config=config)
>>> profile.save()
>>> # Turn it into a JSON-encoded data!
>>> '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> # Eventually, it will be:
>>> """
INSERT INTO myapp_profile
VALUES (42, '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}')
"""
>>> saved_profile = Profile.objects.get(id=profile.id)
>>> """SELECT id, config FROM myapp_profile WHERE id = 42"""
>>> '{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> saved_profile.config
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
```

Python's `json` library

Python's `json` library

```
>>> import json
>>> config = {'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> encoded = json.dumps(config)
>>> encoded
'{"dark_mode": true, "font_size": 2, "color_scheme": "pink"}'
>>> decoded = json.loads(encoded)
>>> decoded
{'dark_mode': True, 'font_size': 2, 'color_scheme': 'pink'}
>>> decoded == config
True
```

A very minimal `JSONField`

A very minimal `JSONField`

```
class JSONField(TextField):
    def get_prep_value(self, value):
        if value is None:
            return value
        return json.dumps(value)
    def from_db_value(self, value, expression, connection):
        if value is None:
            return value
        return json.loads(value)
```

A very minimal `JSONField`

```
class JSONField(TextField):

    def get_prep_value(self, value):

        if value is None:

            return value

        return json.dumps(value)

    def from_db_value(self, value, expression, connection):

        if value is None:

            return value

        return json.loads(value)
```

A very minimal `JSONField`

```
class JSONField(TextField):
    """A very minimal JSONField."""

    def get_prep_value(self, value):
        if value is None:
            return value
        return json.dumps(value)

    def from_db_value(self, value, expression, connection):
        if value is None:
            return value
        return json.loads(value)
```

A very minimal `JSONField`

```
 1 class JSONField(TextField):
 2     """A very minimal JSONField."""
 3
 4     def __init__(self, encoder=None, decoder=None):
 5         ...
 6
 7     def get_prep_value(self, value):
 8         if value is None:
 9             return value
10         return json.dumps(value, cls=self.encoder)
11
12     def from_db_value(self, value, expression, connection):
13         if value is None:
14             return value
15         return json.loads(value, cls=self.decoder)
```

The thing about emptiness

The thing about empty values

The thing about empty values

- `None` ≡ `NULL`
- `' '` > `NULL`

What about...

`"" {} [] null`

The thing about empty values

- `None` ≡ `NULL`
- `''` > `NULL`

What about...

`'""'` `'{}'` `'[]'` `'null'`

Comparison of literals

Python	JSON	SQL
'', ""	""	'""'
{}	{}	'{}'
[]	[]	'[]'
None	null	'null' NULL
Value('null')None	null	'null'
{'something': None}	{"something": null}	'{"something": null}'
[None]	[null]	'[null]'

Differences in the backends

Differences in the backends

```
class JSONField(TextField):
    ...

    def db_type(self, connection):
        db_types = {
            "mysql": "json",
            "oracle": "nclob",
            "postgresql": "jsonb",
            "sqlite": "text",
        }
        return db_types[connection.vendor]
```

Differences in the backends

```
class JSONField(TextField):
    ...

    def db_check(self, connection):
        params = self.db_type_parameters(connection)
        if connection.vendor == "mysql":
            if (
                connection.mysql_is_mariadb and
                connection.mysql_version < (10, 4, 3)
            ):
                return "JSON_VALID(`%(column)s`)" % params
        ...
```

Differences in the backends

```
class JSONField(TextField):
    ...

    def db_check(self, connection):
        ...
        if connection.vendor == "oracle":
            return "%(qn_column)s IS JSON" % params
        if connection.vendor == "sqlite":
            return (
              '(JSON_VALID("%(column)s") '
              'OR "%(column)s" IS NULL)' % params
            )
        return None
```

What else?

Querying

Querying

```
>>> MyModel.objects.filter(some_numeric_field=3)
```

```
SELECT ... WHERE some_numeric_field = 3;
```

```
>>> MyModel.objects.filter(some_numeric_field__gte=3)
```

```
SELECT ... WHERE some_numeric_field >= 3;
```

Lookups

```
>>> MyModel.objects.filter(some_numeric_field=3)
```

```
SELECT ... WHERE some_numeric_field = 3;
```

```
>>> MyModel.objects.filter(some_numeric_field__gte=3)
```

```
SELECT ... WHERE some_numeric_field >= 3;
```

Lookups

```
>>> MyModel.objects.filter(some_numeric_field__exact=3)
```

```
SELECT ... WHERE some_numeric_field = 3;
```

```
>>> MyModel.objects.filter(some_numeric_field__gte=3)
```

```
SELECT ... WHERE some_numeric_field >= 3;
```

Transforms

Transforms

```
>>> MyModel.objects.filter(some_date_field__year=2020)
```

```
SELECT ... WHERE some_date_field
BETWEEN '2020-01-01' AND '2020-12-31';
```

```
>>> MyModel.objects.filter(some_date_field__year__gte=2020)
```

```
SELECT ... WHERE some_date_field >= '2020-01-01';
```

Transforms

```
>>> MyModel.objects.filter(some_date_field__year__exact=2020)
```

```
SELECT ... WHERE some_date_field
BETWEEN '2020-01-01' AND '2020-12-31';
```

```
>>> MyModel.objects.filter(some_date_field__year__gte=2020)
```

```
SELECT ... WHERE some_date_field >= '2020-01-01';
```

JSONField Transforms

JSONField Transforms

```
{
  "name": "Sage",
  "age": 21
}
```

```
>>> MyModel.objects.filter(some_json_field__name='Sage')
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field -> 'name' = 'Sage';
```

```
SELECT ... -- SQLite
WHERE JSON_EXTRACT(some_json_field, '$.name') = 'Sage';
```

```
SELECT ... -- MySQL/MariaDB
WHERE JSON_UNQUOTE(
  JSON_EXTRACT(some_json_field, '$.name')) = 'Sage';
```

```
SELECT ... -- Oracle
WHERE JSON_VALUE(some_json_field, '$.name') = 'Sage';
```

JSONField Transforms

```
{
  "name": "Sage",
  "age": 21,
  "pets": [
    {"name": "Bagol", "species": "cat"}
  ]
}
```

```
>>> MyModel.objects.filter(
        some_json_field__pets__0__name='Bagol')
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field #> {'pets', '0', 'name'} = 'Bagol';
```

```
SELECT ... -- SQLite
WHERE JSON_EXTRACT(
  some_json_field, '$.pets[0].name') = 'Bagol';
```

Still about empty values...

Still about empty values...

```
{
  "name": "Sage",
  "age": 21
}
```

```
{
  "name": "Sage",
  "age": 21,
  "partner": null
}
```

```
>>> MyModel.objects.filter(
        some_json_field__partner=None)
```

```
>>> MyModel.objects.filter(
        some_json_field__partner__isnull=True)
```

Still about empty values...

```
{
  "name": "Sage",
  "age": 21
}
```

```
{
  "name": "Sage",
  "age": 21,
  "partner": null
}
```

```
>>> MyModel.objects.filter(
        some_json_field__partner__exact=None)
```

```
>>> MyModel.objects.filter(
        some_json_field__partner__isnull=True)
```

Still about empty values...

```
>>> MyModel.objects.filter(
        some_json_field__partner__exact=None)
```

```
>>> MyModel.objects.filter(
        some_json_field__partner__isnull=True)
```

```
SELECT ... -- SQLite
WHERE JSON_TYPE(
  some_json_field, '$.pets[0].name') = 'null';
```

JSONField Lookups

JSONField Lookups

Containment

```
{
  "name": "Sage",
  "age": 21,
  "pets": [
    {"name": "Bagol", "species": "cat"}
  ]
}
```

JSONField Lookups

Containment

```
{
  "name": "Sage",
  "age": 21,
  "pets": [
    {"name": "Bagol", "species": "cat"},
    {"name": "Goldy", "species": "goldfish"}
  ]
}
```

JSONField Lookups

Containment

```
{
  "name": "Sage",
  "age": 21,
  "pets": [
    {"name": "Bagol", "species": "cat"},
    {"name": "Goldy", "species": "goldfish"}
  ]
}
```

```
>>> MyModel.objects.filter(
        some_json_field__contains={
            "age": 21,
            "pets": [{"species": "goldfish"}]
        }
    )
```

JSONField Lookups

Containment

```
>>> MyModel.objects.filter(
        some_json_field__contains={
            "age": 21,
            "pets": [{"species": "goldfish"}]
        }
    )
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field @> '{"age": 21, ...}';
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS(some_json_field, '{"age": 21, ...}');
```

JSONField Lookups

Containment

```
{
  "name": "Sage",
  "age": 21
}
```

```
>>> MyModel.objects.filter(
        some_json_field__contained_by={
            "age": 21,
            "name": "Sage",
            "pets": [
              {"name": "Bagol", "species": "cat"},
              {"name": "Goldy", "species": "goldfish"}
            ]
        }
    )
```

JSONField Lookups

Containment

```
>>> MyModel.objects.filter(
        some_json_field__contained_by={
            "age": 21,
            "name": "Sage",
            "pets": [
              {"name": "Bagol", "species": "cat"},
              {"name": "Goldy", "species": "goldfish"}
            ]
        }
    )
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field <@ '{"age": 21, ...}';
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS('{"age": 21, ...}', some_json_field);
```

JSONField Lookups

Key existence

JSONField Lookups

Key existence

```
>>> MyModel.objects.filter(some_json_field__has_key='pets')
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field ? 'pets';
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS_PATH(some_json_field, 'one', '$.pets');
```

```
SELECT ... -- Oracle
WHERE JSON_EXISTS(some_json_field, '$.pets');
```

```
SELECT ... -- SQLite
WHERE JSON_TYPE(some_json_field, '$.pets') IS NOT NULL;
```

JSONField Lookups

Key existence

```
>>> MyModel.objects.filter(
        some_json_field__has_keys=['pets', 'age'])
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field ?& {'pets', 'age'};
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS_PATH(
  some_json_field, 'all', '$.pets', '$.age');
```

JSONField Lookups

Key existence

```
>>> MyModel.objects.filter(
        some_json_field__has_keys=['pets', 'age'])
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS_PATH(
  some_json_field, 'all', '$.pets', '$.age');
```

```
SELECT ... -- Oracle
WHERE (
  JSON_EXISTS(some_json_field, '$.pets') AND
  JSON_EXISTS(some_json_field, '$.age')
);
```

```
SELECT ... -- SQLite
WHERE (
  JSON_TYPE(some_json_field, '$.pets') IS NOT NULL AND
  JSON_TYPE(some_json_field, '$.age') IS NOT NULL
);
```

JSONField Lookups

Key existence

```
SELECT ... -- MySQL, MariaDB
WHERE (
  JSON_CONTAINS_PATH(some_json_field, 'one', '$.pets') AND
  JSON_CONTAINS_PATH(some_json_field, 'one', '$.age')
);
```

```
SELECT ... -- Oracle
WHERE (
  JSON_EXISTS(some_json_field, '$.pets') AND
  JSON_EXISTS(some_json_field, '$.age')
);
```

```
SELECT ... -- SQLite
WHERE (
  JSON_TYPE(some_json_field, '$.pets') IS NOT NULL AND
  JSON_TYPE(some_json_field, '$.age') IS NOT NULL
);
```

JSONField Lookups

Key existence

```
>>> MyModel.objects.filter(
        some_json_field__has_any_keys=['pets', 'age'])
```

```
SELECT ... -- PostgreSQL
WHERE some_json_field ?| {'pets', 'age'};
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS_PATH(
  some_json_field, 'one', '$.pets', '$.age');
```

JSONField Lookups

Key existence

```
>>> MyModel.objects.filter(
        some_json_field__has_any_keys=['pets', 'age'])
```

```
SELECT ... -- MySQL, MariaDB
WHERE JSON_CONTAINS_PATH(
  some_json_field, 'one', '$.pets', '$.age');
```

```
SELECT ... -- Oracle
WHERE (
  JSON_EXISTS(some_json_field, '$.pets') OR
  JSON_EXISTS(some_json_field, '$.age')
);
```

```
SELECT ... -- SQLite
WHERE (
  JSON_TYPE(some_json_field, '$.pets') IS NOT NULL OR
  JSON_TYPE(some_json_field, '$.age') IS NOT NULL
);
```

JSONField Lookups

Key existence

```sql
SELECT ... -- MySQL, MariaDB
WHERE (
  JSON_CONTAINS_PATH(some_json_field, 'one', '$.pets') OR
  JSON_CONTAINS_PATH(some_json_field, 'one', '$.age')
);
```

```sql
SELECT ... -- Oracle
WHERE (
  JSON_EXISTS(some_json_field, '$.pets') OR
  JSON_EXISTS(some_json_field, '$.age')
);
```

```sql
SELECT ... -- SQLite
WHERE (
  JSON_TYPE(some_json_field, '$.pets') IS NOT NULL OR
  JSON_TYPE(some_json_field, '$.age') IS NOT NULL
);
```

Where to go from here?

Where to go from here?

- Optimizations
- Implement unsupported lookups
- `JSON` schema validation

"I only use LTS 😢"

"I only use LTS 😀"

`django-jsonfield-backport` on PyPI

Thank you!

Thank you!

```
{
  "name": "Sage M. Abdullah",
  "username": "laymonage",
  "slides": {
    "hosted": "https://slides.laymonage.com/jsonfield",
    "source": "https://github.com/laymonage/slides-jsonfield"
  }
}
```

Image sources

- https://www.linkedin.com/school/university-of-indonesia/
- https://nypost.com/2020/07/01/friday-the-13th-villain-jason-pushes-mask-wearing-in-psa/