Jack Linke

@JackDLinke

jacklinke.com

Managing Director Watervize - SaaS for Irrigation

👬 Husband 🎤 Public Speaker

🧠 ND 🪖 Veteran 🌈 he/him

Home on the range with Django

Getting comfortable with ranges and range fields

https://github.com/jacklinke/home-on-the-range-with-django/

Disclaimers

- Topic might be challenging for beginners
  - (But exposure to these concepts early may help avoid pitfalls later)
- Postgres-specific

Goals

- Learn why ranges can be more useful than distinct start & end values
- Become familiar with range / interval terminology
- See a number of approaches to using & querying with ranges
- Have resources for further reading & learning

Real Life

- 🌡️ Daily Temperature highs/lows
- 🏠 Budget range for home renovation
- 🎹 Range of frequencies for each musical instrument
- ⛽ Range of gas prices in each region
- 🍲 Range of calories in fast food meals
- 🏈⚽ Range of goals/touchdowns for each team & season
- 🕐 Daily business hours
- 💰 Salary range for different jobs
- 🌱 Suitable soil pH values for different plants
- 🎉 Start & end date/datetime of Events
- 🚙 Typical min/max driving duration between locations

History

- First available in Postgres 9.2 (2011)
- First available in Django in 2015

Typical Approach

- Separate **`start`** and **`stop`** model fields
- Querying with start and stop values
- Quickly gets complicated

Model Constraint Comparison - separate fields

- 2 separate fields (DB doesn't know they are related)
- Have to build logic to ensure you don't end up with...
  - salary_range = $200,000 to $140,000 😬
  - event_dates = 2022/10/21 to 2022/10/16 🤔
- Constraint to prevent overlaps
  - Have to write an expression to trick the DB into treating 2 fields (and boundary) as a range type

Model Constraint Comparison - separate fields

```python
class TsTzRange(Func):
    function = "TSTZRANGE"
    output_field = DateTimeRangeField()


class LockerReservation(models.Model):
    period_start = models.DateTimeField()
    period_end = models.DateTimeField()
    ...

    class Meta:
        constraints = [
            # No Locker should have overlapping reservations
            ExclusionConstraint(
                name="exclude_overlap_locker_res",
                expressions=(
                    (
                        TsTzRange("period_start", "period_end", RangeBoundary()),
                        RangeOperators.OVERLAPS,
                    ),
                    ("locker", RangeOperators.EQUAL),
                ),
                condition=Q(cancelled=False),
            ),
            # Make sure the start value is less than or equal to the end value
            models.CheckConstraint(
                name="prevent_start_gt_end",
                check=~Q(period_start__lte=F("period_end")),
            ),
            ...,
        ]
```

Model Constraint Comparison - range field

- 1 field stores the lower, upper, and boundary information
  - DB knows these are all related
- By default, Postgres won't let you use larger value in lower position
  - salary_range = $140,000 to $200,000 🤑
  - event_dates = 2022/10/16 to 2022/10/21 😎

Model Constraint Comparison - range field

```python
class LockerReservation(auto_prefetch.Model):
    period = DateTimeRangeField()
    ...

    class Meta:
        constraints = [
            # No Locker should have overlapping reservations
            ExclusionConstraint(
                name="excl_overlap_locker_res",
                expressions=[
                    ("period", RangeOperators.OVERLAPS),
                    ("locker", RangeOperators.EQUAL),
                ],
                # Ignore overlaps where the reservation is cancelled
                condition=Q(cancelled=None),
            ),
            ...,
        ]
```

Query Comparison - separate fields

- Queries have to include both fields
- Often unintuitive in which order lookups are used

Query Comparison - separate fields

```
1 now = timezone.now()
2
3 next_12_hours_start = now
4 next_12_hours_end = now + timezone.timedelta(hours=12)
5
6 upcoming_events = Event.objects.filter(
7     event_start__lt=next_12_hours_end,
8     event_end__gte=next_12_hours_start
9 )
```

Query Comparison - separate fields

```
1 now = timezone.now()
2
3 next_12_hours_start = now
4 next_12_hours_end = now + timezone.timedelta(hours=12)
5
6 upcoming_events = Event.objects.filter(
7     event_start__lt=next_12_hours_end,
8     event_end__gte=next_12_hours_start
9 )
```

Query Comparison - range field

- Use a single, intuitive lookup

Query Comparison - range field

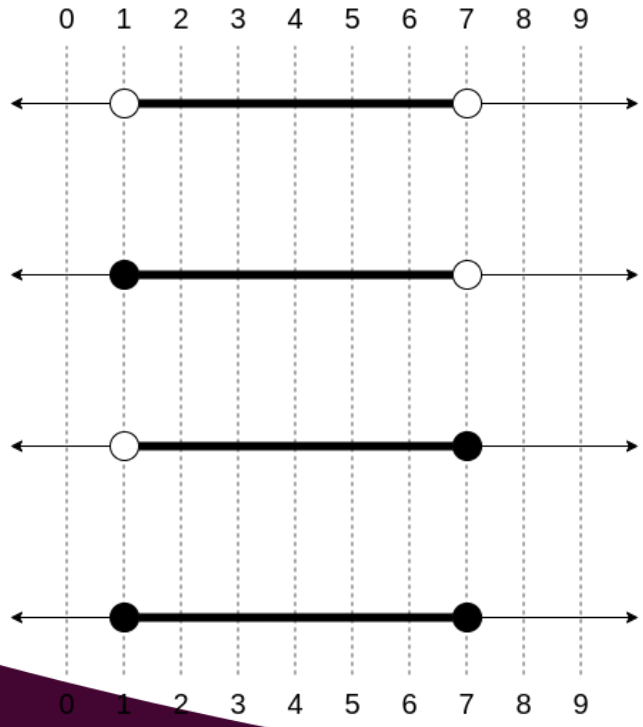
```
now = timezone.now()

next_12_hours = DateTimeTZRange(
    now,
    now + timezone.timedelta(hours=12)
)

upcoming_events = Event.objects.filter(
    event_period__overlap=next_12_hours
)
```

Going back to Elementary School (but no tests - I promise)

Terminology

- Ranges = Intervals
- Inclusive = Closed
- Exclusive = Open

Inclusive vs Exclusive



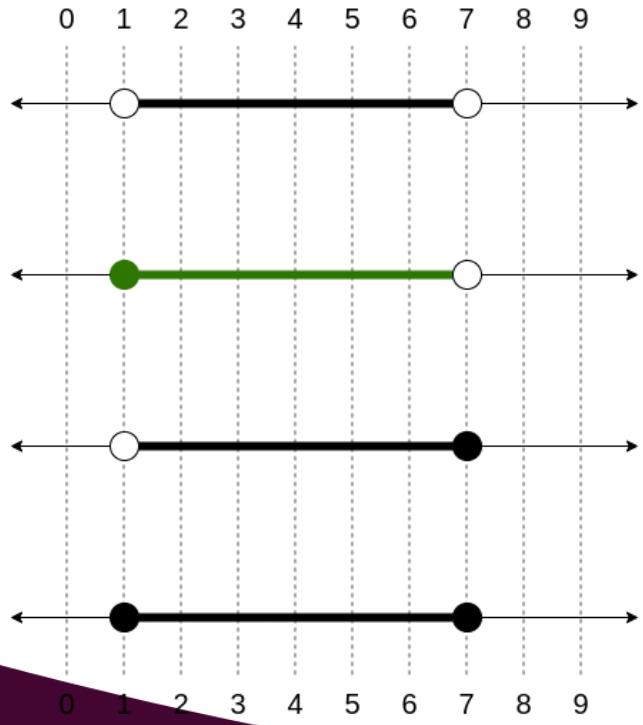
(1, 7)
(exclusive, exclusive)

[1, 7)
[inclusive, exclusive)

(1, 7]
(exclusive, inclusive]

[1, 7]
[inclusive, inclusive]

Inclusive vs Exclusive



(1, 7)
(exclusive, exclusive)

[1, 7)
[inclusive, exclusive)

(1, 7]
(exclusive, inclusive]

[1, 7]
[inclusive, inclusive]

Inclusive vs Exclusive

Another way to look at this

$$[1, 3] = 1 \leq x \leq 3$$

$$[1, 3) = 1 \leq x < 3$$

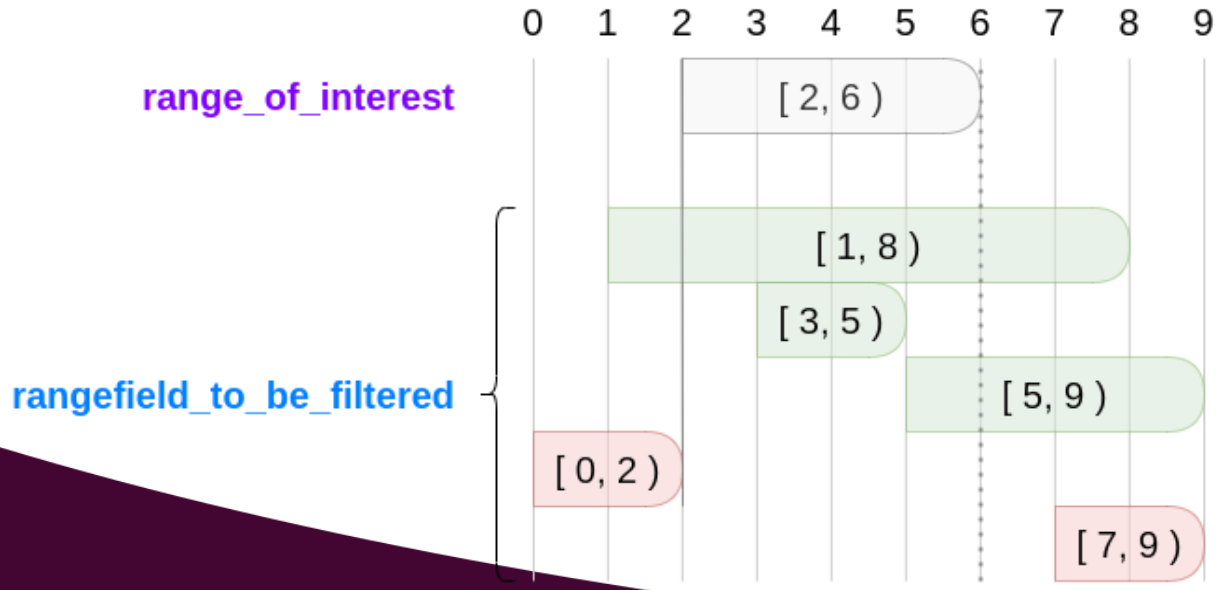
$$(1, 3] = 1 < x \leq 3$$

$$(1, 3) = 1 < x < 3$$

Range Overlaps *Range*

rangefield_to_be_filtered__overlap=range_of_interest

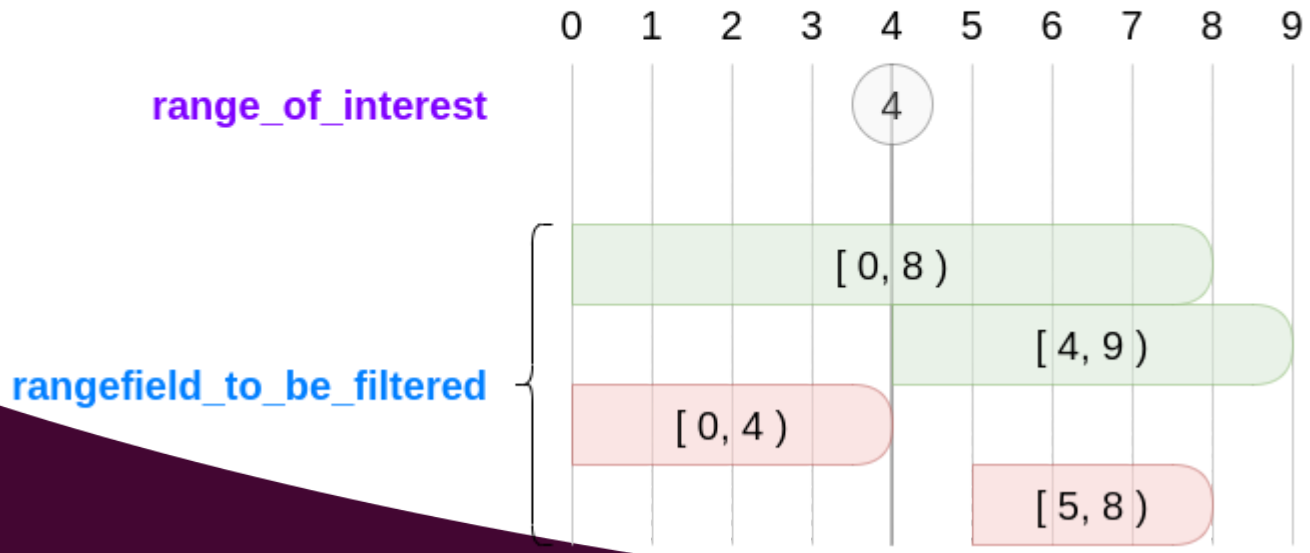
rangefield_to_be_filtered overlaps *range_of_interest (by any amount)*



Range Contains *Element*

rangefield_to_be_filtered__contains=element_of_interest

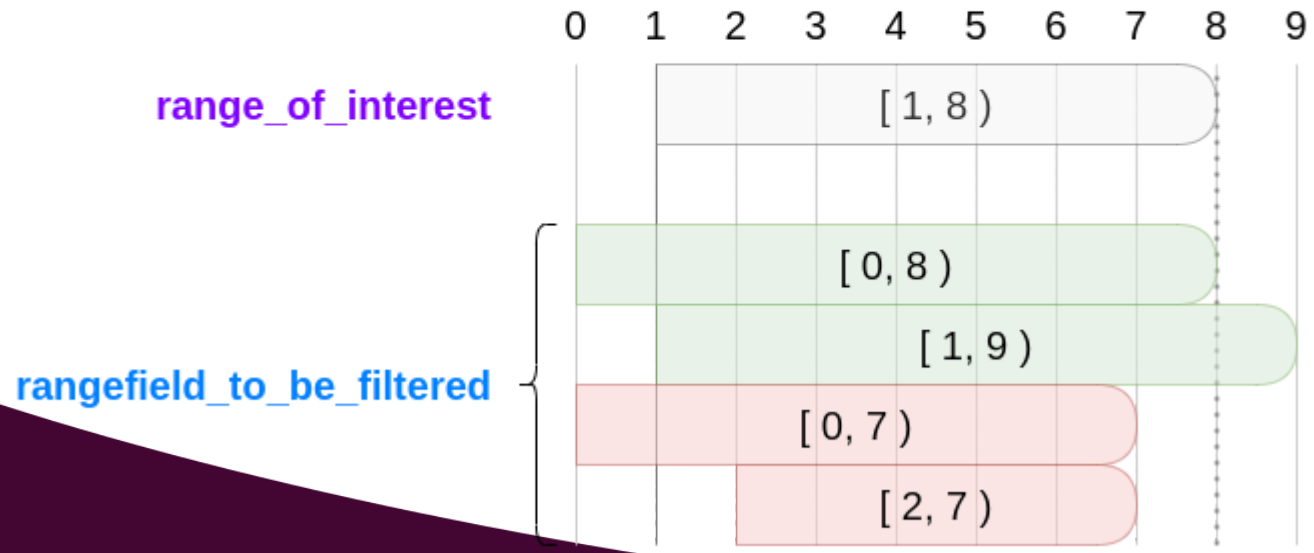
rangefield_to_be_filtered contains *element_of_interest*



Range Contains *Range*

rangefield_to_be_filtered__contains=range_of_interest

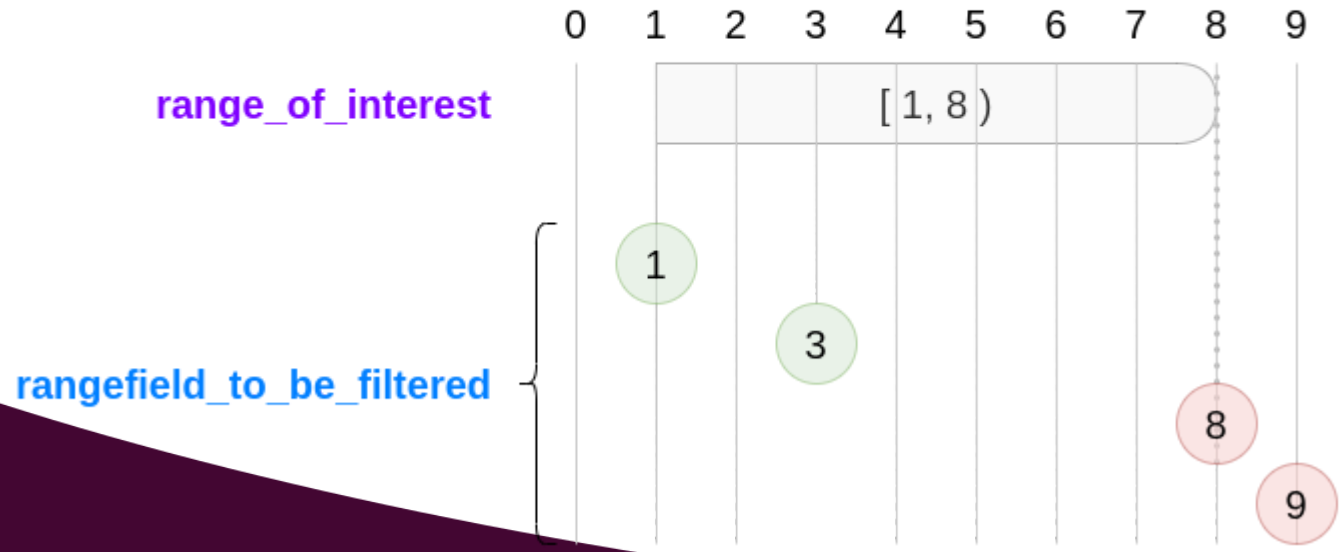
rangefield_to_be_filtered contains *range_of_interest*



Element Contained By *Range*

field_to_be_filtered__contained_by=range_of_interest

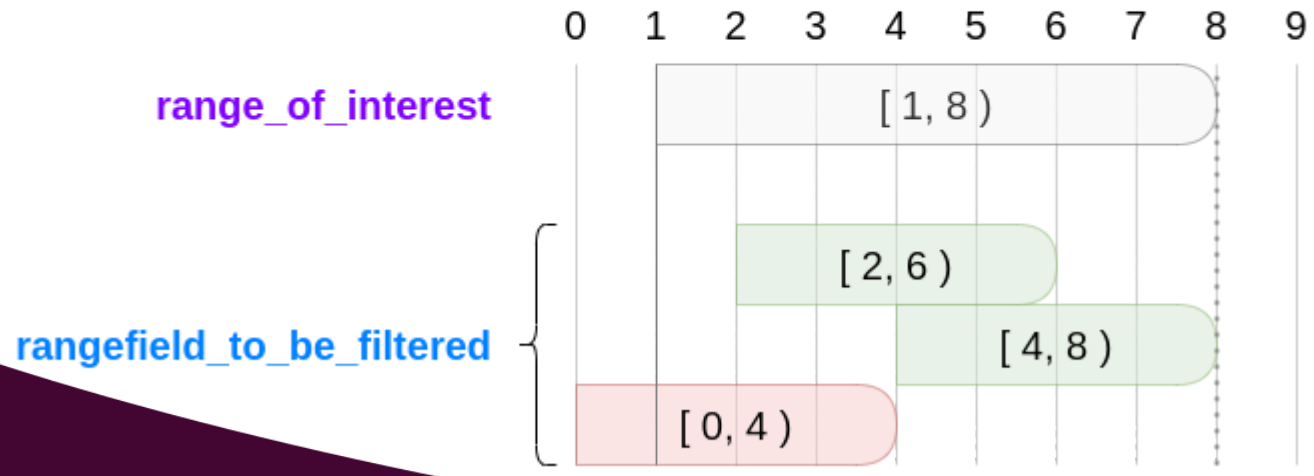
field_to_be_filtered is contained by *range_of_interest*



Range Contained By *Range*

rangefield_to_be_filtered__contained_by=range_of_interest

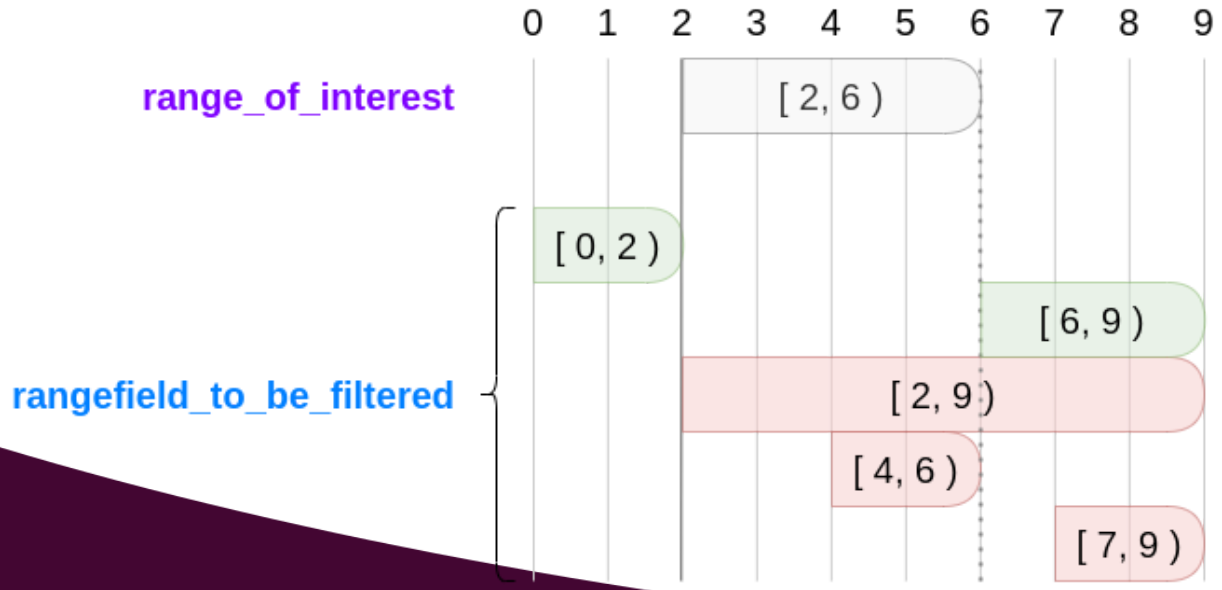
rangefield_to_be_filtered is contained by *range_of_interest*



Range Adjacent To *Range*

rangefield_to_be_filtered__adjacent_to=range_of_interest

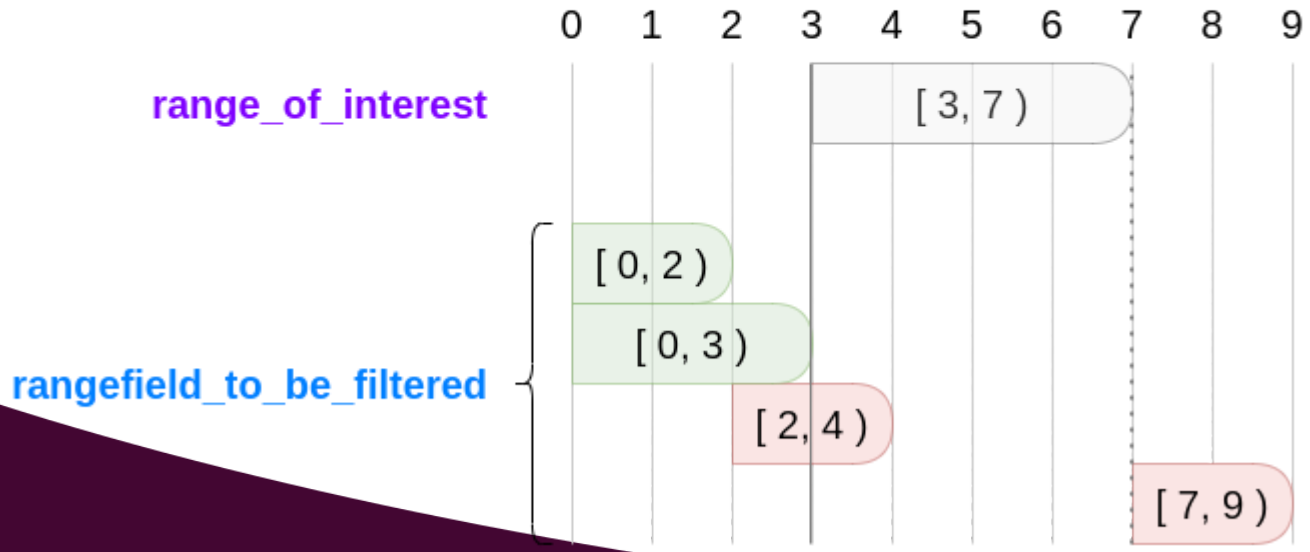
rangefield_to_be_filtered is immediately right or left of *range_of_interest*



Range Fully Less Than *Range*

rangefield_to_be_filtered__fully_lt=range_of_interest

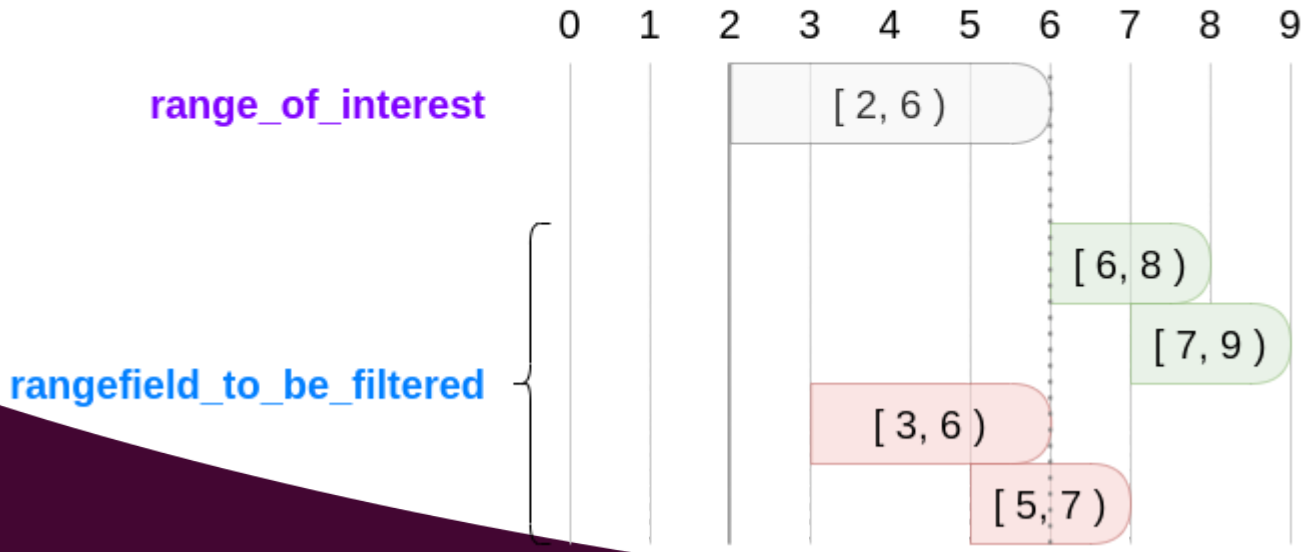
rangefield_to_be_filtered is completely to the left of *range_of_interest*



Range Fully Greater Than *Range*

rangefield_to_be_filtered__fully_gt=range_of_interest

rangefield_to_be_filtered is completely to the right of *range_of_interest*



psycopg2.extras

Provides tools for 'translation'

- NumericRange
  - Postgres: int4range, int8range, numrange
  - Django: IntegerRangeField, BigIntegerRangeField, DecimalRangeField
- DateRange
  - Postgres: daterange
  - Django: DateRangeField

psycopg2.extras (cont'd)

Provides tools for 'translation'

- DateTimeRange
  - Postgres: tsrange
  - Django: DateTimeRangeField (naive dt 💩)
- DateTimeTZRange
  - Postgres: tstzrange
  - Django: DateTimeRangeField

Defining Range Fields in Django

- Using psycopg2.extras classes
  - Bounds default to "[ )"
  - Can be changed for non-discrete range fields with `default_bounds`
  - (DateTimeRangeField and DecimalRangeField)
- Fields also accept tuples as input if no bounds information is necessary

```
1 my_field = IntegerRangeField()
2 my_field = IntegerRangeField(default=(1, 3))
3 my_field = IntegerRangeField(NumericRange(None, 7, "()"))
4 my_field = IntegerRangeField(default_bounds="[]")
```

Lookups based on range bounds

- **`startswith`**: filters on lower bound
- **`endswith`**: filters on upper bound
- **`isempty`**: filters on empty instances
- **`lower_inf`**: unbounded (infinite) or bounded lower bound
  - e.g.: `date.min`
- **`upper_inf`**: unbounded (infinite) or bounded upper bound
  - e.g.: `date.max`
- **`lower_inc`**: filters on inclusive or exclusive lower bounds
- **`upper_inc`**: filters on inclusive or exclusive upper bounds

Lookups based on range bounds

```python
from psycopg2.extras import NumericRange

# From -infinity to infinity
my_first_range = NumericRange(lower=None, upper=None, bounds='[)')

# From 1 (inclusive) to 5 (exclusive)
my_second_range = (1, 5)

# From 7 (inclusive) to infinity
my_third_range = NumericRange(lower=7, upper=None)


# In the model
class MyModel(models.Model):
    my_integer_range = IntegerRangeField()

# In a query
first = MyModel.objects.create(my_integer_range=my_first_object)
second = MyModel.objects.create(my_integer_range=my_second_range)
third = MyModel.objects.create(my_integer_range=my_third_range)
```

Lookups based on range bounds

```
my_first_range = NumericRange(lower=None, upper=None, bounds='[)')
my_second_range = (1, 5)
my_third_range = NumericRange(lower=7, upper=None)

first = MyModel.objects.create(my_integer_range=my_first_object)
second = MyModel.objects.create(my_integer_range=my_second_range)
third = MyModel.objects.create(my_integer_range=my_third_range)


MyModel.objects.filter(my_integer_range__startswith__gt=3)
# <QuerySet [<MyModel: my_third_range>]>

MyModel.objects.filter(my_integer_range__endswith=5)
# <QuerySet [<MyModel: my_second_range>]>
```

Creating your own range types

"every data type that supports a btree operator class (that is, a way of ordering any two given values) can be used to create a range type."

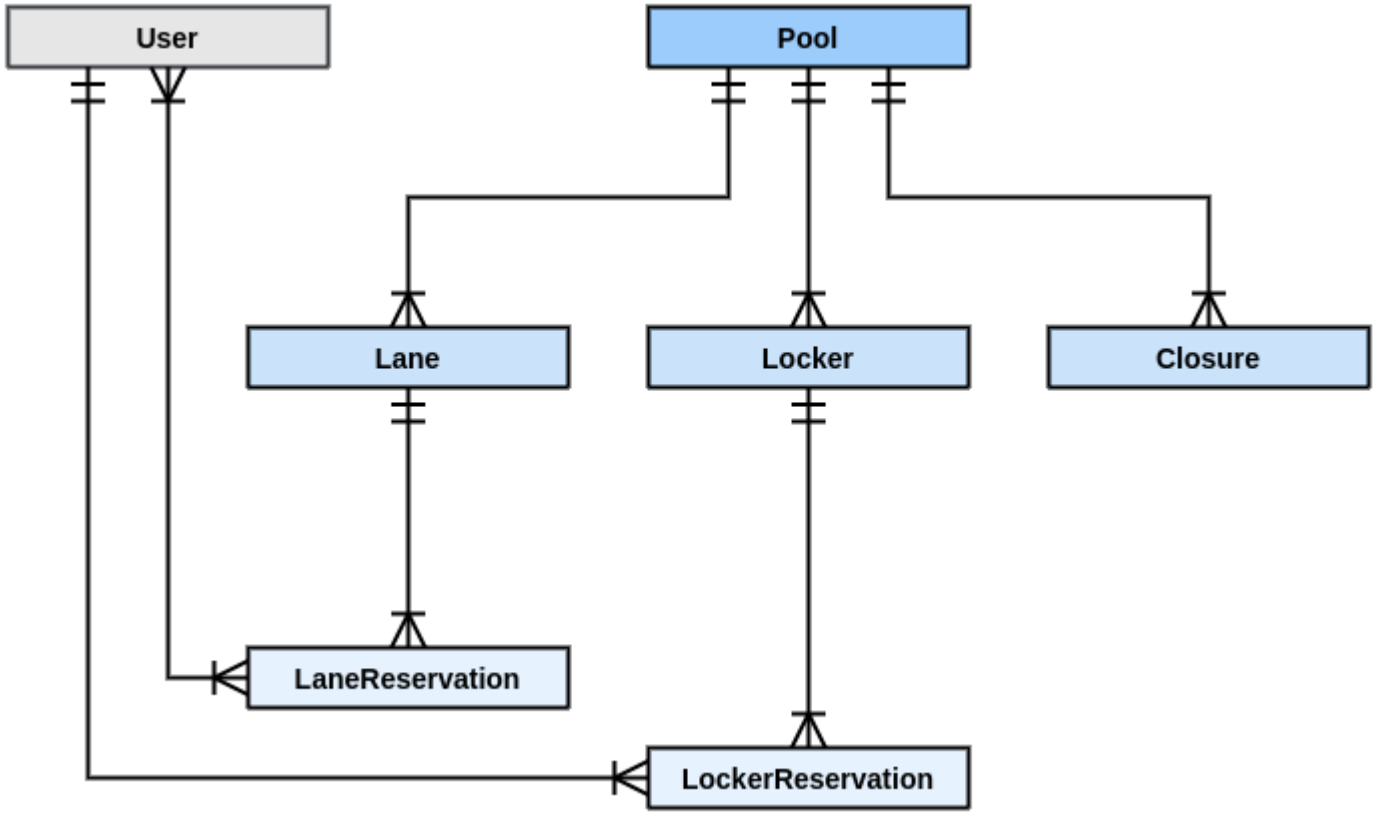
- PostgreSQL 14 Administration Cookbook

Example: GenericIPAddressField (postgres' `inet`) can be extended to create range of IP Addresses

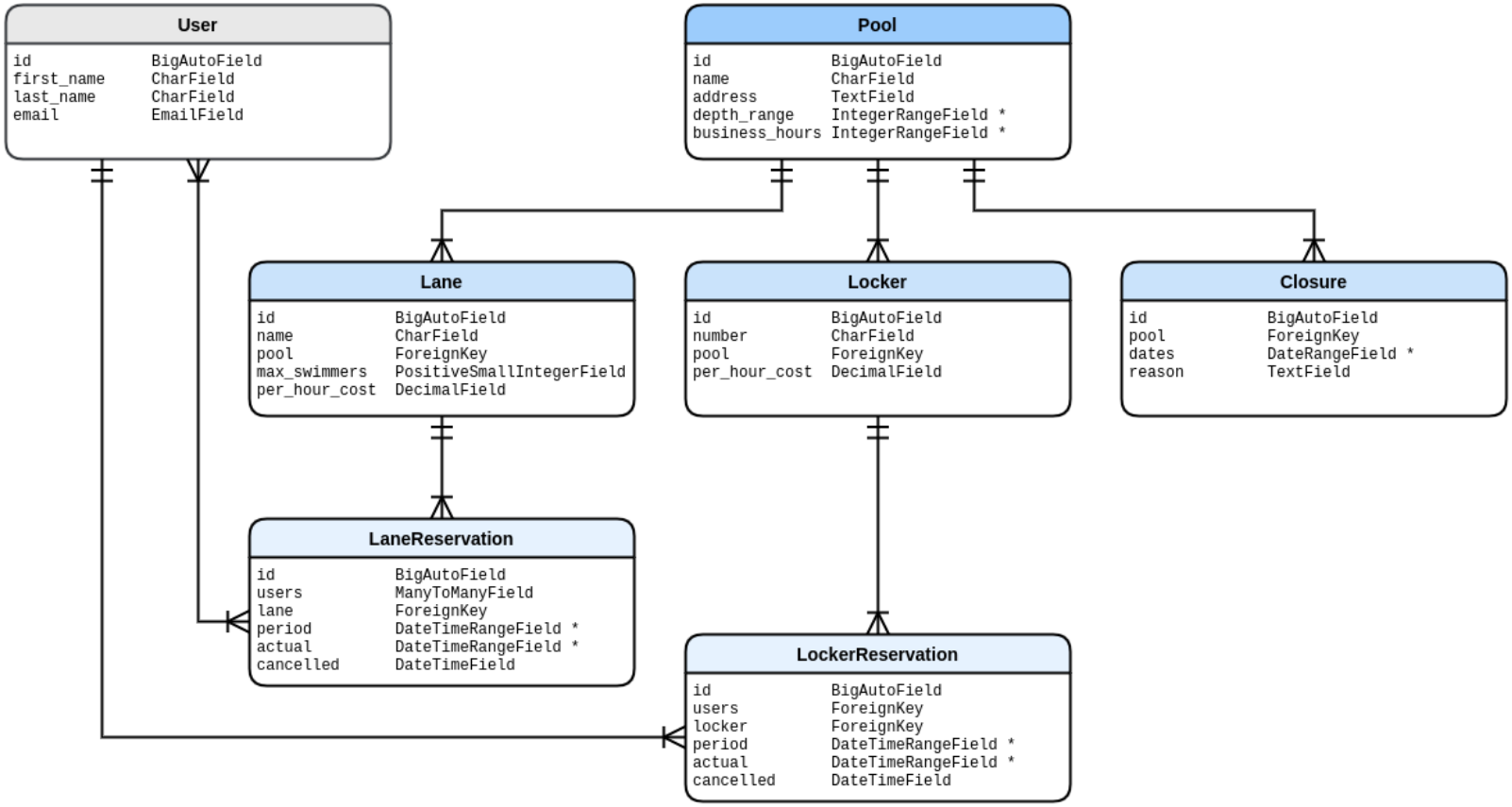
Indexing

Stick with GiST or B-Tree

Example Project



Example Project



Pitfalls

DateTimeRangeField in Admin

Reservation Period:	2022-10-15 11:00:00	2022-10-15 13:00:00
Actual Usage Period:		

Pitfalls

Trying to use `lower` and `upper` as callables

- They aren't.
- You can use `Lower` and `Upper` database functions in queries
- You can use `startswith` and `endswith` lookups

Pitfalls

Trying to use `F()` with the psycopg2.extra classes

They aren't Django classes! May have to use `Cast` to convert to a Django range field

Resources

- psycopg2.extras
- This talk & Example Project
- django-range-merge - use `range_merge` aggregate with Django
- django-generate-series - create sequences with Django's ORM

Final thought

A man works hard to name an interval equal to 24 hours...

...so he calls it a day.