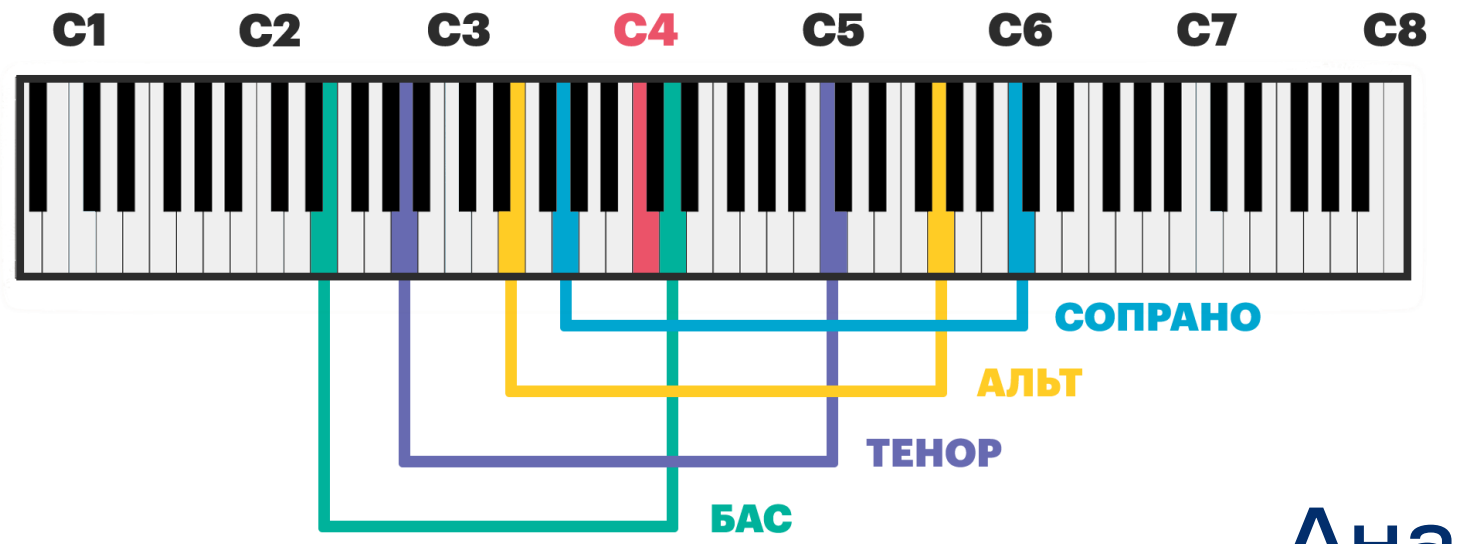
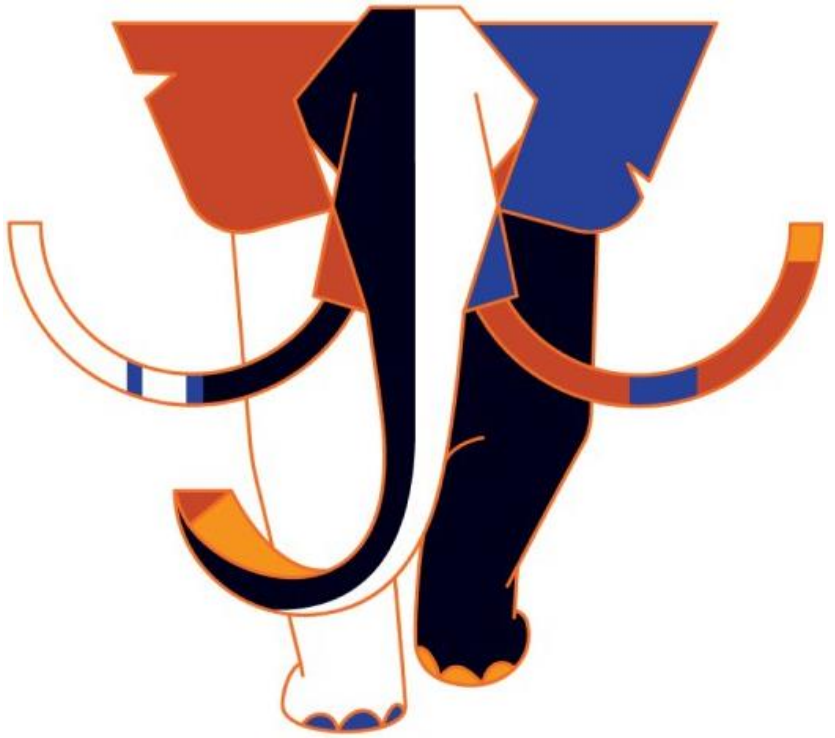
Диапазонные типы в Postgres

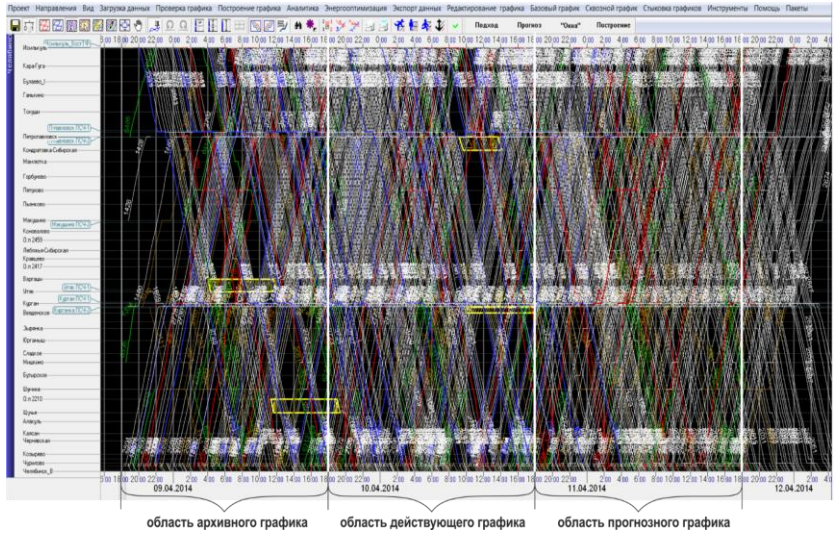




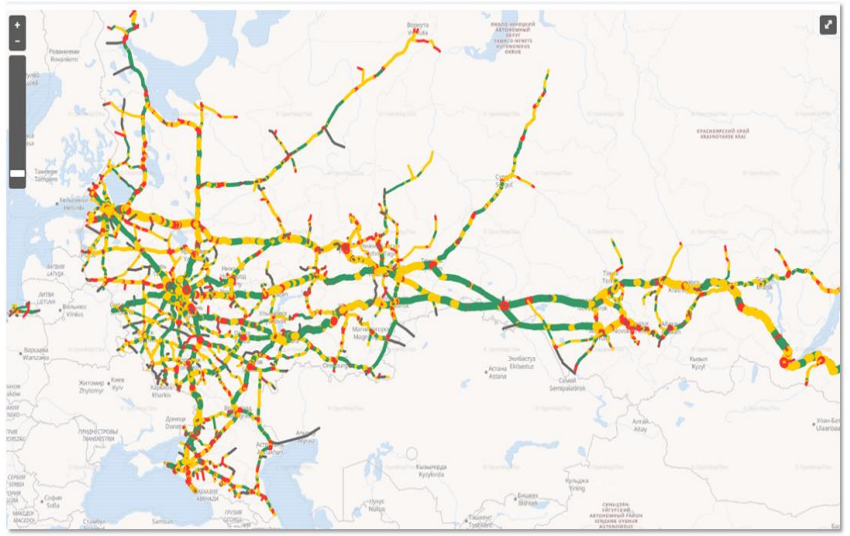
Анатолий Анфиногенов

Postgres и движение поездов

- **ЭЛЬБРУС** (2006 г. – н.в.) – система планирования движения грузовых поездов по энергооптимальным расписаниям. Родился в Oracle, переехал в Postgres.



- **ЭЛЬБРУС–М** (2020 г. – н.в.) – цифровая прогнозная макромодель движения поездопотоков. Изначально создана в Postgres; включает в себя БД ЭЛЬБРУС как подсистему.



Но говорить мы будем не об ЭЛЬБРУС

- В Postgres имеются возможности, о которых многие пришедшие из Oracle или из других СУБД даже не подозревают.
- При миграции новые возможности осваивать некогда.
- После успешного окончания переезда пора обживаться в Postgres по-настоящему!
- Цель этого доклада – возникновение мыслей «*Руки чешутся попробовать!*" и "*А что, так можно было?!*" у слушателей.

Практические задачи *

- **Календари событий** (бронирование объектов, календари пассажирских поездов);
- **Версионность объектов** (с гарантией (а) непересечения и (б) полного покрытия всего диапазона времени жизни объектов);
- **Работа с полубесконечными и бесконечными диапазонами** (по сравнению с использованием `infinity` или "волшебных пирожков").

* Задачи со звёздочкой частенько решаются громоздко

Немного бесконечности

- В Postgres для числовых и временны**х** типов существуют специальные значения бесконечности **infinity** и **-infinity**

```
CREATE TABLE T(N INTEGER,
                  DT TIMESTAMP);

INSERT INTO T(N, DT)
VALUES (-1,'-infinity')
      ,( 0, now()     )
      ,( 1, 'infinity');

SELECT * FROM T ORDER BY DT DESC;
```

n	dt
1	infinity
0	2024-03-14 10:11:04
-1	-infinity

Скажем НЕТ волшебным пирожкам!

- **Задача** – время жизни объектов [Valid_From, Valid_To];
  чему равен Valid_To у активных объектов?

Бесконечность	Волшебный пирожок	NULL как бесконечность
`'infinity'::TIMESTAMP`	`'5683.01.23 02:07:00'`	`NULL::TIMESTAMP`

```
SELECT 'infinity'::TIMESTAMP                AS Good
      ,'5683.01.23 02:07:00'::TIMESTAMP     AS Bad
      , NULL::TIMESTAMP                     AS Ugly;
```

```
|good     |bad                      |ugly     |
|---------|-------------------------|---------|
|infinity|5683-01-23 02:07:00.000|null     |
```

Пример – задача версионности объекта

1. Смоделируем историю развития станций с течением времени – как менялось количество путей на них

```
CREATE TABLE IF NOT EXISTS Stations
(
 St_Name     text
,N_Ways      integer
,Valid_From  date
,Valid_To    date
,CONSTRAINT Stations0_PK PRIMARY KEY(St_Name, Valid_From)
);

COMMENT ON TABLE  Stations              IS 'Станции в развитии во времени';
COMMENT ON COLUMN Stations.St_Name      IS 'Название станции';
COMMENT ON COLUMN Stations.N_Ways       IS 'Количество путей';
COMMENT ON COLUMN Stations.Valid_From IS 'Дата начала действия версии';
COMMENT ON COLUMN Stations.Valid_To    IS 'Дата конца действия версии';
```

Пример – задача версионности объекта

2. Добавим одну станцию – Бирюлёво, которая построена в 1900 году, а после революции на ней было вдвое увеличено число путей

```
INSERT INTO Stations(St_Name, N_Ways, Valid_From, Valid_To) VALUES
('Бирюлёво',10,'1900-01-01','1917-11-07'    ),
('Бирюлёво',20,'1917-11-07','infinity'::date);
```

```
SELECT * FROM Stations ORDER BY Valid_From;
```

st_name	n_ways	valid_from	valid_to
Бирюлёво	10	1900-01-01	1917-11-07
Бирюлёво	20	1917-11-07	infinity

Пример – задача версионности объекта

3. Запросим действующую версию станции Бирюлёво и обратим внимание на план выполнения запроса

```
SELECT * FROM Stations
WHERE St_Name = 'Бирюлёво'
  AND CURRENT_DATE >= Valid_From
  AND CURRENT_DATE <= Valid_To;
```

```
|st_name |n_ways|valid_from|valid_to  |
|--------|------|----------|----------|
|Бирюлёво|20    |1917-11-07|infinity  |
```

```
EXPLAIN (costs OFF ) SELECT * FROM Stations …
```

```
|QUERY PLAN                                                                    |
|------------------------------------------------------------------------------|
|Bitmap Heap Scan on stations                                                  |
| Recheck Cond: (st_name = 'Бирюлёво'::text)                                   |
| Filter: ((CURRENT_DATE >= valid_from) AND (CURRENT_DATE <= valid_to))        |
| -> Bitmap Index Scan on stations_pk                                          |
| Index Cond: ((st_name = 'Бирюлёво'::text) AND (valid_from <= CURRENT_DATE))  |
```

Пример – задача версионности объекта

4. Добавим новую версию Бирюлёво, сделав неактивной предыдущую версию этой станции, но опечатавшись и допустив перекрытие диапазонов времени.

```
UPDATE Stations
SET    Valid_To = '2024-04-16'::date -- Опечатка, «4» вместо «3»!!!
WHERE  St_Name = 'Бирюлёво'
   AND CURRENT_DATE >= Valid_From
   AND CURRENT_DATE <= Valid_To;

INSERT INTO Stations(St_Name, N_Ways, Valid_From, Valid_To) VALUES
('Бирюлёво',30, CURRENT_DATE, 'infinity'::date);
```

* Помимо опечатки тут еще и неатомарная операция. Не делайте так!!!

Пример – задача версионности объекта

5. Запрос действующей версии станции Бирюлёво возвращает две строки. Ваше приложение разрушено, причём сразу это можно и не заметить!

```
SELECT * FROM Stations
WHERE  St_Name = 'Бирюлёво'
   AND CURRENT_DATE >= Valid_From
   AND CURRENT_DATE <= Valid_To;
```

```
|st_name |n_ways|valid_from|valid_to  |
|--------|------|----------|----------|
|Бирюлёво|20    |1917-11-07|2024-04-16|
|Бирюлёво|30    |2024-03-17|infinity  |
```

* Если опечататься в другую сторону, то не вернется ни одной строки. Тоже нехорошо получится.

Пример – задача версионности объекта

6. Предотвратить это можно только контролем непересечения диапазонов времени.

   Контроль непересечения диапазонов в приведенной выше реализации должен выполняться средствами приложения, а не БД, поэтому может не сработать.

   Применение диапазонных типов позволяет эффективно и надёжно решить задачу поддержания версионности объектов.

Определения диапазонных типов

- **Диапазонный тип** – диапазон значений некоторого типа данных (подтипа диапазона);

- **Мультидиапазонный тип** – упорядоченный список несмежных, непустых и отличных от NULL диапазонов

* Мультидиапазон доступен в версии Postgres 14 и старше

Встроенные диапазонные типы

Диапазон	Подтип	Мультидиапазон
int4range	integer	int4multirange
int8range	bigint	int8multirange
numrange	numeric	nummultirange
tsrange	timestamp without time zone	tsmultirange
tstzrange	timestamp with time zone	tstzmultirange
daterange	date	datemultirange

Пользователи могут определять собственные диапазонные типы!

Примеры задания диапазонов

Простые случаи

```
SELECT '[0,5)'::int4range  AS "Полуоткрытый"
      ,'[0,4]'::int4range  AS "Закрытый"
      ,'(-1,5)'::int4range AS "Открытый";
```

```
|Полуоткрытый|Закрытый|Открытый|
|------------|--------|--------|
|[0,5)       |[0,5)   |[0,5)   |
```

Особые случаи

```
SELECT '[5,5)'::int4range AS "Пустой"
      ,'(,4]'::int4range  AS "Полубесконечный"
      ,'(,)'::int4range   AS "Бесконечный";
```

```
|Пустой|Полубесконечный|Бесконечный|
|------|---------------|-----------|
|empty |(,5)           |(,)        |
```

Примеры задания мультидиапазонов

Простые случаи

```
SELECT '{[0,5)}'::int4multirange          AS "Простой"
      ,'{[0,5), [6,9)}'::int4multirange AS "Мульти"
      ,'{[0,7), [6,9)}'::int4multirange AS "Объедин.";
```

```
|Простой|Мульти         |Объедин.|
|-------|---------------|--------|
|{[0,5)}|{[0,5),[6,9)}|{[0,9)} |
```

Особые случаи

```
SELECT '{}'::int4multirange                    AS "Пуст.1"
      ,'{[5,5), (4,5)}'::int4multirange      AS "Пуст.2"
      ,'{(,5), [5,)}'::int4multirange        AS "Беск.1"
      ,'{(,), [0,5), [6,9)}'::int4multirange AS "Беск.2";
```

```
|Пуст.1|Пуст.2|Беск.1|Беск.2|
|------|------|------|------|
|{}    |{}    |{(,)} |{(,)} |
```

Дискретные и непрерывные диапазоны

- **Дискретные** – диапазон, для подтипа которого однозначно определены понятия «следующего» и «предыдущего» элемента для каждого значения (`int4range, int8range, daterange`).
- **Непрерывные** – диапазон, для подтипа которого всегда (или почти всегда) можно найти ещё одно значение между двумя несовпадающими значениями (`numrange, tsrange, tstzrange`).

Канонизация дискретных диапазонов

- **Канонизация** – преобразование равнозначных дискретных диапазонов к единственному представлению.
- По умолчанию `(,)`, `[,]`, `(,]` -> `[,)`
- Если функция канонизации не определена, диапазоны с различным определением будут всегда считаться разными, даже когда они на самом деле представляют одно множество значений.

Конструкторы для диапазонов

Простые случаи

```
SELECT int4range(0, 5, '[)') AS "Полуоткрытый"
      ,int4range(0, 4, '[]') AS "Закрытый"
      ,int4range(-1,5, '()') AS "Открытый";
```

```
|Полуоткрытый|Закрытый|Открытый|
|------------|--------|--------|
|[0,5)       |[0,5)   |[0,5)   |
```

Особые случаи

```
SELECT int4range(   5,   5, '[)') AS "Пустой"
      ,int4range(NULL,   4, '(]') AS "Полубесконечный"
      ,int4range(NULL,NULL, '()') AS "Бесконечный";
```

```
|Пустой|Полубесконечный|Бесконечный|
|------|---------------|-----------|
|empty |(,5)           |(,)        |
```

Конструкторы для мультидиапазонов


```
SELECT int4multirange('[0,4]'::int4range)                                  AS "Простой"
      ,int4multirange('[0,5)'::int4range,'[6,9)'::int4range) AS "Мульти"
      ,int4multirange()                                                    AS "Пустой"
      ,int4multirange('(,5)'::int4range,'[5,)'::int4range)   AS "Беск."
;
```

```
|Простой|Мульти        |Пустой|Беск.|
|-------|--------------|------|-----|
|{[0,5)}|{[0,5),[6,9)}|{}    |{(,)}|
```

Операции с диапазонами и мультидиапазонами

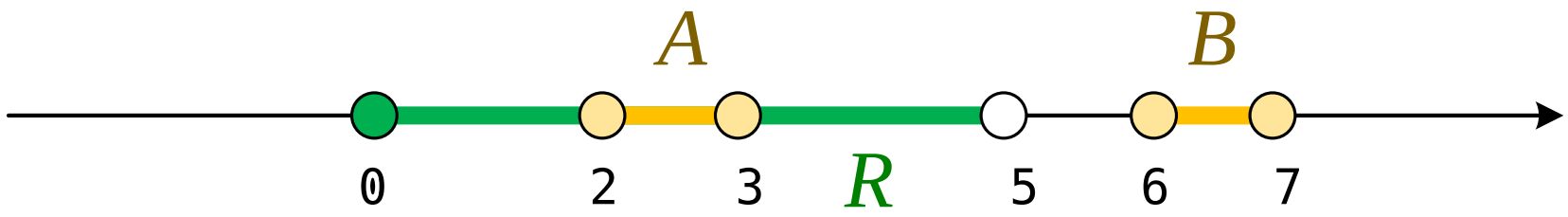
1. Проверка принадлежности элемента x диапазону R, т. е. $x \in R$ – операторы <@, @>


```
SELECT int4range(0,5)       AS "Диапазон"
      ,2 <@ int4range(0,5) AS "x1 ∈ R"
      ,int4range(0,5) @> 6 AS "x2 ∈ R";
```

```
|Диапазон|x1 ∈ R|x2 ∈ R|
|--------|------|------|
|[0,5)   |true  |false |
```

Операции с диапазонами и мультидиапазонами

2. Проверка принадлежности диапазона A диапазону R, т. е. $A \subset R$ – операторы <@, @>

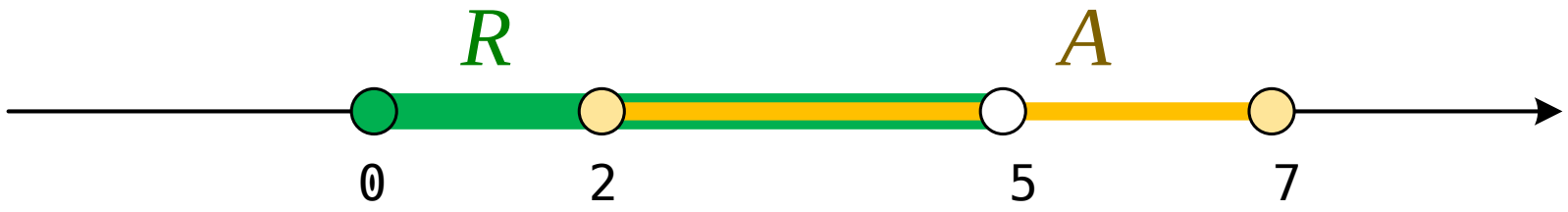



```
SELECT int4range(0,5)       AS "Диапазон R"
      ,int4range(2,3,'[]') AS "Диапазон A"
      ,int4range(6,7,'[]') AS "Диапазон B"
      ,int4range(2,3,'[]') <@ int4range(0,5) AS "A ⊂ R"
      ,int4range(0,5) @> int4range(6,7,'[]') AS "B ⊂ R";
```

```
|Диапазон R|Диапазон A|Диапазон B|A ⊂ R |B ⊂ R|
|----------|----------|----------|------|------|
|[0,5)     |[2,4)     |[6,8)     |true  |false |
```

Операции с диапазонами и мультидиапазонами

3. Объединение и пересечение диапазонов A и R, т. е. $A \cup R$ – оператор **+**, $A \cap R$ – оператор *****

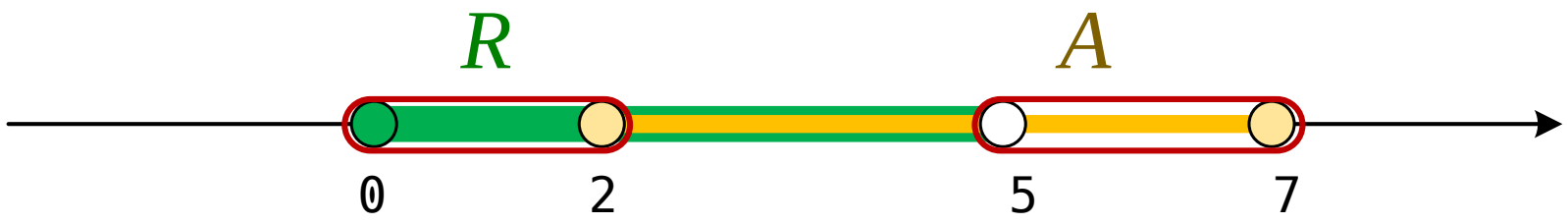



```
SELECT int4range(0,5)       AS "Диапазон R"
      ,int4range(2,7,'[]') AS "Диапазон A"
      ,int4range(0,5) + int4range(2,7,'[]') AS "A ∪ R"
      ,int4range(0,5) * int4range(2,7,'[]') AS "A ∩ R";
```

```
|Диапазон R|Диапазон A|A ∪ R|A ∩ R|
|----------|----------|-----|-----|
|[0,5)     |[2,8)     |[0,8)|[2,5)|
```

Операции с диапазонами и мультидиапазонами

4. Разность диапазонов R и A, т. е. $R \setminus A$ – оператор **-**




```
SELECT int4range(0,5)       AS "Диапазон R"
      ,int4range(2,7,'[]') AS "Диапазон A"
      ,int4range(0,5) - int4range(2,7,'[]') AS "R \ A"
      ,int4range(2,7,'[]') - int4range(0,5) AS "A \ R";
```

```
|Диапазон R|Диапазон A|A \ R|R \ A|
|----------|----------|-----|-----|
|[0,5)     |[2,8)     |[0,2)|[5,8)|
```

5. Логические операции для диапазонов

Оператор	Смысл оператора	Пример оператора
R && A	Диапазоны пересекаются (у A и R есть общие элементы)?	`int4range(3,5) && int4range(2,8) → t`
R << A	Диапазон R располагается строго слева от диапазона A?	`int4range(3,5) << int4range(6,8) → t`
R >> A	Диапазон R располагается строго справа от диапазона A?	`int4range(6,8) >> int4range(3,5) → t`
R &< A	Диапазон R не простирается правее диапазона A?	`int4range(0,6) &< int4range(2,6) → t`
R &> A	Диапазон R не простирается левее диапазона A?	`int4range(2,6) &> int4range(0,6) → t`
R -|- A	Диапазоны R и A примыкают друг к другу?	`numrange(1.1,2.2) -|- numrange(2.2,3.3) → t`

Операции с диапазонами и мультидиапазонами

6. Операции для мультидиапазонов – те же, что и для диапазонов; подробнее – в документации

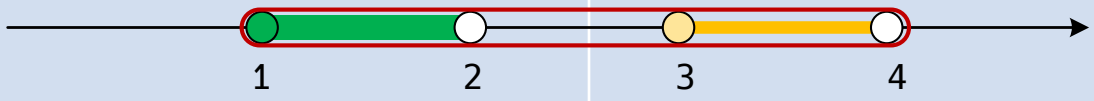
Посмотрим на наш «волшебный пирожок» ♨ :

```
SELECT tsrange( '5683.01.23 02:07:00'::TIMESTAMP
      ,'infinity'::TIMESTAMP)                              AS "♨ - Inf"
      ,tsrange( '5683.01.23 02:07:00'::TIMESTAMP, NULL) AS "♨ - ∞"
      , tsmultirange( tsrange('5683.01.23 02:07:00'::TIMESTAMP, NULL))
      - tsmultirange( tsrange('5683.01.23 02:07:00'::TIMESTAMP
                     ,'infinity'::TIMESTAMP))              AS "Inf - ∞";
```

♨ - Inf	♨ - ∞	Inf - ∞
["5683-01-23 02:07:00",infinity)	["5683-01-23 02:07:00",)	{[infinity,)}

Сюрприз: за бесконечностью есть еще диапазон!

7. Функции для диапазонов

Функция	Описание функции	Пример вызова
`lower(R)`, `upper(R)`	Выдаёт нижнюю/верхнюю границу диапазона R (`NULL` для пустого диапазона или бесконечной границы)	`lower(numrange(1.1, 2.2)) → 1.1` `upper(numrange(1.1, 2.2)) → 2.2`
`isempty(R)`	Диапазон R пуст?	`isempty(numrange(1.1,2.2)) → f`
`lower_inc(R)`, `upper_inc(R)`	Нижняя/верхняя граница диапазона R включена в него?	`lower_inc(numrange(1.1, 2.2)) → t` `upper_inc(numrange(1.1, 2.2)) → f`
`lower_inf(R)`, `upper_inf(R)`	У диапазона R нет нижней/верхней границы? Для `+/-Infinity` возвращается `false`, т.е. `+/-Infinity` не считается бесконечностью(!).	`lower_inf('(,)'::daterange) → t` `upper_inf('(,)'::daterange) → t`
`range_merge(A,B) → R`	Вычисляет наименьший диапазон R, включающий оба диапазона A и B. 	`range_merge('[1,2)'::int4range, '[3,4)'::int4range) → [1,4)`

8. Проверим, как обстоят дела с upper_inf( )

```
SELECT upper(daterange(NULL,NULL))                AS "upper(NULL)"
,upper(daterange(NULL,'infinity'::DATE))          AS "upper(Inf)"
,upper_inf(daterange(NULL,NULL))                  AS "upper_inf(NULL)"
,upper_inf(daterange(NULL,'infinity'::DATE))      AS "upper_inf(Inf)"
,COALESCE(upper(daterange(NULL,NULL))
      ,'infinity'::DATE) >= 'infinity'::DATE AS "Вместо upper_inf()";
```

```
|upper(NULL)|Upper(Inf)|upper_inf(NULL)|upper_inf(Inf)|Вместо upper_inf()|
|-----------|----------|---------------|--------------|------------------|
|NULL       |infinity  |true           |false         |true              |
```

Вывод: формулировать условия бесконечной границы надо аккуратно

9. Функции для мультидиапазонов

Функция	Описание функции	Пример вызова

`lower(M)`, `upper(M)`, `isempty(M)`, `lower_inc(M)`, `upper_inc(M)` `lower_inf(M)`, `upper_inf(M)`	То же самое, что для диапазонов, с заменой на мультидиапазоны M диапазонов R в качестве входных параметров	`lower('{[1.1,2.2)}'::nummultirange) → 1.1`
		`upper('{[1.1,2.2)}'::nummultirange) → 2.2`
		`isempty('{[1.1,2.2)}'::nummultirange) → f`
		`lower_inc('{[1.1,2.2)}'::nummultirange) → t`
		`upper_inc('{[1.1,2.2)}'::nummultirange) → f`
		`lower_inf('{(,)}'::datemultirange) → t`
		`upper_inf('{(,)}'::datemultirange) → t`

`range_merge(M)` `→ R`	Вычисляет наименьший диапазон R, включающий весь мультидиапазон M	`range_merge('{[1,2), [3,4)}'::int4multirange) → [1,4)`

`unnest(M)`	Разворачивает мультидиапазон M в набор диапазонов. Диапазоны читаются в порядке хранения (восходящем)	`unnest('{[1,2), [3,4)}'::int4multirange) →`
		`[1,2)`
		`[3,4)`

10. Агрегатные функции для диапазонов

! В документации в разделе **8.17. Диапазонные типы** вообще не сказано о том, что бывают еще и агрегатные функции для диапазонов!

https://postgrespro.ru/docs/postgresql/16/rangetypes

Агрегатные функции для диапазонов описаны в разделе **9.21. Агрегатные функции**

https://postgrespro.ru/docs/postgresql/16/functions-aggregate

Предложение: добавить упоминание агрегатных функций в п. 8.17

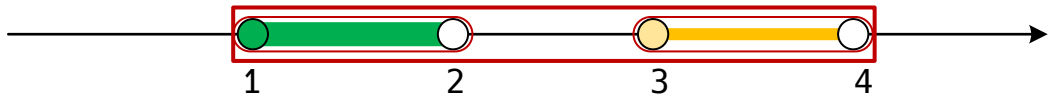
11. Агрегатные функции для диапазонов – список

Функция	Описание функции	Версия Postgres
`range_agg(TABLE(R)) → M`	Вычисляет объединение всех входных значений диапазонов, отличных от NULL.	14
`range_agg(TABLE(M)) → M`	Вычисляет объединение всех входных значений мультидиапазонов, отличных от NULL.	14
`range_intersect_agg (TABLE(R)) → R`	Вычисляет пересечение всех входных значений диапазонов, отличных от NULL. Выходной тип – диапазон!	14
`range_intersect_agg (TABLE(M)) → M`	Вычисляет пересечение всех входных значений мультидиапазонов, отличных от NULL.	15

12. Примеры вызова агрегатных функций

```
WITH CTE_R(R) AS (VALUES ('[1,2)'::int4range),('[3,4)'::int4range)),
CTE_M(M) AS (VALUES ('{[1,2)}'::int4multirange),('{[3,4)}'::int4multirange))
SELECT range_agg(R) AS "U диапазонов"
      ,range_agg(M) AS "U мультидиапазонов"
FROM CTE_R CROSS JOIN CTE_M;
```

```
|U диапазонов |U мультидиапазонов|
|-------------|------------------|
|{[1,2),[3,4)}|{[1,2),[3,4)}     |
```




```
WITH CTE_R(R) AS (VALUES ('[1,6)'::int4range),('[3,9)'::int4range)),
CTE_M(M) AS (VALUES ('{[1,6)}'::int4multirange),('{[3,9)}'::int4multirange))
SELECT range_intersect_agg(R) AS "∩ диапазонов"
      ,range_intersect_agg(M) AS "∩ мультидиапазонов"
FROM CTE_R CROSS JOIN CTE_M;
```

```
|∩ диапазонов|∩ мультидиапазонов|
|------------|------------------|
|[3,6)       |{[3,6)}           |
```

Несколько слов о GiST

- Многие полагают, что GiST – это про геоданные или полнотекстовый поиск.

- Мы собираемся использовать диапазонные типы в условиях `WHERE` и в `CONSTRAINTS`, поэтому по ним должны быть эффективные индексы.

- Привычные B-tree-индексы для этого не годятся, они поддерживают только операторы «больше», «меньше» и «равно». На вопросы вхождения в диапазон (`<@`) и т.п. они не отвечают.

Где почитать о GiST

1) Егор Рогов «PostgreSQL 15 изнутри»

https://postgrespro.ru/education/books/internals

Но в книге не рассмотрен GiST применительно к диапазонным типам.

2) Егор Рогов «Индексы в PostgreSQL — 5»

https://habr.com/ru/companies/postgrespro/articles/333878/

Более ранняя работа, в которой подробно рассмотрен GiST применительно к диапазонным типам.

Особенности индексного метода GiST

- Индексный метод GiST позволяет задать способ распределения данных произвольного типа по сбалансированному дереву и использования этого представления для доступа по некоторому оператору.
- Должна быть задана **функция согласованности** (`consistent`), которая вызывается для каждой индексной записи и определяет, «согласуется» ли предикат записи с условием поиска и определяет, надо ли спускаться в соответствующее поддерево и подходит ли листовая запись.

Пример – решение задачи бронирования

1. Можно ли забронировать домик на заданные даты

```
CREATE TABLE Reservations(Busy daterange);

INSERT INTO Reservations(Busy) VALUES
('[2024-01-01, 2024-01-30)'),
('[2024-02-01, 2024-02-28)'),
('[2024-03-01, 2024-03-30)');


SELECT * FROM Reservations
WHERE Busy && '[2024-02-15, 2024-04-01)';
```

```
|busy                    |
|------------------------|
|[2024-02-01,2024-02-28)|
|[2024-03-01,2024-03-30)|
```

* Пример создан на основе примера в статье (2) Егора Рогова; для наглядности и простоты взят тип `daterange` вместо `tsrange`

Пример – решение задачи бронирования

2. Без индекса поиск вызывает полный перебор

```
EXPLAIN (costs OFF ) SELECT * FROM Reservations
WHERE Busy && '[2024-02-15, 2024-04-01)';
```

```
|QUERY PLAN                                                    |
|--------------------------------------------------------------|
|Seq Scan on reservations                                      |
| Filter: (busy && '[2024-02-15,2024-04-01)'::daterange)       |
```

3. С индексом становится хорошо

```
CREATE INDEX ON Reservations USING GiST(Busy);

EXPLAIN (costs OFF ) SELECT * FROM Reservations
WHERE Busy && '[2024-02-15, 2024-04-01)';
```

```
|QUERY PLAN                                                                |
|--------------------------------------------------------------------------|
|Index Only Scan using reservations_busy_idx on reservations               |
| Index Cond: (busy && '[2024-02-15,2024-04-01)'::daterange)               |
```

Ограничение исключения (EXCLUDE)

- Ограничение исключения EXCLUDE гарантирует, что заданные поля любых двух строк таблицы не будут возвращать `true` в результате применения некоторого оператора, указанного при задании ограничения.

- Для индекса B-tree это оператор =, что означает тривиальное ограничение уникальности.

- Для индекса GiST в случае диапазонов имеют смысл операторы &&(пересечение) и -|-(прилегание).

Пример – решение задачи бронирования

4. Запретим бронирование домика на пересекающиеся интервалы дат с помощью ограничения **EXCLUDE**

```
ALTER TABLE Reservations
ADD EXCLUDE USING GiST(Busy WITH &&);
```

5. Ограничения **EXCLUDE** не позволяет добавить конфликтующий диапазон дат

```
INSERT INTO Reservations(Busy) VALUES
('[2024-01-20, 2024-02-05)');

SQL Error [23P01]: ОШИБКА: конфликтующее значение ключа
нарушает ограничение-исключение "reservations_busy_excl"
Подробности: Ключ (busy)=([2024-01-20,2024-02-05))
конфликтует с существующим ключом (busy)=([2024-01-01,2024-01-30)).
```

Пример – ещё одна задача бронирования

6. Разобравшись с домиком, научимся теперь бронировать номера в отеле: пересекающиеся интервалы дат должны быть запрещены только в пределах номера, а не по всему отелю

```
CREATE TABLE Hotel_Res(Room_Nr integer, Busy daterange);

INSERT INTO Hotel_Res(Room_Nr, Busy) VALUES
(1,'[2024-01-01, 2024-01-30)'),
(1,'[2024-02-01, 2024-02-28)'),
(1,'[2024-03-01, 2024-03-30)'),
(2,'[2024-01-15, 2024-02-10)'),
(2,'[2024-03-20, 2024-04-01)'),
(3,'[2024-01-01, 2024-04-01)');
```

Сюрприз: в GiST нет операции равенства для целых чисел!

Расширение btree_gist

- Поддерживает составные индексы GiST, в которых некоторые столбцы индексируемы только с GiST (в т.ч. диапазоны), а другие — простые типы.

- Поддерживает использование индекса для операции <> («не равно»).

- Поддерживает **EXCLUDE** даже без GiST.

- Поддерживает оператор расстояния <-> и поиск ближайших соседей с помощью индекса GiST.

Пример – ещё одна задача бронирования

7. Запретим бронирование номеров отеля на пересекающиеся интервалы дат

```
CREATE EXTENSION btree_gist;

ALTER TABLE Hotel_Res
ADD EXCLUDE USING GiST(Busy WITH &&, Room_Nr WITH =);
```

8. Ограничения **EXCLUDE** теперь тоже не позволяет добавить конфликтующий диапазон дат

```
INSERT INTO Hotel_Res(Room_Nr, Busy) VALUES
(1,'[2024-01-20, 2024-02-05)');

SQL Error [23P01]: ОШИБКА: конфликтующее значение ключа
нарушает ограничение-исключение "hotel_res_busy_room_nr_excl"
Подробности: Ключ (busy, room_nr)=([2024-01-20,2024-02-05), 1)
конфликтует с существующим ключом (busy, room_nr)=([2024-01-01,2024-01-30), 1).
```

Пример – решение задачи версионности

1. Решим теперь как следует задачу развития станций с течением времени – с использованием диапазонов

```
DROP TABLE IF EXISTS Stations;
CREATE TABLE IF NOT EXISTS Stations
(
 St_Name  text
,N_Ways   integer
,Valid_At daterange
,CONSTRAINT Stations_EX
 EXCLUDE USING GIST (Valid_At WITH &&, St_Name WITH =)
);

COMMENT ON TABLE  Stations           IS 'Станции в разное время';
COMMENT ON COLUMN Stations.St_Name   IS 'Название станции';
COMMENT ON COLUMN Stations.N_Ways    IS 'Количество путей';
COMMENT ON COLUMN Stations.Valid_At  IS 'Период действия версии';
```

Пример – решение задачи версионности

2. Опять добавим станцию Бирюлёво до и после революции

```
INSERT INTO Stations(St_Name, N_Ways, Valid_At) VALUES
('Бирюлёво',10,daterange('1900-01-01','1917-11-07')),
('Бирюлёво',20,daterange('1917-11-07', NULL       ));
```

```
SELECT * FROM Stations ORDER BY Valid_At;
```

```
|st_name |n_ways|valid_at                  |
|--------|------|--------------------------|
|Бирюлёво|10    |[1900-01-01,1917-11-07)|
|Бирюлёво|20    |[1917-11-07,)          |
```

Пример – решение задачи версионности

3. Запросим действующую версию станции Бирюлёво и обратим внимание на план выполнения запроса

```
SELECT * FROM Stations
WHERE St_Name='Бирюлёво' AND CURRENT_DATE <@ Valid_At;
```

```
|st_name |n_ways|valid_at     |
|--------|------|-------------|
|Бирюлёво|20    |[1917-11-07,)|
```

```
EXPLAIN (costs OFF ) SELECT * FROM Stations
WHERE St_Name='Бирюлёво' AND CURRENT_DATE <@ Valid_At;
```

```
|QUERY PLAN                                                                |
|--------------------------------------------------------------------------|
|Index Scan using stations_ex on stations                                  |
| Index Cond: ((valid_at @> CURRENT_DATE) AND (st_name = 'Бирюлёво'::text))|
```

Пример – решение задачи версионности

4. Добавим новую версию станции Бирюлёво, в которой уже 30 путей, при этом сделав неактивной предыдущую версию этой станции.

   На этот раз сделаем это правильным образом.

   Будем использовать `CTE` (`Common Table Expressions`) для того, чтобы выполнить это преобразование атомарно при помощи одного SQL-оператора.

   `CTE` совместно с **`RETURNING`** позволяет передать множество строк, затронутых оператором **`UPDATE`** или **`DELETE`**, на вход следующего SQL-оператора.

Пример – решение задачи версионности


```
WITH CTE_Update AS
(
 UPDATE Stations
 SET    Valid_At = daterange(lower(Valid_At), CURRENT_DATE)
 WHERE  St_Name='Бирюлёво' AND CURRENT_DATE <@ Valid_At
 RETURNING St_Name, Valid_At -- Конвейер для следующего оператора
)
INSERT INTO Stations(St_Name, N_Ways, Valid_At)
SELECT St_Name, 30, daterange(upper(Valid_At),NULL,'[)')
FROM   CTE_Update;
```

Пример – решение задачи версионности

5. Таблица станций теперь выглядит так:

```
SELECT * FROM Stations ORDER BY Valid_At;
```

```
|st_name |n_ways|valid_at                    |
|--------|------|----------------------------|
|Бирюлёво|10    |[1900-01-01,1917-11-07)|
|Бирюлёво|20    |[1917-11-07,2024-03-16)|
|Бирюлёво|30    |[2024-03-16,)           |
```

6. Текущая версия станции Бирюлево выглядит так:

```
SELECT * FROM Stations
WHERE St_Name='Бирюлёво' AND CURRENT_DATE <@ Valid_At;
```

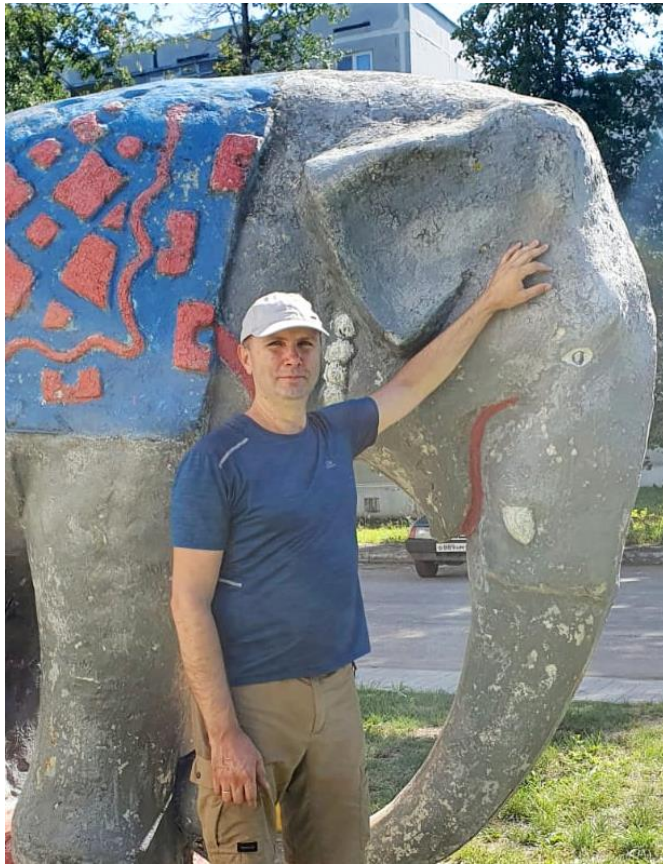
```
|st_name |n_ways|valid_at      |
|--------|------|--------------|
|Бирюлёво|30    |[2024-03-16,)|
```

Подытожим:

- Диапазонные типы в Postgres – не сахар (синтаксический), а мощный и эффективный инструмент!
- Пользуйтесь диапазонными типами, GiST и EXCLUDE в своих приложениях – это рафинирует логику и повышает производительность!
- Пользуйтесь Postgres версии 15+!
- Переехав в Postgres, давайте обживаться в нем по-настоящему!

ВНИИЖТ
НЦ ЦМПЭ

О докладчике



АО «ВНИИЖТ» (АО «Научно-исследовательский институт железнодорожного транспорта»)

Анфиногенов Анатолий Юрьевич

Заместитель директора научного центра – начальник отдела разработки ПО, кандидат физико-математических наук

Работаю уже два десятка лет во ВНИИЖТ над задачами имитационного моделирования и оптимизации железнодорожных перевозок.

Проектировал, разрабатывал и сопровождал БД и серверное ПО для этих задач (PostgreSQL, Oracle, C++, Python), чем и продолжаю заниматься.

Понравился доклад– голосуйте по ссылке:

https://pgconf.ru/talk/1688995

anfinogenov.anatoly@vniizht.ru

+7 (499) 262-45-06