MobilityDB 1.0 Manual de usuario

COLABORADORES

	TÍTULO : MobilityDB 1.0 Manual de usuario	

ACCIÓN	*NOMBRE*	*FECHA*	*FIRMA*
ESCRITO POR	Esteban Zimányi	16 de noviembre de 2021	

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE

Índice general

Índice de figuras

Índice de cuadros

Resumen

MobilityDB es una extensión del sistema de base de datos PostgreSQL y su extensión espacial PostGIS. Permite almacenar en la base de datos objetos temporales y espacio-temporales, es decir, objetos cuyos valores de atributo y/o ubicación evolucionan en el tiempo. MobilityDB incluye funciones para el análisis y procesamiento de objetos temporales y espacio-temporales y proporciona soporte para índices GiST y SP-GiST. MobilityDB es de código abierto y su código está disponible en Github. También está disponible un adaptador para el lenguaje de programación Python en Github.

MobilityDB es desarrollado por el Departamento de Ingeniería Informática y de Decisiones de la Université libre de Bruxelles (ULB) bajo la dirección del Prof. Esteban Zimányi. ULB es un miembro asociado de OGC y miembro del Grupo de trabajo de estandardización de características móviles de OGC (MF-SWG).

Capítulo 1

Introducción

MobilityDB es una extensión de PostgreSQL y PostGIS que proporciona *tipos temporales*. Dichos tipos de datos representan la evolución en el tiempo de los valores de algún tipo de elemento, llamado tipo base del tipo temporal. Por ejemplo, se pueden usar enteros temporales para representar la evolución en el tiempo del número de empleados de un departamento. En este caso, el tipo de datos es *entero temporal* y el tipo base es *entero*. Del mismo modo, se puede utilizar un número flotante temporal para representar la evolución en el tiempo de la temperatura de una habitación. Como otro ejemplo, se puede usar un punto temporal para representar la evolución en el tiempo de la ubicación de un automóvil, como lo reportan los dispositivos GPS. Los tipos temporales son útiles porque representar valores que evolucionan en el tiempo es esencial en muchas aplicaciones, por ejemplo, en aplicaciones de movilidad. Además, los operadores de los tipos base (como los operadores aritméticos y la agregación para números enteros y flotantes, las relaciones espaciales y la distancia para las geometrías) se pueden generalizar intuitivamente cuando los valores evolucionan en el tiempo.

MobilityDB proporciona los siguientes tipos temporales: `tbool`, `tint`, `tfloat`, `ttext`, `tgeompoint` y `tgeogpoint`. Estos tipos temporales se basan, respectivamente, en los tipos de base `bool`, `int`, `float` y `text` proporcionados por PostgreSQL, y en los tipos base de `geometry` y `geography` proporcionados por PostGIS (restringido a puntos 2D o 3D).[1] Además, MobilityDB usa cuatro tipos de tiempo para representar lapsos de tiempo: el tipo `timestamptz` proporcionado por PostgreSQL y tres nuevos tipos que son `period`, `timestampset` y `periodset`. Además, se definen dos tipos de rango en MobilityDB: `intrange` y `floatrange`.

1.1. Comité directivo del proyecto

El comité directivo del proyecto MobilityDB (Project Steering Committee o PSC) coordina la dirección general, los ciclos de publicación, la documentación y los esfuerzos de divulgación para el proyecto MobilityDB. Además, el PSC proporciona soporte general al usuario, acepta y aprueba parches de la comunidad general de MobilityDB y vota sobre diversos problemas relacionados con MobilityDB, como el acceso de commit de los desarrolladores, nuevos miembros del PSC o cambios significativos en la interfaz de programación de aplicaciones (Application Programming Interface o API).

A continuación se detallan los miembros actuales en orden alfabético y sus principales responsabilidades:

- Mohamed Bakli: MobilityDB-docker, versiones distribuidas y en la nube, integración con Citus
- Krishna Chaitanya Bommakanti: MobilityDB SQLAlchemy, MEOS (Mobility Engine Open Source), pyMEOS
- Anita Graser: integración con Moving Pandas y el ecosistema de Python, integración con QGIS
- Darafei Praliaskouski: integración con PostGIS
- Mahmoud Sakr: cofundador del proyecto MobilityDB, MobilityDB workshop, miembro del OGC Moving Feature Standard Working Group (MF-SWG)
- Vicky Vergara: integración con pgRouting

[1] Aunque los puntos temporales 4D se pueden representar, la dimensión M actualmente no se tiene en cuenta.

- Esteban Zimányi (chair): cofoundador del proyecto MobilityDB, coordinación general del proyecto, principal contribuidor del código de backend, BerlinMOD benchmark, MobilityDB-python

1.2. Otros colaboradores del código

- Arthur Lesuisse
- Xinyiang Li
- Maxime Schoemans

1.3. Patrocinadores

1.3.1. Patrocinadores de investigación

Estas son organizaciones de financiación de investigación (en orden alfabético) que han contribuido con financiación monetaria al proyecto MobilityDB.

- European Commission
- Fonds de la Recherche Scientifique (FNRS), Belgium
- Innoviris, Belgium

1.3.2. Patrocinadores corporativos

Estas son entidades corporativas (en orden alfabético) que han contribuido con tiempo de desarrollador o financiación monetaria al proyecto MobilityDB.

- Adonmo, India
- Georepublic, Germany
- Université libre de Bruxelles, Belgium

1.4. Licencias

Las siguientes licencias se pueden encontrar en MobilityDB:

Recurso	**Licencia**
Código MobilityDB	Licencia PostgreSQL
Documentación MobilityDB	Licencia Creative Commons Attribution-Share Alike 3.0

1.5. Instalación

1.5.1. Versión corta

Extraer el tar ball

```
tar xvfz MobilityDB-1.0.tar.gz
cd MobilityDB-1.0
```

Para compilar asumiendo que tiene todas las dependencias en su ruta de búsqueda

```
mkdir build
cd build
cmake  ..
make
sudo make install
```

Una vez que MobilityDB está instalado, debe habilitarse en cada base de datos individual en la que desee usarlo. En éste ejemplo se usa la versión 2.5.5 de PostGIS.

```
createdb mobility
psql mobility -c "CREATE EXTENSION PostGIS WITH VERSION'2.5.5'"
psql mobility -c 'CREATE EXTENSION MobilityDB'
```

1.5.2. Obtener las fuentes

La última versión de MobilityDB se puede encontrar en https://github.com/MobilityDB/MobilityDB/releases/latest

wget

Para descargar esta versión:

```
wget -O mobilitydb-1.0.tar.gz https://github.com/MobilityDB/MobilityDB/archive/v1.0.tar.gz
```

Ir a la Sección 1.5.1 para las instrucciones de extracción y compilación.

git

Para descargar el repositorio

```
git clone https://github.com/MobilityDB/MobilityDB.git
cd MobilityDB
git checkout v1.0
```

Ir a la Sección 1.5.1 para las instrucciones de compilación (no hay tar ball).

1.5.3. Habilitación de la base de datos

MobilityDB es una extensión que depende de PostGIS. Habilitar PostGIS antes de habilitar MobilityDB en la base de datos se puede hacer de la siguiente manera

```
CREATE EXTENSION postgis;
CREATE EXTENSION mobilitydb;
```

Alternativamente, esto se puede hacer con un solo comando usando `CASCADE`, que instala la extensión PostGIS requerida antes de instalar la extensión MobilityDB

```
CREATE EXTENSION mobilitydb CASCADE;
```

1.5.4. Dependencias

Dependencias de compilación

Para poder compilar MobilityDB, asegúrese de que se cumplan las siguientes dependencias:

- Compilador GNU C (`gcc`). Se pueden usar algunos otros compiladores ANSI C, pero pueden causar problemas al compilar algunas dependencias como PostGIS.
- GNU Make (`gmake` o `make`) versión 3.1 o superior. Para muchos sistemas, GNU make es la versión predeterminada de make. Verifique la versión invocando `make -v`.
- PostgreSQL versión 10 o superior. PostgreSQL está disponible en http://www.postgresql.org. Tenga en cuenta que para usar índices SP-GiST para MobilityDB, necesita al menos la versión 11 de PostgreSQL.
- PostGIS versión 2.5. PostGIS está disponible en https://postgis.net/. Actualmente, la versión 3.0 o superior de PostGIS **no es compatible**, esto está previsto para futuras versiones de MobilityDB.
- Biblioteca científica GNU (GSL). GSL está disponible en https://www.gnu.org/software/gsl/. GSL se utiliza para los generadores de números aleatorios.

Tenga en cuenta que PostGIS tiene sus propias dependencias, como Proj4, GEOS, LibXML2 o JSON-C y estas bibliotecas también se utilizan en MobilityDB. Para obtener una matriz de compatibilidad completa de PostgreSQL/PostGIS y una matriz de compatibilidad de PostGIS/GEOS, consulte http://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS.

Dependencias opcionales

Para la documentación del usuario

- Los archivos DocBook DTD y XSL son necesarios para crear la documentación. Para Ubuntu, son proporcionados por los paquetes `docbook` y `docbook-xsl`.
- El validador XML `xmllint` es necesario para validar los archivos XML de la documentación. Para Ubuntu, lo proporciona el paquete `libxml2`.
- El procesador XSLT `xsltproc` es necesario para crear la documentación en formato HTML. Para Ubuntu, lo proporciona el paquete `libxslt`.
- El programa `dblatex` es necesario para crear la documentación en formato PDF. Para Ubuntu, lo proporciona el paquete `dblatex`.
- El programa `dbtoepub` es necesario para construir la documentación en formato EPUB. Para Ubuntu, lo proporciona el paquete `dbtoepub`.

Para la documentación de los desarrolladores

- El programa `doxygen` es necesario para construir la documentación. Para Ubuntu, lo proporciona el paquete `doxygen`.

Ejemplo: instalar dependencias en Linux

Dependencias de base de datos

```
sudo apt-get install postgresql-13 postgresql-server-dev-13 postgresql-13-postgis
```

Dependencias de construcción

```
sudo apt-get install cmake gcc libgsl-dev
```

1.5.5. Configuración

MobilityDB usa el sistema `cmake` para realizar la configuración. El directorio de compilación deber ser diferente del directorio de origen.

Para crear el directorio de compilación

```
mkdir build
```

Para ver las variables que se pueden configurar

```
cd build
cmake -L ..
```

1.5.6. Construir e instalar

Tenga en cuenta que la versión actual de MobilityDB solo se ha probado en sistemas Linux. Puede funcionar en otros sistemas similares a UNIX, pero no se ha probado. Está previsto el soporte para Windows. Buscamos voluntarios que nos ayuden a probar MobilityDB en múltiples plataformas.

Las siguientes instrucciones comienzan desde `path/to/MobilityDB` en un sistema Linux

```
mkdir build
cd build
cmake ..
make
sudo make install
```

Cuando cambia la configuración

```
rm -rf build
```

e inicie el proceso de construcción como se mencionó anteriormente.

1.5.7. Pruebas

MobilityDB utiliza `ctest`, el programa controlador de pruebas de CMake, para realizar pruebas. Este programa ejecutará las pruebas e informará los resultados.

Para ejecutar todas las pruebas

```
ctest
```

Para ejecutar un archivo de prueba dado

```
ctest -R '21_tbox'
```

Para ejecutar un conjunto de archivos de prueba determinados se pueden utilizar comodines

```
ctest -R '22_*'
```

1.5.8. Documentación

La documentación del usuario de MobilityDB se puede generar en formato HTML, PDF y EPUB. Además, la documentación está disponible en inglés y en otros idiomas (actualmente, solo en español). La documentación del usuario se puede generar en todos los formatos y en todos los idiomas, o se pueden especificar formatos y/o idiomas específicos. La documentación del desarrollador de MobilityDB solo se puede generar en formato HTML y en inglés.

Las variables utilizadas para generar la documentación del usuario y del desarrollador son las siguientes:

Variable	Valor por defecto	Comentario
DOC_ALL	BOOL=OFF	La documentación del usuario se genera en formatos HTML, PDF y EPUB.
DOC_HTML	BOOL=OFF	La documentación del usuario se genera en formato HTML.
DOC_PDF	BOOL=OFF	La documentación del usuario se genera en formato PDF.
DOC_EPUB	BOOL=OFF	La documentación del usuario se genera en formato EPUB.
LANG_ALL	BOOL=OFF	La documentación del usuario se genera en inglés y en todas las traducciones disponibles.
ES	BOOL=OFF	La documentación del usuario se genera en inglés y en español.
DOC_DEV	BOOL=OFF	La documentación del desarrollador en inglés se genera en formato HTML

Cuadro 1.1: Variables para la documentación del usuario y del desarrollador

Generar la documentación del usuario y del desarrollador en todos los formatos y en todos los idiomas.

```
cmake -D DOC_ALL=ON -D LANG_ALL=ON -D DOC_DEV=ON ..
make doc
make doc_dev
```

Generar la documentación del usuario en formato HTML y en todos los idiomas.

```
cmake -D DOC_HTML=ON -D LANG_ALL=ON ..
make doc
```

Generar la documentación del usuario en inglés en todos los formatos.

```
cmake -D DOC_ALL=ON ..
make doc
```

Generar la documentación del usuario en formato PDF en inglés y en español.

```
cmake -D DOC_PDF=ON -D ES=ON ..
make doc
```

1.6. Soporte

El soporte de la comunidad de MobilityDB está disponible a través de la página github de MobilityDB, documentación, tutoriales, listas de correo y otros.

1.6.1. Reporte de problemas

Los errores son registrados y manejados en un issue tracker. Por favor siga los siguientes pasos:

1. Busque las entradas para ver si su problema ya ha sido informado. Si es así, agregue cualquier contexto adicional que haya encontrado, o al menos indique que usted también está teniendo el problema. Esto nos ayudará a priorizar los problemas comunes.
2. Si su problema no se ha informado, cree un nuevo asunto para ello.
3. En su informe, incluya instrucciones explícitas para replicar su problema. Las mejores etradas incluyen consultas SQL exactas que se necesitan para reproducir un problema. Reporte también el sistema operativo y las versiones de MobilityDB, PostGIS y PostgreSQL.
4. Se recomienda utilizar el siguiente envoltorio en su problema para determinar el paso que está causando el problema.

```
SET client_min_messages TO debug;
<your code>
SET client_min_messages TO notice;
```

1.6.2. Listas de correo

Hay dos listas de correo para MobilityDB alojadas en el servidor de listas de correo OSGeo:

- Lista de correo de usuarios: http://lists.osgeo.org/mailman/listinfo/mobilitydb-users
- Lista de distribución de desarrolladores: http://lists.osgeo.org/mailman/listinfo/mobilitydb-dev

Para preguntas generales y temas sobre cómo usar MobilityDB, escriba a la lista de correo de usuarios.

Capítulo 2

Tipos de tiempo y tipos de rango

Los tipos temporales se basan en cuatro tipos de tiempo: el tipo `timestamptz` proporcionado por PostgreSQL y tres nuevos tipos que son `period`, `timestampset` y `periodset`.

El tipo `period` es una versión especializada del tipo `tstzrange` (abreviatura de rango de marcas de tiempo con zona horaria) proporcionado por PostgreSQL. El tipo `period` tiene una funcionalidad similar al tipo `tstzrange` pero tiene una implementación más eficiente, en particular es de longitud fija mientras que el tipo `tstzrange` es de longitud variable. Además, los períodos vacíos y los límites infinitos no están permitidos en valores de `period`, mientras están permitidos en valores de `tstzrange`.

Un valor del tipo `period` tiene dos límites, el límite inferior y el límite superior, que son valores de `timestamptz`. Los límites pueden ser inclusivos o exclusivos. Un límite inclusivo significa que el instante límite está incluido en el período, mientras que un límite exclusivo significa que el instante límite no está incluido en el período. En el formato textual de un valor de `period`, los límites inferiores inclusivos y exclusivos están representados, respectivamente, por "`[`" y "`(`". Asimismo, los límites superiores inclusivos y exclusivos se representan, respectivamente, por "`]`" y "`)`". En un valor de `period`, el límite inferior debe ser menor o igual que el límite superior. Un valor de `period` con límites iguales e inclusivos se llama *período instantáneo* y corresponde a un valor de `timestamptz`. Ejemplos de valores de `period` son como sigue:

```
SELECT period '[2012-01-01 08:00:00, 2012-01-03 09:30:00)';
-- Período instantáneo
SELECT period '[2012-01-01 08:00:00, 2012-01-01 08:00:00]';
-- Período erróneo: límites inválidos
SELECT period '[2012-01-01 08:10:00, 2012-01-01 08:00:00]';
-- Período erróneo: período vacío
SELECT period '[2012-01-01 08:00:00, 2012-01-01 08:00:00)';
```

El tipo `timestampset` representa un conjunto de valores `timestamptz` diferentes. Un valor de `timestampset` debe contener al menos un elemento, en cuyo caso corresponde a un valor de `timestamptz`. Los elementos que componen un valor de `timestampset` deben estar ordenados. Ejemplos de valores de `timestampset` son como sigue:

```
SELECT timestampset '{2012-01-01 08:00:00, 2012-01-03 09:30:00}';
-- Conjunto unitario de marcas de tiempo
SELECT timestampset '{2012-01-01 08:00:00}';
-- Conjunto de marcas de tiempo erróneo: elementos desordenados
SELECT timestampset '{2012-01-01 08:10:00, 2012-01-01 08:00:00}';
-- Conjunto de marcas de tiempo erróneo: elementos duplicados
SELECT timestampset '{2012-01-01 08:00:00, 2012-01-01 08:00:00}';
```

Finalmente, el tipo `periodset` representa un conjunto de valores disjuntos de `period`. Un valor de `periodset` debe contener al menos un elemento, en cuyo caso corresponde a un valor de `period`. Los elementos que componen un valor de `periodset` deben estar ordenados. Ejemplos de valores de `periodset` son come sigue:

```
SELECT periodset '{[2012-01-01 08:00:00, 2012-01-01 08:10:00],
  [2012-01-01 08:20:00, 2012-01-01 08:40:00]}';
-- Conjunto unitario de períodos
SELECT periodset '{[2012-01-01 08:00:00, 2012-01-01 08:10:00]}';
-- Conjunto de períodos erróneo: elementos desordenados
SELECT periodset '{[2012-01-01 08:20:00, 2012-01-01 08:40:00],
  [2012-01-01 08:00:00, 2012-01-01 08:10:00]}';
-- Conjunto de períodos erróneo: elementos superpuestos
SELECT periodset '{[2012-01-01 08:00:00, 2012-01-01 08:10:00],
  [2012-01-01 08:05:00, 2012-01-01 08:15:00]}';
```

Los valores del tipo `periodset` son convertidos en *forma normal* de modo que los valores equivalentes tengan representaciones idénticas. Para ello, los valores de períodos consecutivos que son adyacentes se fusionan cuando es posible. Un ejemplo de transformación a forma normal es el siguiente:

```
SELECT periodset '{[2012-01-01 08:00:00, 2012-01-01 08:10:00),
  [2012-01-01 08:10:00, 2012-01-01 08:10:00], (2012-01-01 08:10:00, 2012-01-01 08:20:00]}';
-- "{[2012-01-01 08:00:00+00,2012-01-01 08:20:00+00]}"
```

Además de los tipos de rango nativos proporcionados por PostgreSQL, MobilityDB define dos tipos de rango adicionales: `intrange` (otro nombre para `int4range`) y `floatrange`.

2.1. Funciones y operadores para tipos de tiempo y de rango

A continuación presentamos las funciones y operadores para tipos de tiempo y de rango. Estas funciones y operadores son polimórficos, es decir, sus argumentos pueden ser de varios tipos y el tipo de resultado puede depender del tipo de los argumentos. Para expresar esto en la firma de los operadores, utilizamos la siguiente notación:

- Un conjunto de tipos como `{period,timestampset,periodset}` representa cualquiera de los tipos enumerados,
- `time` representa cualquier tipo de tiempo, es decir, `timestamptz`, `period`, `timestampset` o `periodset`,
- `number` representa cualquier tipo de número, es decir, `integer` o `float`,
- `range` representa cualquier rango de números, es decir, `intrange` o `floatrange`.
- `type[]` representa una matriz de `type`.

Como ejemplo, la firma del operador contiene (`@>`) es como sigue:

```
{timestampset,period,periodset} @> time
```

A continuación, por concisión, la parte de tiempo de las marcas de tiempo se omite en los ejemplos. Recuerde que en ese caso PostgreSQL asume el tiempo `00:00:00`.

2.1.1. Funciones de constructor

El tipo `period` tiene una función constructora que acepta dos o cuatro argumentos. La forma de dos argumentos construye un punto en *forma normal*, es decir, con límite inferior inclusivo y límite superior exclusivo. La forma de cuatro argumentos construye un período con límites especificados por el tercer y cuarto argumento, que son valores booleanos que indican, respectivamente, si los límites izquierdo y derecho son inclusivos o no.

- Constructor para `period`

  `period(timestamptz,timestamptz,left_inc=true,right_inc=false): period`

```
-- Período definido con dos argumentos
SELECT period('2012-01-01 08:00:00', '2012-01-03 08:00:00');
-- [2012-01-01 08:00:00+01, 2012-01-03 08:00:00+01)
-- Período definido con cuatro argumentos
SELECT period('2012-01-01 08:00:00', '2012-01-03 09:30:00', false, true);
-- (2012-01-01 08:00:00+01, 2012-01-03 09:30:00+01]
```

El tipo `timestampset` tiene una función constructora que acepta un solo argumento que es una matriz de valores de `timestamptz`.

- Constructor para `timestampset`

  `timestampset(timestamptz[]): timestampset`

```
SELECT timestampset(ARRAY[timestamptz '2012-01-01 08:00:00', '2012-01-03 09:30:00']);
-- "{2012-01-01 08:00:00+00, 2012-01-03 09:30:00+00}"
```

El tipo `periodset` tiene una función constructora que acepta un solo argumento que es una matriz de valores de `period`.

- Constructor para `periodset`

  `periodset(period[]): periodset`

```
SELECT periodset(ARRAY[period '[2012-01-01 08:00:00, 2012-01-01 08:10:00]',
-- '[2012-01-01 08:20:00, 2012-01-01 08:40:00]']);
```

2.1.2. Conversión de tipos

Los valores de los tipos `timestamptz`, `tstzrange` o los tipos de tiempo se pueden convertir entre sí utilizando la función `CAST` o utilizando la notación `::`.

- Convertir un `timestamptz`a otro tipo de tiempo

  `timestamptz::timestampset`

  `timestamptz::period`

  `timestamptz::periodset`

```
SELECT CAST(timestamptz '2012-01-01 08:00:00' AS timestampset);
-- "{2012-01-01 08:00:00+01}"
SELECT CAST(timestamptz '2012-01-01 08:00:00' AS period);
-- "[2012-01-01 08:00:00+01, 2012-01-01 08:00:00+01]"
SELECT CAST(timestamptz '2012-01-01 08:00:00' AS periodset);
-- "{[2012-01-01 08:00:00+01, 2012-01-01 08:00:00+01]}"
```

- Convertir un `timestampset` a un `periodset`

  `timestampset::periodset`

```
SELECT CAST(timestampset '{2012-01-01 08:00:00, 2012-01-01 08:15:00,
  2012-01-01 08:25:00}' AS periodset);
-- "{[2012-01-01 08:00:00+01, 2012-01-01 08:00:00+01],
  [2012-01-01 08:15:00+01, 2012-01-01 08:15:00+01],
  [2012-01-01 08:25:00+01, 2012-01-01 08:25:00+01]}"
```

- Convertir un `period` a otro tipo de tiempo

  `period::periodset`

  `period::tstzrange`

```
SELECT period '[2012-01-01 08:00:00, 2012-01-01 08:30:00)'::periodset;
-- "{[2012-01-01 08:00:00+01, 2012-01-01 08:30:00+01)}"
SELECT period '[2012-01-01 08:00:00, 2012-01-01 08:30:00)'::tstzrange;
-- "["2012-01-01 08:00:00+01","2012-01-01 08:30:00+01")"
```

- Convertir un `tstzrange` a un `period`

  `tstzrange::period`

```
SELECT tstzrange '[2012-01-01 08:00:00, 2012-01-01 08:30:00)'::period;
-- "[2012-01-01 08:00:00+01, 2012-01-01 08:30:00+01)"
```

2.1.3. Funciones de accesor

- Obtener el tamaño de la memoria en bytes

  `memSize({timestampset,periodset}): integer`

```
SELECT memSize(timestampset '{2012-01-01, 2012-01-02, 2012-01-03}');
-- 104
SELECT memSize(periodset '{[2012-01-01, 2012-01-02], [2012-01-03, 2012-01-04],
  [2012-01-05, 2012-01-06]}');
-- 136
```

- Obtener el límite inferior

  `lower(period): timestamptz`

```
SELECT lower(period '[2011-01-01, 2011-01-05)');
-- "2011-01-01"
```

- Obtener el límite superior

  `upper(period): timestamptz`

```
SELECT upper(period '[2011-01-01, 2011-01-05)');
-- "2011-01-05"
```

- ¿Es el límite inferior inclusivo?

  `lower_inc(period): boolean`

```
SELECT lower_inc(period '[2011-01-01, 2011-01-05)');
-- true
```

- ¿Es el límite superior inclusivo?

  `upper_inc(period): boolean`

```
SELECT upper_inc(period '[2011-01-01, 2011-01-05)');
-- false
```

- Obtener el intervalo de tiempo

  `duration({period,periodset}): interval`

```
SELECT duration(period '[2012-01-01, 2012-01-03)');
-- "2 days"
SELECT duration(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-05)}');
-- "3 days"
```

- Obtener el intervalo de tiempo ignorando las posibles brechas de tiempo

  `timespan({timestampset,period,periodset}): interval`

```
SELECT timespan(timestampset '{2012-01-01, 2012-01-03}');
-- "2 days"
SELECT timespan(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-05)}');
-- "4 days"
```

- Obtener el período en el que se define el conjunto de marcas de tiempo o el conjunto de períodos ignorando las posibles brechas de tiempo

  `period({timestampset,periodset}): period`

```
SELECT period(timestampset '{2012-01-01, 2012-01-03, 2012-01-05}');
-- "[2012-01-01, 2012-01-05]"
SELECT period(periodset '{[2012-01-01, 2012-01-02), [2012-01-03, 2012-01-04)}');
-- "[2012-01-01, 2012-01-04)"
```

- Obtener el número de marcas de tiempo diferentes

  `numTimestamps({timestampset,periodset}): integer`

```
SELECT numTimestamps(timestampset '{2012-01-01, 2012-01-03, 2012-01-04}');
-- 3
SELECT numTimestamps(periodset '{[2012-01-01, 2012-01-03), (2012-01-03, 2012-01-05)}');
-- 3
```

- Obtener la marca de tiempo inicial

  `startTimestamp({timestampset,periodset}): timestamptz`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT startTimestamp(periodset '{[2012-01-01, 2012-01-03), (2012-01-03, 2012-01-05)}');
-- "2012-01-01"
```

- Obtener la marca de tiempo final

  `endTimestamp({timestampset,periodset}): timestamptz`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT endTimestamp(periodset '{[2012-01-01, 2012-01-03), (2012-01-03, 2012-01-05)}');
-- "2012-01-05"
```

- Obtener la enésima marca de tiempo diferente

  `timestampN({timestampset,periodset},integer): timestamptz`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT timestampN(periodset '{[2012-01-01, 2012-01-03), (2012-01-03, 2012-01-05)}', 3);
-- "2012-01-04"
```

- Obtener las marcas de tiempo diferentes

  `timestamps({timestampset,periodset}): timestampset`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT timestamps(periodset '{[2012-01-01, 2012-01-03), (2012-01-03, 2012-01-05)}');
-- "{"2012-01-01", "2012-01-03", "2012-01-05"}"
```

- Obtener el número de períodos

  `numPeriods(periodset): integer`

```
SELECT numPeriods(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-04],
  [2012-01-05, 2012-01-06)}');
-- 3
```

- Obtener el período inicial

  `startPeriod(periodset): period`

```
SELECT startPeriod(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-04],
  [2012-01-05, 2012-01-06)}');
-- "[2012-01-01,2012-01-03)"
```

- Obtener el período final

  `endPeriod(periodset): period`

```
SELECT endPeriod(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-04],
  [2012-01-05, 2012-01-06)}');
-- "[2012-01-05,2012-01-06)"
```

- Obtener el enésimo período

  `periodN(periodset,integer): period`

```
SELECT periodN(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-04],
  [2012-01-05, 2012-01-06)}', 2);
-- "[2012-01-04,2012-01-04]"
```

- Obtener los períodos

  `periods(periodset): period[]`

```
SELECT periods(periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-04],
  [2012-01-05, 2012-01-06)}');
-- "{"[2012-01-01,2012-01-03)", "[2012-01-04,2012-01-04]", "[2012-01-05,2012-01-06)"}"
```

2.1.4. Funciones de modificación

- Desplazar el valor de tiempo con un intervalo

  `shift({timestampset,period,periodset}): {timestampset,period,periodset}`

```
SELECT shift(timestampset '{2001-01-01, 2001-01-03, 2001-01-05}', '1 day'::interval);
-- "{2001-01-02, 2001-01-04, 2001-01-06}"
SELECT shift(period '[2001-01-01, 2001-01-03]', '1 day'::interval);
-- "[2001-01-02, 2001-01-04]"
SELECT shift(periodset '{[2001-01-01, 2001-01-03], [2001-01-04, 2001-01-05]}',
  '1 day'::interval);
-- "{[2001-01-02, 2001-01-04], [2001-01-05, 2001-01-06]}"
```

- Redondear los límites de un rango flotante a un número de decimales

  `setPrecision(floatrange,integer): floatrange`

```
SELECT setPrecision(floatrange '[1.123456789,2.123456789)', 3);
-- "[1.123,2.123)"
SELECT setPrecision(floatrange '(,2.123456789]', 3);
-- "(,2.123]"
SELECT setPrecision(floatrange '[1.123456789, inf)', 3);
-- "[1.123,Infinity)"
```

2.1.5. Operadores de comparación

Los operadores de comparación (=, <, etc.) requieren que los argumentos izquierdo y derecho sean del mismo tipo. Exceptuando la igualdad y la no igualdad, los otros operadores de comparación no son útiles en el mundo real, pero permiten construir índices de árbol B en tipos de tiempo. Para los valores de período, los operadores comparan primero el límite inferior y luego el límite superior. Para los valores de conjunto de marcas de tiempo y conjunto de períodos, los operadores comparan primero los períodos delimitadores y, si son iguales, comparan los primeros N instantes o períodos, donde N es el mínimo del número de instantes o períodos que componen ambos valores.

Los operadores de comparación disponibles para los tipos de tiempo se dan a continuación.

- ¿Son iguales los valores de tiempo?

  `time = time: boolean`

```
SELECT period '[2012-01-01, 2012-01-04)' = period '[2012-01-01, 2012-01-04)';
-- true
```

- ¿Son diferentes los valores de tiempo?

  `time <> time: boolean`

```
SELECT period '[2012-01-01, 2012-01-04)' <> period '[2012-01-03, 2012-01-05)';
-- true
```

- ¿Es el primer valor de tiempo menor que el segundo?

  `time < time: boolean`

```
SELECT timestampset '{2012-01-01, 2012-01-04}' < timestampset '{2012-01-01, 2012-01-05}';
-- true
```

- ¿Es el primer valor de tiempo mayor que el segundo?

  `time > time: boolean`

```
SELECT period '[2012-01-03, 2012-01-04)' > period '[2012-01-02, 2012-01-05)';
-- true
```

- ¿Es el primer valor de tiempo menor o igual que el segundo?

  `time <= time: boolean`

```
SELECT periodset '{[2012-01-01, 2012-01-04)}' <=
  periodset '{[2012-01-01, 2012-01-05), [2012-01-06, 2012-01-07)}';
-- true
```

- ¿Es el primer valor de tiempo mayor o igual que el segundo?

  `time >= time: boolean`

```
SELECT period '[2012-01-03, 2012-01-05)' >= period '[2012-01-03, 2012-01-04)';
-- true
```

2.1.6. Operadores de conjuntos

Los operadores de conjuntos disponibles para los tipos de tiempo se dan a continuación.

- Unión de valores de tiempo

  `time + time: time`

```
SELECT timestampset '{2011-01-01, 2011-01-03, 2011-01-05}' +
  timestampset '{2011-01-03, 2011-01-06}';
-- "{2011-01-01, 2011-01-03, 2011-01-05, 2011-01-06}"
SELECT period '[2011-01-01, 2011-01-05)' + period '[2011-01-03, 2011-01-07)';
-- "[2011-01-01, 2011-01-07)"
SELECT periodset '{[2011-01-01, 2011-01-03), [2011-01-04, 2011-01-05)}' +
  period '[2011-01-03, 2011-01-04)';
-- "{[2011-01-01, 2011-01-05)}"
```

- Intersección de valores de tiempo

  `time * time: time`

```
SELECT timestampset '{2011-01-01, 2011-01-03}' * timestampset '{2011-01-03, 2011-01-05}';
-- "{2011-01-03}"
SELECT period '[2011-01-01, 2011-01-05)' * period '[2011-01-03, 2011-01-07)';
-- "[2011-01-03, 2011-01-05)"
```

- Diferencia de valores de tiempo

  `time - time: time`

```
SELECT period '[2011-01-01, 2011-01-05)' - period '[2011-01-03, 2011-01-07)';
-- "[2011-01-01, 2011-01-03)"
SELECT period '[2011-01-01, 2011-01-05]' - period '[2011-01-03, 2011-01-04]'
-- "{[2011-01-01,2011-01-03), (2011-01-04,2011-01-05]}"
SELECT periodset '{[2011-01-01, 2011-01-06], [2011-01-07, 2011-01-10]}' -
  periodset '{[2011-01-02, 2011-01-03], [2011-01-04, 2011-01-05],
  [2011-01-08, 2011-01-09]}';
-- "{[2011-01-01,2011-01-02), (2011-01-03,2011-01-04), (2011-01-05,2011-01-06],
  [2011-01-07,2011-01-08), (2011-01-09,2011-01-10]}"
```

2.1.7. Operadores topológicos

A continuación se presentan los operadores topológicos disponibles para los tipos de tiempo.

- ¿Se superponen los valores de tiempo (tienen instantes en común)?

  `{timestampset,period,periodset} && {timestampset,period,periodset}: boolean`

```
SELECT period '[2011-01-01, 2011-01-05)' && period '[2011-01-02, 2011-01-07)';
-- true
```

- ¿Contiene el primer valor de tiempo el segundo?

  `{timestampset,period,periodset} @> time: boolean`

```
SELECT period '[2011-01-01, 2011-05-01)' @> period '[2011-02-01, 2011-03-01)';
-- true
SELECT period '[2011-01-01, 2011-05-01)' @> timestamptz '2011-02-01';
-- true
```

- ¿Está el primer valor de tiempo contenido en el segundo?

  `time <@ {timestampset,period,periodset}: boolean`

```
SELECT period '[2011-02-01, 2011-03-01)' <@ period '[2011-01-01, 2011-05-01)';
-- true
SELECT timestamptz '2011-01-10' <@ period '[2011-01-01, 2011-05-01)';
-- true
```

- ¿Es el primer valor de tiempo adyacente al segundo?

  `time -|- time: boolean`

```
SELECT period '[2011-01-01, 2011-01-05)' -|- timestampset '{2011-01-05, 2011-01-07}';
-- true
SELECT periodset '{[2012-01-01, 2012-01-02]}' -|- period '[2012-01-02, 2012-01-03)';
-- false
```

2.1.8. Operadores de posición relativa

En PostgreSQL, los operadores de rango `<<`, `&<`, `>>`, `&>` y `-|-` solo aceptan rangos como argumento izquierdo o derecho. Extendimos estos operadores para rangos númericos de modo que un argumento puede ser un número entero o flotante.

Los operadores de posición relativa disponibles para los tipos de tiempo y tipos de rango se dan a continuación.

- ¿Está el primer número o valor de rango estrictamente a la izquierda del segundo?

  `{number,range} << {number,range}: boolean`

```
SELECT intrange '[15, 20)' << 20;
-- true
```

- ¿Está el primer número o valor de rango estrictamente a la derecha del segundo?

  `{number,range} >> {number,range}: boolean`

```
SELECT intrange '[15, 20)' >> 10;
-- true
```

- ¿No está el primer número o valor de rango a la derecha del segundo?

  `{number,range} &< {number,range}: boolean`

```
SELECT intrange '[15, 20)' &< 18;
-- false
```

- ¿No está el primer número o valor de rango a la izquierda del segundo?

  `{number,range} &> {number,range}: boolean`

```
SELECT period '[2011-01-01, 2011-01-03)' &> period '[2011-01-01, 2011-01-05)';
-- true
SELECT intrange '[15, 20)' &> 30;
-- true
```

- ¿Es el primer número o valor de rango adyacente al segundo?

  `{number,range} -|- {number,range}: boolean`

```
SELECT floatrange '[15, 20)' -|- 20;
-- true
```

- ¿Es el primer valor de tiempo estrictamente anterior al segundo?

  `time <<# time: boolean`

```
SELECT period '[2011-01-01, 2011-01-03)' <<# timestampset '{2011-01-03, 2011-01-05}';
-- true
```

- ¿Es el primer valor de tiempo estrictamente posterior al segundo?

  `time #>> time: boolean`

```
SELECT period '[2011-01-04, 2011-01-05)' #>>
  periodset '{[2011-01-01, 2011-01-04), [2011-01-05, 2011-01-06)}';
-- true
```

- ¿No es el primer valor de tiempo posterior al segundo?

  `time &<# time: boolean`

```
SELECT timestampset '{2011-01-02, 2011-01-05}' &<# period '[2011-01-01, 2011-01-05)';
-- false
```

- ¿No es el primer valor de tiempo anterior al segundo?

  `time #&> time: boolean`

```
SELECT timestamp '2011-01-01' #&> period '[2011-01-01, 2011-01-05)';
-- true
```

2.1.9. Funciones agregadas

Las funciones agregadas temporales generalizan las funciones agregadas tradicionales. Su semántica es que calculan el valor de la función en cada instante de la *unión* de las extensiones temporales de los valores a agregar. En contraste, recuerde que todas las otras funciones que manipulan tipos de tiempo calculan el valor de la función en cada instante de la *intersección* de las extensiones temporales de los argumentos.

Las funciones agregadas temporales son las siguientes:

- La función `tcount` generaliza la función tradicional `count`. El conteo temporal se puede utilizar para calcular en cada momento el número de objetos disponibles (por ejemplo, el número of períodos). La función `tcount` devuelve un entero temporal (ver el Capítulo 3).
- La función `extent` devuelve un período delimitador que engloba un conjunto de valores de tiempo.

Del mismo modo, hay una función agregada para los tipos de rango:

- La función `extent` devuelve un rango delimitador que engloba un conjunto de valores de rango.

La unión es una operación a muy útil para los tipos de tiempo. Como hemos visto en la Sección 2.1.6, podemos calcular la unión de dos valores de tiempo usando el operador +. Sin embargo, también es muy útil tener una versión agregada del operador de unión para combinar un número arbitrario de valores. La función `tunion` se puede utilizar para este propósito.

- Conteo temporal

  `tcount({timestampset,period,periodset}): {tinti,tints}`

```
WITH times(ts) AS (
  SELECT timestampset '{2000-01-01, 2000-01-03, 2000-01-05}' UNION
  SELECT timestampset '{2000-01-02, 2000-01-04, 2000-01-06}' UNION
  SELECT timestampset '{2000-01-01, 2000-01-02}'
)
SELECT tcount(ts) FROM times;
-- "{2@2000-01-01, 2@2000-01-02, 1@2000-01-03, 1@2000-01-04, 1@2000-01-05, 1@2000-01-06}"

WITH periods(ps) AS (
  SELECT periodset '{[2000-01-01, 2000-01-02], [2000-01-03, 2000-01-04]}' UNION
  SELECT periodset '{[2000-01-01, 2000-01-04], [2000-01-05, 2000-01-06]}' UNION
  SELECT periodset '{[2000-01-02, 2000-01-06]}'
)
SELECT tcount(ps) FROM periods;
-- {[2@2000-01-01, 3@2000-01-02], (2@2000-01-02, 3@2000-01-03, 3@2000-01-04],
   (1@2000-01-04, 2@2000-01-05, 2@2000-01-06]}
```

- Período delimitador

  `extent({timestampset,period,periodset}): period`

```
WITH times(ts) AS (
  SELECT timestampset '{2000-01-01, 2000-01-03, 2000-01-05}' UNION
  SELECT timestampset '{2000-01-02, 2000-01-04, 2000-01-06}' UNION
  SELECT timestampset '{2000-01-01, 2000-01-02}'
)
SELECT extent(ts) FROM times;
-- "[2000-01-01, 2000-01-06]"

WITH periods(ps) AS (
  SELECT periodset '{[2000-01-01, 2000-01-02], [2000-01-03, 2000-01-04]}' UNION
  SELECT periodset '{[2000-01-01, 2000-01-04], [2000-01-05, 2000-01-06]}' UNION
  SELECT periodset '{[2000-01-02, 2000-01-06]}'
)
SELECT extent(ps) FROM periods;
-- "[2000-01-01, 2000-01-06]"
```

- Rango delimitador

  `extent(range): range`

```
WITH ranges(r) AS (
  SELECT floatrange '[1, 4)' UNION
  SELECT floatrange '(5, 8)' UNION
  SELECT floatrange '(7, 9)'
)
SELECT extent(r) FROM ranges;
-- "[1,9)"
```

- Unión temporal

  `tunion({timestampset,period,periodset}): {timestampset,periodset}`

```
WITH times(ts) AS (
  SELECT timestampset '{2000-01-01, 2000-01-03, 2000-01-05}' UNION
  SELECT timestampset '{2000-01-02, 2000-01-04, 2000-01-06}' UNION
  SELECT timestampset '{2000-01-01, 2000-01-02}'
)
```

```
SELECT tunion(ts) FROM times;
-- "{2000-01-01, 2000-01-02, 2000-01-03, 2000-01-04, 2000-01-05, 2000-01-06}"
WITH periods(ps) AS (
  SELECT periodset '{[2000-01-01, 2000-01-02], [2000-01-03, 2000-01-04]}' UNION
  SELECT periodset '{[2000-01-02, 2000-01-03], [2000-01-05, 2000-01-06]}' UNION
  SELECT periodset '{[2000-01-07, 2000-01-08]}'
)
SELECT tunion(ps) FROM periods;
-- "{[2000-01-01, 2000-01-04], [2000-01-05, 2000-01-06], [2000-01-07, 2000-01-08]}"
```

2.2. Indexación de tipos de tiempo

Se pueden crear índices GiST y SP-GiST en columnas de tablas de los tipos `timestampset`, `period` y `periodset`. Un ejemplo de creación de un índice GiST en una columna `During` de tipo `period` en una tabla `Reservation` es como sigue:

```
CREATE TABLE Reservation (ReservationID integer PRIMARY KEY, RoomID integer,
  During period);
CREATE INDEX Reservation_During_Idx ON Reservation USING GIST(During);
```

Un índice GiST o SP-GiST puede acelerar las consultas que involucran a los siguientes operadores: =, `&&`, `<@`, `@>`, `-|-`, `<<`, `>>`, `&<` y `&>`.

Además, se pueden crear índices de árbol B para columnas de tabla de un tipo de tiempo. Para estos tipos de índices, básicamente la única operación útil es la igualdad. Hay un orden de clasificación de árbol B definido para valores de tipos de tiempo con los correspondientes operadores < y >, pero el orden es bastante arbitrario y no suele ser útil en el mundo real. El soporte del árbol B está destinado principalmente a permitir la clasificación interna en las consultas, en lugar de la creación de índices reales.

Capítulo 3

Tipos temporales

Hay seis tipos temporales nativos, a saber `tbool`, `tint`, `tfloat`, `ttext`, `tgeompoint` y `tgeogpoint`, que se basan, respectivamente, en los tipos de base `bool`, `int`, `float`, `text`, `geometry` y `geography` (los dos últimos tipos restringidos a puntos 2D o 3D con dimensión Z).

La *interpolación* de un valor temporal establece cómo evoluciona el valor entre instantes sucesivos. La interpolación es *escalonada* cuando el valor permanece constante entre dos instantes sucesivos. Por ejemplo, el número de empleados de un departamento puede representarse con un número entero temporal, lo que indica que su valor es constante entre dos instantes de tiempo. Por otro lado, la interpolación es *lineal* cuando el valor evoluciona linealmente entre dos instantes sucesivos. Por ejemplo, la temperatura de una habitación puede representarse con un número flotante temporal, lo que indica que los valores se conocen en los dos instantes de tiempo pero evolucionan continuamente entre ellos. De manera similar, la ubicación de un vehículo se puede representar mediante un punto temporal en el que la ubicación entre dos lecturas GPS consecutivas se obtiene mediante interpolación lineal. Los tipos temporales basados en tipos base discretos, es decir `tbool`, `tint`, o `ttext` evolucionan necesariamente de manera escalonada. Por otro lado, los tipos temporales basados en tipos base continuos, es decir `tfloat`, `tgeompoint`, o `tgeogpoint` pueden evolucionar de manera lineal o escalonada.

El *subtipo* de un valor temporal establece la extensión temporal en la que se registra la evolución de los valores. Los valores temporales vienen en cuatro subtipos, a saber, instante, conjunto de instantes, secuencia y conjunto de secuencias.

Un valor temporal de subtipo *instante* (brevemente, un *valor de instante*) representa el valor en un instante de tiempo, por ejemplo

```
SELECT tfloat '17@2018-01-01 08:00:00';
```

Un valor temporal de subtipo *conjunto de instantes* (brevemente, un *valor de conjunto de instantes*) representa la evolución del valor en un conjunto de instantes de tiempo, donde los valores entre estos instantes son desconocidos. Un ejemplo es el siguiente:

```
SELECT tfloat '{17@2018-01-01 08:00:00, 17.5@2018-01-01 08:05:00, 18@2018-01-01 08:10:00}';
```

Un valor temporal de subtipo *secuencia* (brevemente, un *valor de secuencia*) representa la evolución del valor durante una secuencia de instantes de tiempo, donde los valores entre estos instantes se interpolan usando una función lineal o escalonada (ver más abajo). Un ejemplo es el siguiente:

```
SELECT tint '(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]';
```

Como puede verse, un valor de secuencia tiene un límite superior e inferior que pueden ser inclusivos (representados por '`[`' y '`]`') o exclusivos (representados por '`(`' y '`)`'). Un valor de secuencia con un único instante como

```
SELECT tint '[10@2018-01-01 08:00:00]';
```

es llamado una *secuencia instantánea*. En ese caso, ambos límites deben ser inclusivos.

El valor de una secuencia temporal se interpreta asumiendo que el período de tiempo definido por cada par de valores consecutivos `v1@t1` y `v2@t2` es inferior inclusivo y superior exclusivo, a menos que sean el primer o el último instantes de la secuencia y, en ese caso, se aplican los límites de toda la secuencia. Además, el valor que toma la secuencia temporal entre dos instantes consecutivos depende de si la interpolación es escalonada o lineal. Por ejemplo, la secuencia temporal anterior representa que el valor es `10` durante `(2018-01-01 08:00:00, 2018-01-01 08:05:00)`, `20` durante `[2018-01-01 08:05:00, 2018-01-01 08:10:00)` y `15` en el instante final `2018-01-01 08:10:00`. Por otro lado, la siguiente secuencia temporal

```
SELECT tfloat '(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]';
```

representa que el valor evoluciona linealmente de `10` a `20` durante `(2018-01-01 08:00:00, 2018-01-01 08:05:00)` y evoluciona de `20` a `15` durante `[2018-01-01 08:05:00, 2018-01-01 08:10:00]`.

Finalmente, un valor temporal de subtipo *conjunto de secuencias* (brevemente, un *valor de conjunto de secuencias*) representa la evolución del valor en un conjunto de secuencias, donde se desconocen los valores entre estas secuencias. Un ejemplo es el siguiente:

```
SELECT tfloat '{[17@2018-01-01 08:00:00, 17.5@2018-01-01 08:05:00],
  [18@2018-01-01 08:10:00, 18@2018-01-01 08:15:00]}';
```

Los valores temporales con subtipo instante o secuencia se denominan *valores temporales unitarios*, mientras que los valores temporales con subtipo conjunto de instantes o de secuencias se llaman *valores temporales de conjunto*. Los valores temporales de conjunto se pueden considerar como una matriz de los valores unitarios correspondientes. Los valores de conjunto temporales deben ser *uniformes*, es decir, deben construirse a partir de valores unitarios del mismo tipo de base y el mismo subtipo.

Los valores de secuencia temporal se convierten en *forma normal* de modo que los valores equivalentes tengan representaciones idénticas. Para ello, los valores instantáneos consecutivos se fusionan cuando es posible. Para la interpolación escalonada, tres valores instantáneos consecutivos se pueden fusionar en dos si tienen el mismo valor. Para la interpolación lineal, tres valores instantáneos consecutivos se pueden fusionar en dos si las funciones lineales que definen la evolución de los valores son las mismas. Ejemplos de transformación en forma normal son los siguientes.

```
SELECT tint '[1@2001-01-01, 2@2001-01-03, 2@2001-01-04, 2@2001-01-05)';
-- "[1@2001-01-01 00:00:00+00, 2@2001-01-03 00:00:00+00, 2@2001-01-05 00:00:00+00)"
SELECT tgeompoint '[Point(1 1)@2001-01-01 08:00:00, Point(1 1)@2001-01-01 08:05:00,
  Point(1 1)@2001-01-01 08:10:00)';
-- "[Point(1 1)@2001-01-01 08:00:00, Point(1 1)@2001-01-01 08:10:00)"
SELECT tfloats(ARRAY[tfloat '[1@2001-01-01, 2@2001-01-03, 3@2001-01-05]']);
-- "{[1@2001-01-01 00:00:00+00, 3@2001-01-05 00:00:00+00]}"
SELECT tgeompoint '[Point(1 1)@2001-01-01 08:00:00, Point(2 2)@2001-01-01 08:05:00,
  Point(3 3)@2001-01-01 08:10:00]';
-- "[Point(1 1)@2001-01-01 08:00:00, Point(3 3)@2001-01-01 08:10:00]"
```

De manera similar, los valores del conjunto de secuencias temporales se convierten en forma normal. Para ello, los valores de secuencia consecutivos se fusionan cuando es posible. Ejemplos de transformación en forma normal son los siguientes.

```
SELECT tints(ARRAY[tint '[1@2001-01-01, 1@2001-01-03)', '[2@2001-01-03, 2@2001-01-05)']);
-- '{[1@2001-01-01 00:00:00+00, 2@2001-01-03 00:00:00+00, 2@2001-01-05 00:00:00+00)}'
SELECT tfloats(ARRAY[tfloat '[1@2001-01-01, 2@2001-01-03)',
  '[2@2001-01-03, 3@2001-01-05]']);
-- '{[1@2001-01-01 00:00:00+00, 3@2001-01-05 00:00:00+00]}'
SELECT tfloats(ARRAY[tfloat '[1@2001-01-01, 3@2001-01-05)', '[3@2001-01-05]']);
-- '{[1@2001-01-01 00:00:00+00, 3@2001-01-05 00:00:00+00]}'
SELECT tgeompoint '{[Point(0 0)@2001-01-01 08:00:00,
```

```
  Point(1 1)@2001-01-01 08:05:00, Point(1 1)@2001-01-01 08:10:00),
  [Point(1 1)@2001-01-01 08:10:00, Point(1 1)@2001-01-01 08:15:00)}';
-- "{[[Point(0 0)@2001-01-01 08:00:00, Point(1 1)@2001-01-01 08:05:00,
  Point(1 1)@2001-01-01 08:15:00)}"
SELECT tgeompoint '{[Point(1 1)@2001-01-01 08:00:00, Point(2 2)@2001-01-01 08:05:00),
  [Point(2 2)@2001-01-01 08:05:00, Point(3 3)@2001-01-01 08:10:00]}';
-- "{[Point(1 1)@2001-01-01 08:00:00, Point(3 3)@2001-01-01 08:10:00]}"
SELECT tgeompoint '{[Point(1 1)@2001-01-01 08:00:00, Point(3 3)@2001-01-01 08:10:00),
  [Point(3 3)@2001-01-01 08:10:00]}';
-- "{[Point(1 1)@2001-01-01 08:00:00, Point(3 3)@2001-01-01 08:10:00]}"
```

Los tipos temporales soportan *modificadores de typo* (o `typmod` en terminología de PostgreSQL), que especifican información adicional para la definición de una columna. Por ejemplo, en la siguiente definición de tabla:

```
CREATE TABLE Department(DeptNo integer, DeptName varchar(25), NoEmps tint(Sequence));
```

el modificador de tipo para el tipo `varchar` es el valor 25, que indica la longitud máxima de los valores de la columna, mientras que el modificador de tipo para el tipo `tint` es la cadena de caracteres `Sequence`, que restringe el subtipo de los valores de la columna para que sean secuencias. En el caso de tipos alfanuméricos temporales (es decir, `tbool`, `tint`, `tfloat` y `ttext`), los valores posibles para el modificador de tipo son `Instant`, `InstantSet`, `Sequence` y `SequenceSet`. Si no se especifica ningún modificador de tipo para una columna, se permiten valores de cualquier subtipo.

Por otro lado, en el caso de tipos de puntos temporales (es decir, `tgeompoint` o `tgeogpoint`) el modificador de tipo se puede utilizar para especificar el subtipo, la dimensionalidad y/o el identificador de referencia espacial (SRID). Por ejemplo, en la siguiente definición de tabla:

```
CREATE TABLE Flight(FlightNo integer, Route tgeogpoint(Sequence, PointZ, 4326));
```

el modificador de typo para el tipo `tgeogpoint` se compone de tres valores, el primero indica el subtipo como arriba, el segundo el tipo espacial de las geografías que componen el punto temporal y el último el SRID de las geografías que componen. Para los puntos temporales, los valores posibles para el primer argumento del modificador de tipo son los anteriores, los del segundo argumento son `Point` o `PointZ` y los del tercer argumento son SRID válidos. Los tres argumentos son opcionales y si alguno de ellos no se especifica para una columna, se permiten valores de cualquier subtipo, dimensionalidad y/o SRID.

Cada tipo temporal está asociado a otro tipo, conocido como su *cuadro delimitador*, que representan su extensión en la dimensión de valor y/o tiempo. El cuadro delimitador de los distintos tipos temporales es el siguiente:

- El tipo `period` para los tipos `tbool` y `ttext`, donde solo se considera la extensión temporal.
- El tipo `tbox` (temporal box) par los tipos `tint` y `tfloat`, donde la extensión del valor se define en la dimensión X y la extensión temporal en la dimensión T.
- El tipo `stbox` (spatiotemporal box) para los tipos `tgeompoint` y `tgeogpoint`, donde la extensión espacial se define en las dimensiones X, Y y Z y la extensión temporal en la dimensión T.

Un amplio conjunto de funciones y operadores son disponibles para realizar varias operaciones en los tipos temporales. Estos se explican en el Capítulo 5. Algunas de estas operaciones, en particular las relacionadas con índices, manipulan cuadros delimitadores por razones de eficiencia.

3.1. Ejemplos de tipos temporales

A continuación se dan ejemplos de uso de tipos alfanuméricos temporales.

```
CREATE TABLE Department(DeptNo integer, DeptName varchar(25), NoEmps tint);
INSERT INTO Department VALUES
  (10, 'Research', tint '[10@2012-01-01, 12@2012-04-01, 12@2012-08-01)'),
  (20, 'Human Resources', tint '[4@2012-02-01, 6@2012-06-01, 6@2012-10-01)');
CREATE TABLE Temperature(RoomNo integer, Temp tfloat);
INSERT INTO Temperature VALUES
  (1001, tfloat '{18.5@2012-01-01 08:00:00, 20.0@2012-01-01 08:10:00}'),
  (2001, tfloat '{19.0@2012-01-01 08:00:00, 22.5@2012-01-01 08:10:00}');
-- Valor en una marca de tiempo
SELECT RoomNo, valueAtTimestamp(Temp, '2012-01-01 08:10:00')
FROM temperature;
-- 1001;
   2001;22.5
-- Restricción a un valor
SELECT DeptNo, atValue(NoEmps, 10)
FROM Department;
-- 10;"[10@2012-01-01 00:00:00+00, 10@2012-04-01 00:00:00+00)"
   20; NULL
-- Restricción a un período
SELECT DeptNo, atPeriod(NoEmps, '[2012-01-01, 2012-04-01]')
FROM Department;
-- 10;"[10@2012-01-01 00:00:00+00, 12@2012-04-01 00:00:00+00]"
   20;"[4@2012-02-01 00:00:00+00, 4@2012-04-01 00:00:00+00]"
-- Comparación temporal
SELECT DeptNo, NoEmps #<= 10
FROM Department;
-- 10;"[t@2012-01-01 00:00:00+00, f@2012-04-01 00:00:00+00, f@2012-08-01 00:00:00+00)"
   20;"[t@2012-04-02 00:00:00+00, t@2012-10-01 00:00:00+00)"
-- Agregación temporal
SELECT tsum(NoEmps)
FROM Department;
-- "{[10@2012-01-01 00:00:00+00, 14@2012-02-01 00:00:00+00, 16@2012-04-01 00:00:00+00,
  18@2012-06-01 00:00:00+00, 6@2012-08-01 00:00:00+00, 6@2012-10-01 00:00:00+00)}"
```

A continuación se dan ejemplos de uso de tipos de puntos temporales.

```
CREATE TABLE Trips(CarId integer, TripId integer, Trip tgeompoint);
INSERT INTO Trips VALUES
  (10, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:00:00, Point(2 0)@2012-01-01 08:10:00,
Point(2 1)@2012-01-01 08:15:00)}'),
  (20, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:05:00, Point(1 1)@2012-01-01 08:10:00,
  Point(3 3)@2012-01-01 08:20:00)}');
-- Valor en una marca de tiempo
SELECT CarId, ST_AsText(valueAtTimestamp(Trip, timestamptz '2012-01-01 08:10:00'))
FROM Trips;
-- 10;"POINT(2 0)"
   20;"POINT(1 1)"
-- Restricción a un valor
SELECT CarId, asText(atValue(Trip, 'Point(2 0)'))
FROM Trips;
-- 10;"{"[POINT(2 0)@2012-01-01 08:10:00+00]"}"
   20; NULL
-- Restricción a un período
SELECT CarId, asText(atPeriod(Trip, '[2012-01-01 08:05:00,2012-01-01 08:10:00]'))
FROM Trips;
-- 10;"{[POINT(1 0)@2012-01-01 08:05:00+00, POINT(2 0)@2012-01-01 08:10:00+00]}"
   20;"{[POINT(0 0)@2012-01-01 08:05:00+00, POINT(1 1)@2012-01-01 08:10:00+00]}"
-- Distancia temporal
SELECT T1.CarId, T2.CarId, T1.Trip <-> T2.Trip
FROM Trips T1, Trips T2
```

```
WHERE T1.CarId < T2.CarId;
-- 10;20;"{[1@2012-01-01 08:05:00+00, 1.4142135623731@2012-01-01 08:10:00+00,
  1@2012-01-01 08:15:00+00)}"
```

3.2. Validez de los tipos temporales

Los valores de los tipos temporales deben satisfacer varias restricciones para que estén bien definidos. Estas restricciones se dan a continuación.

- Las restricciones en el tipo base y el tipo `timestamptz` deben satisfacerse.
- Un valor de secuencia debe estar compuesto por al menos un valor de instante.
- Un valor de secuencia instantánea debe tener límites superior e inferior inclusivos.
- En un valor de secuencia, las marcas de tiempo de los instantes que la componen deben ser diferentes y estar ordenadas.
- En un valor de secuencia con interpolación escalonada, los dos últimos valores deben ser iguales si el límite superior es exclusivo.
- Un valor de conjunto debe estar compuesto por al menos un valor unitario.
- En un valor de conjunto de instantes, los instantes que lo componen deben ser diferentes y ordenados. Esto implica que la extensión temporal de un valor de conjunto de instantes es un conjunto ordenado de valores de `timestamptz` sin duplicados.
- En un valor de conjunto de secuencias, los valores de las secuencias componentes no deben superponerse y deben estar ordenados. Esto implica que la extensión temporal de un valor de conjunto de secuencias es un conjunto ordenado de períodos disjuntos.

Se genera un error cuando no se satisface una de estas restricciones. Ejemplos de valores temporales incorrectos son los siguientes.

```
-- Valor del tipo de base incorrecto
SELECT tbool '1.5@2001-01-01 08:00:00';
-- Valor del tipo base no es un punto
SELECT tgeompoint 'Linestring(0 0,1 1)@2001-01-01 08:05:00';
-- Marca de tiempo incorrecta
SELECT tint '2@2001-02-31 08:00:00';
-- Secuencia vacía
SELECT tint '';
-- Límites incorrectos para la secuencia instantánea
SELECT tint '[1@2001-01-01 09:00:00)';
-- Marcas de tiempo duplicadas
SELECT tint '[1@2001-01-01 08:00:00, 2@2001-01-01 08:00:00]';
-- Marcas de tiempo desordenadas
SELECT tint '[1@2001-01-01 08:10:00, 2@2001-01-01 08:00:00]';
-- Valor final incorrecto
SELECT tint '[1@2001-01-01 08:00:00, 2@2001-01-01 08:10:00)';
-- Conjunto de secuencias vacío
SELECT tints(ARRAY[]);
-- Marcas de tiempo duplicadas
SELECT tinti(ARRAY[tint '1@2001-01-01 08:00:00', '2@2001-01-01 08:00:00']);
-- Períodos superpuestos
SELECT tints(ARRAY[tint '[1@2001-01-01 08:00:00, 1@2001-01-01 10:00:00)',
  '[2@2001-01-01 09:00:00, 2@2001-01-01 11:00:00)']);
```

Capítulo 4

Manipulación de tipos cuadro delimitador

A continuación presentamos las funciones y operadores para tipos cuadro delimitador. Estas funciones y operadores son polimórficos, es decir, sus argumentos pueden ser de varios tipos y el tipo del resultado puede depender del tipo de los argumentos. Para expresar esto, usamos la siguiente notación:

- `box` representa cualquier tipos cuadro delimitador, es decir `tbox` o `stbox`.

A continuación, especificamos con el símbolo  que la función admite puntos 3D y con el símbolo  que la función está disponible para geografías.

4.1. Entrada/salida de tipos cuadro delimitador

Un `tbox` se compone de dimensiones de valor numérico y/o de tiempo. Para cada dimensión, se dan los valores mínimos y máximos. Ejemplos de entrada de valores `tbox` son los siguientes:

```
-- Dimensiones de valor y tiempo
SELECT tbox 'TBOX((1.0, 2000-01-01), (2.0, 2000-01-02))';
-- Sólo dimensión de valor
SELECT tbox 'TBOX((1.0,), (2.0,))';
-- Sólo dimensión de tiempo
SELECT tbox 'TBOX((, 2000-01-01), (, 2000-01-02))';
```

Un `stbox` se compone de dimensiones de valor espacial y/o de tiempo, donde las coordenadas de la dimensión de valor espacial pueden ser 2D o 3D. Para cada dimensión, se dan los valores mínimos y máximos. Las coordenadas pueden ser cartesianas (planas) o geodésicas (esféricas). Se puede especificar el SRID de las coordenadas; si no es el caso, se asume un valor de 0 (desconocido) y 4326 (correspondiente a WGS84), respectivamente, para coordenadas planas y geodésicas. Ejemplos de entrada de valores `stbox` son los siguientes:

```
-- Sólo dimensión de valor con coordenadas X e Y
SELECT stbox 'STBOX((1.0, 2.0), (1.0, 2.0))';
-- Sólo dimensión de valor con coordenadas X, Y y Z
SELECT stbox 'STBOX Z((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))';
-- Dimensiones de valor (con coordenadas X e Y) y de tiempo
SELECT stbox 'STBOX T((1.0, 2.0, 2001-01-03), (1.0, 2.0, 2001-01-03))';
-- Dimensiones de valor (con coordenadas X, Y y Z) y de tiempo
SELECT stbox 'STBOX ZT((1.0, 2.0, 3.0, 2001-01-04), (1.0, 2.0, 3.0, 2001-01-04))';
-- Sólo dimensión de tiempo
SELECT stbox 'STBOX T(( , , 2001-01-03), ( , , 2001-01-03))';
```

```
-- Sólo dimensión de valores con coordenadas geodéticas X, Y y Z
SELECT stbox 'GEODSTBOX((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))';
-- Dimensiones de valor (con coordenadas geodéticas X, Y y Z) y de tiempo
SELECT stbox 'GEODSTBOX T((1.0, 2.0, 3.0, 2001-01-04), (1.0, 2.0, 3.0, 2001-01-04))';
-- Sólo dimensión temporal para cuadro geodético
SELECT stbox 'GEODSTBOX T(( , , 2001-01-03), ( , , 2001-01-03))';
-- Se indica el SRID
SELECT stbox 'SRID=5676;STBOX T((1.0, 2.0, 2001-01-04), (1.0, 2.0, 2001-01-04))';
SELECT stbox 'SRID=4326;GEODSTBOX((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))';
```

4.2. Funciones de constructor

El tipo `tbox` tiene varias funciones constructoras dependiendo de si se dan las dimensiones de valor y/o de tiempo. Estas funciones tienen dos argumentos para los valores mínimo y máximo de tipo `float` y/o dos argumentos para los valores mínimo y máximo de tipo `timestamptz` .

- Constructor para `tbox`

  `tbox(float,float): tbox`

  `tboxt(timestamptz,timestamptz): tbox`

  `tbox(float,timestamptz,float,timestamptz): tbox`

```
-- Dimensiones de valores y de tiempo
SELECT tbox(1.0, '2001-01-01', 2.0, '2001-01-02');
-- Sólo dimensión de valores
SELECT tbox(1.0, 2.0);
-- Sólo dimensión de tiempo
SELECT tboxt('2001-01-01', '2001-01-02');
```

El tipo `stbox` tiene varias funciones constructoras dependiendo de si se dan las dimensiones de coordenadas y/o de tiempo. Además, las coordenadas pueden ser 2D o 3D y pueden ser cartesianas o geodésicas. Estas funciones tienen varios argumentos para los valores de coordenadas mínimo y máximo y/o dos argumentos para el valor mínimo y el máximo de `timestamptz`. El SRID se puede especificar en un último argumento opcional. Si no se da, se asume un valor 0 (respectivamente 4326) por defecto para coordenadas planas (respectivamente geodésicas).

- Constructor para `stbox`

  `stbox(float,float,float,float,integer): stbox`

  `stbox(float,float,float,float,float,float,integer): stbox`

  `stbox(float,float,float,timestamptz,float,float,float,timestamptz,integer): stbox`

  `stboxt(timestamptz,timestamptz,integer): stbox`

  `stbox(float,float,timestamptz,float,float,timestamptz,integer): stbox`

  `stbox(geo, {timestamp,period}): stbox`

```
-- Sólo dimensión de valores con coordenadas X e Y
SELECT stbox(1.0, 2.0, 1.0, 2.0);
-- Sólo dimensión de valores con coordenadas X, Y y Z
SELECT stbox(1.0, 2.0, 3.0, 1.0, 2.0, 3.0);
-- Sólo dimensión de valores con coordenadas X, Y y Z con SRID
SELECT stbox(1.0, 2.0, 3.0, 1.0, 2.0, 3.0);
-- Dimensiones de valor (con coordenadas X e Y) y de tiempo
SELECT stboxt(1.0, 2.0, '2001-01-03', 1.0, 2.0, '2001-01-03');
```

```
-- Dimensiones de valor (con coordenadas X, Y y Z) y de tiempo
SELECT stbox(1.0, 2.0, 3.0, '2001-01-04', 1.0, 2.0, 3.0, '2001-01-04');
-- Sólo dimensión de tiempo
SELECT stboxt('2001-01-03', '2001-01-03');
-- Sólo dimensión de valores con coordenadas geodéticas X, Y y Z
SELECT geodstbox(1.0, 2.0, 3.0, 1.0, 2.0, 3.0);
-- Dimensiones de valor (con coordenadas geodéticas X, Y y Z) y de tiempo
SELECT geodstbox(1.0, 2.0, 3.0, '2001-01-04', 1.0, 2.0, 3.0, '2001-01-03');
-- Sólo dimensión de tiempo para cuadro geodético
SELECT geodstboxt('2001-01-03', '2001-01-03');
SELECT stbox(geometry 'Linestring(1 1 1,2 2 2)', period '[2012-01-03, 2012-01-05]');
-- "STBOX ZT((1,1,1,2012-01-03),(2,2,2,2012-01-05))"
SELECT stbox(geography 'Linestring(1 1 1,2 2 2)', period '[2012-01-03, 2012-01-05]');
-- "GEODSTBOX T((0.99878198,0.017449748,0.017452406,2012-01-03),
(0.99969542,0.034878239,0.034899499,2012-01-05))"
```

4.3. Conversiones de tipo

- Convertir un `tbox` a otro tipo

  `tbox::{floatrange,period}`

```
SELECT tbox 'TBOX((1,2000-01-01),(2,2000-01-02))'::floatrange;
-- "[1,2]"
SELECT tbox 'TBOX((1,2000-01-01),(2,2000-01-02))'::period;
-- "[2000-01-01, 2000-01-02]"
```

- Convertir otro tipo a un `tbox`

  `{integer,float,numeric,intrange,floatrange}::tbox`,

  `{timestamptz,timestampset,period,periodset,tint,tfloat}::tbox`

```
SELECT floatrange '(1.0, 2.0)'::tbox;
-- "TBOX((1,),(2,))"
SELECT periodset '{(2001-01-01,2001-01-02), (2001-01-03,2001-01-04)}'::tbox;
-- "TBOX((,2001-01-01),(,2001-01-04))"
```

- Convertir un `stbox` a otro tipo

  `stbox::{period,box2d,box3d}`

```
SELECT stbox 'STBOX T((1.0, 2.0, 2001-01-01), (3.0, 4.0, 2001-01-03))'::period;
-- "[2000-01-01, 2000-01-03]"
SELECT stbox 'STBOX Z((1 1 1),(3 3 3))'::box2d;
-- "BOX(1 1,3 3)"
SELECT stbox 'STBOX Z((1 1 1),(3 3 3))'::box3d;
-- "BOX3D(1 1 1,3 3 3)"
```

- Convertir otro tipo a un `stbox`

  `{geometry,geography,box2d,box3d}::stbox`

  `{timestamptz,timestampset,period,periodset,tgeompoint,tgeogpoint}::stbox`

```
SELECT geometry 'Linestring(1 1,2 2)'::stbox;
-- "STBOX((1,1),(2,2))"
SELECT periodset '{(2001-01-01,2001-01-02), (2001-01-03,2001-01-04)}'::stbox;
-- "STBOX T((,,2001-01-01),(,,2001-01-04))"
```

4.4. Funciones de accesor

- ¿Tiene dimension X?

  `hasX(box): boolean`

```
SELECT hasX(tbox 'TBOX((, 2000-01-01), (, 2000-01-03))');
-- false
SELECT hasX(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- true
```

- ¿Tiene dimension Z?

  `hasZ(stbox): boolean`

```
SELECT hasZ(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- false
```

- ¿Tiene dimension T?

  `hasT(box): boolean`

```
SELECT hasT(tbox 'TBOX((1.0, 2000-01-01), (3.0, 2000-01-03))');
-- true
SELECT hasT(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- false
```

- Obtener el valor mínimo de X

  `Xmin(box): float`

```
SELECT Xmin(tbox 'TBOX((1.0, 2000-01-01), (3.0, 2000-01-03))');
-- 1
SELECT Xmin(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- 1
```

- Obtener el valor máximo de X

  `Xmax(box): float`

```
SELECT Xmax(tbox 'TBOX((1.0, 2000-01-01), (3.0, 2000-01-03))');
-- 3
SELECT Xmax(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- 3
```

- Obtener el valor mínimo de Y

  `Ymin(stbox): float`

```
SELECT Ymin(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- 2
```

- Obtener el valor máximo de Y

  `Ymax(stbox): float`

```
SELECT Ymax(stbox 'STBOX((1.0, 2.0), (3.0, 4.0))');
-- 4
```

- Obtener el valor mínimo de Z

  `Zmin(stbox): float`

```
SELECT Zmin(stbox 'STBOX Z((1.0, 2.0, 3.0), (4.0, 5.0, 6.0))');
-- 3
```

- Obtener el valor máximo de Z

  `Zmax(stbox): float`

```
SELECT Zmax(stbox 'STBOX Z((1.0, 2.0, 3.0), (4.0, 5.0, 6.0))');
-- 6
```

- Obtener el valor mínimo de T

  `Tmin(box): timestamptz`

```
SELECT Tmin(stbox 'GEODSTBOX T(( , , 2001-01-01), ( , , 2001-01-03))');
-- "2001-01-01"
```

- Obtener el valor máximo de T

  `Tmax(box): timestamptz`

```
SELECT Tmax(stbox 'GEODSTBOX T(( , , 2001-01-01), ( , , 2001-01-03))');
-- "2001-01-03"
```

4.5. Funciones de modificación

Las funciones dadas a continuación expanden los cuadros delimitadores en la dimensión de valor y de tiempo o establecen la precisión de la dimensión de valor. Estas funciones generan un error si la dimensión correspondiente no está presente.

- Extender la dimensión de valor numérico del cuadro delimitador con un valor flotante

  `expandValue(tbox,float): tbox`

```
SELECT expandValue(tbox 'TBOX((1,2012-01-01),(2,2012-01-03))', 1);
-- "TBOX((0,2012-01-01),(3,2012-01-03))"
SELECT expandValue(tbox 'TBOX((,2012-01-01),(,2012-01-03))', 1);
-- The box must have value dimension
```

- Extender la dimensión de valor espacial del cuadro delimitador con un valor flotante

  `expandSpatial(stbox,float): stbox`

```
SELECT expandSpatial(stbox 'STBOX ZT((1,1,1,2012-01-01),(2,2,2,2012-01-03))', 1);
-- "STBOX ZT((0,0,0,2012-01-01),(3,3,3,2012-01-03))"
SELECT expandSpatial(stbox 'STBOX T((,2012-01-01),(,2012-01-03))', 1);
-- The box must have XY dimension
```

- Extender la dimensión temporal del cuadro delimitador con un intervalo de tiempo

  `expandTemporal(box,interval): box`

```
SELECT expandTemporal(tbox 'TBOX((1,2012-01-01),(2,2012-01-03))', interval '1 day');
-- "TBOX((1,2011-12-31),(2,2012-01-04))"
SELECT expandTemporal(stbox 'STBOX ZT((1,1,1,2012-01-01),(2,2,2,2012-01-03))',
  interval '1 day');
-- "STBOX ZT((1,1,1,2011-12-31),(2,2,2,2012-01-04))"
```

- Redondear el valor o las coordenadas del cuadro delimitador a un número de decimales

  `setPrecision(box,integer): box`

```
SELECT setPrecision(tbox 'TBOX((1.12345, 2000-01-01), (2.12345, 2000-01-02))', 2);
-- "TBOX((1.12,2000-01-01),(2.12,2000-01-02))"
SELECT setPrecision(stbox 'STBOX T((1.12345, 1.12345, 2000-01-01),
  (2.12345, 2.12345, 2000-01-02))', 2);
-- "STBOX T((1.12,1.12,2000-01-01),(2.12,2.12,2000-01-02))"
```

4.6. Funciones del sistema de referencia espacial

- Obtener el identificador de referencia espacial

  `SRID(stbox): integer`

```
SELECT SRID(stbox 'STBOX ZT((1.0, 2.0, 3.0, 2000-01-01), (4.0, 5.0, 6.0, 2000-01-02))');
-- 0
SELECT SRID(stbox 'SRID=5676;STBOX T((1.0, 2.0, 2000-01-01), (4.0, 5.0, 2000-01-02))');
-- 5676
SELECT SRID(geodstbox 'GEODSTBOX T((, , 2000-01-01), (, , 2000-01-02))');
-- 4326
```

- Especificar el identificador de referencia espacial

  `setSRID(stbox): stbox`

```
SELECT setSRID(stbox 'STBOX ZT((1.0, 2.0, 3.0, 2000-01-01),
  (4.0, 5.0, 6.0, 2000-01-02))', 5676);
-- "SRID=5676;STBOX ZT((1,2,3,2000-01-01),(4,5,6,2000-01-02))"
```

- Transformar a una referencia espacial diferente

  `transform(stbox,integer): stbox`

```
SELECT transform(stbox 'SRID=4326;STBOX T((2.340088, 49.400250, 2000-01-01),
  (6.575317, 51.553167, 2000-01-02))', 3812);
-- "SRID=3812;STBOX T((502773.429980817,511805.120401577,2000-01-01),
  (803028.908264815,751590.742628986,2000-01-02))"
```

4.7. Funciones aggregadas

- Extensión del cuadro delimitador

  `extent(box): box`

```
WITH boxes(b) AS (
  SELECT tbox 'TBOX((1, 2000-01-01),(3, 2000-01-03))' UNION
  SELECT tbox 'TBOX((5, 2000-01-05),(7, 2000-01-07))' UNION
  SELECT tbox 'TBOX((6, 2000-01-06),(8, 2000-01-08))'
)
SELECT extent(b) FROM boxes;
-- TBOX((1,2000-01-01 00:00:00+01),(8,2000-01-08 00:00:00+01))
WITH boxes(b) AS (
  SELECT stbox 'STBOX Z((1, 1, 1),(3, 3, 3))' UNION
  SELECT stbox 'STBOX Z((5, 5, 5),(7, 7, 7))' UNION
  SELECT stbox 'STBOX Z((6, 6, 6),(8, 8, 8))'
)
SELECT extent(b) FROM boxes;
-- STBOX Z((1,1,1),(8,8,8))
```

4.8. Operadores de comparación

Los operadores de comparación tradicionales (=, <, etc.) se pueden aplicar a tipos de cuadro delimitador. Exceptuando la igualdad y la no igualdad, los otros operadores de comparación no son útiles en el mundo real pero permiten construir índices de árbol B en tipos de cuadro delimitador. Estos operadores comparan primero los valores de tiempo y, si son iguales, comparan los valores.

- ¿Son iguales los cuadros delimitadores?

  `box = box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-01), (1, 2012-01-04))' =
  tbox 'TBOX((2, 2012-01-03), (2, 2012-01-05))';
-- false
```

- ¿Son diferentes los cuadros delimitadores?

  `box <> box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-01), (1, 2012-01-04))' <>
  tbox 'TBOX((2, 2012-01-03), (2, 2012-01-05))'
-- true
```

- ¿Es el primer cuadro delimitador menor que el segundo?

  `box < box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-01), (1, 2012-01-04))' <
  tbox 'TBOX((1, 2012-01-03), (2, 2012-01-05))'
-- true
```

- ¿Es el primer cuadro delimitador mayor que el segundo?

  `box > box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-03), (1, 2012-01-04))' >
  tbox 'TBOX((1, 2012-01-01), (2, 2012-01-05))'
-- true
```

- ¿Es el primer cuadro delimitador menor o igual que el segundo?

  `box <= box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-01), (1, 2012-01-04))' <=
  tbox 'TBOX((2, 2012-01-03), (2, 2012-01-05))'
-- true
```

- ¿Es el primer cuadro delimitador mayor o igual que el segundo?

  `box >= box: boolean`

```
SELECT tbox 'TBOX((1, 2012-01-01), (1, 2012-01-04))' >=
  tbox 'TBOX((2, 2012-01-03), (2, 2012-01-05))'
-- false
```

4.9. Operadores de conjuntos

Los operadores de conjuntos para los tipos de cuadro delimitador son la unión (+) y la intersección ($\star$). En el caso de la unión, los operandos deben tener exactamente las mismas dimensiones, de lo contrario se genera un error. Además, si los operandos no se superponen en todas las dimensiones se genera un error, ya que esto daría como resultado una caja con valores disjuntos, que no se puede representar. El operador calcula la unión en todas las dimensiones que están presentes en ambos argumentos. En el caso de intersección, los operandos deben tener al menos una dimensión común, de lo contrario se genera un error. El operador calcula la intersección en todas las dimensiones que están presentes en ambos argumentos.

- Unión de los cuadros delimitadores

  `box + box: box`

```
SELECT tbox 'TBOX((1,2001-01-01),(3,2001-01-03))' +
  tbox 'TBOX((2,2001-01-02),(4,2001-01-04))';
-- "TBOX((1,2001-01-01),(4,2001-01-04))"
SELECT stbox 'STBOX ZT((1,1,1,2001-01-01),(2,2,2,2001-01-02))' +
  stbox 'STBOX T((2,2,2001-01-01),(3,3,2001-01-03))';
-- ERROR:  Boxes must be of the same dimensionality
SELECT tbox 'TBOX((1,2001-01-01),(3,2001-01-02))' +
  tbox 'TBOX((2,2001-01-03),(4,2001-01-04))';
-- ERROR:  Result of box union would not be contiguous
```

- Intersección de los cuadros delimitadores

  `box * box: box`

```
SELECT tbox 'TBOX((1,2001-01-01),(3,2001-01-03))' *
  tbox 'TBOX((,2001-01-02),(,2001-01-04))';
-- "TBOX((,2001-01-02),(,2001-01-03))"
SELECT stbox 'STBOX ZT((1,1,1,2001-01-01),(3,3,3,2001-01-02))' *
  stbox 'STBOX((2,2),(4,4))';
-- "STBOX((2,2),(3,3))"
```

4.10. Operadores topológicos

Hay cinco operadores topológicos: superposición (`&&`), contiene (`@>`), es contenido (`<@`), mismo (`~=`) y adyacente (`-|-`). Los operadores verifican la relación topológica entre los cuadros delimitadores teniendo en cuenta la dimensión de valor y/o de tiempo para todas las dimensiones que estén presentes en ambos argumentos.

Los operadores topológicos para los cuadros delimitadores se dan a continuación.

- ¿Se superponen los cuadros delimitadores?

  `box && box: boolean`

```
SELECT tbox 'TBOX((1,2001-01-01),(3,2001-01-03))' &&
  tbox 'TBOX((2,2001-01-02),(4,2001-01-04))';
-- true
SELECT stbox 'STBOX T((1,1,2001-01-01),(2,2,2001-01-02))'&&
  stbox 'STBOX T((,2001-01-02),(,2001-01-02))';
-- true
```

- ¿Contiene el primer cuadro delimitador el segundo?

  `box @> box: boolean`

```
SELECT tbox 'TBOX((1,2001-01-01),(4,2001-01-04))' @>
  tbox 'TBOX((2,2001-01-01),(3,2001-01-02))';
-- true
SELECT stbox 'STBOX Z((1,1,1),(3,3,3))' @>
  stbox 'STBOX T((1,1,2001-01-01),(2,2,2001-01-02))';
-- true
```

- ¿Está el primer cuadro delimitador contenido en el segundo?

  `box <@ box: boolean`

```
SELECT tbox 'TBOX((1,2001-01-01),(2,2001-01-02))' <@
  tbox 'TBOX((1,2001-01-01),(2,2001-01-02))';
-- true
SELECT stbox 'STBOX T((1,1,2001-01-01),(2,2,2001-01-02))' <@
  stbox 'STBOX ZT((1,1,1,2001-01-01),(2,2,2,2001-01-02))';
-- true
```

- ¿Son los cuadros delimitadores iguales en sus dimensiones comunes?

  `box ~= box: boolean`

```
SELECT tbox 'TBOX((1,2001-01-01),(2,2001-01-02))' ~=
  tbox 'TBOX((,2001-01-01),(,2001-01-02))';
-- true
SELECT stbox 'STBOX T((1,1,2001-01-01),(3,3,2001-01-03))' ~=
  stbox 'STBOX Z((1,1,1),(3,3,3))';
-- true
```

- ¿Son los cuadros delimitadores adyacentes?

  `box -|- box: boolean`

  Dos cuadros delimitadores son adyacentes si comparten *n* dimensiones y si su intersección tiene como máximo *n*-1 dimensiones.

```
SELECT tbox 'TBOX((1,2001-01-01),(2,2001-01-02))' -|-
  tbox 'TBOX((,2001-01-02),(,2001-01-03))';
-- true
SELECT stbox 'STBOX T((1,1,2001-01-01),(3,3,2001-01-03))' -|-
  stbox 'STBOX T((2,2,2001-01-03),(4,4,2001-01-04))';
-- true
```

4.11. Operadores de posición relativa

Estos operadores consideran la posición relativa de los cuadros delimitadores. Los operadores `<<`, `>>`, `&<` y `&>` consideran el valor X para el tipo `tbox` y las coordenadas X para el tipo `stbox`, los operadores `<<|`, `|>>`, `&<|` y `|&>` consideran las coordenadas Y para el tipo `stbox`, los operadores `<</`, `/>>`, `&</` y `/&>` consideran las coordenadas Z para el tipo `stbox` y los operadores `<<#`, `#>>`, `#&<` y `#&>` consideran la dimensión de tiempo para los tipos `tbox` y `stbox`. Los operadores generan un error si ambos cuadros delimitadores no tienen la dimensión requerida.

Los operadores para la dimensión de valor numérico del tipo `tbox` se dan a continuación.

- ¿Son los valores X del primer cuadro delimitador estrictamente menores que los del segundo?

  `tbox << tbox: boolean`

```
SELECT tbox 'TBOX((1,2012-01-01),(2,2012-01-02))' <<
  tbox 'TBOX((3,2012-01-03),(4,2012-01-04))';
-- true
SELECT tbox 'TBOX((1,2012-01-01),(2,2012-01-02))' <<
  tbox 'TBOX((,2012-01-03),(,2012-01-04))';
-- ERROR:  Boxes must have X dimension
```

- ¿Son los valores X del primer cuadro delimitador estrictamente mayores que los del segundo?

  `tbox >> tbox: boolean`

```
SELECT tbox 'TBOX((3,2012-01-03),(4,2012-01-04))' >>
  tbox 'TBOX((1,2012-01-01),(2,2012-01-02))';
-- true
```

- ¿No son los valores X del primer cuadro delimitador mayores que los del segundo?

  `tbox &< tbox: boolean`

```
SELECT tbox 'TBOX((1,2012-01-01),(4,2012-01-04))' &<
  tbox 'TBOX((3,2012-01-03),(4,2012-01-04))';
-- true
```

- ¿No son los valores X del primer cuadro delimitador menores que los del segundo?

  `tbox &> tbox: boolean`

```
SELECT tbox 'TBOX((1,2012-01-01),(2,2012-01-02))' &>
  tbox 'TBOX((1,2012-01-01),(4,2012-01-04))';
-- true
```

Los operadores para la dimensión de valor espacial del tipo `stbox` se dan a continuación.

- ¿Están los valores X del primer cuadro delimitador estrictamente a la izquierda de los del segundo?

  `stbox << stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' << stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
```

- ¿Están los valores X del primer cuadro delimitador estrictamente a la derecha de los del segundo?

  `stbox >> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' >> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
```

- ¿No están los valores X del primer cuadro delimitador a la derecha de los del segundo?

  `stbox &< stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &< stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
```

- ¿No están los valores X del primer cuadro delimitador a la izquierda de los del segundo?

  `stbox &> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' &> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
```

- ¿Están los valores de Y del primer cuadro delimitador estrictamente debajo de los del segundo?

  `stbox <<| stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' <<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
```

- ¿Están los valores de Y del primer cuadro delimitador estrictamente arriba de los del segundo?

  `stbox |>> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' |>> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
```

- ¿No están los valores de Y del primer cuadro delimitador arriba de los del segundo?
  `stbox &<| stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
```

- ¿No están los valores de Y del primer cuadro delimitador debajo de los del segundo?
  `stbox |&> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' |&> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- false
```

- ¿Están los valores Z del primer cuadro delimitador estrictamente delante de los del segundo?
  `stbox <<| stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' <<| stbox 'STBOX Z((3,3,3),(4,4,4))';
```

- ¿Están los valores Z del primer cuadro delimitador estrictamente detrás de los del segundo?
  `stbox />> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' />> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
```

- ¿No están los valores Z del primer cuadro delimitador detrás de los del segundo?
  `stbox &<| stbox: boolean`

```
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
```

- ¿No están los valores Z del primer cuadro delimitador delante de los del segundo?
  `stbox /&> stbox: boolean`

```
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' /&> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
```

Los operadores para la dimensión temporal de los tipos `tbox` y `stbox` son los siguientes.

- ¿Son los valores de T del primer cuadro delimitador estrictamente anteriores a los del segundo?
  `box <<# box: boolean`

```
SELECT tbox 'TBOX((1,2000-01-01),(2,2000-01-02))' <<#
  tbox 'TBOX((3,2000-01-03),(4,2000-01-04))';
-- true
```

- ¿Son los valores de T del primer cuadro delimitador estrictamente posteriores a los del segundo?

  `box #>> box: boolean`

```
SELECT stbox 'STBOX T((3,3,2000-01-03),(4,4,2000-01-04))'  #>>
  stbox 'STBOX T((1,1,2000-01-01),(2,2,2000-01-02))';
-- true
```

- ¿No son los valores de T del primer cuadro delimitador posteriores a los del segundo?

  `box &<# box: boolean`

```
SELECT tbox 'TBOX((1,2000-01-01),(4,2000-01-04))' &<#
  tbox 'TBOX((3,2000-01-03),(4,2000-01-04))';
-- true
```

- ¿No son los valores de T del primer cuadro delimitador anteriores a los del segundo?

  `box #&> box: boolean`

```
SELECT stbox 'STBOX T((1,1,2000-01-01),(3,3,2000-01-03))' #&>
  stbox 'STBOX T((3,3,2000-01-03),(4,4,2000-01-04))';
-- true
```

4.12. Indexación de los tipos cuadro delimitador

Se pueden crear índices GiST y SP-GiST para columnas de tablas de los tipos `tbox` y `stbox`. Un ejemplo de creación de un índice GiST en una columna `Box` de tipo `stbox` en una tabla `Trips` es el siguiente:

```
CREATE TABLE Trips(TripID integer PRIMARY KEY, Trip tgeompoint, Box stbox);
CREATE INDEX Trips_Box_Idx ON Trips USING GIST(Box);
```

Un índice GiST o SP-GiST puede acelerar las consultas que involucran a los siguientes operadores: `&&`, `<@`, `@>`, `~=`, `-|-`, `<<`, `>>`, `&<`, `&>`, `<<|`, `|>>`, `&<|`, `|&>`, `<</`, `/>>`, `&</`, `/&>`, `<<#`, `#>>`, `&<#` y `#&>`.

Además, se pueden crear índices de árbol B para columnas de tablas de un tipo cuadro delimitador. Para estos tipos de índices, básicamente la única operación útil es la igualdad. Hay un orden de clasificación de árbol B definido para valores de tipos cuadro delimitador con los correspondientes operadores < y >, pero el orden es bastante arbitrario y no suele ser útil en el mundo real. El soporte de árbol B está destinado principalmente a permitir la clasificación interna en las consultas, en lugar de la creación de índices reales.

Capítulo 5

Manipulación de tipos temporales

A continuación presentamos las funciones y operadores para tipos temporales. Estas funciones y operadores son polimórficos, es decir, sus argumentos pueden ser de varios tipos y el tipo del resultado puede depender del tipo de los argumentos. Para expresar esto, usamos la siguiente notación:

- `ttype` representa cualquier tipo temporal,
- `time` representa cualquier tipo de tiempo, es decir, `timestamptz`, `period`, `timestampset` o `periodset`,
- `tnumber` representa cualquier tipo de número temporal, es decir, `tint` o `tfloat`,
- `torder` representa cualquier tipo temporal cuyo tipo de base tiene definido un orden total, es decir, `tint`, `tfloat` o `ttext`,
- `tpoint` representa un tipo de punto temporal, es decir, `tgeompoint` o `tgeogpoint`,
- `ttype_inst` representa cualquier tipo temporal con subtipo instante,
- `ttype_instset` representa cualquier tipo temporal con subtipo conjunto de instantes,
- `ttype_seq` representa cualquier tipo temporal con subtipo secuencia,
- `tdisc_seq` representa cualquier tipo temporal con subtipo secuencia y con tipo de base discreto,
- `tcont_seq` representa cualquier tipo temporal con subtipo secuencia y con tipo de base continuo,
- `ttype_seqset` representa cualquier tipo temporal con subtipo conjunto de secuencias
- `base` representa cualquier tipo de base de un tipo temporal, es decir, `boolean`, `integer`, `float`, `text`, `geometry` o `geography`,
- `number` representa cualquier tipo de base numérico, es decir, `integer` o `float`,
- `numrange` representa cualquier tipo de rango numérico, es decir, `intrange` o `floatrange`,
- `geo` representa los tipos `geometry` o `geography`,
- `point` representa los tipos `geometry` o `geography` restringidos a un punto.
- `type[]` representa una matriz de `type`.

Una forma común de generalizar las operaciones tradicionales a los tipos temporales es aplicar la operación en *cada instante*, lo que da un valor temporal como resultado. En ese caso, la operación sólo se define en la intersección de las extensiones temporales de los operandos; si las extensiones temporales son disjuntas, el resultado es nulo. Por ejemplo, los operadores de comparación temporal, como `#<`, determinan si los valores tomados por sus operandos en cada instante satisfacen la condición y devuelven un booleano temporal. A continuación se dan ejemplos de las diversas generalizaciones de los operadores.

```
-- Comparación temporal
SELECT tint '[2@2001-01-01, 2@2001-01-03)' #< tfloat '[1@2001-01-01, 3@2001-01-03)';
-- "{[f@2001-01-01, f@2001-01-02], (t@2001-01-02, t@2001-01-03)}"
SELECT tfloat '[1@2001-01-01, 3@2001-01-03)' #< tfloat '[3@2001-01-03, 1@2001-01-05)';
-- NULL
-- Adición temporal
SELECT tint '[1@2001-01-01, 1@2001-01-03)' + tint '[2@2001-01-02, 2@2001-01-05)';
-- "[3@2001-01-02, 3@2001-01-03)"
-- Intersección temporal
SELECT tintersects(tgeompoint '[Point(0 1)@2001-01-01, Point(3 1)@2001-01-04)',
geometry 'Polygon((1 0,1 2,2 2,2 0,1 0))');
-- "{[f@2001-01-01, t@2001-01-02, t@2001-01-03], (f@2001-01-03, f@2001-01-04]}"
-- Distancia temporal
SELECT tgeompoint '[Point(0 0)@2001-01-01 08:00:00, Point(0 1)@2001-01-03 08:10:00)' <->
tgeompoint '[Point(0 0)@2001-01-02 08:05:00, Point(1 1)@2001-01-05 08:15:00)';
-- "[0.5@2001-01-02 08:05:00+00, 0.7451840337945 57@2001-01-03 08:10:00+00)"
```

Otro requisito común es determinar si los operandos satisfacen *alguna vez* o *siempre* una condición con respecto a una operación. Estos se pueden obtener aplicando los operadores de comparación alguna vez/siempre. Estos operadores se indican anteponiendo los operadores de comparación tradicionales con, respectivamente, `?` (alguna vez) y `%` (siempre). A continuación se dan ejemplos de operadores de comparación alguna vez y siempre.

```
-- ¿Se cruzan los operandos alguna vez?
SELECT tintersects(tgeompoint '[Point(0 1)@2001-01-01, Point(3 1)@2001-01-04)',
  geometry 'Polygon((1 0,1 2,2 2,2 0,1 0))') ?= true;
-- true
-- ¿Se cruzan los operandos siempre?
SELECT tintersects(tgeompoint '[Point(0 1)@2001-01-01, Point(3 1)@2001-01-04)',
  geometry 'Polygon((0 0,0 2,4 2,4 0,0 0))') %= true;
-- true
-- ¿Es el operando izquierdo alguna vez menor que el derecho?
SELECT (tfloat '[1@2001-01-01, 3@2001-01-03)' #<
  tfloat '[3@2001-01-01, 1@2001-01-03)') ?= true;
-- true
-- ¿Es el operando izquierdo siempre menor que el derecho?
SELECT (tfloat '[1@2001-01-01, 3@2001-01-03)' #<
  tfloat '[2@2001-01-01, 4@2001-01-03)') %= true;
-- true
```

Por razones de eficiencia, algunas operaciones comunes con la semántica alguna vez o siempre se proporcionan de forma nativa. Por ejemplo, la función `intersects` determina si hay un instante en el que los dos argumentos se cruzan espacialmente.

A continuación describimos las funciones y operadores para tipos temporales. Para mayor concisión, en los ejemplos usamos principalmente secuencias compuestas por dos instantes.

5.1. Entrada/salida de tipos temporales

Un valor de instante es un par de la forma `v@t`, donde `v` es un valor del tipo de base y `t` es un valor de `timestamptz`. Un valor de secuencia es un conjunto de valores `v1@t1,...,vn@tn` delimitado por límites superior e inferior, que pueden ser inclusivo (representados por '`[`' y '`]`') o exclusivos (representados por '`(`' y '`)`'). Ejemplos de entrada de valores temporales unitarios son los siguientes:

```
SELECT tbool 'true@2001-01-01 08:00:00';
SELECT tint '1@2001-01-01 08:00:00';
SELECT tfloat '1.5@2001-01-01 08:00:00';
```

```
SELECT ttext 'AAA@2001-01-01 08:00:00';
SELECT tgeompoint 'Point(0 0)@2017-01-01 08:00:05';
SELECT tgeogpoint 'Point(0 0)@2017-01-01 08:00:05';
SELECT tbool '[true@2001-01-01 08:00:00, true@2001-01-03 08:00:00]';
SELECT tint '[1@2001-01-01 08:00:00, 1@2001-01-03 08:00:00]';
SELECT tfloat '[2.5@2001-01-01 08:00:00, 3@2001-01-03 08:00:00, 1@2001-01-04 08:00:00]';
SELECT tfloat '[1.5@2001-01-01 08:00:00]';  -- Instant sequence
SELECT ttext '[BBB@2001-01-01 08:00:00, BBB@2001-01-03 08:00:00]';
SELECT tgeompoint '[Point(0 0)@2017-01-01 08:00:00, Point(0 0)@2017-01-01 08:05:00)';
SELECT tgeogpoint '[Point(0 0)@2017-01-01 08:00:00, Point(0 1)@2017-01-01 08:05:00,
  Point(0 0)@2017-01-01 08:10:00)';
```

La extensión temporal de un valor de instante es un sólo instante, mientras que la extensión temporal de un valor de secuencia es un período definido por el primer y último instantes, así como por los límites superior e inferior.

Un valor temporal de conjunto es un conjunto `{v1,...,vn}` donce cada `vi` es un valor unitario del tipo correspondiente. Ejemplos de entrada de valores temporales de conjunto son los siguientes:

```
SELECT tbool '{true@2001-01-01 08:00:00, false@2001-01-03 08:00:00}';
SELECT tint '{1@2001-01-01 08:00:00, 2@2001-01-03 08:00:00}';
SELECT tfloat '{1.0@2001-01-01 08:00:00, 2.0@2001-01-03 08:00:00}';
SELECT ttext '{AAA@2001-01-01 08:00:00, BBB@2001-01-03 08:00:00}';
SELECT tgeompoint '{Point(0 0)@2017-01-01 08:00:00, Point(0 1)@2017-01-02 08:05:00}';
SELECT tgeogpoint '{Point(0 0)@2017-01-01 08:00:00, Point(0 1)@2017-01-02 08:05:00}';
SELECT tbool '{[false@2001-01-01 08:00:00, false@2001-01-03 08:00:00),
  [true@2001-01-03 08:00:00], (false@2001-01-04 08:00:00, false@2001-01-06 08:00:00]}';
SELECT tint '{[1@2001-01-01 08:00:00, 1@2001-01-03 08:00:00),
  [2@2001-01-04 08:00:00, 3@2001-01-05 08:00:00, 3@2001-01-06 08:00:00]}';
SELECT tfloat '{[1@2001-01-01 08:00:00, 2@2001-01-03 08:00:00, 2@2001-01-04 08:00:00,
  3@2001-01-06 08:00:00]}';
SELECT ttext '{[AAA@2001-01-01 08:00:00, BBB@2001-01-03 08:00:00, BBB@2001-01-04 08:00:00),
  [CCC@2001-01-05 08:00:00, CCC@2001-01-06 08:00:00]}';
SELECT tgeompoint '{[Point(0 0)@2017-01-01 08:00:00, Point(0 1)@2017-01-01 08:05:00),
  [Point(0 1)@2017-01-01 08:10:00, Point(1 1)@2017-01-01 08:15:00)}';
SELECT tgeogpoint '{[Point(0 0)@2017-01-01 08:00:00, Point(0 1)@2017-01-01 08:05:00),
  [Point(0 1)@2017-01-01 08:10:00, Point(1 1)@2017-01-01 08:15:00)}';
```

La extensión temporal de un valor de conjunto de instantes es un conjunto de marcas de tiempo, mientras que la extensión temporal de un valor de conjunto de secuencias es un conjunto de períodos.

Los valores de secuencia o conjunto de secuencias cuyo tipo de base es continuo pueden especificar que la interpolación es escalonada. Si no se especifica, se supone que la interpolación es lineal por defecto.

```
-- Interpolación lineal por defecto
SELECT tfloat '[2.5@2001-01-01, 3@2001-01-03, 1@2001-01-04]';
SELECT tgeompoint '{[Point(2.5 2.5)@2001-01-01, Point(3 3)@2001-01-03],
  [Point(1 1)@2001-01-04, Point(1 1)@2001-01-04]}';
-- Interpolación escalonada
SELECT tfloat 'Interp=Stepwise;[2.5@2001-01-01, 3@2001-01-03, 1@2001-01-04]';
SELECT tgeompoint 'Interp=Stepwise;{[Point(2.5 2.5)@2001-01-01, Point(3 3)@2001-01-03],
  [Point(1 1)@2001-01-04, Point(1 1)@2001-01-04]}';
```

Para los valores de subtipo conjunto de secuencias se supone que todas las secuencias componentes tienen la misma interpolación, ya sea por escalonada o lineal, como en los ejemplos anteriores.

Para los puntos temporales, es posible especificar el identificador de referencia espacial (SRID) utilizando la representación extendida de texto conocido (EWKT) de la siguiente manera:

```
SELECT tgeompoint 'SRID=5435;[Point(0 0)@2000-01-01,Point(0 1)@2000-01-02]'
```

Todas las geometrías componentes serán entonces del SRID dado. Además, cada geometría componente puede especificar su SRID con el formato EWKT como en el siguiente ejemplo

```
SELECT tgeompoint '[SRID=5435;Point(0 0)@2000-01-01,SRID=5435;Point(0 1)@2000-01-02]'
```

Se genera un error si las geometrías componentes no están todas en el mismo SRID o si el SRID de una geometría componente es diferente al del punto temporal.

```
SELECT tgeompoint '[SRID=5435;Point(0 0)@2000-01-01,SRID=4326;Point(0 1)@2000-01-02]';
-- ERROR: Geometry SRID (4326) does not match temporal type SRID (5435)
SELECT tgeompoint 'SRID=5435;[SRID=4326;Point(0 0)@2000-01-01,
  SRID=4326;Point(0 1)@2000-01-02]'
-- ERROR: Geometry SRID (4326) does not match temporal type SRID (5435)
```

5.2. Funciones de constructor

Cada tipo temporal tiene una función de constructor con el mismo nombre que el tipo y un sufijo para el subtipo, donde los sufijos '`_inst`', '`_instset`', '`_seq`' y '`_seqset`' corresponden, respectivamente, a los subtipos instante, conjunto de instantes, secuencia y conjunto de secuencias. Ejemplos son `tint_seq` o `tgeompoint_seqset`. El uso de la función de constructor suele ser más conveniente que escribir una constante literal.

- Un primer conjunto de funciones tiene dos argumentos, un tipo base y un tipo de tiempo, donde el último es un valor de `timestamptz`, `timestampset`, `period` o `periodset` para construir, respectivamente, un valor de subtipo instante, conjunto de instantes, secuencia o conjunto de secuencias. Las funciones para valores de secuencia o de conjunto de secuencias con tipo base continuo tienen además un tercer argumento opcional que es un booleano para indicar si el valor temporal resultante tiene interpolación lineal o no. Por defecto, este argumento es verdadero si no se especifica.
- Otro conjunto de funciones para valores de conjunto de instantes tiene un solo argumento, que es una matriz de valores de instante correspondientes.
- Otro conjunto de funciones para valores de secuencia tiene un argumento para la matriz de valores de instante correspondiente y dos argumentos booleanos opcionales que indican, respectivamente, si los límites izquierdo y derecho son inclusivos o exclusivos. Si estos argumentos no se especifican, se supone que son verdaderos por defecto. Además, las funciones para valores de secuencia con tipo base continuo tienen un argumento booleano adicional que indica si la interpolación es lineal o no. Si este argumento no se especifica, se asume que es verdadero por defecto.
- Otro conjunto de funciones para valores de conjuntos de secuencias tiene un único argumento, que es una matriz de valores de secuencia correspondientes. Para valores de secuencia con tipo de base continuo, la interpolación del valor temporal resultante depende de la interpolación de las secuencias que la componen. Se genera un error si las secuencias que componen la matriz tienen interpolación diferentes.

A continuación, damos las funciones de constructor para los distintos subtipos.

- Constructor para tipos temporales de subtipo instante

  `ttype_inst(base,timestamptz): ttype_inst`

```
SELECT tbool_inst(true, '2001-01-01');
SELECT tint_inst(1, '2001-01-01');
```

- Constructor para tipos temporales de subtipo conjunto de instantes

  `ttype_instset(base,timestampset): ttype_instset`

  `ttype_instset(ttype_inst[]): ttype_instset`

```
SELECT tfloat_instset(1.5, '{2001-01-01, 2001-01-02}');
SELECT ttext_instset('AAA', '{2001-01-01, 2001-01-02}');
SELECT tbool_instset(ARRAY[tbool 'true@2001-01-01 08:00:00','false@2001-01-01 08:05:00']);
SELECT tint_instset(ARRAY[tint '1@2001-01-01 08:00:00', '2@2001-01-01 08:05:00']);
SELECT tfloat_instset(ARRAY[tfloat '1.0@2001-01-01 08:00:00', '2.0@2001-01-01 08:05:00']);
SELECT ttext_instset(ARRAY[ttext 'AAA@2001-01-01 08:00:00', 'BBB@2001-01-01 08:05:00']);
SELECT tgeompoint_instset(ARRAY[tgeompoint 'Point(0 0)@2001-01-01 08:00:00',
  'Point(0 1)@2001-01-01 08:05:00', 'Point(1 1)@2001-01-01 08:10:00']);
SELECT tgeogpoint_instset(ARRAY[tgeogpoint 'Point(1 1)@2001-01-01 08:00:00',
  'Point(2 2)@2001-01-01 08:05:00']);
```

- Constructor para tipos temporales de subtipo secuencia

  `tdisc_seq(base,period): tdisc_seq`

  `tcont_seq(base,period,linear=true): tcont_seq`

  `tdisc_seq(ttype_inst[],left_inc=true,right_inc=true}): tdisc_seq`

  `tcont_seq(ttype_inst[],left_inc=true,right_inc=true,linear=true}): tcont_seq`

```
SELECT tfloat_seq(1.5, '[2001-01-01, 2001-01-02]');
SELECT tgeompoint_seq('Point(0 0)', '[2001-01-01, 2001-01-02]', false);
SELECT tbool_seq(ARRAY[tbool 'true@2001-01-01 08:00:00', 'true@2001-01-03 08:05:00'],
  true, true);
SELECT tint_seq(ARRAY[tint(2,'2001-01-01 08:00:00'), tint(2,'2001-01-01 08:10:00')],
  true, false);
SELECT tfloat_seq(ARRAY[tfloat '2.0@2001-01-01 08:00:00', '3@2001-01-03 08:05:00',
  '1@2001-01-03 08:10:00'], true, false);
SELECT tfloat_seq(ARRAY[tfloat '2.0@2001-01-01 08:00:00', '3@2001-01-03 08:05:00',
  '1@2001-01-03 08:10:00'], true, true, false);
SELECT ttext_seq(ARRAY[ttext('AAA', '2001-01-01 08:00:00'),
  ttext('BBB', '2001-01-03 08:05:00'), ttext('BBB', '2001-01-03 08:10:00')]);
SELECT tgeompoint_seq(ARRAY[tgeompoint 'Point(0 0)@2001-01-01 08:00:00',
  'Point(0 1)@2001-01-03 08:05:00', 'Point(1 1)@2001-01-03 08:10:00']);
SELECT tgeogpoint_seq(ARRAY[tgeogpoint 'Point(0 0)@2001-01-01 08:00:00',
  'Point(0 0)@2001-01-03 08:05:00'], true, true, false);
```

- Constructor para tipos temporales de subtipo conjunto de secuencias

  `tdisc_seqset(base,periodset): tdisc_seqset`

  `tcont_seqset(base,periodset,linear=true): tcont_seqset`

  `ttype_seqset(ttype_seq[]): ttype_seqset`

```
SELECT ttext_seqset('AAA', '{[2001-01-01, 2001-01-02], [2001-01-03, 2001-01-04]}');
SELECT tgeogpoint_seqset('Point(0 0)', '{[2001-01-01, 2001-01-02], [2001-01-03,  ↩
    2001-01-04]}',
  false);
SELECT tbool_seqset(ARRAY[tbool '[false@2001-01-01 08:00:00, false@2001-01-01 08:05:00)',
  '[true@2001-01-01 08:05:00]','(false@2001-01-01 08:05:00, false@2001-01-01 08:10:00)']);
SELECT tint_seqset(ARRAY[tint '[1@2001-01-01 08:00:00, 2@2001-01-01 08:05:00,
  2@2001-01-01 08:10:00, 2@2001-01-01 08:15:00)']);
SELECT tfloat_seqset(ARRAY[tfloat '[1.0@2001-01-01 08:00:00, 2.0@2001-01-01 08:05:00,
  2.0@2001-01-01 08:10:00]', '[2.0@2001-01-01 08:15:00, 3.0@2001-01-01 08:20:00)']);
```

```
SELECT tfloat_seqset(ARRAY[tfloat 'Interp=Stepwise;[1.0@2001-01-01 08:00:00,
  2.0@2001-01-01 08:05:00, 2.0@2001-01-01 08:10:00]',
  'Interp=Stepwise;[3.0@2001-01-01 08:15:00, 3.0@2001-01-01 08:20:00)']);
SELECT ttext_seqset(ARRAY[ttext '[AAA@2001-01-01 08:00:00, AAA@2001-01-01 08:05:00)',
  '[BBB@2001-01-01 08:10:00, BBB@2001-01-01 08:15:00)']);
SELECT tgeompoint_seqset(ARRAY[tgeompoint '[Point(0 0)@2001-01-01 08:00:00,
  Point(0 1)@2001-01-01 08:05:00, Point(0 1)@2001-01-01 08:10:00)',
  '[Point(0 1)@2001-01-01 08:15:00, Point(0 0)@2001-01-01 08:20:00)']);
SELECT tgeogpoint_seqset(ARRAY[tgeogpoint
  'Interp=Stepwise;[Point(0 0)@2001-01-01 08:00:00, Point(0 0)@2001-01-01 08:05:00)',
  'Interp=Stepwise;[Point(1 1)@2001-01-01 08:10:00, Point(1 1)@2001-01-01 08:15:00)']);
SELECT tfloat_seqset(ARRAY[tfloat 'Interp=Stepwise;[1.0@2001-01-01 08:00:00,
  2.0@2001-01-01 08:05:00, 2.0@2001-01-01 08:10:00]',
  '[3.0@2001-01-01 08:15:00, 3.0@2001-01-01 08:20:00)']);
-- ERROR:  Input sequences must have the same interpolation
```

5.3. Conversión de tipos

Un valor temporal se puede convertir en un tipo compatible usando la notación `CAST(ttype1 AS ttype2)` o `ttype1::ttype2`.

- Convertir un valor temporal a un período

  `ttype::period`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::period;
-- "[2001-01-01, 2001-01-03]"
SELECT ttext '(A@2000-01-01, B@2000-01-03, C@2000-01-05]'::period;
-- "(2000-01-01, 2000-01-05]"
```

- Convertir un número temporal a un rango

  `tnumber::range`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::intrange;
-- "[1, 3)"
SELECT tfloat '(1@2000-01-01, 3@2000-01-03, 2@2000-01-05]'::floatrange;
-- "(1, 3]"
SELECT tfloat 'Interp=Stepwise;(1@2000-01-01, 3@2000-01-03, 2@2000-01-05]'::floatrange;
-- "[1, 3]"
```

- Convertir un número temporal number a un `tbox`

  `tnumber::tbox`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::tbox;
-- "TBOX((1,2001-01-01 00:00:00+01),(2,2001-01-03 00:00:00+01))"
SELECT tfloat '(1@2000-01-01, 3@2000-01-03, 2@2000-01-05]'::tbox;
-- "TBOX((1,2000-01-01 00:00:00+01),(3,2000-01-05 00:00:00+01))"
```

- Convertir un punto temporal a un `stbox`

  `tpoint::stbox`

```
SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::stbox;
-- STBOX T((1,1,2001-01-01),(3,3,2001-01-03))
SELECT SELECT tgeogpoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::stbox;;
-- "SRID=4326;GEODSTBOX T
  ((0.9972609281539917,0.017449747771024704,0.01745240643728351,2001-01-01),
  (0.9996954202651978,0.05226423218846321,0.05233595624294383,2001-01-03))"
```

- Convertir un entero temporal en un flotante temporal

  `tint::tfloat`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::tfloat;
-- "[1@2001-01-01 00:00:00+00, 2@2001-01-03 00:00:00+00]"
SELECT tint '[1@2000-01-01, 2@2000-01-03, 3@2000-01-05]'::tfloat;
-- "Interp=Stepwise;[1@2000-01-01, 2@2000-01-03, 3@2000-01-05]"
```

- Convertir un flotante temporal en un entero temporal

  `tfloat::tint`

```
SELECT tfloat 'Interp=Stepwise;[1.5@2001-01-01, 2.5@2001-01-03]'::tint;
-- "[1@2001-01-01 00:00:00+00, 2@2001-01-03 00:00:00+00]"
SELECT tfloat '[1.5@2001-01-01, 2.5@2001-01-03]'::tint;
-- ERROR:  Cannot cast temporal float with linear interpolation to temporal integer
```

- Convertir un punto geométrico temporal en un punto geográfico temporal

  `tgeompoint::tgeogpoint`

```
SELECT asText((tgeogpoint 'Point(0 0)@2001-01-01')::tgeompoint);
-- "{POINT(0 0)@2001-01-01}"
```

- Convertir un punto geográfico temporal en un punto geométrico temporal

  `tgeogpoint::tgeompoint`

```
SELECT asText((tgeompoint '[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02)')::tgeogpoint);
-- "{[POINT(0 0)@2001-01-01, POINT(0 1)@2001-01-02)}"
```

Una forma común de almacenar puntos temporales en PostGIS es representarlos como geometrías de tipo `LINESTRING M` y utilizar la dimensión M para codificar marcas de tiempo como segundos desde 1970-01-01 00:00:00. Estas geometrías aumentadas con tiempo, llamadas *trayectorias*, se pueden validar con la función `ST_IsValidTrajectory` para verificar quel el valor M está creciendo de cada vértice al siguiente. Las trayectorias se pueden manipular con las funciones `ST_ClosestPointOfApproach`, `ST_DistanceCPA` y `ST_CPAWithin`. Los valores de puntos temporales se pueden convertir a/desde trayectorias de PostGIS.

- Convertir un punto temporal en una trayectoria PostGIS

  `tgeompoint::geometry`

  `tgeogpoint::geography`

```
SELECT ST_AsText((tgeompoint 'Point(0 0)@2001-01-01')::geometry);
-- "POINT M (0 0 978307200)"
SELECT ST_AsText((tgeompoint '{Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(1 1)@2001-01-03}')::geometry);
-- "MULTIPOINT M (0 0 978307200,1 1 978393600,1 1 978480000)"
SELECT ST_AsText((tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02)')::geometry);
-- "LINESTRING M (0 0 978307200,1 1 978393600)"
SELECT ST_AsText((tgeompoint '{[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02),
  [Point(1 1)@2001-01-03, Point(1 1)@2001-01-04),
  [Point(1 1)@2001-01-05, Point(0 0)@2001-01-06)}')::geometry);
-- "MULTILINESTRING M ((0 0 978307200,1 1 978393600),(1 1 978480000,1 1 978566400),
  (1 1 978652800,0 0 978739200))"
SELECT ST_AsText((tgeompoint '{[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02),
  [Point(1 1)@2001-01-03],
  [Point(1 1)@2001-01-05, Point(0 0)@2001-01-06)}')::geometry);
-- "GEOMETRYCOLLECTION M (LINESTRING M (0 0 978307200,1 1 978393600),
  POINT M (1 1 978480000),LINESTRING M (1 1 978652800,0 0 978739200))"
```

- Convertir una trayectoria PostGIS a un punto temporal

  `geometry::tgeompoint`

  `geography::tgeogpoint`

```
SELECT asText(geometry 'LINESTRING M (0 0 978307200,0 1 978393600,
  1 1 978480000)'::tgeompoint);
-- "[POINT(0 0)@2001-01-01, POINT(0 1)@2001-01-02, POINT(1 1)@2001-01-03]";
SELECT asText(geometry 'GEOMETRYCOLLECTION M (LINESTRING M (0 0 978307200,1 1 978393600),
  POINT M (1 1 978480000),LINESTRING M (1 1 978652800,0 0 978739200))'::tgeompoint);
-- "{[POINT(0 0)@2001-01-01, POINT(1 1)@2001-01-02], [POINT(1 1)@2001-01-03],
  [POINT(1 1)@2001-01-05, POINT(0 0)@2001-01-06]}"
```

5.4. Funciones de accesor

- Obtener el tamaño de la memoria en bytes

  `memSize(ttype): integer`

```
SELECT memSize(tint '{1@2012-01-01, 2@2012-01-02, 3@2012-01-03}');
-- 280
```

- Obtener el subtipo temporal

  `tempSubtype(ttype): {'Instant','InstantSet','Sequence','SequenceSet'}`

```
SELECT tempSubtype(tint '[1@2012-01-01, 2@2012-01-02, 3@2012-01-03]');
-- "Sequence"
```

- Obtener la interpolación

  `interpolation(ttype): {'Discrete','Stepwise','Linear'}`

```
SELECT interpolation(tfloat '{1@2012-01-01, 2@2012-01-02, 3@2012-01-03}');
-- "Discrete"
SELECT interpolation(tint '[1@2012-01-01, 2@2012-01-02, 3@2012-01-03]');
-- "Stepwise"
SELECT interpolation(tfloat '[1@2012-01-01, 2@2012-01-02, 3@2012-01-03]');
-- "Linear"
SELECT interpolation(tfloat 'Interp=Stepwise;[1@2012-01-01, 2@2012-01-02, 3@2012-01-03]');
-- "Stepwise"
SELECT interpolation(tgeompoint 'Interp=Stepwise;[Point(1 1)@2012-01-01,
  Point(2 2)@2012-01-02, Point(3 3)@2012-01-03]');
-- "Stepwise"
```

- Obtener el valor

  `getValue(ttype_inst): base`

```
SELECT getValue(tint '1@2012-01-01');
-- 1
SELECT ST_AsText(getValue(tgeompoint 'Point(0 0)@2012-01-01'));
-- "POINT(0 0)"
```

- Obtener los valores

  `getValues(ttype): {base[],floatrange[],geo}`

```
SELECT getValues(tint '[1@2012-01-01, 2@2012-01-03]');
-- "{1,2}"
SELECT getValues(tfloat '[1@2012-01-01, 2@2012-01-03)');
-- "{[1,2)}"
SELECT getValues(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 4@2012-01-05)}');
-- "{[1,2),[3,4)}"
SELECT getValues(tfloat 'Interp=Stepwise;{[1@2012-01-01, 2@2012-01-02],
  [3@2012-01-03, 4@2012-01-05]}');
-- "{"[1,1]","[2,2]","[3,3]","[4,4]"}"
SELECT ST_AsText(getValues(tgeompoint '{[Point(0 0)@2012-01-01, Point(0 1)@2012-01-02),
  [Point(0 1)@2012-01-03, Point(1 1)@2012-01-04)}'));
-- "LINESTRING(0 0,0 1,1 1)"
SELECT ST_AsText(getValues(tgeompoint '{[Point(0 0)@2012-01-01, Point(0 1)@2012-01-02),
  [Point(1 1)@2012-01-03, Point(2 2)@2012-01-04)}'));
-- "MULTILINESTRING((0 0,0 1),(1 1,2 2)"
SELECT ST_AsText(getValues(tgeompoint 'Interp=Stepwise;{[Point(0 0)@2012-01-01,
  Point(0 1)@2012-01-02], [Point(0 1)@2012-01-03, Point(1 1)@2012-01-04]}'));
-- "GEOMETRYCOLLECTION(MULTIPOINT(0 0,0 1),MULTIPOINT(0 1,1 1))"
SELECT ST_AsText(getValues(tgeompoint '{Point(0 0)@2012-01-01, Point(0 1)@2012-01-02}'));
-- "MULTIPOINT(0 0,0 1)"
SELECT ST_AsText(getValues(tgeompoint '{[Point(0 0)@2012-01-01, Point(0 1)@2012-01-02),
  [Point(1 1)@2012-01-03, Point(1 1)@2012-01-04),
  [Point(2 1)@2012-01-05, Point(2 2)@2012-01-06)}'));
-- "GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,0 1),LINESTRING(2 1,2 2))"
```

- Obtener el valor inicial

  `startValue(ttype): base`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT startValue(tfloat '(1@2012-01-01, 2@2012-01-03)');
-- 1
```

- Obtener el valor final

  `endValue(ttype): base`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT endValue(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 5@2012-01-05)}');
-- 5
```

- Obtener el valor mínimo

  `minValue(torder): base`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT minValue(tfloat '{1@2012-01-01, 2@2012-01-03, 3@2012-01-05}');
-- 1
```

- Obtener el valor máximo

  `maxValue(torder): base`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT maxValue(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 5@2012-01-05)}');
-- 5
```

- Obtener el rango de valores

  `valueRange(tnumber): numrange`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT valueRange(tfloat '{[2@2012-01-01, 1@2012-01-03), [4@2012-01-03, 6@2012-01-05)}');
-- "[1,6]"
SELECT valueRange(tfloat '{1@2012-01-01, 2@2012-01-03, 3@2012-01-05}');
-- "[1,3])"
```

- Obtener el valor en una marca de tiempo

  `valueAtTimestamp(ttype,timestamptz): base`

```
SELECT valueAtTimestamp(tfloat '[1@2012-01-01, 4@2012-01-04)', '2012-01-02');
-- "2"
```

- Obtener la marca de tiempo

  `getTimestamp(ttype_inst): timestamptz`

```
SELECT getTimestamp(tint '1@2012-01-01');
-- "2012-01-01"
```

- Obtener el tiempo

  `getTime(ttype): periodset`

```
SELECT getTime(tint '[1@2012-01-01, 1@2012-01-15)');
-- "{[2012-01-01, 2012-01-15)}"
```

- Obtener el intervalo de tiempo

  `duration(ttype): interval`

```
SELECT duration(tfloat '[1@2012-01-01, 2@2012-01-03, 2@2012-01-05)');
-- "4 days"
SELECT duration(tfloat '{[1@2012-01-01, 2@2012-01-03), [2@2012-01-04, 2@2012-01-05)}');
-- "3 days"
```

- Obtener el intervalo de tiempo ignorando las posibles brechas de tiempo

  `timespan(ttype): interval`

```
SELECT timespan(tfloat '{1@2012-01-01, 2@2012-01-03, 2@2012-01-05}');
-- "4 days"
SELECT timespan(tfloat '{[1@2012-01-01, 2@2012-01-03), [2@2012-01-04, 2@2012-01-05)}');
-- "4 days"
```

- Obtener el período en el que está definido el valor temporal ignorando las posibles brechas de tiempo

  `period(ttype): period`

```
SELECT period(tint '{1@2012-01-01, 2@2012-01-03, 3@2012-01-05}');
-- "[2012-01-01, 2012-01-05]"
SELECT period(tfloat '{[1@2012-01-01, 1@2012-01-02), [2@2012-01-03, 3@2012-01-04)}');
-- "[2012-01-01, 2012-01-04)"
```

- Obtener el número de instantes diferentes

  `numInstants(ttype): integer`

```
SELECT numInstants(tfloat '{[1@2000-01-01, 2@2000-01-02), (2@2000-01-02, 3@2000-01-03)}');
-- 3
```

- Obtener el instante inicial

  `startInstant(ttype): ttype_inst`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT startInstant(tfloat '{[1@2000-01-01, 2@2000-01-02),
  (2@2000-01-02, 3@2000-01-03)}');
-- "1@2000-01-01"
```

- Obtener el instante final

  `endInstant(ttype): ttype_inst`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT endInstant(tfloat '{[1@2000-01-01, 2@2000-01-02), (2@2000-01-02, 3@2000-01-03)}');
-- "3@2000-01-03"
```

- Obtener el enésimo instante diferente

  `instantN(ttype,integer): ttype_inst`

```
SELECT instantN(tfloat '{[1@2000-01-01, 2@2000-01-02), (2@2000-01-02, 3@2000-01-03)}', 3);
-- "3@2000-01-03"
```

- Obtener los instantes diferentes

  `instants(ttype): ttype_inst[]`

```
SELECT instants(tfloat '{[1@2000-01-01, 2@2000-01-02), (2@2000-01-02, 3@2000-01-03)}');
-- "{"1@2000-01-01","2@2000-01-02","3@2000-01-03"}"
```

- Obtener el número de marcas de tiempo diferentes

  `numTimestamps(ttype): integer`

```
SELECT numTimestamps(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}');
-- 3
```

- Obtener la marca de tiempo inicial

  `startTimestamp(ttype): timestamptz`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT startTimestamp(tfloat '[1@2012-01-01, 2@2012-01-03)');
-- "2012-01-01"
```

- Obtener la marca de tiempo final

  `endTimestamp(ttype): timestamptz`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT endTimestamp(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}');
-- "2012-01-05"
```

- Obtener la enésima marca de tiempo diferente

  `timestampN(ttype,integer): timestamptz`

```
SELECT timestampN(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}', 3);
-- "2012-01-05"
```

- Obtener las marcas de tiempo diferentes

  `timestamps(ttype): timestamptz[]`

```
SELECT timestamps(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 5@2012-01-05)}');
-- "{"2012-01-01", "2012-01-03", "2012-01-05"}"
```

- Obtener el número de secuencias

  `numSequences({ttype_seq,ttype_seqset}): integer`

```
SELECT numSequences(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}');
-- 2
```

- Obtener la secuencia inicial

  `startSequence({ttype_seq,ttype_seqset}): ttype_seq`

```
SELECT startSequence(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}');
-- "[1@2012-01-01, 2@2012-01-03)"
```

- Obtener la secuencia final

  `endSequence({ttype_seq,ttype_seqset}): ttype_seq`

```
SELECT endSequence(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 5@2012-01-05)}');
-- "[3@2012-01-03, 5@2012-01-05)"
```

- Obtener la enésima secuencia

  `sequenceN({ttype_seq,ttype_seqset},integer): ttype_seq`

```
SELECT sequenceN(tfloat '{[1@2012-01-01, 2@2012-01-03),
  [3@2012-01-03, 5@2012-01-05)}', 2);
-- "[3@2012-01-03, 5@2012-01-05)"
```

- Obtener las secuencias

  `sequences({ttype_seq,ttype_seqset}): ttype_seq[]`

```
SELECT sequences(tfloat '{[1@2012-01-01, 2@2012-01-03), [3@2012-01-03, 5@2012-01-05)}');
-- "{"[1@2012-01-01, 2@2012-01-03)", "[3@2012-01-03, 5@2012-01-05)"}"
```

- Obtener los segmentos

  `segments({ttype_seq,ttype_seqset}): ttype_seq[]`

```
SELECT segments(tint '{[1@2012-01-01, 3@2012-01-02, 2@2012-01-03],
  (3@2012-01-03, 5@2012-01-05]}');
-- {"[1@2012-01-01, 1@2012-01-02)","[3@2012-01-02, 3@2012-01-03)","[2@2012-01-03]",
  "(3@2012-01-03, 3@2012-01-05)","[5@2012-01-05]"}
SELECT segments(tfloat '{[1@2012-01-01, 3@2012-01-02, 2@2012-01-03],
  (3@2012-01-03, 5@2012-01-05]}');
-- {"[1@2012-01-01, 3@2012-01-02)","[3@2012-01-02, 2@2012-01-03]",
  "(3@2012-01-03, 5@2012-01-05]"}
```

- ¿Se cruza el valor temporal con la marca de tiempo?

  `intersectsTimestamp(ttype,timestamptz): boolean`

```
SELECT intersectsTimestamp(tint '[1@2012-01-01, 1@2012-01-15)', timestamptz '2012-01-03');
-- true
```

- ¿Se cruza el valor temporal con el conjunto de marcas tiempo?

  `intersectsTimestampSet(ttype,timestampset): boolean`

```
SELECT intersectsTimestampSet(tint '[1@2012-01-01, 1@2012-01-15)',
  timestampset '{2012-01-01, 2012-01-03}');
-- true
```

- ¿Se cruza el valor temporal con el período?

  `intersectsPeriod(ttype,period): boolean`

```
SELECT intersectsPeriod(tint '[1@2012-01-01, 1@2012-01-04)',
  period '[2012-01-01,2012-01-05)');
-- true
```

- ¿Se cruza el valor temporal con el conjunto de períodos?

  `intersectsPeriodSet(ttype,periodset): boolean`

```
SELECT intersectsPeriodSet(tbool '[t@2012-01-01, f@2012-01-15]',
  periodset '{[2012-01-01, 2012-01-03), [2012-01-05, 2012-01-07)}');
-- true
```

- Obtener el promedio ponderado en el tiempo

  `twAvg(tnumber): float`

```
SELECT twAvg(tfloat '{[1@2012-01-01, 2@2012-01-03), [2@2012-01-04, 2@2012-01-06)}');
-- 1.75
```

5.5. Funciones de transformación

Un valor temporal se puede transformar en otro subtipo. Se genera un error si los subtipos son incompatibles.

- Transformar un tipo temporal a otro subtipo

  `ttype_inst(ttype): ttype_inst`

  `ttype_instset(ttype): ttype_instset`

  `ttype_seq(ttype): ttype_seq`

  `ttype_seqset(ttype): ttype_seqset`

```
SELECT tbool_inst(tbool '{[true@2001-01-01]}');
-- "t@2001-01-01 00:00:00+00"
SELECT tbool_inst(tbool '{[true@2001-01-01, true@2001-01-02]}');
-- ERROR: Cannot transform input to a temporal instant
SELECT tbool_instset(tbool 'true@2001-01-01');
-- "{t@2001-01-01}"
SELECT tint_seq(tint '1@2001-01-01');
```

```
-- "[1@2001-01-01]"
SELECT tfloat_seqset(tfloat '2.5@2001-01-01');
-- "{[2.5@2001-01-01]}"
SELECT tfloat_seqset(tfloat '{2.5@2001-01-01, 1.5@2001-01-02, 3.5@2001-01-02}');
-- "{[2.5@2001-01-01],[1.5@2001-01-02],[3.5@2001-01-03]}"
```

- Transformar un valor temporal con tipo de base continuo de interpolación escalonada a lineal

  `toLinear(ttype): ttype`

```
SELECT toLinear(tfloat 'Interp=Stepwise;[1@2000-01-01, 2@2000-01-02,
  1@2000-01-03, 2@2000-01-04]');
-- "{[1@2000-01-01, 1@2000-01-02), [2@2000-01-02, 2@2000-01-03),
  [1@2000-01-03, 1@2000-01-04), [2@2000-01-04]}"
SELECT asText(toLinear(tgeompoint 'Interp=Stepwise;{[Point(1 1)@2000-01-01,
  Point(2 2)@2000-01-02], [Point(3 3)@2000-01-05, Point(4 4)@2000-01-06]}'));
-- "{[POINT(1 1)@2000-01-01, POINT(1 1)@2000-01-02), [POINT(2 2)@2000-01-02],
  [POINT(3 3)@2000-01-05, POINT(3 3)@2000-01-06), [POINT(4 4)@2000-01-06]}"
```

- Anexar un instante temporal a un valor temporal

  `appendInstant(ttype,ttype_inst): ttype`

```
SELECT appendInstant(tint '1@2000-01-01', tint '1@2000-01-02');
-- "{1@2000-01-01, 1@2000-01-02}"
SELECT appendInstant(tint_seq(tint '1@2000-01-01'), tint '1@2000-01-02');
-- "[1@2000-01-01, 1@2000-01-02]"
SELECT asText(appendInstant(tgeompoint '{[Point(1 1 1)@2000-01-01,
  Point(2 2 2)@2000-01-02], [Point(3 3 3)@2000-01-04, Point(3 3 3)@2000-01-05]}',
  tgeompoint 'Point(1 1 1)@2000-01-06'));
-- "{[POINT Z (1 1 1)@2000-01-01, POINT Z (2 2 2)@2000-01-02],
  [POINT Z (3 3 3)@2000-01-04, POINT Z (3 3 3)@2000-01-05, POINT Z (1 1 1)@2000-01-06]}"
```

- Fusionar los valores temporales

  `merge(ttype,ttype): ttype`

  `merge(ttype[]): ttype`

  Los valores pueden compartir una sola marca de tiempo, en ese caso, los valores temporales se unen en el resultado si su valor en la marca de tiempo común es el mismo; de lo contrario, se genera un error.

```
SELECT merge(tint '1@2000-01-01', tint '1@2000-01-02');
-- "{1@2000-01-01, 1@2000-01-02}"
SELECT merge(tint '[1@2000-01-01, 2@2000-01-02]', tint '[2@2000-01-02, 1@2000-01-03]');
-- "[1@2000-01-01, 2@2000-01-02, 1@2000-01-03]"
SELECT merge(tint '[1@2000-01-01, 2@2000-01-02]', tint '[3@2000-01-03, 1@2000-01-04]');
-- "{[1@2000-01-01, 2@2000-01-02], [3@2000-01-03, 1@2000-01-04]}"
SELECT merge(tint '[1@2000-01-01, 2@2000-01-02]', tint '[1@2000-01-02, 2@2000-01-03]');
-- ERROR:  Both arguments have different value at their overlapping timestamp
SELECT asText(merge(tgeompoint '{[Point(1 1 1)@2000-01-01,
  Point(2 2 2)@2000-01-02], [Point(3 3 3)@2000-01-04, Point(3 3 3)@2000-01-05]}',
  tgeompoint '{[Point(3 3 3)@2000-01-05, Point(1 1 1)@2000-01-06]}'));
-- "{[POINT Z (1 1 1)@2000-01-01, POINT Z (2 2 2)@2000-01-02],
  [POINT Z (3 3 3)@2000-01-04, POINT Z (3 3 3)@2000-01-05, POINT Z (1 1 1)@2000-01-06]}"

SELECT merge(ARRAY[tint '1@2000-01-01', '1@2000-01-02']);
-- "{1@2000-01-01, 1@2000-01-02}"
SELECT merge(ARRAY[tint '{1@2000-01-01, 2@2000-01-02}', '{2@2000-01-02, 3@2000-01-03}']);
-- "{1@2000-01-01, 2@2000-01-02, 3@2000-01-03}"
```

```
SELECT merge(ARRAY[tint '{1@2000-01-01, 2@2000-01-02}', '{3@2000-01-03, 4@2000-01-04}']);
-- "{1@2000-01-01, 2@2000-01-02, 3@2000-01-03, 4@2000-01-04}"
SELECT merge(ARRAY[tint '[1@2000-01-01, 2@2000-01-02]', '[2@2000-01-02, 1@2000-01-03]']);
-- "[1@2000-01-01, 2@2000-01-02, 1@2000-01-03]"
SELECT merge(ARRAY[tint '[1@2000-01-01, 2@2000-01-02]', '[3@2000-01-03, 4@2000-01-04]']);
-- "{[1@2000-01-01, 2@2000-01-02], [3@2000-01-03, 4@2000-01-04]}"
SELECT merge(ARRAY[tgeompoint '{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02],
  [Point(3 3)@2000-01-03, Point(4 4)@2000-01-04]}', '{[Point(4 4)@2000-01-04,
  Point(3 3)@2000-01-05], [Point(6 6)@2000-01-06, Point(7 7)@2000-01-07]}']);
-- "{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02], [Point(3 3)@2000-01-03,
  Point(4 4)@2000-01-04, Point(3 3)@2000-01-05],
  [Point(6 6)@2000-01-06, Point(7 7)@2000-01-07]}"
SELECT merge(ARRAY[tgeompoint '{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02]}',
  '{[Point(2 2)@2000-01-02, Point(1 1)@2000-01-03]}']);
-- "[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02, Point(1 1)@2000-01-03]"
```

- Desplazar el intervalo de tiempo del valor temporal con un intervalo

  `shift(ttype,interval): ttype`

```
SELECT shift(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}', '1 day');
-- "{1@2001-01-02, 2@2001-01-04, 1@2001-01-06}"
SELECT shift(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day');
-- "[1@2001-01-02, 2@2001-01-04]"
SELECT asText(shift(tgeompoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-03],
  [Point(2 2)@2001-01-04, Point(1 1)@2001-01-05]}', '1 day'));
-- "{[POINT(1 1)@2001-01-02, POINT(2 2)@2001-01-04],
  [POINT(2 2)@2001-01-05, POINT(1 1)@2001-01-06]}"
```

- Escalear el intervalo de tiempo del valor temporal a un intervalo. Si el intervalo de tiempo del valor temporal es cero (por ejemplo, para un instante temporal), el resultado es el valor temporal. El intervalo dado debe ser estrictamente mayor que cero.

  `tscale(ttype,interval): ttype`

```
SELECT tscale(tint '1@2001-01-01', '1 day');
-- "1@2001-01-01"
SELECT tscale(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}', '1 day');
-- "{1@2001-01-01 00:00:00+01, 2@2001-01-01 12:00:00+01, 1@2001-01-02 00:00:00+01}"
SELECT tscale(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day');
-- "[1@2001-01-01, 2@2001-01-02]"
SELECT asText(tscale(tgeompoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02,
  Point(1 1)@2001-01-03], [Point(2 2)@2001-01-04, Point(1 1)@2001-01-05]}', '1 day'));
-- "{[POINT(1 1)@2001-01-01 00:00:00+01, POINT(2 2)@2001-01-01 06:00:00+01,
  POINT(1 1)@2001-01-01 12:00:00+01], [POINT(2 2) @2001-01-01 18:00:00+01,
  POINT(1 1)@2001-01-02 00:00:00+01]}"
SELECT tscale(tint '1@2001-01-01', '-1 day');
-- ERROR:  The duration must be a positive interval: -1 days
```

- Desplazar y escalear el intervalo de tiempo del valor temporal a los dos intervalos. Esta función combina en un solo paso las funciones shift y tscale.

  `shiftTscale(ttype,interval,interval): ttype`

```
SELECT shiftTscale(tint '1@2001-01-01', '1 day', '1 day');
-- "1@2001-01-02"
SELECT shiftTscale(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}', '1 day', '1 day');
-- "{1@2001-01-02 00:00:00+01, 2@2001-01-02 12:00:00+01, 1@2001-01-03 00:00:00+01}"
SELECT shiftTscale(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day', '1 day');
```

```
-- "[1@2001-01-02, 2@2001-01-03]"
SELECT asText(shiftTscale(tgeompoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02,
  Point(1 1)@2001-01-03], [Point(2 2)@2001-01-04, Point(1 1)@2001-01-05]}',
  '1 day', '1 day'));
-- "{[POINT(1 1)@2001-01-02 00:00:00+01, POINT(2 2)@2001-01-02 06:00:00+01,
  POINT(1 1)@2001-01-02 12:00:00+01], [POINT(2 2) @2001-01-02 18:00:00+01,
  POINT(1 1)@2001-01-03 00:00:00+01]}"
```

5.6. Funciones de restricción

5.6.1. Funciones de selección

Estas funciones restringen el valor temporal con respecto a una extensión de valores o de tiempo.

- Restringir a un valor

  `atValue(ttype,base): ttype`

```
SELECT atValue(tint '[1@2012-01-01, 1@2012-01-15)', 1);
-- "[1@2012-01-01, 1@2012-01-15)"
SELECT asText(atValue(tgeompoint '[Point(0 0 0)@2012-01-01, Point(2 2 2)@2012-01-03)',
  'Point(1 1 1)'));
-- "{[POINT Z (1 1 1)@2012-01-02]}"
```

- Restringir a una matriz de valores

  `atValues(ttype,base[]): ttype`

```
SELECT atValues(tfloat '[1@2012-01-01, 4@2012-01-4)', ARRAY[1, 3, 5]);
-- "{[1@2012-01-01], [3@2012-01-03]}"
SELECT asText(atValues(tgeompoint '[Point(0 0)@2012-01-01, Point(2 2)@2012-01-03)',
  ARRAY[geometry 'Point(0 0)', 'Point(1 1)']));
-- "{[POINT(0 0)@2012-01-01 00:00:00+00], [POINT(1 1)@2012-01-02 00:00:00+00]}"
```

- Restringir a un rango de valores

  `atRange(tnumber,numrange): ttype`

```
SELECT atRange(tfloat '[1@2012-01-01, 4@2012-01-4)', floatrange '[1,3]');
-- "[1@2012-01-01, 3@2012-01-03]"
```

- Restringir a una matriz de rangos de valores

  `atRanges(tnumber,numrange[]): ttype`

```
SELECT atRanges(tfloat '[1@2012-01-01, 5@2012-01-05)',
  ARRAY[floatrange '[1,2]', '[3,4]']);
-- "{[1@2012-01-01, 2@2012-01-02],[3@2012-01-03, 4@2012-01-04]}"
```

- Restringir al valor mínimo

  `atMin(torder): torder`

  La función devuelve nulo si el valor mínimo sólo ocurre en límites exclusivos.

```
SELECT atMin(tint '{1@2012-01-01, 2@2012-01-03, 1@2012-01-05}');
-- "{1@2012-01-01, 1@2012-01-05}"
SELECT atMin(tint '(1@2012-01-01, 3@2012-01-03]');
-- "{(1@2012-01-01, 1@2012-01-03)}"
SELECT atMin(tfloat '(1@2012-01-01, 3@2012-01-03]');
-- NULL
SELECT atMin(ttext '{(AA@2012-01-01, AA@2012-01-03), (BB@2012-01-03, AA@2012-01-05]}');
-- "{(AA@2012-01-01, AA@2012-01-03), [AA@2012-01-05]}"
```

- Restringir al valor máximo

  `atMax(torder): torder`

  La función devuelve nulo si el valor máximo sólo ocurre en límites exclusivos.

```
SELECT atMax(tint '{1@2012-01-01, 2@2012-01-03, 3@2012-01-05}');
-- "{3@2012-01-05}"
SELECT atMax(tfloat '(1@2012-01-01, 3@2012-01-03)');
-- NULL
SELECT atMax(tfloat '{(2@2012-01-01, 1@2012-01-03), [2@2012-01-03, 2@2012-01-05)}');
-- "{[2@2012-01-03, 2@2012-01-05]}"
SELECT atMax(ttext '{(AA@2012-01-01, AA@2012-01-03), (BB@2012-01-03, AA@2012-01-05]}');
-- "{("BB"@2012-01-03, "BB"@2012-01-05)}"
```

- Restringir a una geometría

  `atGeometry(tgeompoint,geometry): tgeompoint`

  Tenga en cuenta que está permitido mezclar geometrías 2D/3D, pero el cálculo sólo se realiza en 2D.

```
SELECT asText(atGeometry(tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
-- "{"[POINT(1 1)@2012-01-02, POINT(2 2)@2012-01-03]"}"
SELECT astext(atGeometry(tgeompoint '[Point(0 0 0)@2000-01-01, Point(4 4 4)@2000-01-05]',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
-- "{[POINT Z (1 1 1)@2000-01-02, POINT Z (2 2 2)@2000-01-03]}"
```

- Restringir a una marca de tiempo

  `atTimestamp(ttype,timestamptz): ttype_inst`

```
SELECT atTimestamp(tfloat '[1@2012-01-01, 5@2012-01-05)', '2012-01-02');
-- "2@2012-01-02"
```

- Restringir a un conjunto de marcas de tiempo

  `atTimestampSet(ttype,timestampset): {ttype_inst,ttype_instset}`

```
SELECT atTimestampSet(tint '[1@2012-01-01, 1@2012-01-15)',
  timestampset '{2012-01-01, 2012-01-03}');
-- "{1@2012-01-01, 1@2012-01-03}"
```

- Restringir a un período

  `atPeriod(ttype,period): ttype`

```
SELECT atPeriod(tfloat '{[1@2012-01-01, 3@2012-01-03), [3@2012-01-04, 1@2012-01-06)}',
  '[2012-01-02,2012-01-05)');
-- "{[2@2012-01-02, 3@2012-01-03), [3@2012-01-04, 2@2012-01-05)}"
```

- Restringir a un conjunto de períodos

  `atPeriodSet(ttype,periodset): ttype`

```
SELECT atPeriodSet(tint '[1@2012-01-01, 1@2012-01-15)',
  periodset '{[2012-01-01, 2012-01-03), [2012-01-04, 2012-01-05)}');
-- "{[1@2012-01-01, 1@2012-01-03),[1@2012-01-04, 1@2012-01-05)}"
```

- Restringir a un `tbox`

  `atTbox(tnumber,tbox): tnumber`

```
SELECT atTbox(tfloat '[0@2012-01-01, 3@2012-01-04)',
  tbox 'TBOX((0, 2012-01-02), (2, 2012-01-04))');
-- "{[1@2012-01-02, 2@2012-01-03]}"
```

- Restringir a un `stbox`

  `atStbox(tgeompoint,stbox): tgeompoint`

```
SELECT asText(atStbox(tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)',
  stbox 'STBOX T((0, 0, 2012-01-02), (2, 2, 2012-01-04))'));
-- "{[POINT(1 1)@2012-01-02, POINT(2 2)@2012-01-03]}"
```

5.6.2. Funciones de diferencia

Estas funciones restringen el valor temporal con respecto al complemento de una extensión de valores o de tiempo.

- Diferencia con un valor

  `minusValue(ttype,base): ttype`

```
SELECT minusValue(tint '[1@2012-01-01, 2@2012-01-02, 2@2012-01-03)', 1);
-- "{[2@2012-01-02, 2@2012-01-03)}"
SELECT asText(minusValue(tgeompoint '[Point(0 0 0)@2012-01-01, Point(2 2 2)@2012-01-03)',
  'Point(1 1 1)'));
-- "{[POINT Z (0 0 0)@2012-01-01, POINT Z (1 1 1)@2012-01-02),
  (POINT Z (1 1 1)@2012-01-02, POINT Z (2 2 2)@2012-01-03)}"
```

- Diferencia con una matriz de valores

  `minusValues(ttype,base[]): ttype`

```
SELECT minusValues(tfloat '[1@2012-01-01, 4@2012-01-4)', ARRAY[2, 3]);
-- "{[1@2012-01-01, 2@2012-01-02), (2@2012-01-02, 3@2012-01-03),
  (3@2012-01-03, 4@2012-01-04)}"
SELECT asText(minusValues(tgeompoint '[Point(0 0 0)@2012-01-01, Point(3 3 3)@2012-01-04)',
  ARRAY[geometry 'Point(1 1 1)', 'Point(2 2 2)']));
-- "{[POINT Z (0 0 0)@2012-01-01, POINT Z (1 1 1)@2012-01-02),
  (POINT Z (1 1 1)@2012-01-02, POINT Z (2 2 2)@2012-01-03),
  (POINT Z (2 2 2)@2012-01-03, POINT Z (3 3 3)@2012-01-04)}"
```

- Diferencia con un rango de valores

  `minusRange(tnumber,numrange): ttype`

```
SELECT minusRange(tfloat '[1@2012-01-01, 4@2012-01-4)', floatrange '[2,3]');
-- "{[1@2012-01-01, 2@2012-01-02), (3@2012-01-03, 4@2012-01-04)}"
```

- Diferencia con una matriz de rangos de valores

  `minusRanges(tnumber,numrange[]): ttype`

```
SELECT minusRanges(tfloat '[1@2012-01-01, 5@2012-01-05)',
  ARRAY[floatrange '[1,2]', '[3,4]']);
-- "{(2@2012-01-02, 3@2012-01-03), (4@2012-01-04, 5@2012-01-05)}"
```

- Diferencia con el valor mínimo

  `minusMin(torder): torder`

```
SELECT minusMin(tint '{1@2012-01-01, 2@2012-01-03, 1@2012-01-05}');
-- "{2@2012-01-03}"
SELECT minusMin(tfloat '[1@2012-01-01, 3@2012-01-03]');
-- "{(1@2012-01-01, 3@2012-01-03]}"
SELECT minusMin(tfloat '(1@2012-01-01, 3@2012-01-03)');
-- "{(1@2012-01-01, 3@2012-01-03)}"
SELECT minusMin(tint '{[1@2012-01-01, 1@2012-01-03), (1@2012-01-03, 1@2012-01-05)}');
-- NULL
```

- Diferencia con el valor máximo

  `minusMax(torder): torder`

```
SELECT minusMax(tint '{1@2012-01-01, 2@2012-01-03, 3@2012-01-05}');
-- "{1@2012-01-01, 2@2012-01-03}"
SELECT minusMax(tfloat '[1@2012-01-01, 3@2012-01-03]');
-- "{[1@2012-01-01, 3@2012-01-03)}"
SELECT minusMax(tfloat '(1@2012-01-01, 3@2012-01-03)');
-- "{(1@2012-01-01, 3@2012-01-03)}"
SELECT minusMax(tfloat '{[2@2012-01-01, 1@2012-01-03), [2@2012-01-03, 2@2012-01-05)}');
-- "{(2@2012-01-01, 1@2012-01-03)}"
SELECT minusMax(tfloat '{[1@2012-01-01, 3@2012-01-03), (3@2012-01-03, 1@2012-01-05)}');
-- "{[1@2012-01-01, 3@2012-01-03), (3@2012-01-03, 1@2012-01-05)}"
```

- Diferencia con una geometría

  `minusGeometry(tgeompoint,geometry): tgeompoint`

  Tenga en cuenta que está permitido mezclar geometrías 2D/3D, pero el cálculo sólo se realiza en 2D.

```
SELECT asText(minusGeometry(tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
-- "{[POINT(0 0)@2012-01-01, POINT(1 1)@2012-01-02), (POINT(2 2)@2012-01-03,
  POINT(3 3)@2012-01-04)}"
SELECT astext(minusGeometry(tgeompoint '[Point(0 0 0)@2000-01-01,
  Point(4 4 4)@2000-01-05]', geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
-- "{[POINT Z (0 0 0)@2000-01-01, POINT Z (1 1 1)@2000-01-02),
  (POINT Z (2 2 2)@2000-01-03, POINT Z (4 4 4)@2000-01-05]}"
```

- Diferencia con una marca de tiempo

  `minusTimestamp(ttype,timestamptz): ttype`

```
SELECT minusTimestamp(tfloat '[1@2012-01-01, 5@2012-01-05)', '2012-01-02');
-- "{[1@2012-01-01, 2@2012-01-02), (2@2012-01-02, 5@2012-01-05)}"
```

- Diferencia con un conjunto de marcas de tiempo

  `minusTimestampSet(ttype,timestampset): ttype`

```
SELECT minusTimestampSet(tint '[1@2012-01-01, 1@2012-01-15)',
  timestampset '{2012-01-02, 2012-01-03}');
-- "{[1@2012-01-01, 1@2012-01-02), (1@2012-01-02, 1@2012-01-03),
(1@2012-01-03, 1@2012-01-15)}"
```

- Diferencia con un período

  `minusPeriod(ttype,period): ttype`

```
SELECT minusPeriod(tfloat '{[1@2012-01-01, 3@2012-01-03), [3@2012-01-04, 1@2012-01-06)}',
  '[2012-01-02,2012-01-05)');
-- "{[1@2012-01-01, 2@2012-01-02), [2@2012-01-05, 1@2012-01-06)}"
```

- Diferencia con un conjunto de períodos

  `minusPeriodSet(ttype,periodset): ttype`

```
SELECT minusPeriodSet(tint '[1@2012-01-01, 1@2012-01-15)',
  periodset '{[2012-01-02, 2012-01-03), [2012-01-04, 2012-01-05)}');
-- "{[1@2012-01-01, 1@2012-01-02), [1@2012-01-03, 1@2012-01-04),
  [1@2012-01-05, 1@2012-01-15)}"
```

- Diferencia con un `tbox`

  `minusTbox(tnumber,tbox): tnumber`

```
SELECT minusTbox(tfloat '[0@2012-01-01, 3@2012-01-04)',
  tbox 'TBOX((0, 2012-01-02), (2, 2012-01-04))');
-- "{[0@2012-01-01, 1@2012-01-02), (2@2012-01-03, 3@2012-01-04)}"
```

- Diferencia con un `stbox`

  `minusStbox(tgeompoint,stbox): tgeompoint`

```
SELECT asText(minusStbox(tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)',
  stbox 'STBOX T((0, 0, 2012-01-02), (2, 2, 2012-01-04))'));
-- "{[POINT(0 0)@2012-01-01, POINT(1 1)@2012-01-02),
  (POINT(2 2)@2012-01-03, POINT(3 3)@2012-01-04)}"
```

5.7. Operadores de comparación

5.7.1. Operadores de comparación tradicionales

Los operadores de comparación tradicionales (=, <, etc.) requieren que los operandos izquierdo y derecho sean del mismo tipo base. Excepto la igualdad y la no igualdad, los otros operadores de comparación no son útiles en el mundo real pero permiten que los índices de árbol B se construyan sobre tipos temporales. Estos operadores comparan los períodos delimitadores (ver la Sección 2.1.5), después los cuadros delimitadores (ver la Sección 4.8) y si son iguales, entonces la comparación depende del subtipo. Para los valores de instante, primero comparan las marcas de tiempo y, si son iguales, comparan los valores. Para los valores de secuencia y conjunto de instantes, comparan los primeros N instantes, donde N es el mínimo del número de instantes que componen ambos valores. Finalmente, para los valores de conjuntos de secuencias, comparan los primeros N valores de secuencia, donde N es el mínimo del número de secuencias que componen ambos valores.

Los operadores de igualdad y no igualdad consideran la representación equivalente para diferentes subtipos como se muestra a continuación.

```
SELECT tint '1@2001-01-01' = tint '{1@2001-01-01}';
-- true
SELECT tfloat '1.5@2001-01-01' = tfloat '[1.5@2001-01-01]';
-- true
SELECT ttext 'AAA@2001-01-01' = ttext '{[AAA@2001-01-01]}';
-- true
SELECT tgeompoint '{Point(1 1)@2001-01-01, Point(2 2)@2001-01-02}' =
  tgeompoint '{[Point(1 1)@2001-01-01], [Point(2 2)@2001-01-02]}';
-- true
SELECT tgeogpoint '[Point(1 1 1)@2001-01-01, Point(2 2 2)@2001-01-02]' =
  tgeogpoint '{[Point(1 1 1)@2001-01-01], [Point(2 2 2)@2001-01-02]}';
-- true
```

- ¿Son iguales los valores temporales?

  `ttype = ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' = tint '[2@2012-01-03, 2@2012-01-05)';
-- false
```

- ¿Son diferentes los valores temporales?

  `ttype <> ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' <> tint '[2@2012-01-03, 2@2012-01-05)'
-- true
```

- ¿Es el primer valor temporal menor que el segundo?

  `ttype < ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' < tint '[2@2012-01-03, 2@2012-01-05)'
-- true
```

- ¿Es el primer valor temporal mayor que el segundo?

  `ttype > ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' > tint '[2@2012-01-03, 2@2012-01-05)'
-- false
```

- ¿Es el primer valor temporal menor o igual que el segundo?

  `ttype <= ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' <= tint '[2@2012-01-03, 2@2012-01-05)'
-- true
```

- ¿Es el primer valor temporal mayor o igual que el segundo?

  `ttype >= ttype: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' >= tint '[2@2012-01-03, 2@2012-01-05)'
-- false
```

5.7.2. Operadores de comparación alguna vez y siempre

Una posible generalización de los operadores de comparación tradicionales (=, <>, <, <=, etc.) a tipos temporales consiste en determinar si la comparación es alguna vez o siempre verdadera. En este caso, el resultado es un valor booleano. MobilityDB proporciona operadores para probar si la comparación de un valor temporal y un valor del tipo base es alguna vez o siempre verdadera. Estos operadores se indican anteponiendo los operadores de comparación tradicionales con, respectivamente, `?` (alguna vez) y `%` (siempre). Algunos ejemplos son `?=`, `%<>` o `?<=`. La igualdad y la no igualdad alguna vez/siempre están disponibles para todos los tipos temporales, mientras que las desigualdades alguna vez/siempre sólo están disponibles para los tipos temporales cuyo tipo base tiene un orden total definido, es decir, `tint`, `tfloat` o `ttext`. Las comparaciones alguna vez y siempre son operadores inversos: por ejemplo, `?=` es el inverso de `%<>` y `?>` es el inverso de `%<=`.

- ¿Es el valor temporal alguna vez igual al valor?

  `ttype ?= base: boolean`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' ?= 2;
-- true
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' ?= 3;
-- true
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(2 2)@2012-01-04)' ?=
  geometry 'Point(1 1)';
-- true
```

- ¿Es el valor temporal alguna vez diferente del valor?

  `ttype ?<> base: boolean`

```
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' ?<> 2;
-- false
SELECT tfloat '[2@2012-01-01, 2@2012-01-04)' ?<> 2;
-- true
SELECT tgeompoint '[Point(1 1)@2012-01-01, Point(1 1)@2012-01-04)' ?<>
  geometry 'Point(1 1)';
-- true
```

- ¿Es el valor temporal alguna vez menor que el valor?

  `tnumber ?< number: boolean`

```
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' ?< 2;
-- "{[t@2012-01-01, f@2012-01-02, f@2012-01-04)}"
SELECT tint '[2@2012-01-01, 2@2012-01-05)' ?< tfloat '[1@2012-01-03, 3@2012-01-05)';
-- "{[f@2012-01-03, f@2012-01-04], (t@2012-01-04, t@2012-01-05)}"
```

- ¿Es el valor temporal alguna vez mayor que el valor?

  `tnumber ?> number: boolean`

```
SELECT tint '[1@2012-01-03, 1@2012-01-05)' ?> 1;
-- "[f@2012-01-03, f@2012-01-05)"
```

- ¿Es el valor temporal alguna vez menor o igual que el valor?

  `tnumber ?<= number: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-05)' ?<= tfloat '{2@2012-01-03, 3@2012-01-04}';
-- "{t@2012-01-03, t@2012-01-04}"
```

- ¿Es el valor temporal alguna vez mayor o igual que el valor?

  `tnumber ?>= number: boolean`

```
SELECT 'AAA'::text ?> ttext '{[AAA@2012-01-01, AAA@2012-01-03),
  [BBB@2012-01-04, BBB@2012-01-05)}';
-- "{[f@2012-01-01, f@2012-01-03), [t@2012-01-04, t@2012-01-05)}"
```

- ¿Es el valor temporal siempre igual que el valor?

  `ttype %= base: boolean`

  La función no tiene en cuenta si los límites son inclusivos o no.

```
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' %= 2;
-- true
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' %= 3;
-- true
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(2 2)@2012-01-04)' %=
  geometry 'Point(1 1)';
-- true
```

- ¿Es el valor temporal siempre diferente que el valor?

  `ttype %<> base: boolean`

```
SELECT tfloat '[1@2012-01-01, 3@2012-01-04)' %<> 2;
-- false
SELECT tfloat '[2@2012-01-01, 2@2012-01-04)' %<> 2;
-- true
SELECT tgeompoint '[Point(1 1)@2012-01-01, Point(1 1)@2012-01-04)' %<>
  geometry 'Point(1 1)';
-- true
```

- ¿Es el valor temporal siempre menor que el valor?

  `tnumber %< number: boolean`

```
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' %< 2;
-- "{[t@2012-01-01, f@2012-01-02, f@2012-01-04)}"
SELECT tint '[2@2012-01-01, 2@2012-01-05)' %< tfloat '[1@2012-01-03, 3@2012-01-05)';
-- "{[f@2012-01-03, f@2012-01-04], (t@2012-01-04, t@2012-01-05)}"
```

- ¿Es el valor temporal siempre mayor que el valor?

  `tnumber %> number: boolean`

```
SELECT tint '[1@2012-01-03, 1@2012-01-05)' %> 1;
-- "[f@2012-01-03, f@2012-01-05)"
```

- ¿Es el valor temporal siempre menor o igual que el valor?

  `tnumber %<= number: boolean`

```
SELECT tint '[1@2012-01-01, 1@2012-01-05)' %<= tfloat '{2@2012-01-03, 3@2012-01-04}';
-- "{t@2012-01-03, t@2012-01-04}"
```

- ¿Es el valor temporal siempre mayor o igual que el valor?

  `tnumber %>= number: boolean`

```
SELEC T 'AAA'::text %> ttext '{[AAA@2012-01-01, AAA@2012-01-03),
  [BBB@2012-01-04, BBB@2012-01-05)}';
-- "{[f@2012-01-01, f@2012-01-03), [t@2012-01-04, t@2012-01-05)}"
```

5.7.3. Operadores de comparación temporal

Otra posible generalización de los operadores de comparación tradicionales (=, <>, <, <=, etc.) a tipos temporales consiste en determinar si la comparación es verdadera o falsa en cada instante. En este caso, el resultado es un booleano temporal. Los operadores de comparación temporal se indican anteponiendo los operadores de comparación tradicionales con `#`. Algunos ejemplos son `#=` o `#<=`. La igualdad y no igualdad temporal están disponibles para todos los tipos temporales, mientras que las desigualdades temporales sólo están disponibles para los tipos temporales cuyo tipo base tiene un orden total definido, es decir, `tint`, `tfloat` o `ttext`.

- Igual temporal

  `{base,ttype} #= {base,ttype}: tbool`

```
SELECT tfloat '[1@2012-01-01, 2@2012-01-04)' #= 3;
-- "{[f@2012-01-01, f@2012-01-04)}"
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' #= tint '[1@2012-01-01, 1@2012-01-04)';
-- "{[t@2012-01-01], (f@2012-01-01, f@2012-01-04)}"
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' #= tfloat '[4@2012-01-02, 1@2012-01-05)';
-- "{[f@2012-01-02, t@2012-01-03], (f@2012-01-03, f@2012-01-04)}"
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(2 2)@2012-01-03)' #=
  geometry 'Point(1 1)';
-- "{[f@2012-01-01, t@2012-01-02], (f@2012-01-02, f@2012-01-03)}"
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(2 2)@2012-01-03)' #=
  tgeompoint '[Point(0 2)@2012-01-01, Point(2 0)@2012-01-03)';
-- "{[f@2012-01-01], (t@2012-01-01, t@2012-01-03)}"
```

- Diferente temporal

  `{base,ttype} #<> {base,ttype}: tbool`

```
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' #<> 2;
-- "{[t@2012-01-01, f@2012-01-02], (t@2012-01-02, 2012-01-04)}"
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' #<> tint '[2@2012-01-02, 2@2012-01-05)';
-- "{[f@2012-01-02], (t@2012-01-02, t@2012-01-04)}"
```

- Menor que temporal

  `{base,torder} #< {base,torder}: tbool`

```
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' #< 2;
-- "{[t@2012-01-01, f@2012-01-02, f@2012-01-04)}"
SELECT tint '[2@2012-01-01, 2@2012-01-05)' #< tfloat '[1@2012-01-03, 3@2012-01-05)';
-- "{[f@2012-01-03, f@2012-01-04], (t@2012-01-04, t@2012-01-05)}"
```

- Mayor que temporal

  `{base,torder} #> {base,torder}: tbool`

```
SELECT 1 #> tint '[1@2012-01-03, 1@2012-01-05)';
-- "[f@2012-01-03, f@2012-01-05)"
```

- Menor o igual que temporal

  `{base,torder} #<= {base,torder}: tbool`

```
SELECT tint '[1@2012-01-01, 1@2012-01-05)' #<= tfloat '{2@2012-01-03, 3@2012-01-04}';
-- "{t@2012-01-03, t@2012-01-04}"
```

- Mayor o igual que temporal

  `{base,torder} #>= {base,torder}: tbool`

```
SELECT 'AAA'::text #> ttext '{[AAA@2012-01-01, AAA@2012-01-03),
  [BBB@2012-01-04, BBB@2012-01-05)}';
-- "{[f@2012-01-01, f@2012-01-03), [t@2012-01-04, t@2012-01-05)}"
```

5.8. Operadores de cuadro delimitador

Estos operadores prueban si los cuadros delimitadores de sus argumentos satisfacen el predicado y dan como resultado un valor booleano. Como se indica en el Capítulo 3, el cuadro delimitador asociado a un tipo temporal depende del tipo base: es el tipo `period` para los tipos `tbool` y `ttext`, el tipo `tbox` para los tipos `tint` y `tfloat` y el tipo `stbox` para los tipos `tgeompoint` y `tgeogpoint`. Además, como se dijo en la Sección 4.3, muchos tipos PostgreSQL, PostGIS o MobilityDB se pueden convertir a los tipos `tbox` y `stbox`. Por ejemplo, los tipos numéricos y los rangos se pueden convertir al tipo `tbox`, los tipos `geometry` y `geography` se pueden convertir al tipo `stbox` y los tipos de tiempo y los tipos temporales se pueden convertir a los tipos `tbox` y `stbox`.

Un primer conjunto de operadores considera las relaciones topológicas entre los cuadros delimitadores. Hay cinco operadores topológicos: superposicion (`&&`), contiene (`@>`), está contenido (`<@`), mismo (`~=`) y adyacente (`-|-`). Los argumentos de estos operadores pueden ser un tipo base, una cuadro delimitador o un tipo temporal y los operadores verifican la relación topológica teniendo en cuenta el valor y/o la dimensión temporal según el tipo de los argumentos.

Otro conjunto de operadores considera la posición relativa de los cuadros delimitadores. Los operadores <<, >>, &< y &> consideran la dimensión de valor para los tipos `tint` y `tfloat` y las coordenadas X para los tipos `tgeompoint` y `tgeogpoint`, los operadores <<|, |>>, &<| y |&> consideran las coordenadas Y para los tipos `tgeompoint` y `tgeogpoint`, los operadores <</, />>, &</ y /&> consideran las coordenadas Z para los tipos `tgeompoint` y `tgeogpoint` y los operadores <<#, #>>, #&< y #&> consideran la dimensión tiempo para todos los tipos temporales.

Finalmente, cabe destacar que los operadores de cuadro delimitador permiten mezclar geometrías 2D/3D pero en ese caso, el cálculo sólo se realiza en 2D.

Refiérase a la Sección 4.10 y a la Sección 4.11 para los operadores de cuadro delimitador.

5.9. Funciones y operadores matemáticos

- Adición temporal

  `{number,tnumber} + {number,tnumber}: tnumber`

```
SELECT tint '[2@2012-01-01, 2@2012-01-04)' + 1.5;
-- "[3.5@2012-01-01, 3.5@2012-01-04)"
SELECT tint '[2@2012-01-01, 2@2012-01-04)' + tfloat '[1@2012-01-01, 4@2012-01-04)';
-- "[3@2012-01-01, 6@2012-01-04)"
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' +
  tfloat '{[1@2012-01-01, 2@2012-01-02), [1@2012-01-02, 2@2012-01-04)}';
-- "{[2@2012-01-01, 4@2012-01-04), [3@2012-01-02, 6@2012-01-04)}"
```

- Resta temporal

  `{number,tnumber} - {number,tnumber}: tnumber`

```
SELECT tint '[1@2012-01-01, 1@2012-01-04)' - tint '[2@2012-01-03, 2@2012-01-05)';
-- "[-1@2012-01-03, -1@2012-01-04)"
SELECT tfloat '[3@2012-01-01, 6@2012-01-04)' - tint '[2@2012-01-01, 2@2012-01-04)';
-- "[1@2012-01-01, 4@2012-01-04)"
```

- Multiplicación temporal

  `{number,tnumber} * {number,tnumber}: tnumber`

```
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' * 2;
-- "[2@2012-01-01, 8@2012-01-04)"
SELECT tfloat '[1@2012-01-01, 4@2012-01-04)' * tint '[2@2012-01-01, 2@2012-01-04)';
-- "[2@2012-01-01, 8@2012-01-04)"
SELECT tfloat '[1@2012-01-01, 3@2012-01-03)' * '[3@2012-01-01, 1@2012-01-03)'
-- "{[3@2012-01-01, 4@2012-01-02, 3@2012-01-03)}"
```

- División temporal

  `{number,tnumber} / {number,tnumber}: tnumber`

  La función genera un error si el denominador es alguna vez igual a cero durante el intervalo de tiempo común de los argumentos.

```
SELECT 2 / tfloat '[1@2012-01-01, 3@2012-01-04)';
-- "[2@2012-01-01, 1@2012-01-02 12:00:00+00, 0.6666666666666667@2012-01-04)"
SELECT tfloat '[1@2012-01-01, 5@2012-01-05)' / '[5@2012-01-01, 1@2012-01-05)'
-- "{[0.2@2012-01-01, 1@2012-01-03,2012-01-03, 5@2012-01-03,2012-01-05)}"
SELECT 2 / tfloat '[-1@2000-01-01, 1@2000-01-02]'
```

```
-- ERROR:  Division by zero
SELECT tfloat '[-1@2000-01-04, 1@2000-01-05]' / tfloat '[-1@2000-01-01, 1@2000-01-05]'
-- "[-2@2000-01-04, 1@2000-01-05]"
```

- Redondear los valores a un número de posiciones decimales

  `round(tfloat,integer): tfloat`

```
SELECT round(tfloat '[0.785398163397448@2000-01-01, 2.356194490192340@2000-01-02]', 2);
-- "[0.79@2000-01-01, 2.36@2000-01-02]"
```

- Convertir de radianes a grados

  `degrees(tfloat): tfloat`

```
SELECT degrees(tfloat '[0.785398163397448@2000-01-01, 2.356194490192340@2000-01-02]');
-- "[45@2000-01-01, 135@2000-01-02]"
```

- Obtener la derivada sobre el tiempo del número flotante temporal en unidades por segundo

  `derivative(tfloat): tfloat`

  El número flotante temporal debe tener interpolación linear

```
SELECT derivative(tfloat '{[0@2000-01-01, 10@2000-01-02, 5@2000-01-03],
  [1@2000-01-04, 0@2000-01-05]}') * 3600 * 24;
--  Interp=Stepwise;{[-10@2000-01-01, 5@2000-01-02, 5@2000-01-03],
  [1@2000-01-04, 1@2000-01-05]}
SELECT derivative(tfloat 'Interp=Stepwise;[0@2000-01-01, 10@2000-01-02, 5@2000-01-03]');
-- ERROR:  The temporal value must have linear interpolation
```

5.10. Operadores booleanos

- Y temporal

  `{boolean,tbool} & {boolean,tbool}: tbool`

```
SELECT tbool '[true@2012-01-03, true@2012-01-05)' &
  tbool '[false@2012-01-03, false@2012-01-05)';
-- "[f@2012-01-03, f@2012-01-05)"
SELECT tbool '[true@2012-01-03, true@2012-01-05)' &
  tbool '{[false@2012-01-03, false@2012-01-04),
  [true@2012-01-04, true@2012-01-05)}';
-- "{[f@2012-01-03, t@2012-01-04, t@2012-01-05)}"
```

- O temporal

  `{boolean,tbool} | {boolean,tbool}: tbool`

```
SELECT tbool '[true@2012-01-03, true@2012-01-05)' |
  tbool '[false@2012-01-03, false@2012-01-05)';
-- "[t@2012-01-03, t@2012-01-05)"
```

- No temporal

  `~tbool: tbool`

```
SELECT ~tbool '[true@2012-01-03, true@2012-01-05)';
-- "[f@2012-01-03, f@2012-01-05)"
```

5.11. Funciones y operadores de texto

- Concatenación de texto temporal

  `{text,ttext} || {text,ttext}: ttext`

```
SELECT ttext '[AA@2012-01-01, AA@2012-01-04)' || text 'B';
-- "["AAB"@2012-01-01, "AAB"@2012-01-04)"
SELECT ttext '[AA@2012-01-01, AA@2012-01-04)' || ttext '[BB@2012-01-02, BB@2012-01-05)';
-- "["AABB"@2012-01-02, "AABB"@2012-01-04)"
SELECT ttext '[A@2012-01-01, B@2012-01-03, C@2012-01-04]' ||
  ttext '{[D@2012-01-01, D@2012-01-02), [E@2012-01-02, E@2012-01-04)}';
-- "{["DA"@2012-01-01, "EA"@2012-01-02, "EB"@2012-01-03, "EB"@2012-01-04)}"
```

- Transformar a mayúsculas

  `upper(ttext): ttext`

```
SELECT upper(ttext '[AA@2000-01-01, bb@2000-01-02]');
-- "["AA"@2000-01-01, "BB"@2000-01-02]"
```

- Transformar a minúsculas

  `lower(ttext): ttext`

```
SELECT lower(ttext '[AA@2000-01-01, bb@2000-01-02]');
-- "["aa"@2000-01-01, "bb"@2000-01-02]"
```

5.12. Funciones y operadores espaciales

A continuación, especificamos con el símbolo 🧊 que la función admite puntos 3D y con el símbolo 🌐 que la función está disponible para geografías.

5.12.1. Funciones de entrada/salida

- Obtener la representación de texto conocido (Well-Known Text o WKT) 🧊🌐

  `asText({tpoint,tpoint[],geo[]}): {text,text[]}`

```
SELECT asText(tgeompoint 'SRID=4326;[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-02)') ↩
    ;
-- "[POINT Z (0 0 0)@2012-01-01 00:00:00+00, POINT Z (1 1 1)@2012-01-02 00:00:00+00)"
SELECT asText(ARRAY[geometry 'Point(0 0)', 'Point(1 1)']);
-- "{"POINT(0 0)","POINT(1 1)"}"
```

- Obtener la representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `asEWKT({tpoint,tpoint[],geo[]}): {text,text[]}`

```
SELECT asEWKT(tgeompoint 'SRID=4326;[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-02)') ↩
    ;
-- "SRID=4326;[POINT Z (0 0 0)@2012-01-01 00:00:00+00,
  POINT Z (1 1 1)@2012-01-02 00:00:00+00)"
SELECT asEWKT(ARRAY[geometry 'SRID=5676;Point(0 0)', 'SRID=5676;Point(1 1)']);
-- "{"SRID=5676;POINT(0 0)","SRID=5676;POINT(1 1)"}"
```

- Obtener la representación JSON de característica móviles (Moving Features)

  `asMFJSON(tpoint,maxdecdigits integer=15,options integer=0): bytea`

  El último argumento `options` puede usarse para agregar BBOX y/o CRS en la salida MFJSON:

  - 0: significa que no hay opción (valor por defecto)
  - 1: MFJSON BBOX
  - 2: MFJSON Short CRS (e.g EPSG:4326)
  - 4: MFJSON Long CRS (e.g urn:ogc:def:crs:EPSG::4326)

```
SELECT asMFJSON(tgeompoint 'Point(1 2)@2019-01-01 18:00:00.15+02');
-- "{"type":"MovingPoint","coordinates":[1,2],"datetimes":"2019-01-01T17:00:00.15+01",
  "interpolations":["Discrete"]}"
SELECT asMFJSON(tgeompoint 'SRID=4326;
  Point(50.813810 4.384260)@2019-01-01 18:00:00.15+02', 2, 3);
-- "{"type":"MovingPoint","crs":{"type":"name","properties":{"name":"EPSG:4326"}},
  "stBoundedBy":{"bbox":[50.81,4.38,50.81,4.38],
  "period":{"begin":"2019-01-01 17:00:00.15+01","end":"2019-01-01 17:00:00.15+01"}},
  "coordinates":[50.81,4.38],"datetimes":"2019-01-01T17:00:00.15+01",
  "interpolations":["Discrete"]}"
```

- Obtener la representación binaria conocida (Well-Known Binary o WKB)

  `asBinary(tpoint): bytea`

  `asBinary(tpoint,endian text): bytea`

  El resultado se codifica utilizando la codificación little-endian (NDR) o big-endian (XDR). Si no se especifica ninguna codificación, se utiliza la codificación de la máquina.

```
SELECT asBinary(tgeompoint 'Point(1 2 3)@2012-01-01');
-- "\x0191000000000000f03f0000000000000040000000000000084000fce0136a580100"
```

- Obtener la representación extendida binaria conocida (Extended Well-Known Binary o EWKB)

  `asEWKB(tpoint): bytea`

  `asEWKB(tpoint,endian text): bytea`

  El resultado se codifica utilizando la codificación little-endian (NDR) o big-endian (XDR). Si no se especifica ninguna codificación, se utiliza la codificación de la máquina.

```
SELECT asEWKB(tgeogpoint 'SRID=7844;Point(1 2 3)@2012-01-01');
-- "\x01f1a41e0000000000000000f03f0000000000000040000000000000084000fce0136a580100"
```

- Obtener la representación hexadecimal extendida binaria conocida (Extended Well-Known Binary o EWKB) en formato texto

  `asHexEWKB(tpoint): text`

  `asHexEWKB(tpoint,endian text): text`

  El resultado se codifica utilizando la codificación little-endian (NDR) o big-endian (XDR). Si no se especifica ninguna codificación, se utiliza NDR.

```
SELECT asHexEWKB(tgeompoint 'SRID=3812;Point(1 2 3)@2012-01-01');
-- "01D1E40E0000000000000000F03F0000000000000040000000000000084000FCE0136A580100"
```

- Entrar un punto temporal geométrico en una representación de texto conocido (Well-Known Text o WKT)

  `tgeompointFromText(text): tgeompoint`

```
SELECT asEWKT(tgeompointFromText(text '[POINT(1 2)@2000-01-01, POINT(3 4)@2000-01-02]')) ↩
    ;
-- "[POINT(1 2)@2000-01-01, POINT(3 4)@2000-01-02]"
```

- Entrar un punto temporal geográfico en una representación de texto conocido (Well-Known Text o WKT)

  `tgeogpointFromText(text): tgeogpoint`

```
SELECT asEWKT(tgeogpointFromText(text '[POINT(1 2)@2000-01-01, POINT(3 4)@2000-01-02]')) ↩
    ;
-- "SRID=4326;[POINT(1 2)@2000-01-01, POINT(3 4)@2000-01-02]"
```

- Entrar un punto temporal geométrico en una representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `tgeompointFromEWKT(text): tgeompoint`

```
SELECT asEWKT(tgeompointFromEWKT(text 'SRID=3812;[POINT(1 2)@2000-01-01,
  POINT(3 4)@2000-01-02]'));
-- "SRID=3812;[POINT(1 2)@2000-01-01 00:00:00+01, POINT(3 4)@2000-01-02 00:00:00+01]"
```

- Entrar un punto temporal geográfico en una representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `tgeogpointFromEWKT(text): tgeogpoint`

```
SELECT asEWKT(tgeogpointFromEWKT(text 'SRID=7844;[POINT(1 2)@2000-01-01,
  POINT(3 4)@2000-01-02]'));
-- "SRID=7844;[POINT(1 2)@2000-01-01, POINT(3 4)@2000-01-02]"
```

- Entrar un punto temporal geométrico en una representación JSON de características móviles (Moving Features)

  `tgeompointFromMFJSON(text): tgeompoint`

```
SELECT asEWKT(tgeompointFromMFJSON(text '{"type":"MovingPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[50.81,4.38],
  "datetimes":"2019-01-01T17:00:00.15+01","interpolations":["Discrete"]}'));
-- "SRID=4326;POINT(50.81 4.38)@2019-01-01 17:00:00.15+01"
```

- Entrar un punto temporal geográfico en una representación JSON de características móviles (Moving Features)
  `tgeogpointFromMFJSON(text): tgeogpoint`

```
SELECT asEWKT(tgeogpointFromMFJSON(text '{"type":"MovingPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[50.81,4.38],
  "datetimes":"2019-01-01T17:00:00.15+01","interpolations":["Discrete"]}'));
-- "SRID=4326;POINT(50.81 4.38)@2019-01-01 17:00:00.15+01"
```

- Entrar un punto temporal geométrico en una representación binaria conocida (WKB)
  `tgeompointFromBinary(bytea): tgeompoint`

```
SELECT asEWKT(tgeompointFromBinary(
  '\x018100000000000f03f0000000000000040005c6c29ffffffff'));
-- "POINT(1 2)@2000-01-01"
```

- Entrar un punto temporal geográfico en una representación binaria conocida (WKB)
  `tgeogpointFromBinary(bytea): tgeogpoint`

```
SELECT asEWKT(tgeompointFromBinary(
  '\x01b1000000000000f03f000000000000f03f000000000000f03f005c6c29ffffffff'));
-- "SRID=4326;POINT Z (1 1 1)@2000-01-01"
```

- Entrar un punto temporal geométrico en una representación extendida binaria conocida (EWKB)
  `tgeompointFromEWKB(bytea): tgeompoint`

```
SELECT asEWKT(tgeompointFromEWKB(
  '\x01c1e40e0000000000000000f03f0000000000000040005c6c29ffffffff'));
-- "SRID=3812;POINT(1 2)@2000-01-01"
```

- Entrar un punto temporal geográfico en una representación extendida binaria conocida (EWKB)
  `tgeogpointFromEWKB(bytea): tgeogpoint`

```
SELECT asEWKT(tgeogpointFromEWKB(
  '\x01f1a41e0000000000000000f03f000000000000f03f000000000000f03f005c6c29ffffffff'));
-- "SRID=7844;POINT Z (1 1 1)@2000-01-01"
```

- Entrar un punto temporal geométrico en una representación hexadecimal extendida binaria conocida (HexEWKB)
  `tgeompointFromHexEWKB(text): tgeompoint`

```
SELECT asEWKT(tgeompointFromHexEWKB(
  '01C1E40E0000000000000000F03F0000000000000040005C6C29FFFFFFFF'));
-- "SRID=3812;POINT(1 2)@2000-01-01"
```

- Entrar un punto temporal geográfico en una representación hexadecimal extendida binaria conocida (HexEWKB)
  `tgeogpointFromHexEWKB(text): tgeogpoint`

```
SELECT asEWKT(tgeogpointFromHexEWKB(
  '01F1A41E0000000000000000F03F000000000000F03F000000000000F03F005C6C29FFFFFFFF'));
-- "SRID=7844;POINT Z (1 1 1)@2000-01-01"
```

5.12.2. Funciones de sistema de referencia espacial

- Obtener el identificador de referencia espacial
  `SRID(tpoint): integer`

```
SELECT SRID(tgeompoint 'Point(0 0)@2012-01-01');
-- 0
```

- Establecer el identificador de referencia espacial
  `setSRID(tpoint): tpoint`

```
SELECT asEWKT(setSRID(tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-02)', 4326) ↩
    );
-- "SRID=4326;[POINT(0 0)@2012-01-01 00:00:00+00, POINT(1 1)@2012-01-02 00:00:00+00)"
```

- Transformar a una referencia espacial diferente
  `transform(tpoint,integer): tpoint`

```
SELECT asEWKT(transform(tgeompoint 'SRID=4326;Point(4.35 50.85)@2012-01-01', 3812));
-- "SRID=3812;POINT(648679.018035303 671067.055638114)@2012-01-01 00:00:00+00"
```

5.12.3. Funciones de accessor

- Obtener los valores de las coordenadas X como un número flotante temporal
  `getX(tpoint): tfloat`

```
SELECT getX(tgeompoint '{Point(1 2)@2000-01-01, Point(3 4)@2000-01-02,
  Point(5 6)@2000-01-03}');
-- "{1@2000-01-01, 3@2000-01-02, 5@2000-01-03}"
SELECT getX(tgeogpoint 'Interp=Stepwise;[Point(1 2 3)@2000-01-01, Point(4 5 6)@2000 ↩
    -01-02,
  Point(7 8 9)@2000-01-03]');
-- "Interp=Stepwise;[1@2000-01-01, 4@2000-01-02, 7@2000-01-03]"
```

- Obtener los valores de las coordenadas Y como un número flotante temporal
  `getY(tpoint): tfloat`

```
SELECT getY(tgeompoint '{Point(1 2)@2000-01-01, Point(3 4)@2000-01-02,
  Point(5 6)@2000-01-03}');
-- "{2@2000-01-01, 4@2000-01-02, 6@2000-01-03}"
SELECT getY(tgeogpoint 'Interp=Stepwise;[Point(1 2 3)@2000-01-01, Point(4 5 6)@2000 ↩
    -01-02,
  Point(7 8 9)@2000-01-03]');
-- "Interp=Stepwise;[2@2000-01-01, 5@2000-01-02, 8@2000-01-03]"
```

- Obtener los valores de las coordenadas Z como un número flotante temporal
  `getZ(tpoint): tfloat`

```
SELECT getZ(tgeompoint '{Point(1 2)@2000-01-01, Point(3 4)@2000-01-02,
  Point(5 6)@2000-01-03}');
-- The temporal point do not have Z dimension
SELECT getZ(tgeogpoint 'Interp=Stepwise;[Point(1 2 3)@2000-01-01, Point(4 5 6)@2000 ↩
    -01-02,
  Point(7 8 9)@2000-01-03]');
-- "Interp=Stepwise;[3@2000-01-01, 6@2000-01-02, 9@2000-01-03]"
```

- Devuelve verdadero si el punto temporal no se auto-intersecta espacialmente

  `isSimple(tpoint): boolean`

  Nótese que un punto temporal de conjunto de secuencias es simple si cada una de las secuencias que lo componen es simple.

```
SELECT isSimple(tgeompoint '[Point(0 0)@2000-01-01, Point(1 1)@2000-01-02,
  Point(0 0)@2000-01-03]');
-- false
SELECT isSimple(tgeompoint '[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000-01-02,
  Point(2 0 2)@2000-01-03, Point(0 0 0)@2000-01-04]');
-- true
SELECT isSimple(tgeompoint '{[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000-01-02],
  [Point(1 1 1)@2000-01-03, Point(0 0 0)@2000-01-04]}');
-- true
```

- Obtener la longitud atravesada por el punto temporal

  `length(tpoint): float`

```
SELECT length(tgeompoint '[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000-01-02]');
-- 1.73205080756888
SELECT length(tgeompoint '[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000-01-02,
  Point(0 0 0)@2000-01-03]');
-- 3.46410161513775
SELECT length(tgeompoint 'Interp=Stepwise;[Point(0 0 0)@2000-01-01,
  Point(1 1 1)@2000-01-02, Point(0 0 0)@2000-01-03]');
-- 0
```

- Obtener la longitud acumulada atravesada por el punto temporal

  `cumulativeLength(tpoint): tfloat_seq`

```
SELECT round(cumulativeLength(tgeompoint '{[Point(0 0)@2000-01-01, Point(1 1)@2000 ↩
    -01-02,
  Point(1 0)@2000-01-03], [Point(1 0)@2000-01-04, Point(0 0)@2000-01-05]}'), 6);
-- {[0@2000-01-01, 1.414214@2000-01-02, 2.414214@2000-01-03],
  [2.414214@2000-01-04, 3.414214@2000-01-05]}
SELECT cumulativeLength(tgeompoint 'Interp=Stepwise;[Point(0 0 0)@2000-01-01,
  Point(1 1 1)@2000-01-02, Point(0 0 0)@2000-01-03]');
-- Interp=Stepwise;[0@2000-01-01, 0@2000-01-03]
```

- Obtener la velocidad del punto temporal en unidades por segundo

  `speed(tpoint): tfloat_seqset`

  El punto temporal debe tener interpolación linear

```
SELECT speed(tgeompoint '{[Point(0 0)@2000-01-01, Point(1 1)@2000-01-02,
  Point(1 0)@2000-01-03], [Point(1 0)@2000-01-04, Point(0 0)@2000-01-05]}') * 3600 * 24;
-- "Interp=Stepwise;{[1.4142135623731@2000-01-01, 1@2000-01-02, 1@2000-01-03],
  [1@2000-01-04, 1@2000-01-05]}"
SELECT speed(tgeompoint 'Interp=Stepwise;[Point(0 0)@2000-01-01, Point(1 1)@2000-01-02,
  Point(1 0)@2000-01-03]');
-- ERROR:  The temporal value must have linear interpolation
```

- Obtener el centroide ponderado en el tiempo

`twCentroid(tgeompoint): point`

```
SELECT ST_AsText(twCentroid(tgeompoint '{[Point(0 0 0)@2012-01-01,
  Point(0 1 1)@2012-01-02, Point(0 1 1)@2012-01-03, Point(0 0 0)@2012-01-04)}'));
-- "POINT Z (0 0.666666666666667 0.666666666666667)"
```

- Obtener el acimut temporal

`azimuth(tpoint): tfloat`

```
SELECT degrees(azimuth(tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-02,
  Point(1 1 1)@2012-01-03, Point(0 0 0)@2012-01-04)'));
-- "Interp=Stepwise;{[45@2012-01-01, 45@2012-01-02], [225@2012-01-03, 225@2012-01-04)}"
```

- Obtener el rumbo temporal

`bearing({tpoint, point}, {tpoint, point}): tfloat`

```
SELECT degrees(bearing(tgeompoint '[Point(1 1)@2012-01-01, Point(3 3)@2012-01-03]',
  geometry 'Point(2 2)'));
-- [45@2012-01-01, 0@2012-01-02, 225@2012-01-03]
SELECT round(degrees(bearing(tgeompoint '[Point(0 0)@2012-01-01, Point(2 0)@2012 ↩
    -01-03]',
  tgeompoint '[Point(2 1)@2012-01-01, Point(0 1)@2012-01-03]')), 3);
--  [63.435@2012-01-01, 0@2012-01-02, 296.565@2012-01-03]
SELECT round(degrees(bearing(tgeompoint '[Point(2 1)@2012-01-01, Point(0 1)@2012 ↩
    -01-03]',
  tgeompoint '[Point(0 0)@2012-01-01, Point(2 0)@2012-01-03]')), 3);
-- [243.435@2012-01-01, 116.565@2012-01-03]
```

Tenga en cuenta que esta función actualmente no acepta dos puntos geográficos temporales.

5.12.4. Funciones de manipulación

- Redondear los valores de las coordenadas a un número de decimales

`setPrecision(tpoint,integer): tpoint`

```
SELECT asText(setPrecision(tgeompoint '{Point(1.12345 1.12345 1.12345)@2000-01-01,
  Point(2 2 2)@2000-01-02, Point(1.12345 1.12345 1.12345)@2000-01-03}', 2));
-- "{POINT Z (1.12 1.12 1.12)@2000-01-01, POINT Z (2 2 2)@2000-01-02,
  POINT Z (1.12 1.12 1.12)@2000-01-03}"
SELECT asText(setPrecision(tgeogpoint 'Point(1.12345 1.12345)@2000-01-01', 2));
-- "POINT(1.12 1.12)@2000-01-01"
```

- Devuelve una matriz de fragmentos del punto temporal que son simples
  `makeSimple(tpoint): tgeompoint[]`

```
SELECT asText(makeSimple(tgeompoint '[Point(0 0)@2000-01-01, Point(1 1)@2000-01-02,
  Point(0 0)@2000-01-03]'));
-- {"[POINT(0 0)@2000-01-01, POINT(1 1)@2000-01-02)",
   "[POINT(1 1)@2000-01-02, POINT(0 0)@2000-01-03]"}
SELECT asText(makeSimple(tgeompoint '[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000-01-02,
  Point(2 0 2)@2000-01-03, Point(0 0 0)@2000-01-04]'));
-- {"[POINT Z (0 0 0)@2000-01-01, POINT Z (1 1 1)@2000-01-02, POINT Z (2 0 2)@2000 ↩
    -01-03,
   POINT Z (0 0 0)@2000-01-04]"}
SELECT asText(makeSimple(tgeompoint '[Point(0 0)@2000-01-01, Point(1 1)@2000-01-02,
  Point(0 1)@2000-01-03, Point(1 0)@2000-01-04]'));
-- {"[POINT(0 0)@2000-01-01, POINT(1 1)@2000-01-02, POINT(0 1)@2000-01-03)",
   "[POINT(0 1)@2000-01-03, POINT(1 0)@2000-01-04]"}
SELECT asText(makeSimple(tgeompoint '{[Point(0 0 0)@2000-01-01, Point(1 1 1)@2000 ↩
    -01-02],
  [Point(1 1 1)@2000-01-03, Point(0 0 0)@2000-01-04]}'));
-- {"{[POINT Z (0 0 0)@2000-01-01, POINT Z (1 1 1)@2000-01-02],
   [POINT Z (1 1 1)@2000-01-03, POINT Z (0 0 0)@2000-01-04]}"}
```

- Simplificar un punto temporal usando una generalización del algoritmo de Douglas-Peucker
  `simplify(tpoint,distance float): tpoint`
  `simplify(tpoint,distance float,speed float): tpoint`
  La primera versión elimina los puntos cuya distancia es menor que la distancia pasada como segundo argumento, que se especifica en las unidades del sistema de coordenadas. La segunda versión elimina los puntos cuya distancia es menor que la distancia pasada como segundo argumento siempre que la diferencia de velocidad entre el punto y el punto correspondiente en la versión simplificada sea menor que la velocidad pasada como tercer argumento, que se especifica en unidades por segundo. Observe que la simplificación se aplica sólo a secuencias temporales o conjuntos de secuencias con interpolación lineal. En todos los demás casos, se devuelve una copia del punto temporal dado.

```
-- Solo distancia especificada
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 1.5)));
-- "LINESTRING(0 4,1 1,4 3,5 0,6 4)"
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 2)));
-- "LINESTRING(0 4,5 0,6 4)"
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 4)));
-- "LINESTRING(0 4,6 4)"

-- Solo diferencia de velocidad especificada
SELECT round(speed(tgeompoint '[Point(0 4)@2000-01-01, Point(1 1)@2000-01-02,
  Point(2 3)@2000-01-03, Point(3 1)@2000-01-04, Point(4 3)@2000-01-05,
  Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]') * 1e5, 2);
-- "Interp=Stepwise;[3.66@2000-01-01, 2.59@2000-01-02, 3.66@2000-01-05,
  4.77@2000-01-06, 4.77@2000-01-07]"

-- Se especifican tanto la distancia como la diferencia de velocidad
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 4, 1 / 1e5)));
```

```
-- "LINESTRING(0 4,1 1,2 3,3 1,4 3,5 0,6 4)"
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 4, 2 / 1e5)));
-- "LINESTRING(0 4,1 1,5 0,6 4)"
SELECT ST_AsText(trajectory(simplify(tgeompoint '[Point(0 4)@2000-01-01,
  Point(1 1)@2000-01-02, Point(2 3)@2000-01-03, Point(3 1)@2000-01-04,
  Point(4 3)@2000-01-05, Point(5 0)@2000-01-06, Point(6 4)@2000-01-07]', 4, 3 / 1e5)));
-- "LINESTRING(0 4,6 4)"
```

Un uso típico de la función `simplify` es reducir el tamaño de un conjunto de datos, en particular con fines de visualización.

- Construir una geometría/geografía con medida M a partir de un punto temporal y un número flotante temporal

  `geoMeasure(tpoint,tfloat,segmentize=false): geo`

  El último argumento `segmentize` establece si el valor resultado ya sea es un `Linestring M` o un `MultiLinestring M` donde cada componente es un segmento de dos puntos.

```
SELECT st_astext(geoMeasure(tgeompoint '{Point(1 1 1)@2000-01-01,
  Point(2 2 2)@2000-01-02}', '{5@2000-01-01, 5@2000-01-02}'));
-- "MULTIPOINT ZM (1 1 1 5,2 2 2 5)"
SELECT st_astext(geoMeasure(tgeogpoint '{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02],
  [Point(1 1)@2000-01-03, Point(1 1)@2000-01-04]}',
  '{[5@2000-01-01, 5@2000-01-02],[7@2000-01-03, 7@2000-01-04]}'));
-- "GEOMETRYCOLLECTION M (LINESTRING M (1 1 5,2 2 5),POINT M (1 1 7))"
SELECT st_astext(geoMeasure(tgeompoint '[Point(1 1)@2000-01-01,
  Point(2 2)@2000-01-02, Point(1 1)@2000-01-03]',
  '[5@2000-01-01, 7@2000-01-02, 5@2000-01-03]', true));
-- "MULTILINESTRING M ((1 1 5,2 2 5),(2 2 7,1 1 7))"
```

Una visualización típica de los datos de movilidad es mostrar en un mapa la trayectoria del objeto móvil utilizando diferentes colores según la velocidad. La Figura 5.1 muestra el resultado de la consulta a continuación usando una rampa de color en QGIS.

```
WITH Temp(t) AS (
  SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-05,
    Point(2 0)@2012-01-08, Point(3 1)@2012-01-10, Point(4 0)@2012-01-11]'
)
SELECT ST_AsText(geoMeasure(t, round(speed(t) * 3600 * 24, 2), true))
FROM Temp;
-- "MULTILINESTRING M ((0 0 0.35,1 1 0.35),(1 1 0.47,2 0 0.47),(2 0 0.71,3 1 0.71),
  (3 1 1.41,4 0 1.41))"
```

La siguiente expresión se usa en QGIS para lograr esto. La función `scale_linear` transforma el valor M de cada segmento componente al rango [0, 1]. Este valor luego se pasa a la función `ramp_color`.

```
ramp_color(
  'RdYlBu',
  scale_linear(
    m(start_point(geometry_n($geometry,@geometry_part_num))),
    0, 2, 0, 1)
)
```

- Transformar un punto geométrico temporal en el espacio de coordenadas de un Mapbox Vector Tile. El resultado es un par compuesto de un valor `geometry` y una matriz de valores de marca de tiempo asociados codificados como época de Unix

  `asMVTGeom(tpoint,bounds,extent=4096,buffer=256,clip=TRUE): geom_times`

  Los parámetros son los siguientes:

Figura 5.1: Visualización de la velocidad de un objeto móvil usando una rampa de color en QGIS.

- `tpoint` es el punto temporal para transformar
- `bounds` es un `stbox` que define los límites geométricos del contenido del mosaico sin búfer
- `extent` es la extensión del mosaico en el espacio de coordenadas del mosaico
- `buffer` es la distancia del búfer en el espacio de coordenadas de mosaico
- `clip` es un booleano que determina si las geometrías resultantes y las marcas de tiempo deben recortarse o no

```
SELECT ST_AsText((mvt).geom), (mvt).times
FROM (SELECT asMVTGeom(tgeompoint '[Point(0 0)@2000-01-01, Point(100 100)@2000-01-02]',
  stbox 'STBOX((40,40),(60,60))') AS mvt ) AS t;
-- LINESTRING(-256 4352,4352 -256) | {946714680,946734120}
SELECT ST_AsText((mvt).geom), (mvt).times
FROM (SELECT asMVTGeom(tgeompoint '[Point(0 0)@2000-01-01, Point(100 100)@2000-01-02]',
  stbox 'STBOX((40,40),(60,60))', clip:=false) AS mvt ) AS t;
-- LINESTRING(-8192 12288,12288 -8192) | {946681200,946767600}
```

5.12.5. Funciones y operadores de distancia

- Obtener la distancia más pequeña que haya existido
  `{geo,tpoint} |=| {geo,tpoint}: float`

```
SELECT tgeompoint '[Point(0 0)@2012-01-02, Point(1 1)@2012-01-04, Point(0 0)@2012-01-06)'
  |=| geometry 'Linestring(2 2,2 1,3 1)';
-- 1
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03, Point(0 0)@2012-01-05)'
  |=| tgeompoint '[Point(2 0)@2012-01-02, Point(1 1)@2012-01-04, Point(2 2)@2012-01-06)';
-- 0.5
SELECT tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
  Point(0 0 0)@2012-01-05)' |=| tgeompoint '[Point(2 0 0)@2012-01-02,
  Point(1 1 1)@2012-01-04, Point(2 2 2)@2012-01-06)';
-- 0.5
SELECT tgeompoint 'Interp=Stepwise;(Point(1 1)@2000-01-01, Point(3 1)@2000-01-03]' |=|
  geometry 'Linestring(1 3,2 2,3 3)';
-- 1.4142135623731
```

El operador `|=|` se puede utilizar para realizar una búsqueda de vecino más cercano utilizando un índice GiST (ver la Sección 5.7). Esta función corresponde a la función `ST_DistanceCPA` proporcionada por PostGIS, aunque este última requiere que ambos argumentos sean una trayectoria.

```
SELECT ST_DistanceCPA(
  tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
    Point(0 0 0)@2012-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04,
    Point(2 2 2)@2012-01-06)'::geometry);
-- 0.5
```

- Obtener el instante del primer punto temporal en el que los dos argumentos están a la distancia más cercana

  `nearestApproachInstant({geo,tpoint},{geo,tpoint}): tpoint`

  La función sólo devuelve el primer instante que encuentre si hay más de uno. El instante resultante puede tener un límite exclusivo.

```
SELECT asText(NearestApproachInstant(tgeompoint '(Point(1 1)@2000-01-01,
  Point(3 1)@2000-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- "POINT(2 1)@2000-01-02"
SELECT asText(NearestApproachInstant(tgeompoint 'Interp=Stepwise;(Point(1 1)@2000-01-01,
  Point(3 1)@2000-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- "POINT(1 1)@2000-01-01"
SELECT asText(NearestApproachInstant(tgeompoint '(Point(1 1)@2000-01-01,
  Point(2 2)@2000-01-03]', tgeompoint '(Point(1 1)@2000-01-01, Point(4 1)@2000-01-03]')) ↩
    ;
-- "POINT(1 1)@2000-01-01"
SELECT asText(nearestApproachInstant(tgeompoint '[Point(0 0 0)@2012-01-01,
  Point(1 1 1)@2012-01-03, Point(0 0 0)@2012-01-05)', tgeompoint
  '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04, Point(2 2 2)@2012-01-06)'));
-- "POINT Z (0.75 0.75 0.75)@2012-01-03 12:00:00+00"
```

La función `nearestApproachInstant` generaliza the la función PostGIS `ST_ClosestPointOfApproach`. Primero, la última función requiere que ambos argumentos sean trayectorias. Segundo, la función `nearestApproachInstant` devuelve tanto el punto como la marca de tiempo del punto de aproximación más cercano, mientras que la función PostGIS sólo proporciona la marca de tiempo como se muestra a continuación.

```
SELECT to_timestamp(ST_ClosestPointOfApproach(
  tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
    Point(0 0 0)@2012-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04,
    Point(2 2 2)@2012-01-06)'::geometry));
-- "2012-01-03 12:00:00+00"
```

- Obtener la línea que conecta el punto de aproximación más cercano

  `shortestLine({geo,tpoint},{geo,tpoint}): geo`

  La función sólo devolverá la primera línea que encuentre si hay más de una.

```
SELECT ST_AsText(shortestLine(tgeompoint '(Point(1 1)@2000-01-01,
  Point(3 1)@2000-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- "LINESTRING(2 1,2 2)"
SELECT ST_AsText(shortestLine(tgeompoint 'Interp=Stepwise;(Point(1 1)@2000-01-01,
  Point(3 1)@2000-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- "LINESTRING(1 1,2 2)"
SELECT ST_AsText(shortestLine(
  tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
    Point(0 0 0)@2012-01-05)',
  tgeompoint '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04,
    Point(2 2 2)@2012-01-06)'));
-- "LINESTRING Z (0.75 0.75 0.75,1.25 0.75 0.75)"
```

La función `shortestLine` se puede utilizar para obtener el resultado proporcionado por la función PostGIS `ST_CPAWithin` cuando ambos argumentos son trayectorias como se muestra a continuación.

```
SELECT ST_Length(shortestLine(
  tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
```

```
    Point(0 0 0)@2012-01-05)',
  tgeompoint '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04,
    Point(2 2 2)@2012-01-06)')) <= 0.5;
-- true
SELECT ST_CPAWithin(
  tgeompoint '[Point(0 0 0)@2012-01-01, Point(1 1 1)@2012-01-03,
    Point(0 0 0)@2012-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2012-01-02, Point(1 1 1)@2012-01-04,
    Point(2 2 2)@2012-01-06)'::geometry, 0.5);
-- true
```

El operador de distancia temporal, denotado `<->`, calcula la distancia en cada instante de la intersección de las extensiones temporales de sus argumentos y da como resultado un número flotante temporal. Calcular la distancia temporal es útil en muchas aplicaciones de movilidad. Por ejemplo, un grupo en movimiento (también conocido como convoy o bandada) se define como un conjunto de objetos que se mueven cerca unos de otros durante un intervalo de tiempo prolongado. Esto requiere calcular la distancia temporal entre dos objetos en movimiento.

El operador de distancia temporal acepta una geometría/geografía restringida a un punto o un punto temporal como argumentos. Observe que los tipos temporales sólo consideran la interpolación lineal entre valores, mientras que la distancia es una raíz de una función cuadrática. Por lo tanto, el operador de distancia temporal proporciona una aproximación lineal del valor de distancia real para los puntos de secuencia temporal. En este caso, los argumentos se sincronizan en la dimensión de tiempo y para cada uno de los segmentos de línea que componen los argumentos, se calcula la distancia espacial entre el punto inicial, el punto final y el punto de aproximación más cercano, como se muestra en los ejemplos a continuación.

- Obtener la distancia temporal

  `{point,tpoint} <-> {point,tpoint}: tfloat`

```
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03)' <->
  geometry 'Point(0 1)';
-- "[1@2012-01-01, 0.707106781186548@2012-01-02, 1@2012-01-03)"
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03)' <->
  tgeompoint '[Point(0 1)@2012-01-01, Point(1 0)@2012-01-03)';
-- "[1@2012-01-01, 0@2012-01-02, 1@2012-01-03)"
SELECT tgeompoint '[Point(0 1)@2012-01-01, Point(0 0)@2012-01-03)' <->
  tgeompoint '[Point(0 0)@2012-01-01, Point(1 0)@2012-01-03)';
-- "[1@2012-01-01, 0.707106781186548@2012-01-02, 1@2012-01-03)"
SELECT tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-02)' <->
  tgeompoint '[Point(0 1)@2012-01-01, Point(1 2)@2012-01-02)';
-- "[1@2012-01-01,1@2012-01-02)"
```

5.12.6. Relaciones espaciales

Las relaciones topológicas como `ST_Intersects` y las relaciones de distancia como `ST_DWithin` pueden ser generalizadas a los puntos temporales. Los argumentos de estas funciones generalizadas son un punto temporal y un tipo base (es decir, un `geometry` o un `geography`) o dos puntos temporales. Además, ambos argumentos deben ser del mismo tipo base, es decir, estas funciones no permiten merzclar un punto de geometría temporal (o una geometría) y un punto de geografía temporal (o una geografía).

Hay dos versiones de las relaciones espaciales:

- Las *relaciones possibles* aplican la función topológica o de distancia tradicional a la unión de todos los valores tomados por el punto temporal (que es un `geometry` o `geography`) y resulta en un `boolean`. Ejemplos son las funciones `intersects` y `dwithin`.
- Las *relaciones temporales* se definen con la semántica temporal, es decir, la función topológica o de distancia tradicional se calcula en cada instante y da como resultado un `tbool`. Ejemplos son las funciones `tintersects` y `tdwithin`.

La semántica de las relaciones posibles varía según la relación y el tipo de argumentos. Por ejemplo, la siguiente consulta

```
SELECT intersects(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 1)@2012-01-01, Point(1 1)@2012-01-03)');
```

comprueba si el punto temporal se cruzó alguna vez con la geometría, ya que la consulta es conceptualmente equivalente a la siguiente

```
SELECT ST_Intersects(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  geometry 'Linestring(0 1,1 1)');
```

donde la segunda geometría se obtiene aplicando la función `trajectory` al punto temporal. Por otro lado, la consulta

```
SELECT contains(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 1)@2012-01-01, Point(1 1)@2012-01-03)');
```

comprueba si la geometría siempre contiene el punto temporal. Finalmente, la siguiente consulta

```
SELECT intersects(tgeompoint '[Point(0 1)@2012-01-01, Point(1 0)@2012-01-03)',
  tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03)');
```

comprueba si los puntos temporales pueden cruzarse en el espacio independientemente del tiempo, ya que la consulta anterior es conceptualmente equivalente a la siguiente

```
SELECT ST_Intersects('Linestring(0 1,1 0)', 'Linestring(0 0,1 1)');
```

Las relaciones posibles se utilizan normalmente en combinación con un índice espacio-temporal al calcular las relaciones temporales. Por ejemplo, la siguiente consulta

```
SELECT T.TripId, R.RegionId, tintersects(T.Trip, R.Geom)
FROM Trips T, Regions R
WHERE intersects(T.Trip, R.Geom)
```

que verifica si un viaje `T` (que es un punto temporal) se cruza con una región `R` (que es una geometría) beneficiará de un índice espacio-temporal en la columna `T.Trip` dado que la función `intersects` realiza automáticamente la comparación del cuadro delimitador `T.Trip && R.Geom`. Esto se explica más adelante en este documento.

No todas las relaciones espaciales disponibles en PostGIS tienen una generalización significativa para los puntos temporales. Se define una versión generalizada de las siguientes relaciones para los puntos geométricos temporales: `intersects`, `disjoint`, `dwithin`, `contains`, y `touches`, mientras que para los puntos geográficos temporales solo se definen las tres primeras. Además, no todas las combinaciones de parámetros son significativas para una función generalizada dada. Por ejemplo, mientras que `tcontains(geometry,tpoint)` es significativo, `tcontains(tpoint,geometry)` es significativo solo cuando la geometría es un solo punto, y `tcontains(tpoint,tpoint)` es equivalente a `tintersects(tpoint, geometry)`. Por esta razón, solo se define la primera combinación de parámetros para `contains` y `tcontains`.

Finalmente, cabe destacar que las relaciónes temporales permiten mezclar geometrías 2D/3D pero en ese caso, el cálculo sólo se realiza en 2D.

5.12.7. Relaciones topológicas posibles

- Puede contener

  `contains({geo,tgeompoint},{geo,tgeompoint}): boolean`

```
SELECT contains(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03)');
-- true
```

- Puede ser disjunto

  `disjoint({geo,tgeompoint},{geo,tgeompoint}): boolean`

```
SELECT disjoint(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(1 1)@2012-01-03)');
-- false
```

- Puede estar a distancia de ⬡

  `dwithin({geo,tpoint},{geo,tpoint},float): boolean`

```
SELECT dwithin(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint 'Point(0 2 1)@2000-01-01,Point(2 2 1)@2000-01-02', 1)
-- true
SELECT dwithin(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint 'Point(0 2 2)@2000-01-01,Point(2 2 2)@2000-01-02', 1)
-- false
```

- Puede cruzarse ⬡

  `intersects({geo,tpoint},{geo,tpoint}): boolean`

```
SELECT intersects(geometry 'Polygon((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))',
  tgeompoint '[Point(0 0 1)@2012-01-01, Point(1 1 1)@2012-01-03)');
-- false
SELECT intersects(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint '[Point(0 0 1)@2012-01-01, Point(1 1 1)@2012-01-03)');
-- true
```

- Puede tocar

  `touches({geo,tgeompoint},{geo,tgeompoint}): boolean`

```
SELECT touches(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(0 1)@2012-01-03)');
-- true
```

5.12.8. Relaciones topológicas temporales

- Contiene temporal

  `tcontains({geo,tgeompoint},{geo,tgeompoint}): tbool`

```
SELECT tcontains(geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)');
-- "{[f@2012-01-01, f@2012-01-02], (t@2012-01-02, f@2012-01-03, f@2012-01-04)}"
```

- Disjunto temporal

```
tdisjoint({geo,tgeompoint},{geo,tgeompoint}): tbool
```

```
SELECT tdisjoint(geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)');
-- "{[t@2012-01-01, f@2012-01-02, f@2012-01-03], (t@2012-01-03, t@2012-01-04]}"
SELECT tdisjoint(tgeompoint '[Point(0 3)@2012-01-01, Point(3 0)@2012-01-05)',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-05)');
-- "{[t@2012-01-01, f@2012-01-03], (t@2012-01-03, t@2012-01-05)}"
```

- Estar a distancia de temporal

```
tdwithin({geo,tpoint},{geo,tpoint},float): tbool
```

```
SELECT tdwithin(geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 0)@2012-01-04)', 1);
-- "{[f@2012-01-01, t@2012-01-02, t@2012-01-03], (f@2012-01-03, f@2012-01-04)}"
SELECT tdwithin(tgeompoint '[Point(1 0)@2000-01-01, Point(1 4)@2000-01-05]',
  tgeompoint 'Interp=Stepwise;[Point(1 2)@2000-01-01, Point(1 3)@2000-01-05]', 1);
-- "{[f@2000-01-01, t@2000-01-02, t@2000-01-04], (f@2000-01-04, t@2000-01-05]}"
```

- Intersección temporal

```
tintersects({geo,tpoint},{geo,tpoint}): tbool
```

```
SELECT tintersects(geometry 'MultiPoint(1 1,2 2)',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)');
-- "{[f@2012-01-01, t@2012-01-02], (f@2012-01-02, t@2012-01-03],
  (f@2012-01-03, f@2012-01-04]}"
SELECT tintersects(tgeompoint '[Point(0 3)@2012-01-01, Point(3 0)@2012-01-05)',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 3)@2012-01-05)');
-- "{[f@2012-01-01, t@2012-01-03], (f@2012-01-03, f@2012-01-05)}"
```

- Toca temporal

```
ttouches({geo,tgeompoint},{geo,tgeompoint}): tbool
```

```
SELECT ttouches(geometry 'Polygon((1 0,1 2,2 2,2 0,1 0))',
  tgeompoint '[Point(0 0)@2012-01-01, Point(3 0)@2012-01-04)');
-- "{[f@2012-01-01, t@2012-01-02, t@2012-01-03], (f@2012-01-03, f@2012-01-04]}"
```

5.13. Funciones de similaridad

- Obtener la distancia de Fréchet discreta entre dos valores temporales

```
frechetDistance({tnumber, tgeo}, {tnumber, tgeo}): float
```

Esta función tiene una complejidad de espacio lineal ya que solo dos filas de la matriz de distancia son asignadas en la memoria. Sin embargo, su complejidad de tiempo es cuadrática en el número de instantes de los valores temporales. Por lo tanto, la función requerirá un tiempo considerable para valores temporales con gran número de instantes.

```
SELECT frechetDistance(tfloat '[1@2012-01-01, 3@2012-01-03, 1@2012-01-06]',
 tfloat '[1@2012-01-01, 1.5@2012-01-02, 2.5@2012-01-03, 1.5@2012-01-04, 1.5@2012-01-05]');
-- 0.5
SELECT round(frechetDistance(tgeompoint '[Point(1 1)@2012-01-01, Point(3 3)@2012-01-03,
  Point(1 1)@2012-01-05]', tgeompoint '[Point(1.1 1.1)@2012-01-01,
  Point(2.5 2.5)@2012-01-02, Point(4 4)@2012-01-03, Point(3 3)@2012-01-04,
  Point(1.5 2)@2012-01-05]')::numeric, 6);
-- 1.414214
```

- Obtener las parejas de correspondencia entre dos valores temporales con respecto a la distancia de Fréchet discreta

  `frechetDistancePath({tnumber, tgeo}, {tnumber, tgeo}): pairs`

  Esta función requiere ubicar en memoria una matriz de distancias cuyo tamaño es cuadrático en el número de instantes de los valores temporales. Por tanto, la función fallará para valores temporales con gran número de instantes dependiendo de la memoria disponible.

```
SELECT frechetDistancePath(tfloat '[1@2012-01-01, 3@2012-01-03, 1@2012-01-06]',
 tfloat '[1@2012-01-01, 1.5@2012-01-02, 2.5@2012-01-03, 1.5@2012-01-04, 1.5@2012-01-05]');
-- (0,0)
   (1,0)
   (2,1)
   (3,2)
   (4,2)
SELECT frechetDistancePath(tgeompoint '[Point(1 1)@2012-01-01, Point(3 3)@2012-01-03,
  Point(1 1)@2012-01-05]', tgeompoint '[Point(1.1 1.1)@2012-01-01,
  Point(2.5 2.5)@2012-01-02, Point(4 4)@2012-01-03, Point(3 3)@2012-01-04,
  Point(1.5 2)@2012-01-05]');
-- (0,0)
   (1,1)
   (2,1)
   (3,1)
   (4,2)
```

- Obtener la distancia de distorsión de tiempo dinámica (Dynamic Time Warp o DTW) entre dos valores temporales

  `dynamicTimeWarp({tnumber, tgeo}, {tnumber, tgeo}): float`

  Esta función tiene una complejidad de espacio lineal ya que solo dos filas de la matriz de distancia son asignadas en la memoria. Sin embargo, su complejidad de tiempo es cuadrática en el número de instantes de los valores temporales. Por lo tanto, la función requerirá un tiempo considerable para valores temporales con gran número de instantes.

```
SELECT dynamicTimeWarp(tfloat '[1@2012-01-01, 3@2012-01-03, 1@2012-01-06]',
 tfloat '[1@2012-01-01, 1.5@2012-01-02, 2.5@2012-01-03, 1.5@2012-01-04, 1.5@2012-01-05]');
-- 2
SELECT round(dynamicTimeWarp(tgeompoint '[Point(1 1)@2012-01-01, Point(3 3)@2012-01-03,
  Point(1 1)@2012-01-05]', tgeompoint '[Point(1.1 1.1)@2012-01-01,
  Point(2.5 2.5)@2012-01-02, Point(4 4)@2012-01-03, Point(3 3)@2012-01-04,
  Point(1.5 2)@2012-01-05]')::numeric, 6);
-- 3.380776
```

- btener las parejas de correspondencia entre dos valores temporales con respecto a la distancia de distorsión de tiempo dinámica (Dynamic Time Warp o DTW)

  `dynamicTimeWarpPath({tnumber, tgeo}, {tnumber, tgeo}): pairs`

  `frechetDistancePath({tnumber, tgeo}, {tnumber, tgeo}): pairs`

  Esta función requiere ubicar en memoria una matriz de distancias cuyo tamaño es cuadrático en el número de instantes de los valores temporales. Por tanto, la función fallará para valores temporales con gran número de instantes dependiendo de la memoria disponible.

```
SELECT dynamicTimeWarpPath(tfloat '[1@2012-01-01, 3@2012-01-03, 1@2012-01-06]',
 tfloat '[1@2012-01-01, 1.5@2012-01-02, 2.5@2012-01-03, 1.5@2012-01-04, 1.5@2012-01-05]');
-- (0,0)
   (1,0)
   (2,1)
   (3,2)
   (4,2)
SELECT dynamicTimeWarpPath(tgeompoint '[Point(1 1)@2012-01-01, Point(3 3)@2012-01-03,
  Point(1 1)@2012-01-05]', tgeompoint '[Point(1.1 1.1)@2012-01-01,
  Point(2.5 2.5)@2012-01-02, Point(4 4)@2012-01-03, Point(3 3)@2012-01-04,
  Point(1.5 2)@2012-01-05]');
-- (0,0)
   (1,1)
   (2,1)
   (3,1)
   (4,2)
```

5.14. Mosaicos multidimensionales

Los mosaicos multidimensionales es el mecanismo que se utiliza para dividir el dominio de valores temporales en intervalos o mosaicos de un número variable de dimensiones. En el caso de una sola dimensión, el dominio se puede dividir por valor o por tiempo utilizando intervalos del mismo ancho o la misma duración, respectivamente. Para los números temporales, el dominio se puede dividir en mosaicos bidimensionales del mismo ancho para la dimensión de valor y la misma duración para la dimensión de tiempo. Para los puntos temporales, el dominio se puede dividir en el espacio en mosaicos bidimensionales o tridimensionales, dependiendo del número de dimensiones de las coordenadas espaciales. Finalmente, para los puntos temporales, el dominio se puede dividir por espacio y por tiempo usando mosaicos tridimensionales o tetradimensionales. Además, los valores temporales también se pueden fragmentar de acuerdo con una malla multidimensional definida sobre el dominio subyacente.

Los mosaicos multidimensionales se pueden utilizar para diversos fines. Por ejemplo, se pueden utilizar para calcular histogramas multidimensionales, donde los valores temporales se agregan de acuerdo con la partición subyacente del dominio. Por otro lado, el mosaico multidimensional se puede utilizar para distribuir un conjunto de datos en un grupo de servidores, donde cada servidor contiene una partición del conjunto de datos. La ventaja de este mecanismo de partición es que conserva la proximidad en el espacio y el tiempo, a diferencia de los mecanismos de partición tradicionales basados en hash utilizados en entornos de big data.

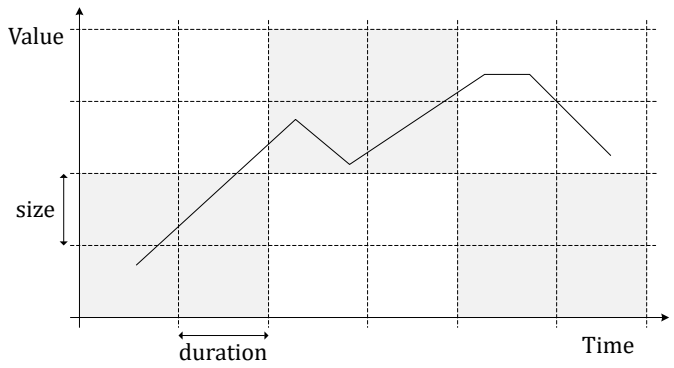

Figura 5.2: Mosaicos multidimensionales para números flotantes temporales.

La Figura 5.2 ilustra un mosaico multidimensional para números flotantes temporales. El dominio bidimensional se divide en mosaicos que tienen el mismo tamaño para la dimensión de valor y la misma duración para la dimensión de tiempo. Suponga que este esquema de mosaicos se usa para distribuir un conjunto de datos en un clúster de seis servidores, como sugiere el patrón gris en la figura. En este caso, los valores se fragmentan para que cada servidor reciba los datos de mosaicos contiguos. Esto implica en particular que cuatro nodos recibirán un fragmento del número flotante temporal que se muestra en la figura. Una ventaja de

esta distribución de datos basada en mosaicos multidimensionales es que reduce los datos que deben intercambiarse entre nodos cuando se procesan consultas, un proceso que generalmente se denomina *reshuffling*.

Muchas de las funciones de esta sección son *funciones de retorno de conjuntos* (también conocidas como *funciones de tabla*) ya que normalmente devuelven más de un valor. En este caso, las funciones están marcadas con el símbolo `{}`.

5.14.1. Operaciones de intervalos

- Obtener un conjunto de parejas (índice, intervalo) que cubre el rango o el período con intervalos de la misma amplitud o duración alineados con el origen `{}`.

  Si el origen no se especifica, su valor se establece por defecto en 0 para los rangos y en lunes 3 de enero de 2000 para los períodos. Los índices empiezan en 1.

  ```
  bucketList(bounds range,width number,origin number=0): setof index_range
  bucketList(bounds period,duration interval,origin timestamptz='2000-01-03'):
   setof index_period
  ```

```
SELECT (bl).index, (bl).range
FROM (SELECT bucketList(tint '[15@2000-01-01, 25@2000-01-10]'::intrange, 2) AS bl) t;
-- 1 | [14,16)
   2 | [16,18)
   3 | [18,20)
   ...
SELECT bucketList(tfloat '[-1@2000-01-01, -10@2000-01-10]'::floatrange, 2, -7);
-- (1,"[-11,-9)")
   (2,"[-9,-7)")
   (3,"[-7,-5)")
   ...
SELECT (bl).index, (bl).period
FROM (SELECT bucketList(tfloat '[1@2000-01-15, 10@2000-01-25]'::period,'2 days') AS bl) t;
-- 1 | [2000-01-15, 2000-01-17)
   2 | [2000-01-17, 2000-01-19)
   3 | [2000-01-19, 2000-01-21)
   ...
SELECT bucketList(tfloat '[1@2000-01-15, 10@2000-01-25]'::period, '2 days', '2000-01-02');
-- (1,"[2000-01-14, 2000-01-16)")
   (2,"[2000-01-16, 2000-01-18)")
   (3,"[2000-01-18, 2000-01-20)")
   ...
```

- Obtener el valor inicial del intervalo que contiene el número de entrada.

  Si el origen no se especifica, su valor se establece por defecto en 0.

  ```
  valueBucket(value number,width number, origin number=0): number
  ```

```
SELECT valueBucket(3, 2);
-- 2
SELECT valueBucket(3.5, 2.5, 1.5);
-- 1.5
```

- Obtener el rango en el espacio de intervalos que contiene el número de entrada.

  Si el origen no se especifica, su valor se establece por defecto en 0.

  ```
  rangeBucket(value number,width number,origin number=0): range
  ```

```
SELECT rangeBucket(2, 2);
-- [2,4)
SELECT rangeBucket(2, 2, 1);
-- [1,3)
SELECT rangeBucket(2, 2.5);
-- [0,2.5)
SELECT rangeBucket(2, 2.5, 1.5);
-- [1.5,4)
```

- Obtener el valor inicial del intervalo que contiene la marca de tiempo de entrada.

  Si el origen no se especifica, su valor se establece por defecto en lunes 3 de enero de 2000.

  ```
  timeBucket(time timestamptz,duration interval,origin timestamptz='2000-01-03'):
   timestamptz
  ```

```
SELECT timeBucket(timestamptz '2020-05-01', interval '2 days');
-- 2020-04-29 01:00:00+02
SELECT timeBucket(timestamptz '2020-05-01', interval '2 days', timestamptz '2020-01-01');
-- 2020-04-30 01:00:00+02
```

- Obtener el período en el espacio de intervalos que contiene la marca de tiempo de entrada.

  Si el origen no se especifica, su valor se establece por defecto en lunes 3 de enero de 2000.

  ```
  periodBucket(time timestamptz,duration interval,origin timestamptz='2000-01-03'):
   period
  ```

```
SELECT periodBucket('2000-01-04', interval '1 week');
-- [2000-01-03, 2000-01-10)
SELECT periodBucket('2000-01-04', interval '1 week', '2000-01-07');
-- [1999-12-31, 2000-01-07)
```

5.14.2. Operaciones de malla

- Obtener un conjunto de parejas (índice, mosaico) que cubre el cuadro delimitador con mosaicos multidimensionales del mismo tamaño y duración. {}

  Si el origen de las dimensiones de valores y/o de tiempo no se especifican, su valor se establece por defecto en 0 o `'Point (0 0 0)'` para la dimensión de valores (según el tipo de cuadro delimitador) y en el lunes 3 de enero de 2000 para la dimensión de tiempo.

  ```
  multidimGrid(bounds tbox,size float,duration interval,vorigin float=0,
   torigin timestamptz='2000-01-03'): setof index_box
  multidimGrid(bounds stbox,size float,sorigin geometry='Point(0 0 0)'):
   setof index_box
  multidimGrid(bounds stbox,size float,duration interval,sorigin geometry='Point(0 0 0)',
   torigin timestamptz='2000-01-03'): setof index_box
  ```

  En el caso de una malla espacio-temporal, el SRID de las coordenadas de los mosaicos está determinado por el del cuadro de entrada y el tamaño se da en las unidades del SRID. Si se especifica el origen de las coordenadas espaciales, que debe ser un punto, su dimensionalidad y SRID deben ser iguales al del cuadro delimitador, de lo contrario se genera un error.

```
SELECT (gr).index, (gr).box
FROM (SELECT multidimGrid(tfloat '[15@2000-01-15, 25@2000-01-25]'::tbox, 2.0, '2 days')
  AS gr) t;
-- 1 | TBOX((14,2000-01-15),(16,2000-01-17))
   2 | TBOX((16,2000-01-15),(18,2000-01-17))
   3 | TBOX((18,2000-01-15),(20,2000-01-17))
   ...
SELECT multidimGrid(tfloat '[15@2000-01-15, 25@2000-01-25]'::tbox, 2.0, '2 days', 11.5);
-- (1,"TBOX((13.5,2000-01-15),(15.5,2000-01-17))")
   (2,"TBOX((15.5,2000-01-15),(17.5,2000-01-17))")
   (3,"TBOX((17.5,2000-01-15),(19.5,2000-01-17))")
   ...
SELECT multidimGrid(tgeompoint '[Point(3 3)@2000-01-15,
  Point(15 15)@2000-01-25]'::stbox, 2.0);
-- (1,"STBOX((2,2),(4,4))")
   (2,"STBOX((4,2),(6,4))")
   (3,"STBOX((6,2),(8,4))")
   ...
SELECT multidimGrid(tgeompoint 'SRID=3812;[Point(3 3)@2000-01-15,
  Point(15 15)@2000-01-25]'::stbox, 2.0, geometry 'Point(3 3)');
-- (1,"SRID=3812;STBOX((3,3),(5,5))")
   (2,"SRID=3812;STBOX((5,3),(7,5))")
   (3,"SRID=3812;STBOX((7,3),(9,5))")
   ...
SELECT multidimGrid(tgeompoint '[Point(3 3 3)@2000-01-15,
  Point(15 15 15)@2000-01-25]'::stbox, 2.0, geometry 'Point(3 3 3)');
-- (1,"STBOX Z((3,3,3),(5,5,5))")
   (2,"STBOX Z((5,3,3),(7,5,5))")
   (3,"STBOX Z((7,3,3),(9,5,5))")
   ...
SELECT multidimGrid(tgeompoint '[Point(3 3)@2000-01-15,
  Point(15 15)@2000-01-25]'::stbox, 2.0, interval '2 days');
-- (1,"STBOX T((2,2,2000-01-15),(4,4,2000-01-17))")
   (2,"STBOX T((4,2,2000-01-15),(6,4,2000-01-17))")
   (3,"STBOX T((6,2,2000-01-15),(8,4,2000-01-17))")
   ...
SELECT multidimGrid(tgeompoint '[Point(3 3 3)@2000-01-15,
  Point(15 15 15)@2000-01-25]'::stbox, 2.0, '2 days', 'Point(3 3 3)', '2000-01-15');
-- (1,"STBOX ZT((3,3,3,2000-01-15),(5,5,5,2000-01-17))")
   (2,"STBOX ZT((5,3,3,2000-01-15),(7,5,5,2000-01-17))")
   (3,"STBOX ZT((7,3,3,2000-01-15),(9,5,5,2000-01-17))")
   ...
```

- Obtener el mosaico de la malla multidimensional que contiene el valor y la marca de tiempo.

  Si el origen de las dimensiones de valores y/o de tiempo no se especifican, su valor se establece por defecto en 0 o `'Point(0 0 0)'` para la dimensión de valores y en el lunes 3 de enero de 2000 para la dimensión de tiempo, respectivamente.

  ```
  multidimTile(value float,time timestamptz,size float,duration interval,
   vorigin float=0.0,torigin timestamptz='2000-01-03'): tbox
  multidimTile(point geometry,size float,sorigin geometry='Point(0 0 0)'): stbox
  multidimTile(point geometry,time timestamptz,size float,duration interval,sorigin
   geometry='Point(0 0 0)',torigin timestamptz='2000-01-03'): stbox
  ```

  En el caso de una malla espacio-temporal, el SRID de las coordenadas de los mosaicos está determinado por el punto de entrada y la amplitud espacial se da en las unidades del SRID. Si se especifica el origen de las coordenadas espaciales, que debe ser un punto, su dimensionalidad y SRID deben ser iguales al del cuadro delimitador, de lo contrario se genera un error.

```
SELECT multidimTile(15, '2000-01-15', 2, interval '2 days');
```

```
-- TBOX((14,2000-01-15),(16,2000-01-17))
SELECT multidimTile(15, '2000-01-15', 2, interval '2 days', 1, '2000-01-02');
-- TBOX((15,2000-01-14),(17,2000-01-16))
SELECT multidimTile(geometry 'Point(1 1 1)', 2.0);
-- STBOX Z((0,0,0),(2,2,2))
SELECT multidimTile(geometry 'Point(1 1)', '2000-01-01', 2.0, interval '2 days');
-- STBOX T((0,0,2000-01-01),(2,2,2000-01-03))
SELECT multidimTile(geometry 'Point(1 1)', '2000-01-01', 2.0, '2 days', 'Point(1 1)',
  '2000-01-02');
-- STBOX T((1,1,1999-12-31),(3,3,2000-01-02))
```

5.14.3. Operaciones de fragmentación

Estas operaciones fragmentan un valor temporal con respecto a una secuencia de intervalos (ver la Sección 5.14.1) o una malla multidimensional (ver la Sección 5.14.2).

- Fragmentar el número temporal con respecto a intervalos de valores. {}

  `valueSplit(value tnumber,width number,origin number=0): setof number_tnumber`

  Si el origen de los valores no se especifica, su valor se establece por defecto en 0.

```
SELECT (sp).number, (sp).tnumber
FROM (SELECT valueSplit(tint '[1@2012-01-01, 2@2012-01-02, 5@2012-01-05, 10@2012-01-10]',
  2) AS sp) t;
--  0 | {[1@2012-01-01 00:00:00+01, 1@2012-01-02 00:00:00+01)}
    2 | {[2@2012-01-02 00:00:00+01, 2@2012-01-05 00:00:00+01)}
    4 | {[5@2012-01-05 00:00:00+01, 5@2012-01-10 00:00:00+01)}
   10 | {[10@2012-01-10 00:00:00+01]}
SELECT valueSplit(tfloat '[1@2012-01-01, 10@2012-01-10)', 2.0, 1.0);
-- (1,"{[1@2012-01-01 00:00:00+01, 3@2012-01-03 00:00:00+01)}")
   (3,"{[3@2012-01-03 00:00:00+01, 5@2012-01-05 00:00:00+01)}")
   (5,"{[5@2012-01-05 00:00:00+01, 7@2012-01-07 00:00:00+01)}")
   (7,"{[7@2012-01-07 00:00:00+01, 9@2012-01-09 00:00:00+01)}")
   (9,"{[9@2012-01-09 00:00:00+01, 10@2012-01-10 00:00:00+01)}")
```

- Fragmentar el valor temporal con respecto a intervalos de tiempo.  {}

  `timeSplit(value ttype,duration interval,origin timestamptz='2000-01-03'):`
  `setof time_temp`

  Si el origen del tiempo no se especifica, su valor se establece por defecto en el lunes 3 de enero de 2000.

```
SELECT (ts).time, (ts).temp
FROM (SELECT timeSplit(tfloat '[1@2012-01-01, 10@2012-01-10)', '2 days') AS ts) t;
-- 2011-12-31 | [1@2012-01-01, 2@2012-01-02)
   2012-01-02 | [2@2012-01-02, 4@2012-01-04)
   2012-01-04 | [4@2012-01-04, 6@2012-01-06)
   ...
SELECT (ts).time, astext((ts).temp) AS temp
FROM (SELECT timeSplit(tgeompoint '[Point(1 1)@2012-01-01, Point(10 10)@2012-01-10]',
  '2 days', '2012-01-01') AS ts) AS t;
-- 2012-01-01 | [POINT Z (1 1 1)@2012-01-01, POINT Z (3 3 3)@2012-01-03)
   2012-01-03 | [POINT Z (3 3 3)@2012-01-03, POINT Z (5 5 5)@2012-01-05)
   2012-01-05 | [POINT Z (5 5 5)@2012-01-05, POINT Z (7 7 7)@2012-01-07)
   ...
```

Observe que se puede fragmentar un valor temporal en intervalos de tiempo cíclicos (en lugar de lineales). Los siguientes dos ejemplos muestran cómo fragmentar un valor temporal por hora y por día de la semana.

```
SELECT (ts).time::time as hour, merge((ts).temp) as temp
FROM (SELECT timeSplit(tfloat '[1@2012-01-01, 10@2012-01-03]', '1 hour') AS ts) t
GROUP BY hour ORDER BY hour;

-- 00:00:00 | {[1@2012-01-01 00:00:00+01, 1.1875@2012-01-01 01:00:00+01),
               [5.5@2012-01-02 00:00:00+01, 5.6875@2012-01-02 01:00:00+01)}
   01:00:00 | {[1.1875@2012-01-01 01:00:00+01, 1.375@2012-01-01 02:00:00+01),
               [5.6875@2012-01-02 01:00:00+01, 5.875@2012-01-02 02:00:00+01)}
   02:00:00 | {[1.375@2012-01-01 02:00:00+01, 1.5625@2012-01-01 03:00:00+01),
               [5.875@2012-01-02 02:00:00+01, 6.0625@2012-01-02 03:00:00+01)}
   03:00:00 | {[1.5625@2012-01-01 03:00:00+01, 1.75@2012-01-01 04:00:00+01),
               [6.0625@2012-01-02 03:00:00+01, 6.25@2012-01-02 04:00:00+01)}
   ...
SELECT EXTRACT(DOW FROM (ts).time) as dow_no, TO_CHAR((ts).time, 'Dy') as dow,
  asText(setPrecision(merge((ts).temp), 2)) as temp
FROM (SELECT timeSplit(tgeompoint '[Point(1 1)@2012-01-01, Point(10 10)@2012-01-14)',
  '1 hour') AS ts) t
GROUP BY dow, dow_no ORDER BY dow_no;
-- 0 | Sun | {[POINT(1 1)@2012-01-01, POINT(1.69 1.69)@2012-01-02),
              [POINT(5.85 5.85)@2012-01-08, POINT(6.54 6.54)@2012-01-09)}
   1 | Mon | {[POINT(1.69 1.69)@2012-01-02, POINT(2.38 2.38)@2012-01-03),
              [POINT(6.54 6.54)@2012-01-09, POINT(7.23 7.23)@2012-01-10)}
   2 | Tue | {[POINT(2.38 2.38)@2012-01-03, POINT(3.08 3.08)@2012-01-04),
              [POINT(7.23 7.23)@2012-01-10, POINT(7.92 7.92)@2012-01-11)}
   ...
```

- Fragmentar el número temporal con respecto a los mosaicos de una malla de valores y de tiempo. {}

  `valueTimeSplit(value tumber,width number,duration interval,vorigin number=0, torigin timestamptz='2000-01-03'): setof number_time_tnumber`

  Si el origen de los valores y/o el tiempo no se especifican, su valor se establece por defecto en 0 y en el lunes 3 de enero de 2000, respectivamente.

```
SELECT (sp).number, (sp).time, (sp).tnumber
FROM (SELECT valueTimeSplit(tint '[1@2012-01-01, 2@2012-01-02, 5@2012-01-05,
  10@2012-01-10]', 5, '5 days') AS sp) t;
-- 0 | 2011-12-31 | [1@2012-01-01, 2@2012-01-02, 2@2012-01-05)
   5 | 2012-01-05 | [5@2012-01-05, 5@2012-01-10)
  10 | 2012-01-10 | [10@2012-01-10]
SELECT (sp).number, (sp).time, (sp).tnumber
FROM (SELECT valueTimeSplit(tfloat '[1@2012-01-01, 10@2012-01-10)', 5.0, '5 days', 1.0,
  '2012-01-01') AS sp) t;
-- 1 | 2012-01-01 | [1@2012-01-01, 6@2012-01-06)
   6 | 2012-01-06 | [6@2012-01-06, 10@2012-01-10)
```

- Fragmentar el punto temporal con respecto a los mosaicos de una malla espacial. {}

  `spaceSplit(value tgeompoint,width float,origin geometry='Point(0 0 0)', bitmatrix=TRUE): setof point_tpoint`

  Si el origen del espacio no se especifica, su valor se establece por defecto en `'Point(0 0 0)'`. Si no se especifica el argumento `bitmatrix`, el cálculo utilizará una matriz de bits para acelerar el proceso.

```
SELECT ST_AsText((sp).point) AS point, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceSplit(tgeompoint '[Point(1 1)@2020-03-01, Point(10 10)@2020-03-10]',
  2.0) AS sp) t;
-- POINT(0 0) | {[POINT(1 1)@2020-03-01, POINT(2 2)@2020-03-02)}
   POINT(2 2) | {[POINT(2 2)@2020-03-02, POINT(4 4)@2020-03-04)}
   POINT(4 4) | {[POINT(4 4)@2020-03-04, POINT(6 6)@2020-03-06)}
   ...
SELECT ST_AsText((sp).point) AS point, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceSplit(tgeompoint '[Point(1 1 1)@2020-03-01,
  Point(10 10 10)@2020-03-10]', 2.0, 'Point(1 1 1)') AS sp) t;
-- POINT Z (1 1 1) | {[POINT Z (1 1 1)@2020-03-01, POINT Z (3 3 3)@2020-03-03)}
   POINT Z (3 3 3) | {[POINT Z (3 3 3)@2020-03-03, POINT Z (5 5 5)@2020-03-05)}
   POINT Z (5 5 5) | {[POINT Z (5 5 5)@2020-03-05, POINT Z (7 7 7)@2020-03-07)}
   ...
```

- Fragmentar el punto temporal con respecto a los mosaicos de una malla espacio-temporal. {}

  ```
  spaceTimeSplit(value tgeompoint,size float,duration interval,sorigin
   geometry='Point(0 0 0)',torigin timestamptz='2000-01-03', bitmatrix=TRUE):
   setof point_time_tpoint
  ```

  Si el origen del espacio y/o el tiempo no se especifica, su valor se establece por defecto en `'Point(0 0 0)'` y en el lunes 3 de enero de 2000, respectivamente. Si no se especifica el argumento `bitmatrix`, el cálculo utilizará una matriz de bits para acelerar el proceso.

```
SELECT ST_AsText((sp).point) AS point, (sp).time, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceTimeSplit(tgeompoint '[Point(1 1)@2020-03-01, Point(10 10)@2020-03-10]',
  2.0, '2 days') AS sp) t;
-- POINT(0 0) | 2020-03-01 | {[POINT(1 1)@2020-03-01, POINT(2 2)@2020-03-02)}
   POINT(2 2) | 2020-03-01 | {[POINT(2 2)@2020-03-02, POINT(3 3)@2020-03-03]}
   POINT(2 2) | 2020-03-03 | {[POINT(3 3)@2020-03-03, POINT(4 4)@2020-03-04)}
   ...
SELECT ST_AsText((sp).point) AS point, (sp).time, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceTimeSplit(tgeompoint '[Point(1 1 1)@2020-03-01,
  Point(10 10 10)@2020-03-10]', 2.0, '2 days', 'Point(1 1 1)', '2020-03-01') AS sp) t;
-- POINT Z(1 1 1) | 2020-03-01 | {[POINT Z(1 1 1)@2020-03-01, POINT Z(3 3 3)@2020-03-03)}
   POINT Z(3 3 3) | 2020-03-03 | {[POINT Z(3 3 3)@2020-03-03, POINT Z(5 5 5)@2020-03-05)}
   POINT Z(5 5 5) | 2020-03-05 | {[POINT Z(5 5 5)@2020-03-05, POINT Z(7 7 7)@2020-03-07)}
   ...
```

5.15. Funciones agregadas

Las funciones agregadas temporales generalizan las funciones agregadas tradicionales. Su semántica es que calculan el valor de la función en cada instante de la *unión* de las extensiones temporales de los valores a agregar. En contraste, recuerde que todas las otras funciones que manipulan tipos temporales calculan el valor de la función en cada instante de la *intersección* de las extensiones temporales de los argumentos.

Las funciones agregadas temporales son las siguientes:

- Para todos los tipos temporales, la función `tcount` generaliza la función traditional `count`. El conteo temporal se puede utilizar para calcular en cada momento el número de objetos disponibles (por ejemplo, el número de coches en un área).
- Para todos los tipos temporales, la función `extent` devuelve un cuadro delimitador que engloba un conjunto de valores temporales. Dependiendo del tipo de base, el resultado de esta función puede ser un `period`, un `tbox` o un `stbox`.
- Para el tipo booleano temporal, las funciones `tand` y `tor` generalizan las funciones traditionales `and` y `or`.

- Para los tipos numéricos temporales hay dos tipos de funciones agregadas temporales. Las funciones `tmin`, `tmax`, `tsum` y `tavg` generalizan las funciones traditionales `min`, `max`, `sum` y `avg`. Además, las funciones `wmin`, `wmax`, `wcount`, `wsum` y `wavg` son versiones de ventana (o acumulativas) de las funciones tradicionales que, dado un intervalo de tiempo w, calculan el valor de la función en un instante t considerando los valores durante el intervalo [t-w, t]. Todas las funciones agregadas de ventana están disponibles para enteros temporales, mientras que para flotantes temporales sólo son significativos el mínimo y el máximo de ventana.
- Para el tipo texto temporal, las funciones `tmin` y `tmax` generalizan las funciones traditionales `min` y `max`.
- Finalmente, para puntos temporales, la función `tcentroid` generaliza la función `ST_Centroid` proporcionada por PostGIS. Por ejemplo, dado un conjunto de objetos que se mueven juntos (es decir, un convoy o una bandada), el centroide temporal producirá un punto temporal que representa en cada instante el centro geométrico (o el centro de masa) de todos los objetos en movimiento.

En los ejemplos que siguen, suponemos que las tablas `Department` y `Trip` contienen las dos tuplas introducidas en la Sección 3.1.

- Conteo temporal

  `tcount(ttype): {tintinstset,tint_seqset}`

```
SELECT tcount(NoEmps) FROM Department;
-- "{[1@2012-01-01, 2@2012-02-01, 1@2012-08-01, 1@2012-10-01)}"
```

- Extensión del cuadro delimitador

  `extent(temp): {period,tbox,stbox}`

```
SELECT extent(noEmps) FROM Department;
-- "TBOX((4,2012-01-01 00:00:00+01),(12,2012-10-01 00:00:00+02))"
SELECT extent(Trip) FROM Trips;
-- "STBOX T((0,0,2012-01-01 08:00:00+01),(3,3,2012-01-01 08:20:00+01))"
```

- Y temporal

  `tand(tbool): tbool`

```
SELECT tand(NoEmps #> 6) FROM Department;
-- "{[t@2012-01-01, f@2012-04-01, f@2012-10-01)}"
```

- O temporal

  `tor(tbool): tbool`

```
SELECT tor(NoEmps #> 6) FROM Department;
-- "{[t@2012-01-01, f@2012-08-01, f@2012-10-01)}"
```

- Mínimo temporal

  `tmin(ttype): {ttype_instset,ttype_seqset}`

```
SELECT tmin(NoEmps) FROM Department;
-- "{[10@2012-01-01, 4@2012-02-01, 6@2012-06-01, 6@2012-10-01)}"
```

- Máximo temporal

  `tmax(ttype): {ttype_instset,ttype_seqset}`

```
SELECT tmax(NoEmps) FROM Department;
-- "{[10@2012-01-01, 12@2012-04-01, 6@2012-08-01, 6@2012-10-01)}"
```

- Suma temporal

  `tsum(tnumber): {tnumberinstset,tnumberseqset}`

```
SELECT tsum(NoEmps) FROM Department;
-- "{[10@2012-01-01, 14@2012-02-01, 16@2012-04-01, 18@2012-06-01, 6@2012-08-01,
  6@2012-10-01)}"
```

- Promedio temporal

  `tavg(tnumber): {tfloatinstset,tfloat_seqset}`

```
SELECT tavg(NoEmps) FROM Department;
-- "{[10@2012-01-01, 10@2012-02-01), [7@2012-02-01, 7@2012-04-01),
  [8@2012-04-01, 8@2012-06-01), [9@2012-06-01, 9@2012-08-01),
  [6@2012-08-01, 6@2012-10-01)"
```

- Mínimo de ventana

  `wmin(tnumber,interval): {tnumberinstset,tnumberseqset}`

```
SELECT wmin(NoEmps, interval '2 days') FROM Department;
-- "{[10@2012-01-01, 4@2012-04-01, 6@2012-06-03, 6@2012-10-03)}"
```

- Máximo de ventana

  `wmax(tnumber,interval): {tnumberinstset,tnumberseqset}`

```
SELECT wmax(NoEmps, interval '2 days') FROM Department;
-- "{[10@2012-01-01, 12@2012-04-01, 6@2012-08-03, 6@2012-10-03)}"
```

- Conteo de ventana

  `wcount(tnumber,interval): {tintinstset,tint_seqset}`

```
SELECT wcount(NoEmps, interval '2 days') FROM Department;
-- "{[1@2012-01-01, 2@2012-02-01, 3@2012-04-01, 2@2012-04-03, 3@2012-06-01, 2@2012-06-03,
  1@2012-08-03, 1@2012-10-03)}"
```

- Suma de ventana

  `wsum(tint,interval): {tintinstset,tint_seqset}`

```
SELECT wsum(NoEmps, interval '2 days') FROM Department;
-- "{[10@2012-01-01, 14@2012-02-01, 26@2012-04-01, 16@2012-04-03, 22@2012-06-01,
  18@2012-06-03, 6@2012-08-03, 6@2012-10-03)}"
```

- Promedio de ventana

  `wavg(tint,interval): {tfloatinstset,tfloat_seqset}`

```
SELECT wavg(NoEmps, interval '2 days') FROM Department;
-- "{[10@2012-01-01, 10@2012-02-01), [7@2012-02-01, 7@2012-04-01),
  [8.6666666666667@2012-04-01, 8.6666666666667@2012-04-03),
  [8@2012-04-03, 8@2012-06-01),
  [7.3333333333333@2012-06-01, 7.3333333333333@2012-06-03),
  [9@2012-06-03, 9@2012-08-03), [6@2012-08-03, 6@2012-10-03)}"
```

- Centroide temporal

  `tcentroid(tgeompoint): tgeompoint`

```
SELECT tcentroid(Trip) FROM Trips;
-- "{[POINT(0 0)@2012-01-01 08:00:00+00, POINT(1 0)@2012-01-01 08:05:00+00),
  [POINT(0.5 0)@2012-01-01 08:05:00+00, POINT(1.5 0.5)@2012-01-01 08:10:00+00,
  POINT(2 1.5)@2012-01-01 08:15:00+00),
  [POINT(2 2)@2012-01-01 08:15:00+00, POINT(3 3)@2012-01-01 08:20:00+00)}"
```

5.16. Funciones de utilidad

- Versión de la extensión MobilityDB

  `mobilitydb_version(): text`

```
SELECT mobilitydb_version();
-- "MobilityDB 1.0"
```

- Versión de la extensión MobilityDB y de sus dependencias

  `mobilitydb_full_version(): text`

```
SELECT mobilitydb_full_version();
-- "MobilityDB 1.0 PostgreSQL 12.3 PostGIS 2.5"
```

5.17. Indexación de tipos temporales

Se pueden crear índices GiST y SP-GiST para columnas de tabla de tipos temporales. El índice GiST implementa un árbol R para tipos alfanuméricos temporales y para tipos de puntos temporales. El índice SP-GiST implementa un árbol cuádruple para tipos alfanuméricos temporales y un árbol óctuple para tipos de puntos temporales. Ejemplos de creación de índices son los siguientes:

```
CREATE INDEX Department_NoEmps_Gist_Idx ON Department USING Gist(NoEmps);
CREATE INDEX Trips_Trip_SPGist_Idx ON Trips USING SPGist(Trip);
```

Los índices GiST y SP-GiST almacenan el cuadro delimitador para los tipos temporales. Como se explica en el Capítulo 3, estos son

- el tipo `period` para los tipos `tbool` y `ttext`,

- el tipo `tbox` par los tipos `tint` y `tfloat`,
- el tipo `stbox` para los tipos `tgeompoint` y `tgeogpoint`.

Un índice GiST o SP-GiST puede acelerar las consultas que involucran a los siguientes operadores (consulte la Sección 5.7 para obtener más información):

- `<<`, `&<`, `&>`, `>>`, que sólo consideran la dimensión de valores en tipos alfanuméricos temporales,
- `<<`, `&<`, `&>`, `>>`, `<<|`, `&<|`, `|&>`, `|>>`, `&</`, `<</`, `/>>` y `/&>`, que sólo consideran la dimensión espacial en tipos de puntos temporales,
- `&<#`, `<<#`, `#>>`, `#&>`, que sólo consideran la dimensión temporal para todos los tipos temporales,
- `&&`, `@>`, `<@` y `~=`, que consideran tantas dimensiones como compartan la columna indexada y el argumento de consulta. Estos operadores trabajan en cuadros delimitadores (es decir, `period`, `tbox` o `stbox`), no los valores completos.

Además, un índice GiST puede acelerar las consultas de vecinos más cercanos que involucran el operador `|=|`.

Por ejemplo, dado el índice definido anteriormente en la tabla `Department` y una consulta que implica una condición con el operador `&&` (superposición), si el argumento derecho es un flotante temporal, entonces se consideran tanto el valor como las dimensiones de tiempo para filtrar las tuplas de la relación, mientras que si el argumento derecho es un valor flotante, un rango flotante o un tipo de tiempo, entonces el valor o la dimensión de tiempo se utilizará para filtrar las tuplas de la relación. Además, se puede construir un cuadro delimitador a partir de un valor/rango y/o una marca de tiempo/período, que se puede usar para filtrar las tuplas de la relación. Ejemplos de consultas que utilizan el índice en la tabla `Department` definida anteriormente se dan a continuación.

```
SELECT * FROM Department WHERE NoEmps && 5;
SELECT * FROM Department WHERE NoEmps && intrange '[1, 5)';
SELECT * FROM Department WHERE NoEmps && timestamptz '2012-04-01';
SELECT * FROM Department WHERE NoEmps && period '[2012-04-01, 2012-05-01)';
SELECT * FROM Department WHERE NoEmps &&
tbox(intrange '[1, 5)', period '[2012-04-01, 2012-05-01)');
SELECT * FROM Department WHERE NoEmps &&
tfloat '{[1@2012-01-01, 1@2012-02-01), [5@2012-04-01, 5@2012-05-01)}';
```

Del mismo modo, los ejemplos de consultas que utilizan el índice en la tabla `Trips` definida anteriormente se dan a continuación.

```
SELECT * FROM Trips WHERE Trip && geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))';
SELECT * FROM Trips WHERE Trip && timestamptz '2001-01-01';
SELECT * FROM Trips WHERE Trip && period '[2001-01-01, 2001-01-05)';
SELECT * FROM Trips WHERE Trip &&
stbox(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))', period '[2001-01-01, 2001-01-05]');
SELECT * FROM Trips WHERE Trip &&
tgeompoint '{[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02, Point(1 1)@2001-01-05)}';
```

Finalmente, se pueden crear índices de árbol B para columnas de tabla de todos los tipos temporales. Para este tipo de índice, la única operación útil es la igualdad. Hay un orden de clasificación de árbol B definido para valores de tipos temporales, con los correspondientes operadores <, <=, > y >=, pero el orden es bastante arbitrario y no suele ser útil en el mundo real. El soporte de árbol B para tipos temporales está destinado principalmente a permitir la clasificación interna en las consultas, en lugar de la creación de índices reales.

Para acelerar varias de las funciones en el Capítulo 5, se puede agregar en la cláusula `WHERE` de las consultas una comparación de cuadro delimitador que hace uso de los índices disponibles. Por ejemplo, este sería típicamente el caso de las funciones que proyectan los tipos temporales a las dimensiones de valor/espacio y/o tiempo. Esto filtrará las tuplas con un índice como se muestra en la siguiente consulta.

```
SELECT atPeriod(T.Trip, period(2001-01-01, 2001-01-02))
FROM Trips T
-- Filtro de índice con cuadro delimitador
WHERE T.Trip && period(2001-01-01, 2001-01-02)
```

En el caso de los puntos temporales, todas las relaciones espaciales con la semántica posible (ver la Sección 5.12.6) incluyen automáticamente una comparación de cuadro delimitador que hará uso de cualquier índice que esté disponible en los puntos temporales. Por esta razón, la primera versión de las relaciones se usa típicamente para filtrar las tuplas con la ayuda de un índice al calcular las relaciones temporales como se muestra en la siguiente consulta.

```
SELECT tintersects(T.Trip, R.Geom)
FROM Trips T, Regions R
-- Filtro de índice con cuadro delimitador
WHERE intersects(T.Trip, R.Geom);
```

5.18. Estadísticas y selectividad para tipos temporales

5.18.1. Colecta de estadísticas

El planificador de PostgreSQL se basa en información estadística sobre el contenido de las tablas para generar el plan de ejecución más eficiente para las consultas. Estas estadísticas incluyen una lista de algunos de los valores más comunes en cada columna y un histograma que muestra la distribución aproximada de datos en cada columna. Para tablas grandes, se toma una muestra aleatoria del contenido de la tabla, en lugar de examinar cada fila. Esto permite analizar tablas grandes en poco tiempo. La información estadística es recopilada por el comando `ANALYZE` y es almacenada en la tabla de catálogo `pg_statistic`. Dado que diferentes tipos de estadísticas pueden ser apropiados para diferentes tipos de datos, la tabla sólo almacena estadísticas muy generales (como el número de valores nulos) en columnas dedicadas. Todo lo demás se almacena en cinco “slots”, que son pares de columnas de matriz que almacenan las estadísticas de una columna de un tipo arbitrario.

Las estadísticas recopiladas para tipos de tiempo y tipos temporales se basan en las recopiladas por PostgreSQL para tipos escalares y tipos de rango. Para tipos escalares, como `float`, se recopilan las siguientes estadísticas:

1. fracción de valores nulos,
2. ancho promedio, en bytes, de valores no nulos,
3. número de diferentes valores no nulos,
4. matriz de los valores más comunes y matriz de sus frecuencias,
5. histograma de valores, donde se excluyen los valores más comunes,
6. correlación entre el orden de filas físico y lógico.

Para los tipos de rango, como `tstzrange`, se recopilan tres histogramas adicionales:

7. histograma de longitud de rangos no vacíos,
8. histogramas de límites superior e inferior.

Para geometrías, además de (1)–(3), se recopilan las siguientes estadísticas:

9. número de dimensiones de los valores, cuadro delimitador N-dimensional, número de filas en la tabla, número de filas en la muestra, número de valores no nulos,
10. Histograma N-dimensional que divide el cuadro delimitador en varias celdas y mantiene la proporción de valores que se cruzan con cada celda.

Las estadísticas recopiladas para columnas de los nuevos tipos de tiempo `timestampset`, `period` y `periodset` replican las recopiladas por PostgreSQL para `tstzrange`. Esto es claro para el tipo `period`, que es equivalente a `tszrange`, excepto que los períodos no pueden estar vacíos. Para los tipos `timestampset` y `periodset`, un valor se convierte en su cuadro delimitador que es un `period` y luego se recopilan las estadísticas del tipo `period`.

Las estadísticas recopiladas para columnas de los tipos temporales dependen del subtipo temporal y del tipo base. Además de las estadísticas (1)–(3) que se recopilan para todos los tipos temporales, las estadísticas se recopilan para la dimensiónes de tiempo y de valor de forma independiente. Más precisamente, se recopilan las siguientes estadísticas para la dimensión de tiempo:

- Para columnas de subtipo instante, las estadísticas (4)–(6) se recopilan para las marcas de tiempo.
- Para columnas de otro subtipo, las estadísticas (7)–(8) se recopilan para los períodos delimitadores.

Las siguientes estadísticas se recopilan para la dimensión de valores:

- Para columnas de tipos temporales con interpolación escalonada (es decir, `tbool`, `ttext` o `tint`):
  - Para el subtipo instante, las estadísticas (4)–(6) se recopilan para los valores.
  - Para todas los demás subtipos, las estadísticas (7)–(8) se recopilan para los valores.
- Para columnas de tipos temporales flotantes (es decir, `tfloat`):
  - Para el subtipo instante, las estadísticas (4)–(6) se recopilan para los valores.
  - Para todas los demás subtipos, las estadísticas (7)–(8) se recopilan por los rangos delimitadores de valores.
- Para columnas de tipos de puntos temporales (es decir, `tgeompoint` y `tgeogpoint`) las estadísticas (9)–(10) se compilan para los puntos.

5.18.2. Estimación de la selectividad de los operadores

Los operadores booleanos en PostgreSQL se pueden asociar con dos funciones de selectividad, que calculan la probabilidad de que un valor de un tipo dado coincida con un criterio dado. Estas funciones de selectividad se basan en las estadísticas recopiladas. Hay dos tipos de funciones de selectividad. Las funciones de selectividad de *restricción* intentan estimar el porcentaje de filas en una tabla que satisfacen una condición en la cláusula `WHERE` de la forma `column OP constant`. Por otro lado, las funciones de selectividad de *unión* intentan estimar el porcentaje de filas en una tabla que satisfacen una condición en la cláusula `WHERE` de la forma `table1.column1 OP table2.column2`.

MobilityDB define 23 clases de operadores booleanos (como=, `<`, `&&`, `<<`, etc.), cada uno de los cuales puede tener como argumentos izquierdo o derecho un tipo PostgreSQL (como `integer`, `timestamptz`, etc.) o un nuevo tipo MobilityDB (como `period`, `tint_seq`, etc.). Como consecuencia, existe un número muy elevado de operadores con diferentes argumentos a considerar para las funciones de selectividad. El enfoque adoptado fue agrupar estas combinaciones en clases correspondientes a las dimensiones de valor o de tiempo. Las clases corresponden al tipo de estadísticas recopiladas como se explica en la sección anterior.

Actualmente, sólo están implementadas las funciones de selectividad de restricción para tipos temporales, mientras que las funciones de selectividad de unión dan un valor de selectividad predeterminado según el operador. Está previsto implementar las funciones de selectividad de unión en el futuro.

Capítulo 6

Puntos de red temporales

Los puntos temporales que hemos considerado hasta ahora representan el movimiento de objetos que pueden moverse libremente en el espacio ya que se supone que pueden cambiar su posición de un lugar a otro sin ninguna restricción de movimiento. Este es el caso de los animales y de los objetos voladores como aviones o drones. Sin embargo, en muchos casos, los objetos no se mueven libremente en el espacio sino dentro de redes integradas espacialmente, como rutas o ferrocarriles. En este caso, es necesario tener en cuenta de las redes integradas al describir los movimientos de estos objetos en movimiento. Los puntos de red temporales tienen en cuenta estos requisitos.

En comparación con los puntos temporales de espacio libre, los puntos basados en red tienen las siguientes ventajas:

- Los puntos de red temporales proporcionan restricciones que reflejan los movimientos reales de los objetos en movimiento.
- La información geométrica no se almacena con el punto móvil, sino de una vez por todas en las redes fijas. De esta forma, las representaciones e interpolaciones de la ubicación son más precisas.
- Los puntos de red temporales son más eficientes en términos de almacenamiento de datos, actualización de ubicación, formulación de consultas e indexación. Estos aspectos se tratan más adelante en este documento.

Los puntos de red temporales se basan en pgRouting, una extensión de PostgreSQL para desarrollar aplicaciones de enrutamiento de red y realizar análisis de gráficos. Por lo tanto, los puntos de red temporal asumen que la red subyacente está definida en una tabla llamada `ways`, que tiene al menos tres columnas: `gid` que contiene el identificador de ruta único, `length` que contiene la longitud de la ruta y `the_geom` que contiene la geometría de la ruta.

Hay dos tipos de red estáticos, `npoint` (abreviatura de *network point*) y `nsegment` (abreviatura de *network segment*), que representan, respectivamente, un punto y un segmento de una ruta. Un valor `npoint` se compone de un identificador de ruta y un número flotante en el rango [0,1] que determina una posición relativa de la ruta, donde 0 corresponde al comienzo de la ruta y 1 al final de la ruta. Un valor de `nsegment` se compone de un identificador de ruta y dos números flotantes en el rango [0,1] que determinan las posiciones relativas de inicio y finalización. Un valor de `nsegment` cuyas posiciones inicial y final son iguales corresponde a un valor de `npoint`.

El tipo `npoint` sirve como tipo base para definir el tipo punto de red temporal `tnpoint`. El tipo `tnpoint` tiene una funcionalidad similar al tipo de punto temporal `tgeompoint` con la excepción de que solo considera dos dimensiones. Por lo tanto, todas las funciones y operadores descritos anteriormente para el tipo `tgeompoint` también son aplicables para el tipo `tnpoint`. Además, hay funciones específicas definidas para el tipo `tnpoint`.

6.1. Tipos de red estáticos

Un valor `npoint` es un par de la forma `(rid, position)` donde `rid` es un valor `bigint` que representa un identificador de ruta y `position` es un valor `float` en el rango [0,1] que indica su posición relativa. Los valores 0 y 1 de `position` denotan, respectivamente, la posición inicial y final de la ruta. La distancia de la ruta entre un valor de `npoint` y la posición inicial de la ruta con el identificador `rid` se calcula multiplicando `position` por `length`, donde este último es la longitud de la ruta. Ejemplos de entrada de valores de puntos de red son los siguientes:

```
SELECT npoint 'Npoint(76, 0.3)';
SELECT npoint 'Npoint(64, 1.0)';
```

La función de constructor para puntos de red tiene un argumento para el identificador de ruta y un argumento para la posición relativa. Un ejemplo de un valor de punto de red definido con la función constructora es el siguiente:

```
SELECT npoint(76, 0.3);
```

Un valor `nsegment` es un triple de la forma `(rid, startPosition, endPosition)` donde `rid` es un valor `bigint` que representa un identificador de ruta y `startPosition` y `endPosition` son valores de `float` en el rango [0,1] tal que `startPosition` $\leq$ `endPosition`. Semánticamente, un segmento de red representa un conjunto de puntos de red `(rid, position)` con `startPosition` $\leq$ `position` $\leq$ `endPosition`. Si `startPosition = 0` y `endPosition = 1`, el segmento de red es equivalente a la ruta completa. Si `startPosition = endPosition`, el segmento de red representa un único punto de red. Ejemplos de entrada de valores de puntos de red son los siguientes:

```
SELECT nsegment 'Nsegment(76, 0.3, 0.5)';
SELECT nsegment 'Nsegment(64, 0.5, 0.5)';
SELECT nsegment 'Nsegment(64, 0.0, 1.0)';
SELECT nsegment 'Nsegment(64, 1.0, 0.0)';
-- convertido a nsegment 'Nsegment(64, 0.0, 1.0)';
```

Como se puede ver en el último ejemplo, los valores `startPosition` y `endPosition` se invertirán para asegurar que la condición `startPosition` $\leq$ `endPosition` siempre se satisface. La función de constructor para segmentos de red tiene un argumento para el identificador de ruta y dos argumentos opcionales para las posiciones inicial y final. Los ejemplos de valores de segmento de red definidos con la función constructora son los siguientes:

```
SELECT nsegment(76, 0.3, 0.3);
SELECT nsegment(76); -- se asume que las posiciones inicial y final son 0 y 1  ↩
    respectivamente
SELECT nsegment(76, 0.5); -- se asume que la posición final es 1
```

Los valores del tipo `npoint` se pueden convertir al tipo `nsegment` usando un `CAST` explícito o usando la notación `::` como se muestra a continuación.

```
SELECT npoint(76, 0.33)::nsegment;
```

Los valores de los tipos de red estáticos deben satisfacer varias restricciones para que estén bien definidos. Estas restricciones se dan a continuación.

- El identificador de ruta `rid` debe encontrarse en la columna `gid` de la tabla `ways`.
- Los valores de `position`, `startPosition` y `endPosition` deben estar en el rango [0,1]. Se genera un error cuando no se cumple una de estas restricciones.

Ejemplos de valores de tipo de red estática incorrectos son los siguientes.

```
-- incorrect rid value
SELECT npoint 'Npoint(87.5, 1.0)';
-- incorrect position value
SELECT npoint 'Npoint(87, 2.0)';
-- rid value not found in the ways table
SELECT npoint 'Npoint(99999999, 1.0)';
```

Damos a continuación las funciones y operadores para los tipos de redes estáticas.

6.1.1. Funciones de constructor

- Constructor de puntos de red

  `npoint(bigint,double precision): npoint`

```
SELECT npoint(76, 0.3);
```

- Constructor de segmentos de red

  `nsegment(bigint,double precision,double precision): nsegment`

```
SELECT nsegment(76, 0.3, 0.5);
```

6.1.2. Funciones de transformación

- Establecer la precisión de la(s) posición(es) del punto de red or el segmento de red en el número de posiciones decimales

  `setPrecision({npoint,nsegment},int): {npoint,nsegment}`

```
SELECT setPrecision(npoint(76, 0.123456789), 6);
--  NPoint(76,0.123457)
SELECT setPrecision(nsegment(76, 0.123456789, 0.223456789), 6);
--  NSegment(76,0.123457,0.223457)
```

6.1.3. Funciones de accesor

- Obtener el identificador de ruta

  `route({npoint,nsegment}): bigint`

```
SELECT route(npoint 'Npoint(63, 0.3)');
-- 63
SELECT route(nsegment 'Nsegment(76, 0.3, 0.3)');
-- 76
```

- Obtener la posición

  `getPosition(npoint): float`

```
SELECT getPosition(npoint 'Npoint(63, 0.3)');
-- 0.3
```

- Obtener la posición inicial

  `startPosition(npoint): float`

```
SELECT startPosition(nsegment 'Nsegment(76, 0.3, 0.5)');
-- 0.3
```

- Obtener la posición final

  `endPosition(npoint): float`

```
SELECT endPosition(nsegment 'Nsegment(76, 0.3, 0.5)');
-- 0.5
```

6.1.4. Funciones espaciales

- Obtener el identificador de referencia espacial

  `SRID({npoint,nsegment}): int`

```
SELECT SRID(nspoint 'Npoint(76, 0.3)');
-- 5676
SELECT SRID(nsegment 'Nsegment(76, 0.3, 0.5)');
-- 5676
```

Los valores de los tipos `npoint` y `nsegment` se pueden convertir al tipo `geometry` usando un `CAST` explícito o usando la notación `::` como se muestra a continuación.

- Convertir un punto de red en una geometría

  `{npoint,nsegment}::geometry`

```
SELECT ST_AsText(npoint(76, 0.33)::geometry);
-- POINT(21.6338731332283 50.0545869554067)
SELECT ST_AsText(nsegment(76, 0.33, 0.66)::geometry);
-- LINESTRING(21.6338731332283 50.0545869554067,30.7475989651999 53.9185062927473)
SELECT ST_AsText(nsegment(76, 0.33, 0.33)::geometry);
-- POINT(21.6338731332283 50.0545869554067)
```

De manera similar, los valores de tipo `geometry` de subtipo `point` o `linestring` (restringidos a dos puntos) se pueden convertir, respectivamente, en valores `npoint` y `nsegment` usando un `CAST` explícito o usando la notación `::`. Para ello se debe encontrar la ruta que interseca los puntos dados, donde se asume una tolerancia de 0.00001 unidades (dependiendo del sistema de coordenadas) por lo que se considera que un punto y una ruta que están cerca se cruzan. Si no se encuentra dicha ruta, se devuelve un valor nulo.

- Convertir una geometría en un punto de red

  `geometry::{npoint,nsegment}`

```
SELECT geometry 'Point(279.269156511873 811.497076880187)'::npoint;
-- NPoint(3,0.781413)
SELECT geometry 'LINESTRING(406.729536784738 702.58583437902,
  383.570801314823 845.137059419277)'::nsegment;
-- NSegment(3,0.6,0.9)
SELECT geometry 'Point(279.3 811.5)'::npoint;
-- NULL
SELECT geometry 'LINESTRING(406.7 702.6,383.6 845.1)'::nsegment;
-- NULL
```

Dos valores `npoint` pueden tener diferentes identificadores de ruta pero pueden representar el mismo punto espacial en la intersección de las dos rutas. La función `equals` se utiliza para verificar la igualdad espacial de los puntos de la red.

- Igualdad espacial para puntos de red

```
equals(npoint, npoint)::Boolean
```

```
WITH inter(geom) AS (
  SELECT st_intersection(t1.the_geom, t2.the_geom)
  FROM ways t1, ways t2 WHERE t1.gid = 1 AND t2.gid = 2),
fractions(f1, f2) AS (
  SELECT ST_LineLocatePoint(t1.the_geom, i.geom), ST_LineLocatePoint(t2.the_geom, i.geom)
  FROM ways t1, ways t2, inter i WHERE t1.gid = 1 AND t2.gid = 2)
SELECT equals(npoint(1, f1), npoint(2, f2)) FROM fractions;
-- true
```

6.1.5. Operadores de comparación

Los operadores de comparación (=, < y así sucesivamente) para tipos de red estáticos requieren que los argumentos izquierdo y derecho sean del mismo tipo. Excepto la igualdad y la desigualdad, los otros operadores de comparación no son útiles en el mundo real pero permiten que los índices de árbol B se construyan en tipos de red estáticos.

- ¿Son iguales los valores?

```
{npoint,nsegment} = {npoint,nsegment}
```

```
SELECT npoint 'Npoint(3, 0.5)' = npoint 'Npoint(3, 0.5)';
-- true
SELECT nsegment 'Nsegment(3, 0.5, 0.5)' = nsegment 'Nsegment(3, 0.5, 0.6)';
-- false
```

- ¿Son diferentes los valores?

```
{npoint,nsegment} <> {npoint,nsegment}
```

```
SELECT npoint 'Npoint(3, 0.5)' <> npoint 'Npoint(3, 0.6);
-- true
SELECT nsegment 'Nsegment(3, 0.5, 0.5)' <> nsegment 'Nsegment(3, 0.5, 0.5)';
-- false
```

- ¿El primer valor es menor que el segundo?

```
{npoint,nsegment} < {npoint,nsegment}
```

```
SELECT nsegment 'Nsegment(3, 0.5, 0.5)' < nsegment 'Nsegment(3, 0.5, 0.6)';
-- true
```

- ¿El primer valor es mayor que el segundo?

```
{npoint,nsegment} > {npoint,nsegment}
```

```
SELECT nsegment 'Nsegment(3, 0.5, 0.5)' > nsegment 'Nsegment(2, 0.5, 0.5)';
-- true
```

- ¿El primer valor es menor o igual que el segundo?

```
{npoint,nsegment} <= {npoint,nsegment}
```

```
SELECT npoint 'Npoint(1, 0.5)' <= npoint 'Npoint(2, 0.5)';
-- true
```

- ¿El primer valor es mayor o igual que el segundo?
  `{npoint,nsegment} >= {npoint,nsegment}`

```
SELECT npoint 'Npoint(1, 0.6)' >= npoint 'Npoint(1, 0.5)';
-- true
```

6.2. Puntos de red temporales

El tipo de punto de red temporal `tnpoint` permite representar el movimiento de objetos en una red. Corresponde al tipo de punto temporal `tgeompoint` restringido a coordenadas bidimensionales. Como todos los demás tipos temporales, se presenta en cuatro subtipos, a saber, instante, conjunto de instantes, secuencia y conjunto de secuencias. A continuación se dan ejemplos de valores de `tnpoint` en estos subtipos.

```
SELECT tnpoint 'Npoint(1, 0.5)@2000-01-01';
SELECT tnpoint '{Npoint(1, 0.3)@2000-01-01, Npoint(1, 0.5)@2000-01-02,
  Npoint(1, 0.5)@2000-01-03}';
SELECT tnpoint '[Npoint(1, 0.2)@2000-01-01, Npoint(1, 0.4)@2000-01-02,
  Npoint(1, 0.5)@2000-01-03]';
SELECT tnpoint '{[Npoint(1, 0.2)@2000-01-01, Npoint(1, 0.4)@2000-01-02,
  Npoint(1, 0.5)@2000-01-03], [Npoint(2, 0.6)@2000-01-04, Npoint(2, 0.6)@2000-01-05]}';
```

El tipo de punto de red temporal acepta modificadores de tipo (o `typmod` en la terminología de PostgreSQL). Los valores posibles para el modificador de tipo son `Instant`, `InstantSet`, `Sequence` y `SequenceSet`. Si no se especifica ningún modificador de tipo para una columna, se permiten valores de cualquier subtipo.

```
SELECT tnpoint(Sequence) '[Npoint(1, 0.2)@2000-01-01, Npoint(1, 0.4)@2000-01-02,
  Npoint(1, 0.5)@2000-01-03]';
SELECT tnpoint(Sequence) 'Npoint(1, 0.2)@2000-01-01';
-- ERROR: Temporal type (Instant) does not match column type (Sequence)
```

Los valores de puntos de red temporales del subtipo de secuencia deben definirse en una única ruta. Por lo tanto, se necesita un valor de subtipo de conjunto de secuencias para representar el movimiento de un objeto que atraviesa varias rutas, incluso si no hay un espacio temporal. Por ejemplo, en el siguiente valor

```
SELECT tnpoint '{[NPoint(1, 0.2)@2001-01-01, NPoint(1, 0.5)@2001-01-03),
  [NPoint(2, 0.4)@2001-01-03, NPoint(2, 0.6)@2001-01-04)}';
```

el punto de red cambia su ruta en 2001-01-03.

Los valores de puntos de red temporal del subtipo de secuencia o conjunto de secuencias se convierten en una forma normal para que los valores equivalentes tengan representaciones idénticas. Para ello, los valores instantáneos consecutivos se fusionan cuando es posible. Tres valores instantáneos consecutivos se pueden fusionar en dos si las funciones lineales que definen la evolución de los valores son las mismas. Los ejemplos de transformación a una forma normal son los siguientes.

```
SELECT tnpoint '[NPoint(1, 0.2)@2001-01-01, NPoint(1, 0.4)@2001-01-02,
  NPoint(1, 0.6)@2001-01-03)';
-- [NPoint(1,0.2)@2001-01-01, NPoint(1,0.6)@2001-01-03)
SELECT tnpoint '{[NPoint(1, 0.2)@2001-01-01, NPoint(1, 0.3)@2001-01-02,
  NPoint(1, 0.5)@2001-01-03), [NPoint(1, 0.5)@2001-01-03, NPoint(1, 0.7)@2001-01-04)}';
-- {[NPoint(1,0.2)@2001-01-01, NPoint(1,0.3)@2001-01-02, NPoint(1,0.7)@2001-01-04)}
```

6.3. Validez de los puntos de red temporal

Los valores de los puntos de red temporal deben satisfacer las restricciones especificadas en la Sección 3.2 para que estén bien definidos. Se genera un error cuando no se cumple una de estas restricciones. Ejemplos de valores incorrectos son los siguientes.

```
-- no se permiten valores nulos
SELECT tnpoint 'NULL@2001-01-01 08:05:00';
SELECT tnpoint 'Point(0 0)@NULL';
-- el tipo base no es un punto de red
SELECT tnpoint 'Point(0 0)@2001-01-01 08:05:00';
-- múltiples rutas en una secuencia
SELECT tnpoint '[Npoint(1, 0.2)@2001-01-01 09:00:00, Npoint(2, 0.2)@2001-01-01 09:05:00)';
```

6.4. Constructores para puntos de red temporales

- Constructor para puntos de red temporal de subtipo instante

  `tnpoint_inst(val npoint,t timestamptz):tnpoint_inst`

```
SELECT tnpoint_inst('Npoint(1, 0.5)', '2000-01-01');
-- NPoint(1,0.5)@2000-01-01
```

- Constructor para puntos de red temporal de subtipo conjunto de instantes

  `tnpoint_instset(tnpoint[]):tnpoint_instset`

  `tnpoint_instset(npoint,timestampset):tnpoint_instset`

```
SELECT tnpoint_instset(ARRAY[tnpoint 'Npoint(1, 0.3)@2000-01-01',
  'Npoint(1, 0.5)@2000-01-02', 'Npoint(1, 0.5)@2000-01-03']);
-- {NPoint(1,0.3)@2000-01-01, NPoint(1,0.5)@2000-01-02, NPoint(1,0.5)@2000-01-03}
SELECT tnpoint_instset('Npoint(1, 0.3)', '{2000-01-01, 2000-01-03, 2000-01-05}');
-- {NPoint(1,0.3)@2000-01-01, NPoint(1,0.3)@2000-01-03, NPoint(1,0.3)@2000-01-05}
```

- Constructor para puntos de red temporal de subtipo secuencia

  `tnpoint_seq(tnpoint[],lower_inc boolean=true,upper_inc boolean=true,`
  `linear boolean=true):tnpoint_seq`

  `tnpoint_seq(npoint,period,linear boolean=true):tnpoint_seq`

```
SELECT tnpoint_seq(ARRAY[tnpoint 'Npoint(1, 0.2)@2000-01-01', 'Npoint(1, 0.4)@2000-01-02',
  'Npoint(1, 0.5)@2000-01-03']);
-- [NPoint(1,0.2)@2000-01-01, NPoint(1,0.4)@2000-01-02, NPoint(1,0.5)@2000-01-03]
SELECT tnpoint_seq(npoint 'Npoint(1, 0.2)', '[2000-01-01, 2000-01-03]', false);
-- Interp=Stepwise;[NPoint(1,0.2)@2000-01-01, NPoint(1,0.2)@2000-01-03]
```

- Constructor para puntos de red temporal de subtipo conjunto de secuencias

  `tnpoint_seqset(tnpoint[]):tnpoint_seqset`

  `tnpoint_seqset(npoint,periodset,boolean=true):tnpoint_seqset`

```
SELECT tnpoint_seqset(ARRAY[tnpoint '[Npoint(1, 0.2)@2000-01-01, Npoint(1, 0.4)@2000 ↩
    -01-02,
  Npoint(1, 0.5)@2000-01-03]', '[Npoint(2, 0.6)@2000-01-04, Npoint(2, 0.6)@2000-01-05]']);
-- {[NPoint(1,0.2)@2000-01-01, NPoint(1,0.4)@2000-01-02, NPoint(1,0.5)@2000-01-03],
  [NPoint(2,0.6)@2000-01-04, NPoint(2,0.6)@2000-01-05]}
```

6.5. Conversión de puntos de red temporales

Un valor de punto de red temporal se puede convertir en y desde un punto de geometría temporal. Esto se puede hacer usando un `CAST` explícito o usando la notación `::`. Se devuelve un valor nulo si alguno de los valores de puntos de geometría que componen no se puede convertir en un valor `npoint`.

- Convertir un punto de red temporal en un punto de geometría temporal

  `tnpoint::tgeompoint`

```
SELECT astext((tnpoint '[NPoint(1, 0.2)@2001-01-01,
  NPoint(1, 0.3)@2001-01-02)')::tgeompoint);
-- [POINT(23.057077727326 28.7666335767956)@2001-01-01,
  POINT(48.7117553116406 20.9256801894708)@2001-01-02)
```

- Convertir un punto de geometría temporal en un punto de red temporal

  `tgeompoint::tnpoint`

```
SELECT tgeompoint '[POINT(23.057077727326 28.7666335767956)@2001-01-01,
  POINT(48.7117553116406 20.9256801894708)@2001-01-02)'::tnpoint
-- [NPoint(1,0.2)@2001-01-01, NPoint(1,0.3)@2001-01-02)
SELECT tgeompoint '[POINT(23.057077727326 28.7666335767956)@2001-01-01,
  POINT(48.7117553116406 20.9)@2001-01-02)'::tnpoint
-- NULL
```

Damos a continuación las funciones y operadores para los tipos de puntos de red.

6.6. Funciones y operadores para los tipos de puntos de red

Todas las funciones para tipos temporales descritas en el Capítulo 5 se pueden aplicar para tipos de puntos de red temporales. Por lo tanto, en las firmas de las funciones, la notación `base` también representa un `npoint` y las notaciones `ttype`, `tpoint` y `tgeompoint` también representan un `tnpoint`. Además, las funciones que tienen un argumento de tipo `geometry` aceptan además un argumento de tipo `npoint`. Para evitar la redundancia, a continuación solo presentamos algunos ejemplos de estas funciones y operadores para puntos de red temporales.

- Transformar un punto de red temporal en otro subtipo

  `tnpoint_inst(tnpoint): tnpoint_inst`

  `tnpoint_instset(tnpoint): tnpoint_instset`

  `tnpoint_seq(tnpoint): tnpoint_seq`

  `tnpoint_seqset(tnpoint): tnpoint_seqset`

```
SELECT tnpoint_seqset(tnpoint 'NPoint(1, 0.5)@2001-01-01');
-- {[NPoint(1,0.5)@2001-01-01]}
```

- Establecer la precisión de la fracción del punto de red temporal en el número de lugares decimales

  `setPrecision(tnpoint,integer): tnpoint`

```
SELECT setPrecision(tnpoint '{[NPoint(1, 0.123456789)@2012-01-01, NPoint(1, 0.5)@2012 ↩
    -01-02)}', 6);
-- {[NPoint(1,0.123457)@2012-01-01 00:00:00+01, NPoint(1,0.5)@2012-01-02 00:00:00+01)}
```

- Obtener los valores

  `getValues(tnpoint): npoint[]`

```
SELECT getValues(tnpoint '{[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.5)@2012-01-02)}');
-- {"NPoint(1,0.3)","NPoint(1,0.5)"}
SELECT getValues(tnpoint '{[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.3)@2012-01-02)}');
-- {"NPoint(1,0.3)"}
```

- Obtener el valor en una marca de tiempo

  `valueAtTimestamp(tnpoint,timestamptz): npoint`

```
SELECT valueAtTimestamp(tnpoint '[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.5)@2012-01-03)',
  '2012-01-02');
-- NPoint(1,0.4)
```

- Obtener la longitud atravesada por el punto de red temporal

  `length(tnpoint): float`

```
SELECT length(tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]');
-- 54.3757408468784
```

- Obtener la longitud acumulada atravesada por el punto de red temporal

  `cumulativeLength(tnpoint): tfloat`

```
SELECT cumulativeLength(tnpoint '{[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02,
  NPoint(1, 0.5)@2000-01-03], [NPoint(1, 0.6)@2000-01-04, NPoint(1, 0.7)@2000-01-05]}');
-- {[0@2000-01-01, 54.3757408468784@2000-01-02, 54.3757408468784@2000-01-03],
  [54.3757408468784@2000-01-04, 81.5636112703177@2000-01-05]}
```

- Obtener la velocidad del punto de red temporal en unidades por segundo

  `speed({tnpoint_seq, tpoint_seqset}): tfloat_seqset`

```
SELECT speed(tnpoint '[NPoint(1, 0.1)@2000-01-01, NPoint(1, 0.4)@2000-01-02,
  NPoint(1, 0.6)@2000-01-03]') * 3600 * 24;
-- Interp=Stepwise;[21.4016800272077@2000-01-01, 14.2677866848051@2000-01-02,
  14.2677866848051@2000-01-03]
```

- Construir el cuadro delimitador a partir de un punto de red y, opcionalmente, una marca de tiempo o un período

  `stbox(npoint): stbox`

  `stbox(npoint,{timestamptz,period}): stbox`

```
SELECT stbox(npoint 'NPoint(1,0.3)');
-- STBOX((48.711754,20.92568),(48.711758,20.925682))
SELECT stbox(npoint 'NPoint(1,0.3)', timestamptz '2000-01-01');
-- STBOX T((62.786633,80.143555,2000-01-01),(62.786636,80.143562,2000-01-01))
SELECT stbox(npoint 'NPoint(1,0.3)', period '[2000-01-01,2000-01-02]');
-- STBOX T((62.786633,80.143555,2000-01-01),(62.786636,80.143562,2000-01-02))
```

- Obtener el centroide ponderado en el tiempo

  `twCentroid(tnpoint): geometry(Point)`

```
SELECT st_astext(twCentroid(tnpoint '{[NPoint(1, 0.3)@2012-01-01,
  NPoint(1, 0.5)@2012-01-02, NPoint(1, 0.5)@2012-01-03, NPoint(1, 0.7)@2012-01-04)}'));
-- POINT(79.9787466444847 46.2385558051041)
```

- Obtener el acimut temporal

  `azimuth(tnpoint): tfloat`

```
SELECT azimuth(tnpoint '[NPoint(2, 0.3)@2012-01-01, NPoint(2, 0.7)@2012-01-02]');
-- {[0.974681063778863@2012-01-01 00:00:00+01,
  0.974681063778863@2012-01-01 23:54:36.721091+01),
  [3.68970843029227@2012-01-01 23:54:36.721091+01,
  3.68970843029227@2012-01-02 00:00:00+01)}
```

  Dado que la geometría subyacente asociada a una ruta puede tener varios vértices, el valor de acimut puede cambiar entre instantes del punto de red temporal de entrada, como se muestra en el ejemplo anterior.

- Obtener el instante del primer punto de red temporal en el que los dos argumentos están a la distancia más cercana

  `nearestApproachInstant({geo,npoint,tpoint},{geo,npoint,tpoint}): tpoint`

```
SELECT nearestApproachInstant(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', geometry 'Linestring(50 50,55 55)');
-- NPoint(2,0.349928)@2012-01-01 02:59:44.402905+01
SELECT nearestApproachInstant(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', npoint 'NPoint(1, 0.5)');
-- NPoint(2,0.592181)@2012-01-01 17:31:51.080405+01
```

- Obtener la distancia más pequeña entre los dos argumentos

  `nearestApproachDistance({geo,npoint,tpoint},{geo,npoint,tpoint}): float`

```
SELECT nearestApproachDistance(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', geometry 'Linestring(50 50,55 55)');
-- 1.41793220500979
SELECT nearestApproachDistance(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', npoint 'NPoint(1, 0.5)');
-- NPoint(2,0.592181)@2012-01-01 17:31:51.080405+01
```

  La función `nearestApproachDistance` tiene un operador asociado `|=|` que se puede utilizar para realizar búsquedas más cercanas utilizando un índice GiST (ver Sección 5.17).

- Obtener la línea que conecta el punto de aproximación más cercano entre los dos argumentos

  `shortestLine({geo,npoint,tpoint},{geo,npoint,tpoint}): geometry`

  The function will only return the first line that it finds if there are more than one

```
SELECT st_astext(shortestLine(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', geometry 'Linestring(50 50,55 55)'));
-- LINESTRING(50.7960725266492 48.8266286733015,50 50)
SELECT st_astext(shortestLine(tnpoint '[NPoint(2, 0.3)@2012-01-01,
  NPoint(2, 0.7)@2012-01-02]', npoint 'NPoint(1, 0.5)'));
-- LINESTRING(77.0902838115125 66.6659083092593,90.8134936900394 46.4385792121146)
```

- Restringir a un valor

  `atValue(tnpoint,base): tnpoint`

```
SELECT atValue(tnpoint '[NPoint(2, 0.3)@2012-01-01, NPoint(2, 0.7)@2012-01-03]',
  'NPoint(2, 0.5)');
-- {[NPoint(2,0.5)@2012-01-02]}
```

- Restringir a una geometría

  `atGeometry(tnpoint,geometry): tnpoint`

```
SELECT atGeometry(tnpoint '[NPoint(2, 0.3)@2012-01-01, NPoint(2, 0.7)@2012-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
```

- Diferencia con un valor

  `minusValue(tnpoint,base): tnpoint`

```
SELECT minusValue(tnpoint '[NPoint(2, 0.3)@2012-01-01, NPoint(2, 0.7)@2012-01-03]',
  'NPoint(2, 0.5)');
-- {[NPoint(2,0.3)@2012-01-01, NPoint(2,0.5)@2012-01-02),
  (NPoint(2,0.5)@2012-01-02, NPoint(2,0.7)@2012-01-03]}
```

- Diferencia con una geometría

  `minusGeometry(tnpoint,geometry): tnpoint`

```
SELECT minusGeometry(tnpoint '[NPoint(2, 0.3)@2012-01-01, NPoint(2, 0.7)@2012-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
-- {(NPoint(2,0.342593)@2012-01-01 05:06:40.364673+01,
  NPoint(2,0.7)@2012-01-03 00:00:00+01]}
```

- Operadores de comparación tradicionales

  `tnpoint = tnpoint: boolean`

  `tnpoint <> tnpoint: boolean`

  `tnpoint < tnpoint: boolean`

  `tnpoint > tnpoint: boolean`

  `tnpoint <= tnpoint: boolean`

  `tnpoint >= tnpoint: boolean`

```
SELECT tnpoint '{[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-02),
  [NPoint(1, 0.3)@2001-01-02, NPoint(1, 0.5)@2001-01-03]}' =
  tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.5)@2001-01-03]';
-- true
SELECT tnpoint '{[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.5)@2001-01-03]}' <>
  tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.5)@2001-01-03]';
-- false
SELECT tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' <
  tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.6)@2001-01-03]';
-- true
```

- Operadores de comparación temporales

  `tnpoint #= tnpoint: tbool`

  `tnpoint #<> tnpoint: tbool`

```
SELECT tnpoint '[NPoint(1, 0.2)@2012-01-01, NPoint(1, 0.4)@2012-01-03)' #=
  npoint 'NPoint(1, 0.3)';
-- {[f@2012-01-01, t@2012-01-02], (f@2012-01-02, f@2012-01-03)}
SELECT tnpoint '[NPoint(1, 0.2)@2012-01-01, NPoint(1, 0.8)@2012-01-03)' #<>
  tnpoint '[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.7)@2012-01-03)';
-- {[t@2012-01-01, f@2012-01-02], (t@2012-01-02, t@2012-01-03)}
```

- Operadores de igualdad siempre y alguna vez

  `tnpoint ?= tnpoint: boolean`

  `tnpoint &= tnpoint: boolean`

```
SELECT tnpoint '[Npoint(1, 0.2)@2012-01-01, Npoint(1, 0.4)@2012-01-04)' ?= Npoint(1, 0.3);
-- true
SELECT tnpoint '[Npoint(1, 0.2)@2012-01-01, Npoint(1, 0.2)@2012-01-04)' &= Npoint(1, 0.2);
-- true
```

- Operadores de posición relativa

  `tnpoint << tnpoint: boolean`

  `tnpoint &< tnpoint: boolean`

  `tnpoint >> tnpoint: boolean`

  `tnpoint &> tnpoint: boolean`

  `tnpoint <<| tnpoint: boolean`

  `tnpoint &<| tnpoint: boolean`

  `tnpoint |>> tnpoint: boolean`

  `tnpoint |&> tnpoint: boolean`

  `tnpoint <<# tnpoint: boolean`

  `tnpoint &<# tnpoint: boolean`

  `tnpoint #>> tnpoint: boolean`

  `tnpoint |&> tnpoint: boolean`

```
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' <<
  npoint 'NPoint(1, 0.2)'
-- false
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' <<|
```

```
  stbox(npoint 'NPoint(1, 0.5)')
-- false
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' &>
  npoint 'NPoint(1, 0.3)'::geometry
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' >>#
  tnpoint '[NPoint(1, 0.3)@2000-01-03, NPoint(1, 0.5)@2000-01-05]'
-- true
```

- Operadores topológicos

```
tnpoint && tnpoint: boolean
tnpoint <@ tnpoint: boolean
tnpoint @> tnpoint: boolean
tnpoint ~= tnpoint: boolean
tnpoint -|- tnpoint: boolean
```

```
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' &&
  npoint 'NPoint(1, 0.5)'
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]' @>
  stbox(npoint 'NPoint(1, 0.5)')
-- true
SELECT npoint 'NPoint(1, 0.5)'::geometry <@
  tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-02]'
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' ~=
  tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.35)@2000-01-02,
  NPoint(1, 0.5)@2000-01-03]'
-- true
```

- Obtener la distancia más pequeña entre los dos argumentos

```
tgeompoint |=| tnpoint: float
```

```
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' |=|
  npoint 'NPoint(1, 0.2)';
-- 2.34988300875063
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' |=|
  geometry 'Linestring(2 2,2 1,3 1)';
-- 82.2059262761477
```

- Obtener la distancia temporal

```
tgeompoint <-> tnpoint: tfloat
```

```
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' <->
  npoint 'NPoint(1, 0.2)';
-- [2.34988300875063@2000-01-02 00:00:00+01, 2.34988300875063@2000-01-03 00:00:00+01]
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' <->
  geometry 'Point(50 50)';
-- [25.0496666945044@2000-01-01 00:00:00+01, 26.4085688426232@2000-01-03 00:00:00+01]
SELECT tnpoint '[NPoint(1, 0.3)@2000-01-01, NPoint(1, 0.5)@2000-01-03]' <->
  tnpoint '[NPoint(1, 0.3)@2000-01-02, NPoint(1, 0.5)@2000-01-04]'
-- [2.34988300875063@2000-01-02 00:00:00+01, 2.34988300875063@2000-01-03 00:00:00+01]
```

- Relaciones posibles espaciales

```
contains(geometry,tnpoint): boolean
disjoint({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
intersects({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
touches({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
dwithin({geometry,npoint,tnpoint},{geometry,npoint,tnpoint},float): boolean
```

```
SELECT contains(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)');
-- false
SELECT disjoint(npoint 'NPoint(2, 0.0)',
  tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)');
-- true
SELECT intersects(tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)',
  tnpoint '[NPoint(2, 0.0)@2012-01-01, NPoint(2, 1)@2012-01-03)');
-- false
```

- Relaciones espaciales temporales

```
tcontains(geometry,tnpoint): boolean
tdisjoint({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
tintersects({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
ttouches({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}): boolean
tdwithin({geometry,npoint,tnpoint},{geometry,npoint,tnpoint},float): boolean
```

```
SELECT tdisjoint(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)');
-- {[t@2012-01-01 00:00:00+01, t@2012-01-03 00:00:00+01)}
SELECT tdwithin(tnpoint '[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.5)@2012-01-03)',
  tnpoint '[NPoint(1, 0.5)@2012-01-01, NPoint(1, 0.3)@2012-01-03)', 1);
-- {[t@2012-01-01 00:00:00+01, t@2012-01-01 22:35:55.379053+01],
  (f@2012-01-01 22:35:55.379053+01, t@2012-01-02 01:24:04.620946+01,
  t@2012-01-03 00:00:00+01)}
```

6.7. Funciones agregadas

Las tres funciones agregadas para puntos de red temporales se ilustran a continuación.

- Conteo temporal

```
tcount(tnpoint): {tint_instset,tint_seqset}
```

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2012-01-02, NPoint(1, 0.4)@2012-01-04)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2012-01-03, NPoint(1, 0.5)@2012-01-05)' )
SELECT tcount(Temp)
FROM Temp
-- {[1@2012-01-01, 2@2012-01-02, 1@2012-01-04, 1@2012-01-05)}
```

- Conteo de ventana

  `wcount(tnpoint): {tint_instset,tint_seqset}`

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2012-01-02, NPoint(1, 0.4)@2012-01-04)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2012-01-03, NPoint(1, 0.5)@2012-01-05)' )
SELECT wcount(Temp, '1 day')
FROM Temp
-- {[1@2012-01-01, 2@2012-01-02, 3@2012-01-03, 2@2012-01-04, 1@2012-01-05,
  1@2012-01-06)}
```

- Centroide temporal

  `tcentroid(tnpoint): tgeompoint`

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2012-01-01, NPoint(1, 0.3)@2012-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2012-01-01, NPoint(1, 0.4)@2012-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2012-01-01, NPoint(1, 0.5)@2012-01-03)' )
SELECT astext(tcentroid(Temp))
FROM Temp
-- {[POINT(72.451531682218 76.5231414472853)@2012-01-01,
  POINT(55.7001249027598 72.9552602410653)@2012-01-03)}
```

6.8. Indexación de puntos de red temporales

Se pueden crear índices GiST y SP-GiST para columnas de tabla de puntos de redes temporales. A continuación, se muestra un ejemplo de creación de índice:

```
CREATE INDEX Trips_Trip_SPGist_Idx ON Trips USING SPGist(Trip);
```

Los índices GiST y SP-GiST almacenan el cuadro delimitador para los puntos de la red temporal, que es un `stbox` y, por lo tanto, almacena las coordenadas absolutas del espacio subyacente.

Un índice GiST o SP-GiST puede acelerar las consultas que involucran a los siguientes operadores:

- `<<`, `&<`, `&>`, `>>`, `<<|`, `&<|`, `|&>`, `|>>`, que solo consideran la dimensión espacial en puntos de la red temporal,
- `<<#`, `&<#`, `#&>`, `#>>`, que solo consideran la dimensión temporal en puntos de la red temporal,
- `&&`, `@>`, `<@`, `~=`, and `-|-`, que consideran tantas dimensiones como compartan la columna indexada y el argumento de consulta.

Estos operadores trabajan con cuadros delimitadores, no con los valores completos. Además, un índice GiST puede acelerar las consultas del vecino más cercano que involucran al operador `|=|`.

Apéndice A

Referencia de MobilityDB

A.1. Funciones y operadores para tipos de tiempo y tipos de rango

A.1.1. Funciones de constructor

- period: Constructor para `period`
- timestampset: Constructor para `timestampset`
- periodset: Constructor para `periodset`

A.1.2. Conversión de tipos

- timestamptz::time: Convertir un `timestamptz` a otro tipo de tiempo
- timestampset::periodset: Convertir un `timestampset` a un `periodset`
- period::type: Convertir un `period` a otro tipo de tiempo
- tstzrange::period: Convertir un `tstzrange` a un `period`

A.1.3. Funciones de accesor

- memSize: Obtener el tamaño de la memoria en bytes
- lower: Obtener el límite inferior
- upper: Obtener el límite superior
- lower_inc: ¿Es el límite inferior inclusivo?
- upper_inc: ¿Es el límite superior inclusivo?
- duration: Obtener el intervalo de tiempo
- timespan: Obtener el intervalo de tiempo ignorando las posibles brechas de tiempo
- period: Obtener el período en el que se define el conjunto de marcas de tiempo o el conjunto de períodos ignorando las posibles brechas de tiempo
- numTimestamps: Obtener el número de marcas de tiempo diferentes
- startTimestamp: Obtener la marca de tiempo inicial

- endTimestamp: Obtener la marca de tiempo final
- timestampN: Obtener la enésima marca de tiempo diferente
- timestamps: Obtener las marcas de tiempo diferentes
- numPeriods: Obtener el número de períodos
- startPeriod: Obtener el período inicial
- endPeriod: Obtener el período final
- periodN: Obtener el enésimo período
- periods: Obtener los períodos

A.1.4. Funciones de modificación

- shift: Desplazar el valor de tiempo con un intervalo
- setPrecision: Redondear los límites de un rango flotante a un número de decimales

A.1.5. Operadores de comparación

- =: ¿Son iguales los valores de tiempo?
- <>: ¿Son diferentes los valores de tiempo?
- <: ¿Es el primer valor de tiempo menor que el segundo?
- >: ¿Es el primer valor de tiempo mayor que el segundo?
- <=: ¿Es el primer valor de tiempo menor o igual que el segundo?
- >=: ¿Es el primer valor de tiempo mayor o igual que el segundo?

A.1.6. Operadores de conjuntos

- +: Unión de valores de tiempo
- *: Intersección de valores de tiempo
- -: Diferencia de valores de tiempo

A.1.7. Operadores topológicos y de posición relativa

- &&: ¿Se superponen los valores de tiempo (tienen instantes en común)?
- @>: ¿Contiene el primer valor de tiempo el segundo?
- <@: ¿Está el primer valor de tiempo contenido en el segundo?
- -|-: ¿Es el primer valor de tiempo adyacente al segundo?
- <<: ¿Está el primer número o valor de rango estrictamente a la izquierda del segundo?
- >>: ¿Está el primer número o valor de rango estrictamente a la izquierda del segundo?
- &<: ¿No está el primer número o valor de rango a la derecha del segundo?
- &>: ¿No está el primer número o valor de rango a la izquierda del segundo?

- -|-: ¿Es el primer número o valor de rango adyacente al segundo?
- <<#: ¿Es el primer valor de tiempo estrictamente anterior al segundo?
- #>>: ¿Es el primer valor de tiempo estrictamente posterior al segundo?
- &<#: ¿No es el primer valor de tiempo posterior al segundo?
- #&>: ¿No es el primer valor de tiempo anterior al segundo?

A.1.8. Funciones agregadas

- tcount: Conteo temporal
- extent: Extensión del período delimitador
- extent: Extensión del rango delimitador
- tunion: Unión temporal

A.2. Funciones y operadores para tipos de cuadro delimitadores

A.2.1. Funciones de constructor

- tbox: Constructor para `tbox`
- stbox, stboxt: Constructor para `stbox`

A.2.2. Conversión de tipos

- tbox::type: Convertir un `tbox` a otro tipo
- type::tbox: Convertir otro tipo a un `tbox`
- stbox::type: Convertir un `stbox` a otro tipo
- type::stbox: Convertir otro tipo a un `stbox`

A.2.3. Funciones de accesor

- hasX: ¿Tiene dimension X?
- hasZ: ¿Tiene dimension Z?
- hasT: ¿Tiene dimension T?
- Xmin: Obtener el valor mínimo de X
- Xmax: Obtener el valor máximo de X
- Ymin: Obtener el valor mínimo de Y
- Ymax: Obtener el valor máximo de Y
- Zmin: Obtener el valor mínimo de Z
- Zmax: Obtener el valor máximo de Z
- Tmin: Obtener el valor mínimo de T
- Tmax: Obtener el valor máximo de T

A.2.4. Funciones de modificación

- expandValue: Extender la dimensión de valor numérico del cuadro delimitador con un valor flotante
- expandSpatial: Extender la dimensión de valor espacial del cuadro delimitador con un valor flotante
- expandTemporal: Extender la dimensión temporal del cuadro delimitador con un intervalo de tiempo
- setPrecision: Redondear el valor o las coordenadas del cuadro delimitador a un número de decimales

A.2.5. Funciones del sistema de referencia espacial

- SRID: Obtener el identificador de referencia espacial
- setSRID: Especificar el identificador de referencia espacial
- transform: Transformar a una referencia espacial diferente

A.2.6. Funciones agregadas

- extent: Extensión del cuadro delimitador

A.2.7. Operadores de comparación

- =: ¿Son iguales los cuadros delimitadores?
- <>: ¿Son diferentes los cuadros delimitadores?
- <: ¿Es el primer cuadro delimitador menor que el segundo?
- >: ¿Es el primer cuadro delimitador mayor que el segundo?
- <=: ¿Es el primer cuadro delimitador menor o igual que el segundo?
- >=: ¿Es el primer cuadro delimitador menor que el segundo?

A.2.8. Operadores de conjuntos

- +: Union de los cuadros delimitadores
- *: Intersección de los cuadros delimitadores

A.2.9. Operadores topológicos

- &&: ¿Se superponen los cuadros delimitadores?
- @>: ¿Contiene el primer cuadro delimitador el segundo?
- <@: ¿Está el primer cuadro delimitador contenido en el segundo?
- ~=: ¿Son los cuadros delimitadores iguales en sus dimensiones comunes?
- -|-: ¿Son los cuadros delimitadores adyacentes?

A.2.10. Operadores de posición relativa

- <<: ¿Son los valores X del primer cuadro delimitador estrictamente mayores que los del segundo?
- >>: ¿Son los valores X del primer cuadro delimitador estrictamente mayores que los del segundo?
- &<: ¿No son los valores X del primer cuadro delimitador mayores que los del segundo?
- &>: ¿No son los valores X del primer cuadro delimitador menores que los del segundo?
- <<: ¿Están los valores X del primer cuadro delimitador estrictamente a la izquierda de los del segundo?
- >>: ¿Están los valores X del primer cuadro delimitador estrictamente a la derecha de los del segundo?
- &<: ¿No están los valores X del primer cuadro delimitador a la derecha de los del segundo?
- &>: ¿No están los valores X del primer cuadro delimitador a la izquierda de los del segundo?
- <<|: ¿Están los valores de Y del primer cuadro delimitador estrictamente debajo de los del segundo?
- |>>: ¿Están los valores de Y del primer cuadro delimitador estrictamente arriba de los del segundo?
- &<|: ¿No están los valores de Y del primer cuadro delimitador debajo de los del segundo?
- |&>: ¿No están los valores de Y del primer cuadro delimitador debajo de los del segundo?
- <</: ¿Están los valores Z del primer cuadro delimitador estrictamente delante de los del segundo?
- />>: ¿Están los valores Z del primer cuadro delimitador estrictamente detrás de los del segundo?
- &</: ¿No están los valores Z del primer cuadro delimitador detrás de los del segundo?
- /&>: ¿No están los valores Z del primer cuadro delimitador delante de los del segundo?
- <<#: ¿Son los valores de T del primer cuadro delimitador estrictamente anteriores a los del segundo?
- #>>: ¿Son los valores de T del primer cuadro delimitador estrictamente posteriores a los del segundo?
- &<#: ¿No son los valores de T del primer cuadro delimitador posteriores a los del segundo?
- #&>: ¿No son los valores de T del primer cuadro delimitador anteriores a los del segundo?

A.3. Funciones y operadores para tipos temporales

A.3.1. Funciones de constructor

- ttype_inst: Constructor para tipos temporales de subtipo instante
- ttype_instset: Constructor para tipos temporales de subtipo conjunto de instantes
- ttype_seq: Constructor para tipos temporales de subtipo secuencia
- ttype_seqset: Constructor para tipos temporales de subtipo conjunto de secuencias

A.3.2. Conversión de tipos

- ttype::period: Convertir un valor temporal a un período
- tnumber::range: Convertir un número temporal a un rango
- tnumber::tbox: Convertir un número temporal number a un `tbox`
- tpoint::stbox: Convertir un punto temporal a un `stbox`
- tint::tfloat: Convertir un entero temporal en un flotante temporal
- tfloat::tint: Convertir un flotante temporal en un entero temporal
- tgeompoint::tgeogpoint: Convertir un punto geométrico temporal en un punto geográfico temporal
- tgeogpoint::tgeompoint: Convertir un punto geográfico temporal en un punto geométrico temporal
- tgeompoint::geometry, tgeogpoint::geography: Convertir un punto temporal en una trayectoria PostGIS
- geometry::tgeompoint, geography::tgeogpoint: Convertir una trayectoria PostGIS en un punto temporal

A.3.3. Funciones de transformación

- ttype_inst, ttype_instset, ttype_seq, ttype_seqset: Transformar un valor temporal en otro subtipo
- toLinear: Transformar un valor temporal con tipo de base continuo de interpolación escalonada a lineal
- appendInstant: Anexar un instante temporal a un valor temporal
- merge: Fusionar los valores temporales

A.3.4. Funciones de accesor

- memSize: Obtener el tamaño de la memoria en bytes
- tempSubtype: Obtener el subtipo temporal
- interpolation: Obtener la interpolación
- getValue: Obtener el valor
- getValues: Obtener los valores
- startValue: Obtener el valor inicial
- endValue: Obtener el valor final
- minValue: Obtener el valor mínimo
- maxValue: Obtener el valor máximo
- valueRange: Obtener el rango de valores
- valueAtTimestamp: Obtener el valor en una marca de tiempo
- getTimestamp: Obtener la marca de tiempo
- getTime: Obtener el tiempo
- duration: Obtener el intervalo de tiempo
- timespan: Obtener el intervalo de tiempo ignorando las posibles brechas de tiempo
- period: Obtener el período en el que está definido el valor temporal ignorando las posibles brechas de tiempo

- numInstants: Obtener el número de instantes diferentes
- startInstant: Obtener el instante inicial
- endInstant: Obtener el instante final
- instantN: Obtener el enésimo instante diferente
- instants: Obtener los instantes diferentes
- numTimestamps: Obtener el número de marcas de tiempo diferentes
- startTimestamp: Obtener la marca de tiempo inicial
- endTimestamp: Obtener la marca de tiempo final
- timestampN: Obtener la enésima marca de tiempo diferente
- timestamps: Obtener las marcas de tiempo diferentes
- numSequences: Obtener el número de secuencias
- startSequence: Obtener la secuencia inicial
- endSequence: Obtener la secuencia final
- sequenceN: Obtener la enésima secuencia
- sequences: Obtener las secuencias
- segments: Obtener los segmentos
- shift: Desplazar el intervalo de tiempo del valor temporal con un intervalo
- tscale: Escalear el intervalo de tiempo del valor temporal a un intervalo.
- shiftTscale: Desplazar y escalear el intervalo de tiempo del valor temporal a los dos intervalos.
- intersectsTimestamp: ¿Se cruza el valor temporal con la marca de tiempo?
- intersectsTimestampSet: ¿Se cruza el valor temporal con el conjunto de marcas tiempo?
- intersectsPeriod: ¿Se cruza el valor temporal con el período?
- intersectsPeriodSet: ¿Se cruza el valor temporal con el conjunto de períodos?
- twAvg: Obtener el promedio ponderado en el tiempo

A.3.5. Funciones de restricción

A.3.5.1. Funciones de selección

- atValue: Restringir a un valor
- atValues: Restringir a una matriz de valores
- atRange: Restringir a un rango de valores
- atRanges: Restringir a una matriz de rangos de valores
- atMin: Restringir al valor mínimo
- atMax: Restringir al valor máximo
- atGeometry: Restringir a una geometría

- atTimestamp: Restringir a una marca de tiempo
- atTimestampSet: Restringir a un conjunto de marcas de tiempo
- atPeriod: Restringir a un período
- atPeriodSet: Restringir a un conjunto de períodos
- atTbox: Restringir a un `tbox`
- atStbox: Restringir a un `stbox`

A.3.5.2. Funciones de diferencia

- minusValue: Diferencia con un valor
- minusValues: Diferencia con una matriz de valores
- minusRange: Diferencia con un rango de valores
- minusRanges: Diferencia con una matriz de rangos de valores
- minusMin: Diferencia con el valor mínimo
- minusMax: Diferencia con el valor máximo
- minusGeometry: Diferencia con una geometría
- minusTimestamp: Diferencia con una marca de tiempo
- minusTimestampSet: Diferencia con un conjunto de marcas de tiempo
- minusPeriod: Diferencia con un período
- minusPeriodSet: Diferencia con un conjunto de períodos
- minusTbox: Diferencia con un `tbox`
- minusStbox: Diferencia con un `stbox`

A.3.6. Operadores de comparación

A.3.6.1. Operadores de comparación tradicionales

- =: ¿Son iguales los valores temporales?
- <>: ¿Son diferentes los valores temporales?
- <: ¿Es el primer valor temporal menor que el segundo?
- >: ¿Es el primer valor temporal mayor que el segundo?
- <=: ¿Es el primer valor temporal menor o igual que el segundo?
- >=: ¿Es el primer valor temporal mayor o igual que el segundo?

A.3.6.2. Operadores de comparación alguna vez y siempre

- ?=: ¿Es el valor temporal alguna vez igual al valor?
- ?<>: ¿Es el valor temporal alguna vez diferente del valor?
- ?<: ¿Es el valor temporal alguna vez menor que el valor?
- ?>: ¿Es el valor temporal alguna vez mayor que el valor?
- ?<=: ¿Es el valor temporal alguna vez menor o igual que el valor?
- ?>=: ¿Es el valor temporal alguna vez mayor o igual que el valor?
- %=: ¿Es el valor temporal siempre igual que el valor?
- %<>: ¿Es el valor temporal siempre diferente que el valor?
- %<: ¿Es el valor temporal siempre menor que el valor?
- %>: ¿Es el valor temporal siempre mayor que el valor?
- %<=: ¿Es el valor temporal siempre menor o igual que el valor?
- %>=: ¿Es el valor temporal siempre mayor o igual que el valor?

A.3.6.3. Operadores de comparación temporal

- #=: Igual temporal
- #<>: Diferente temporal
- #<: Menor que temporal
- #>: Mayor que temporal
- #<=: Menor o igual que temporal
- #>=: Mayor o igual que temporal

A.3.7. Funciones y operadores matemáticos

- +: Adición temporal
- -: Resta temporal
- *: Multiplicación temporal
- /: División temporal
- round: Redondear los valores a un número de posiciones decimales
- degrees: Convertir de radianes a grados
- derivative: Obtener la derivada sobre el tiempo del número temporal en unidades por segundo

A.3.8. Operadores booleanos

- &: Y temporal
- |: O temporal
- ~: No temporal

A.3.9. Funciones y operadores de texto

- ||: Concatenación de texto temporal
- upper: Transformar a mayúsculas
- lower: Transformar a minúsculas

A.3.10. Funciones espaciales

A.3.10.1. Funciones de entrada/salida

- asText: Obtener la representación de texto conocido (Well-Known Text o WKT)
- asEWKT: Obtener la representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- asMFJSON: Obtener la representación JSON de características móviles (Moving Features)
- asBinary: Obtener la representación binaria conocida (Well-Known Binary o WKB)
- asEWKB: Obtener la representación extendida binaria conocida (Extended Well-Known Binary o EWKB)
- asHexEWKB: Obtener la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB) en formato texto
- tgeompointFromText: Entrar un punto temporal geométrico en representación de texto conocido (Well-Known Text o WKT)
- tgeogpointFromText: Entrar un punto temporal geográfico en representación de texto conocido (Well-Known Text o WKT)
- tgeompointFromEWKT: Entrar un punto temporal geométrico en representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- tgeogpointFromEWKT: Entrar un punto temporal geográfico en representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- tgeompointFromMFJSON: Entrar un punto temporal geométrico en una representación JSON de características móviles (Moving Features)
- tgeogpointFromMFJSON: Entrar un punto temporal geográfico en una representación JSON de características móviles (Moving Features)
- tgeompointFromBinary: Entrar un punto temporal geométrico en representación binaria conocida (Well-Known Binary o WKB)
- tgeogpointFromBinary: Entrar un punto temporal geográfico en representación binaria conocida (Well-Known Binary o WKB)
- tgeompointFromEWKB: Entrar un punto temporal geométrico en una representación extendida binaria conocida (Extended Well-Known Binary o EWKB)
- tgeogpointFromEWKB: Entrar un punto temporal geográfico en una representación extendida binaria conocida (Extended Well-Known Binary o EWKB)
- tgeompointFromHexEWKB: Entrar un punto temporal geométrico en representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB) en formato texto
- tgeogpointFromHexEWKB: Entrar un punto temporal geográfico en representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB) en formato texto

A.3.10.2. Funciones de sistema de referencia espacial

- SRID: Obtener el identificador de referencia espacial
- setSRID: Establecer el identificador de referencia espacial
- transform: Transformar a una referencia espacial diferente

A.3.10.3. Funciones de accesor

- getX: Obtener los valores de las coordenadas X como un número flotante temporal
- getY: Obtener los valores de las coordenadas Y como un número flotante temporal
- getZ: Obtener los valores de las coordenadas Z como un número flotante temporal
- isSimple: Devuelve verdadero si el punto temporal no se auto-intersecta espacialmente
- length: Obtener la longitud atravesada por el punto temporal
- cumulativeLength: Obtener la longitud acumulada atravesada por el punto temporal
- speed: Obtener la velocidad del punto temporal en unidades por segundo
- twCentroid: Obtener el centroide ponderado en el tiempo
- azimuth: Obtener el acimut temporal
- bearing: Obtener el rumbo temporal

A.3.10.4. Funciones de manipulación

- setPrecision: Redondear los valores de las coordenadas a un número de decimales
- makeSimple: Devuelve una matriz de fragmentos del punto temporal que son simples
- simplify: Simplificar un punto temporal usando una generalización del algoritmo de Douglas-Peucker
- geoMeasure: Construir una geometría/geografía con medida M a partir de un punto temporal y un número flotante temporal
- asMVTGeom: Transformar un punto geométrico temporal en el espacio de coordenadas de un Mapbox Vector Tile

A.3.10.5. Operadores de distancia

- |=|: Obtener la distancia mínima que haya existido
- nearestApproachInstant: Obtener el instante del primer punto temporal en el que los dos argumentos están a la distancia más cercana
- shortestLine: Obtener la línea que conecta el punto de aproximación más cercano
- <->: Obtener la distancia temporal

A.3.10.6. Relaciones posibles

- contains: Puede contener
- disjoint: Puede ser disjunto
- dwithin: Puede estar a distancia de
- intersects: Puede cruzarse
- touches: Puede tocar

A.3.10.7. Relaciones temporales

- tcontains: Contiene temporal
- tdisjoint: Disjunto temporal
- tdwithin: Estar a distancia de temporal
- tintersects: Intersección temporal
- ttouches: Toca temporal

A.3.11. Funciones de similaridad

- frechetDistance: Obtener la distancia de Fréchet discreta entre dos valores temporales
- frechetDistancePath: Obtener las parejas de correspondencia entre dos valores temporales con respecto a la distancia de Fréchet discreta
- dynamicTimeWarp: Obtener la distancia de distorsión de tiempo dinámica (Dynamic Time Warp) entre dos valores temporales
- dynamicTimeWarpPath: Obtener las parejas de correspondencia entre dos valores temporales con respecto a la distancia de distorsión de tiempo dinámica (Dynamic Time Warp)

A.3.12. Mosaicos multidimensionales

A.3.12.1. Operaciones de intervalos

- bucketList: Obtener un conjunto de parejas (índice, intervalo) que cubre el rango o el período con intervalos de la misma amplitud alineados con el origen
- valueBucket: Obtener el valor inicial del intervalo que contiene el número de entrada.
- rangeBucket: Obtener el rango en el espacio de intervalos contiene el número de entrada.
- timeBucket: Obtener el valor inicial del intervalo que contiene la marca de tiempo de entrada.
- periodBucket: Obtener el período en el espacio de intervalos que contiene la marca de tiempo de entrada.

A.3.12.2. Operaciones de malla

- multidimGrid: Obtener un conjunto de parejas (índices, mosaico) que cubre el cuadro delimitador con mosaicos multidimensionales del mismo tamaño y duración
- multidimTile: Obtener el mosaico de la malla multidimensional que contiene el valor y la marca de tiempo

A.3.12.3. Operaciones de fragmentación

- valueSplit: Fragmentar el número temporal con respecto a intervalos de valores
- timeSplit: Fragmentar el valor temporal con respecto a intervalos de tiempo
- valueTimeSplit: Fragmentar el número temporal con respecto a los mosaicos de una malla de valores y de tiempo
- spaceSplit: Fragmentar el punto temporal con respecto a los mosaicos de una malla espacial
- spaceTimeSplit: Fragmentar el punto temporal con respecto a los mosaicos de una malla espacio-temporal

A.3.13. Funciones agregadas

- tcount: Conteo temporal
- extent: Extensión del cuadro delimitador
- tand: Y temporal
- tor: O temporal
- tmin: Mínimo temporal
- tmax: Máximo temporal
- tsum: Suma temporal
- tavg: Promedio temporal
- wmin: Mínimo de ventana
- wmax: Máximo de ventana
- wcount: Conteo de ventana
- wsum: Suma de ventana
- wavg: Promedio de ventana
- tcentroid: Centroide temporal

A.3.14. Funciones de utilidad

- mobilitydb_version: Versión de la extensión MobilityDB
- mobilitydb_full_version: Versión de la extensión MobilityDB y de sus dependencias

A.4. Funciones y operadores para puntos de red temporales

A.4.1. Tipos de red estáticos

A.4.1.1. Funciones de constructor

- npoint: Constructor para puntos de red
- nsegment: Constructor para segmentos de red

A.4.1.2. Funciones de modificación

- setPrecision: Establecer la precisión de la(s) posición(es) del punto de red or el segmento de red en el número de posiciones decimales

A.4.1.3. Funciones de accesor

- route: Obtener el identificador de ruta
- getPosition: Obtener la fracción de posición
- startPosition: Obtener la posición inicial
- endPosition: Obtener la posición final
- SRID: Obtener el identificador de referencia espacial

A.4.1.4. Funciones espaciales

- :::: Convertir un punto de red en una geometría
- :::: Convertir una geometría en un punto de red
- equals: Igualdad espacial para puntos de red

A.4.1.5. Operadores de comparación

- =: ¿Son iguales los valores?
- <>: ¿Son diferentes los valores?
- <: ¿El primer valor es menor que el segundo?
- >: ¿El primer valor es mayor que el segundo?
- <=: ¿El primer valor es menor o igual que el segundo?
- >=: ¿El primer valor es mayor o igual que el segundo?

A.4.2. Puntos de red temporales

A.4.2.1. Constructores

- tnpoint_inst: Constructor para puntos de red temporal de subtipo instante
- tnpoint_instset: Constructor para puntos de red temporal de subtipo conjunto de instantes
- tnpoint_seq: Constructor para puntos de red temporal de subtipo secuencia
- tnpoint_seqset: Constructor para puntos de red temporal de subtipo conjunto de secuencias

A.4.2.2. Conversión de tipos

- :::: Convertir un punto de red temporal en un punto de geometría temporal
- :::: Convertir un punto de geometría temporal en un punto de red temporal

A.4.2.3. Funciones y operadores

- tnpoint_inst, tnpoint_instset, tnpoint_seq, tnpoint_seqset: Transformar un punto de red temporal en otro subtipo
- setPrecision: Establecer la precisión de la fracción del punto de red temporal en el número de posiciones decimales
- getValues: Obtener los valores
- valueAtTimestamp: Obtener el valor en una marca de tiempo
- length: Obtener la longitud atravesada por el punto de red temporal
- cumulativeLength: Obtener la longitud acumulada atravesada por el punto de red temporal
- speed: Obtener la velocidad del punto de red temporal en unidades por segundo
- stbox: Construir el cuadro delimitador a partir de un punto de red y, opcionalmente, una marca de tiempo o un período
- twCentroid: Obtener el centroide ponderado en el tiempo
- azimuth: Obtener el acimut temporal

- nearestApproachInstant: Obtener el instante del primer punto de red temporal en el que los dos argumentos están a la distancia más cercana
- nearestApproachDistance: Obtener la distancia más pequeña entre los dos argumentos
- shortestLine: Obtener la línea que conecta el punto de aproximación más cercano entre los dos argumentos
- atValue: Restringir a un valor
- atGeometry: Restringir a una geometría
- minusValue: Diferencia con un valor
- minusGeometry: Diferencia con una geometría
- =, <>, <, >, <=, >=: Operadores de comparación tradicionales
- #=, #<>: Operadores de comparación temporales
- ?=, &=: Operadores de igualdad siempre y alguna vez
- <<, &<, >>, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, |&>: Operadores de posición relativa
- &&, <@, @>, ~=, -|-: Operadores topológicos
- |=|: Obtener la distancia más pequeña entre los dos argumentos
- <->: Obtener la distancia temporal
- contains, disjoint, intersects, touches, dwithin: Relaciones espaciales posibles
- tcontains, tdisjoint, tintersects, ttouches, tdwithin: Relaciones espaciales temporales

A.4.2.4. Funciones agregadas

- tcount: Conteo temporal
- wcount: Conteo de ventana
- tcentroid: Centroide temporal

Apéndice B

Generador de datos sintéticos

En muchas circunstancias, es necesario tener un conjunto de datos de prueba para evaluar enfoques de implementación alternativos o realizar evaluaciones comparativas. A menudo se requiere que tal conjunto de datos tenga requisitos particulares en tamaño o en las características intrínsecas de sus datos. Incluso si un conjunto de datos del mundo real pudiera estar disponible, puede que no sea ideal para tales experimentos por múltiples razones. Por lo tanto, un generador de datos sintéticos que pueda personalizarse para producir datos de acuerdo con los requisitos dados suele ser la mejor solución. Obviamente, los experimentos con datos sintéticos deben complementarse con experimentos con datos del mundo real para tener una comprensión profunda del problema en cuestión.

MobilityDB proporciona un generador simple de datos sintéticos que se puede utilizar para tales fines. En particular, se utilizó este generador de datos para generar la base de datos utilizada para las pruebas de regresión en MobilityDB. El generador de datos está programado en PL/pgSQL para que se pueda personalizar fácilmente. Se encuentra en el directorio `datagen` en el repositorio. En este apéndice, presentamos brevemente la funcionalidad básica del generador. Primero enumeramos las funciones que generan valores aleatorios para varios tipos de datos de PostgreSQL, PostGIS y MobilityDB y luego damos ejemplos de cómo se usan estas funciones para generar tablas de dichos valores. Los parámetros de las funciones no están especificados, consulte los archivos fuente donde se pueden encontrar explicaciones detalladas sobre los distintos parámetros.

B.1. Generador para tipos PostgreSQL

- `random_bool`: Generar un booleano aleatorio
- `random_int`: Generar un entero aleatorio
- `random_int_array`: Generar una matriz de enteros aleatorios
- `random_intrango`: Generar un rango aleatorio de enteros
- `random_float`: Generar un número flotante aleatorio
- `random_float_array`: Generar una matriz de números flotantes aleatorios
- `random_floatrango`: Generar un rango aleatorio de números flotantes
- `random_text`: Generar un texto aleatorio
- `random_timestamptz`: Generar una marca de tiempo con zona horaria aleatoria
- `random_timestamptz_array`: Generar una matriz de marcas de tiempo con zona horaria aleatorias
- `random_minutos`: Generar un intervalo de minutos aleatorio
- `random_tstzrange`: Generar rango aleatorio de marcas de tiempo con zona horaria
- `random_tstzrango_array`: Generar una matriz de rangos aleatorios de marcas de tiempo con zona horaria

B.2. Generador para tipos PostGIS

- `random_geom_point`: Generar un punto geométrico 2D aleatorio
- `random_geom_point3D`: Generar un punto geométrico 3D aleatorio
- `random_geog_point`: Generar un punto geográfico 2D aleatorio
- `random_geog_point3D`: Generar un punto geográfico 3D aleatorio
- `random_geom_point_array`: Generar una matriz de puntos geométricos 2D aleatorios
- `random_geom_point3D_array`: Generar una matriz de puntos geométricos 3D aleatorios
- `random_geog_point_array`: Generar una matriz puntos geográficos 2D aleatorios
- `random_geog_point3D_array`: Generar una matriz de puntos geográficos 3D aleatorios
- `random_geom_linestring`: Generar una cadena lineal geométrica 2D aleatoria
- `random_geom_linestring3D`: Generar una cadena lineal geométrica 3D aleatoria
- `random_geog_linestring`: Generar una cadena lineal geográfica 2D aleatoria
- `random_geog_linestring3D`: Generar una cadena lineal geográfica 3D aleatoria
- `random_geom_polygon`: Generar un polígono geométrico 2D sin agujeros aleatorio
- `random_geom_polygon3D`: Generar un polígono geométrico 3D sin agujeros aleatorio
- `random_geog_polygon`: Generar un polígono geográfico 2D sin agujeros aleatorio
- `random_geog_polygon3D`: Generar un polígono geográfico 3D sin agujeros aleatorio
- `random_geom_multipoint`: Generar un multipunto geométrico 2D aleatorio
- `random_geom_multipoint3D`: Generar un multipunto geométrico 3D aleatorio
- `random_geog_multipoint`: Generar un multipunto geográfico 2D aleatorio
- `random_geog_multipoint3D`: Generar un multipunto geográfico 3D aleatorio
- `random_geom_multilinestring`: Generar una multicadena lineal geométrica 2D aleatoria
- `random_geom_multilinestring3D`: Generar una multicadena lineal geométrica 3D aleatoria
- `random_geog_multilinestring`: Generar una multicadena lineal geográfica 2D aleatoria
- `random_geog_multilinestring3D`: Generar una multicadena lineal geográfica 3D aleatoria
- `random_geom_multipolygon`: Generar un multipolígono geométrico 2D sin agujeros aleatorio
- `random_geom_multipolygon3D`: Generar un multipolígono geométrico 3D sin agujeros aleatorio
- `random_geog_multipolygon`: Generar un multipolígono geográfico 2D sin agujeros aleatorio
- `random_geog_multipolygon3D`: Generar un multipolígono geográfico 3D sin agujeros aleatorio

B.3. Generador para tipos de tiempo y de cuadro delimitador MobilityDB

- `random_timestampset`: Generar un `timestampset` aleatorio
- `random_period`: Generar un `period` aleatorio
- `random_period_array`: Generar una matriz de valores `period` aleatorios
- `random_periodset`: Generar un `periodset` aleatorio
- `random_tbox`: Generar un `tbox` aleatorio
- `random_stbox`: Generar un `stbox` 2D aleatorio
- `random_stbox3D`: Generar un `stbox` 3D aleatorio
- `random_geodstbox`: Generar un `stbox` geodético 2D aleatorio
- `random_geodstbox3D`: Generar un `stbox` geodético 3D aleatorio

B.4. Generador para tipos temporales MobilityDB

- `random_tbool_inst`: Generar un booleano temporal de subtipo instante aleatorio
- `random_tint_inst`: Generar un entero temporal de subtipo instante aleatorio
- `random_tfloat_inst`: Generar un flotante temporal de subtipo instante aleatorio
- `random_ttext_inst`: Generar un texto temporal de subtipo instante aleatorio
- `random_tgeompoint_inst`: Generar un punto geométrico temporal 2D de subtipo instante aleatorio
- `random_tgeompoint3D_inst`: Generar un punto geométrico temporal 3D de subtipo instante aleatorio
- `random_tgeogpoint_inst`: Generar un punto geográfico temporal 2D de subtipo instante aleatorio
- `random_tgeogpoint3D_inst`: Generar un punto geográfico temporal 3D de subtipo instante aleatorio
- `random_tbool_instset`: Generar un booleano temporal de subtipo conjunto de instantes aleatorio
- `random_tint_instset`: Generar un entero temporal de subtipo conjunto de instantes aleatorio
- `random_tfloat_instset`: Generar un flotante temporal de subtipo conjunto de instantes aleatorio
- `random_ttext_instset`: Generar un texto temporal de subtipo conjunto de instantes aleatorio
- `random_tgeompoint_instset`: Generar un punto temporal geométrico 2D de subtipo conjunto de instantes aleatorio
- `random_tgeompoint3D_instset`: Generar un punto geométrico temporal 3D de subtipo conjunto de instantes aleatorio
- `random_tgeogpoint_instset`: Generar un punto geográfico temporal 2D de subtipo conjunto de instantes aleatorio
- `random_tgeogpoint3D_instset`: Generar un punto geográfico temporal 3D de subtipo conjunto de instantes aleatorio
- `random_tbool_seq`: Generar un booleano temporal de subtipo secuencia aleatorio
- `random_tint_seq`: Generar un entero temporal de subtipo secuencia aleatorio
- `random_tfloat_seq`: Generar un flotante temporal de subtipo secuencia aleatorio
- `random_ttext_seq`: Generar un texto temporal de subtipo secuencia aleatorio
- `random_tgeompoint_seq`: Generar un punto geométrico temporal 2D de subtipo secuencia aleatorio
- `random_tgeompoint3D_seq`: Generar un punto geométrico temporal 3D de subtipo secuencia aleatorio

- `random_tgeogpoint_seq`: Generar un punto geográfico temporal 2D de subtipo secuencia aleatorio
- `random_tgeogpoint3D_seq`: Generar un punto geográfico temporal 3D de subtipo secuencia aleatorio
- `random_tbool_seqset`: Generar un booleano temporal de subtipo conjunto de secuencias aleatorio
- `random_tint_seqset`: Generar un entero temporal de subtipo conjunto de secuencias aleatorio
- `random_tfloat_seqset`: Generar un flotante temporal de subtipo conjunto de secuencias aleatorio
- `random_ttext_seqset`: Generar un texto temporal de subtipo conjunto de secuencias aleatorio
- `random_tgeompoint_seqset`: Generar un punto geométrico temporal 2D de subtipo conjunto de secuencias aleatorio
- `random_tgeompoint3D_seqset`: Generar un punto geométrico temporal 3D de subtipo conjunto de secuencias aleatorio
- `random_tgeogpoint_seqset`: Generar un punto geográfico temporal 2D de subtipo conjunto de secuencias aleatorio
- `random_tgeogpoint3D_seqset`: Generar un punto geográfico temporal 3D de subtipo conjunto de secuencias aleatorio

B.5. Generación de tablas con valores aleatorios

Los archivos `/datagen/general/create_test_tables_temporal.sql` and `/datagen/point/create_test_tabl` dan ejemplos de utilización de las funciones que generan valores aleatorios listadas arriba. Por ejemplo, el primer archivo define la función siguiente.

```
CREATE OR REPLACE FUNCTION create_test_tables_temporal(size int DEFAULT 100)
RETURNS text AS $$
DECLARE
  perc int;
BEGIN
perc := size * 0.01;
IF perc < 1 THEN perc := 1; END IF;

-- ... Table generation ...

RETURN 'The End';
END;
$$ LANGUAGE 'plpgsql';
```

La función tiene un parámetro `size` que define el número de filas en las tablas. Si no se proporciona, crea por defecto tablas de 100 filas. La función define una variable `perc` que calcula el 1 % del tamaño de las tablas. Este parámetro se utiliza, por ejemplo, para generar tablas con un 1 % de valores nulos. A continuación ilustramos algunos de los comandos que generan tablas.

La creación de una tabla `tbl_float` que contiene valores aleatorios `float` en el rango [0,100] con 1 % de valores nulos se da a continuación.

```
CREATE TABLE tbl_float AS
/* Add perc NULL valores */
SELECT k, NULL AS f
FROM generate_series(1, perc) AS k UNION
SELECT k, random_float(0, 100)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_tbox` que contiene valores aleatorios `tbox` donde los límites de los valores están en el rango [0,100] y los límites de las marcas de tiempo están en el rango [2001-01-01, 2001-12-31] se da a continuación.

```
CREATE TABLE tbl_tbox AS
/* Add perc NULL valores */
SELECT k, NULL AS b
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tbox(0, 100, '2001-01-01', '2001-12-31', 10, 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_floatrange` que contiene valores aleatorios `floatrange` donde los límites de los valores están en el rango [0,100] y la máxima diferencia entre los límites inferiores y superiores es 10 se da a continuación.

```
CREATE TABLE tbl_floatrango AS
/* Add perc NULL valores */
SELECT k, NULL AS f
FROM generate_series(1, perc) AS k UNION
SELECT k, random_floatrango(0, 100, 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_timestampset` que contiene valores aleatorios `timestampset` que tienen entre 5 y 10 marcas de tiempo donde las marcas de tiempo están en el rango [2001-01-01, 2001-12-31] y el máximo intervalo entre marcas de tiempo consecutivas es 10 minutos se da a continuación.

```
CREATE TABLE tbl_timestampset AS
/* Add perc NULL valores */
SELECT k, NULL AS ts
FROM generate_series(1, perc) AS k UNION
SELECT k, random_timestampset('2001-01-01', '2001-12-31', 10, 5, 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_period` que contiene valores aleatorios `period` donde las marcas de tiempo están en el rango [2001-01-01, 2001-12-31] y la máxima diferencia entre los límites inferiores y superiores es 10 minutos se da a continuación.

```
CREATE TABLE tbl_period AS
/* Add perc NULL valores */
SELECT k, NULL AS p
FROM generate_series(1, perc) AS k UNION
SELECT k, random_period('2001-01-01', '2001-12-31', 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_geom_point` que contiene valores aleatorios `geometry` 2D point valores, donde las coordenadas x e y están en el rango [0, 100] y en SRID 3812 se da a continuación.

```
CREATE TABLE tbl_geom_point AS
SELECT 1 AS k, geometry 'SRID=3812;point empty' AS g UNION
SELECT k, random_geom_point(0, 100, 0, 100, 3812)
FROM generate_series(2, size) k;
```

Observe que la tabla contiene un valor de punto vacío. Si no se proporciona el SRID, se establece de forma predeterminada en 0.

La creación de una tabla `tbl_geog_point3D` que contiene valores aleatorios `geography` 3D point valores, donde las coordenadas x, y, y z están, respectivament, en los rangos [-10, 32], [35, 72] y [0, 1000] y en SRID 7844 se da a continuación.

```
CREATE TABLE tbl_geog_point3D AS
SELECT 1 AS k, geography 'SRID=7844;pointZ empty' AS g UNION
SELECT k, random_geog_point3D(-10, 32, 35, 72, 0, 1000, 7844)
FROM generate_series(2, size) k;
```

Tenga en cuenta que los valores de latitud y longitud se eligen para cubrir aproximadamente la Europa continental. Si no se proporciona el SRID, se establece de forma predeterminada en 4326.

La creación de una tabla `tbl_geom_linestring` que contiene valores aleatorios `geometry` 2D linestring valores que tienen entre 5 y 10 vértices, donde las coordenadas x e y están en el rango [0, 100] y en SRID 3812 y la máxima diferencia entre valores de coordenadas consecutivos es 10 unidades en el SRID subyacente se da a continuación.

```
CREATE TABLE tbl_geom_linestring AS
SELECT 1 AS k, geometry 'linestring empty' AS g UNION
SELECT k, random_geom_linestring(0, 100, 0, 100, 10, 5, 10, 3812)
FROM generate_series(2, size) k;
```

La creación de una tabla `tbl_geom_linestring` que contiene valores aleatorios `geometry` 2D linestring valores que tienen entre 5 y 10 vértices, donde las coordenadas x e y están en el rango [0, 100] y la máxima diferencia entre valores de coordenadas consecutivos es 10 unidades en el SRID subyacente se da a continuación.

```
CREATE TABLE tbl_geom_linestring AS
SELECT 1 AS k, geometry 'linestring empty' AS g UNION
SELECT k, random_geom_linestring(0, 100, 0, 100, 10, 5, 10)
FROM generate_series(2, size) k;
```

La creación de una tabla `tbl_geom_polygon3D` que contiene valores aleatorios `geometry` 3D polygon valores sin agujeros, que tienen entre 5 y 10 vértices, donde las coordenadas x, y, y z están en el rango [0, 100] y la máxima diferencia entre valores de coordenadas consecutivos es 10 unidades en el SRID subyacente se da a continuación.

```
CREATE TABLE tbl_geom_polygon3D AS
SELECT 1 AS k, geometry 'polygon Z empty' AS g UNION
SELECT k, random_geom_polygon3D(0, 100, 0, 100, 0, 100, 10, 5, 10)
FROM generate_series(2, size) k;
```

La creación de una tabla `tbl_geom_multipoint` que contiene valores aleatorios `geometry` 2D multipunto valores que tienen entre 5 y 10 points, donde las coordenadas x e y están en el rango [0, 100] y la máxima diferencia entre valores de coordenadas consecutivos es 10 unidades en el SRID subyacente se da a continuación.

```
CREATE TABLE tbl_geom_multipoint AS
SELECT 1 AS k, geometry 'multipunto empty' AS g UNION
SELECT k, random_geom_multipoint(0, 100, 0, 100, 10, 5, 10)
FROM generate_series(2, size) k;
```

La creación de una tabla `tbl_geog_multilinestring` que contiene valores aleatorios `geography` 2D multilinestring valores que tienen entre 5 y 10 linestrings, cada una teniendo entre 5 y 10 vértices, donde las coordenadas x e y estan, respectivamente, en el rangos [-10, 32] y [35, 72] y la máxima diferencia entre valores de coordenadas consecutivos es 10 se da a continuación.

```
CREATE TABLE tbl_geog_multilinestring AS
SELECT 1 AS k, geography 'multilinestring empty' AS g UNION
SELECT k, random_geog_multilinestring(-10, 32, 35, 72, 10, 5, 10, 5, 10)
FROM generate_series(2, size) k;
```

La creación de una tabla `tbl_geometry3D` que contiene valores aleatorios `geometry` 3D de varios tipos se da a continuación. Esta función requiere que las tablas para los diversos tipos de geometría se hayan creado previamente.

```
CREATE TABLE tbl_geometry3D (
  k serial PRIMARY KEY,
  g geometry);
INSERT INTO tbl_geometry3D(g)
(SELECT g FROM tbl_geom_point3D ORDER BY k LIMIT (size * 0.1)) UNION ALL
(SELECT g FROM tbl_geom_linestring3D ORDER BY k LIMIT (size * 0.1)) UNION ALL
(SELECT g FROM tbl_geom_polygon3D ORDER BY k LIMIT (size * 0.2)) UNION ALL
(SELECT g FROM tbl_geom_multipoint3D ORDER BY k LIMIT (size * 0.2)) UNION ALL
(SELECT g FROM tbl_geom_multilinestring3D ORDER BY k LIMIT (size * 0.2)) UNION ALL
(SELECT g FROM tbl_geom_multipolygon3D ORDER BY k LIMIT (size * 0.2));
```

La creación de una tabla `tbl_tbool_inst` que contiene valores aleatorios `tbool` valores de subtipo instante donde las marcas de tiempo están en el rango [2001-01-01, 2001-12-31] se da a continuación.

```
CREATE TABLE tbl_tbool_inst AS
/* Add perc NULL valores */
SELECT k, NULL AS inst
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tbool_inst('2001-01-01', '2001-12-31')
FROM generate_series(perc+1, size) k;
/* Add perc duplicates */
UPDATE tbl_tbool_inst t1
SET inst = (SELECT inst FROM tbl_tbool_inst t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
/* Add perc rows con the same timestamp */
UPDATE tbl_tbool_inst t1
SET inst = (SELECT tboolinst(random_bool(), getTimestamp(inst))
  FROM tbl_tbool_inst t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
```

Como se puede ver arriba, la tabla tiene un porcentaje de valores nulos, de duplicados y de filas con la misma marca de tiempo.

La creación de una tabla `tbl_tint_instset` que contiene valores aleatorios `tint` valores de subtipo conjunto de instantes que tienen entre 5 y 10 marcas de tiempo donde the integer valores están en el rango [0, 100], las marcas de tiempo están en el rango [2001-01-01, 2001-12-31], la máxima diferencia entre dos valores consecutivos es 10 y el máximo intervalo entre dos instantes consecutivos es 10 minutos se da a continuación.

```
CREATE TABLE tbl_tint_instset AS
/* Add perc NULL valores */
SELECT k, NULL AS ti
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tint_instset(0, 100, '2001-01-01', '2001-12-31', 10, 10, 5, 10) AS ti
FROM generate_series(perc+1, size) k;
/* Add perc duplicates */
UPDATE tbl_tint_instset t1
SET ti = (SELECT ti FROM tbl_tint_instset t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
/* Add perc rows con the same timestamp */
UPDATE tbl_tint_instset t1
SET ti = (SELECT ti + random_int(1, 2) FROM tbl_tint_instset t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
/* Add perc rows that meet */
UPDATE tbl_tint_instset t1
SET ti = (SELECT shift(ti, endTimestamp(ti)-startTimestamp(ti))
```

```
  FROM tbl_tint_instset t2 WHERE t2.k = t1.k+perc)
WHERE t1.k in (SELECT i FROM generate_series(1 + 6*perc, 7*perc) i);
/* Add perc rows that overlap */
UPDATE tbl_tint_instset t1
SET ti = (SELECT shift(ti, date_trunc('minute',(endTimestamp(ti)-startTimestamp(ti))/2))
  FROM tbl_tint_instset t2 WHERE t2.k = t1.k+2)
WHERE t1.k in (SELECT i FROM generate_series(1 + 8*perc, 9*perc) i);
```

Como se puede ver arriba, la tabla tiene un porcentaje de valores nulos, de duplicados, de filas con la misma marca de tiempo, de filas que se encuentran y de filas que se superponen.

La creación de una tabla `tbl_tfloat_seq` que contiene valores aleatorios `tfloat` valores de subtipo secuencia que tienen entre 5 y 10 marcas de tiempo donde the float valores están en el rango [0, 100], las marcas de tiempo están en el rango [2001-01-01, 2001-12-31], la máxima diferencia entre dos valores consecutivos es 10 y el máximo intervalo entre dos instantes consecutivos es 10 minutos se da a continuación.

```
CREATE TABLE tbl_tfloat_seq AS
/* Add perc NULL valores */
SELECT k, NULL AS seq
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tfloat_seq(0, 100, '2001-01-01', '2001-12-31', 10, 10, 5, 10) AS seq
FROM generate_series(perc+1, size) k;
/* Add perc duplicates */
UPDATE tbl_tfloat_seq t1
SET seq = (SELECT seq FROM tbl_tfloat_seq t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
/* Add perc tuples with the same timestamp */
UPDATE tbl_tfloat_seq t1
SET seq = (SELECT seq + random_int(1, 2) FROM tbl_tfloat_seq t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
/* Add perc tuples that meet */
UPDATE tbl_tfloat_seq t1
SET seq = (SELECT shift(seq, timespan(seq)) FROM tbl_tfloat_seq t2 WHERE t2.k = t1.k+perc)
WHERE t1.k in (SELECT i FROM generate_series(1 + 6*perc, 7*perc) i);
/* Add perc tuples that overlap */
UPDATE tbl_tfloat_seq t1
SET seq = (SELECT shift(seq, date_trunc('minute',timespan(seq)/2))
  FROM tbl_tfloat_seq t2 WHERE t2.k = t1.k+perc)
WHERE t1.k in (SELECT i FROM generate_series(1 + 8*perc, 9*perc) i);
```

La creación de una tabla `tbl_ttext_seqset` que contiene valores aleatorios `ttext` de subtipo conjunto de secuencias que tienen entre 5 y 10 sequences, cada una teniendo entre 5 y 10 marcas de tiempo, donde los valores de texto tienen máximo 10 caracteres, las marcas de tiempo están en el rango [2001-01-01, 2001-12-31] y el máximo intervalo entre dos instantes consecutivos es 10 minutos se da a continuación.

```
CREATE TABLE tbl_ttext_seqset AS
/* Add perc NULL valores */
SELECT k, NULL AS ts
FROM generate_series(1, perc) AS k UNION
SELECT k, random_ttext_seqset('2001-01-01', '2001-12-31', 10, 10, 5, 10, 5, 10) AS ts
FROM generate_series(perc+1, size) AS k;
/* Add perc duplicates */
UPDATE tbl_ttext_seqset t1
SET ts = (SELECT ts FROM tbl_ttext_seqset t2 WHERE t2.k = t1.k+perc)
WHERE k in (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
/* Add perc tuples con the same timestamp */
UPDATE tbl_ttext_seqset t1
SET ts = (SELECT ts || text 'A' FROM tbl_ttext_seqset t2 WHERE t2.k = t1.k+perc)
```

```
WHERE k in (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
/* Add perc tuples that meet */
UPDATE tbl_ttext_seqset t1
SET ts = (SELECT shift(ts, timespan(ts)) FROM tbl_ttext_seqset t2 WHERE t2.k = t1.k+perc)
WHERE t1.k in (SELECT i FROM generate_series(1 + 6*perc, 7*perc) i);
/* Add perc tuples that overlap */
UPDATE tbl_ttext_seqset t1
SET ts = (SELECT shift(ts, date_trunc('minute', timespan(ts)/2))
  FROM tbl_ttext_seqset t2 WHERE t2.k = t1.k+perc)
WHERE t1.k in (SELECT i FROM generate_series(1 + 8*perc, 9*perc) i);
```

La creación de una tabla `tbl_tgeompoint_instset` que contiene valores aleatorios `tgeompoint` 2D valores de subtipo conjunto de instantes que tienen entre 5 y 10 instantes, donde the x e y coordenadas están en el rango [0, 100] y en SRID 3812, las marcas de tiempo están en el rango [2001-01-01, 2001-12-31], la máxima diferencia entre coordenadas successivas máximo 10 unidades en el SRID subyacente y el máximo intervalo entre dos instantes consecutivos es 10 minutos se da a continuación.

```
CREATE TABLE tbl_tgeompoint_instset AS
SELECT k, random_tgeompoint_instset(0, 100, 0, 100, '2001-01-01', '2001-12-31',
  10, 10, 5, 10, 3812) AS ti
FROM generate_series(1, size) k;
/* Add perc duplicates */
UPDATE tbl_tgeompoint_instset t1
SET ti = (SELECT ti FROM tbl_tgeompoint_instset t2 WHERE t2.k = t1.k+perc)
WHERE k IN (SELECT i FROM generate_series(1, perc) i);
/* Add perc tuples con the same timestamp */
UPDATE tbl_tgeompoint_instset t1
SET ti = (SELECT setPrecision(ti,6) FROM tbl_tgeompoint_instset t2 WHERE t2.k = t1.k+perc)
WHERE k IN (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
/* Add perc tuples that meet */
UPDATE tbl_tgeompoint_instset t1
SET ti = (SELECT shift(ti, endTimestamp(ti)-startTimestamp(ti))
  FROM tbl_tgeompoint_instset t2 WHERE t2.k = t1.k+perc)
WHERE t1.k IN (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
/* Add perc tuples that overlap */
UPDATE tbl_tgeompoint_instset t1
SET ti = (SELECT shift(ti, date_trunc('minute',(endTimestamp(ti)-startTimestamp(ti))/2))
  FROM tbl_tgeompoint_instset t2 WHERE t2.k = t1.k+2)
WHERE t1.k IN (SELECT i FROM generate_series(1 + 6*perc, 7*perc) i);
```

Finalmente, la creación de una tabla `tbl_tgeompoint3D_seqset` que contiene valores aleatorios `tgeompoint` 3D valores de subtipo conjunto de secuencias que tienen entre 5 y 10 sequences, cada una teniendo entre 5 y 10 marcas de tiempo, donde las coordenadas x, y, y z están en el rango [0, 100] y en SRID 3812, las marcas de tiempo están en el rango [2001-01-01, 2001-12-31], la máxima diferencia entre coordenadas successivas máximo 10 unidades en el SRID subyacente y el máximo intervalo entre dos instantes consecutivos es 10 minutos se da a continuación.

```
DROP TABLE IF EXISTS tbl_tgeompoint3D_seqset;
CREATE TABLE tbl_tgeompoint3D_seqset AS
SELECT k, random_tgeompoint3D_seqset(0, 100, 0, 100, 0, 100, '2001-01-01', '2001-12-31',
  10, 10, 5, 10, 5, 10, 3812) AS ts
FROM generate_series(1, size) AS k;
/* Add perc duplicates */
UPDATE tbl_tgeompoint3D_seqset t1
SET ts = (SELECT ts FROM tbl_tgeompoint3D_seqset t2 WHERE t2.k = t1.k+perc)
WHERE k IN (SELECT i FROM generate_series(1, perc) i);
/* Add perc tuples con the same timestamp */
UPDATE tbl_tgeompoint3D_seqset t1
SET ts = (SELECT setPrecision(ts,3) FROM tbl_tgeompoint3D_seqset t2 WHERE t2.k = t1.k+perc)
WHERE k IN (SELECT i FROM generate_series(1 + 2*perc, 3*perc) i);
```

```
/* Add perc tuples that meet */
UPDATE tbl_tgeompoint3D_seqset t1
SET ts = (SELECT shift(ts, timespan(ts)) FROM tbl_tgeompoint3D_seqset t2 WHERE t2.k = t1.k+ ↩
    perc)
WHERE t1.k IN (SELECT i FROM generate_series(1 + 4*perc, 5*perc) i);
/* Add perc tuples that overlap */
UPDATE tbl_tgeompoint3D_seqset t1
SET ts = (SELECT shift(ts, date_trunc('minute', timespan(ts)/2))
  FROM tbl_tgeompoint3D_seqset t2 WHERE t2.k = t1.k+perc)
WHERE t1.k IN (SELECT i FROM generate_series(1 + 6*perc, 7*perc) i);
```

B.6. Generador para tipos de red temporales

- `random_fraction`: Generar una fracción aleatoria en el rango [0,1]
- `random_npoint`: Genera un punto de red aleatorio
- `random_nsegment`: Genera un segmento de red aleatorio
- `random_tnpoint_inst`: Generar un punto de red temporal de subtipo instant aleatorio
- `random_tnpoint_instset`: Generar un punto de red temporal de subtipo conjunto de instantes aleatorio
- `random_tnpoint_seq`: Generar un punto de red temporal de subtipo secuencia aleatorio
- `random_tnpoint_seqset`: Generar un punto de red temporal de subtipo conjunto de secuencias aleatorio

El archivo `/datagen/npoint/create_test_tables_tnpoint.sql` da ejemplos de utilización de las funciones que generan valores aleatorios listadas arriba.

Capítulo 7

Índice alfabético