MobilityDB 1.1 Manual de usuario

COLABORADORES			
	TÍTULO : MobilityDB 1.1 Manual de usuario		
ACCIÓN	*NOMBRE*	*FECHA*	*FIRMA*
ESCRITO POR	Esteban Zimányi	5 de abril de 2024	

HISTORIAL DE REVISIONES			
NÚMERO	FECHA	MODIFICACIONES	NOMBRE

Índice general

1. Introducción **1**

1.1. Comité directivo del proyecto . . . 1

1.2. Otros colaboradores del código . . . 2

1.3. Patrocinadores . . . 2

1.4. Licencias . . . 2

1.5. Instalación a partir de las fuentes . . . 2

1.5.1. Versión corta . . . 2

1.5.2. Obtener las fuentes . . . 3

1.5.3. Habilitación de la base de datos . . . 4

1.5.4. Dependencias . . . 4

1.5.5. Configuración . . . 5

1.5.6. Construir e instalar . . . 5

1.5.7. Pruebas . . . 5

1.5.8. Documentación . . . 6

1.6. Instalación a partir de binarios . . . 6

1.6.1. Distribuciones de Linux basadas en Debian . . . 6

1.6.2. Windows . . . 6

1.7. Soporte . . . 6

1.7.1. Reporte de problemas . . . 7

1.7.2. Listas de correo . . . 7

1.8. Migración de la versión 1.0 a la versión 1.1 . . . 7

2. Tipos de conjunto y de rango **9**

2.1. Entrada y salida . . . 10

2.2. Constructores . . . 13

2.3. Conversión de tipos . . . 13

2.4. Accesores . . . 15

2.5. Transformaciones . . . 18

2.6. Sistema de referencia espacial . . . 20

2.7. Operaciones de conjuntos . . . 21

2.8. Operaciones de cuadro delimitador . . . 22
2.8.1. Operaciones topológicas . . . 22
2.8.2. Operaciones de posición . . . 23
2.9. Operaciones de distancia . . . 24
2.10. Comparaciones . . . 25
2.11. Agregaciones . . . 26
2.12. Indexación . . . 27

3. Tipos de cuadro delimitador **28**
3.1. Entrada y salida . . . 28
3.2. Constructores . . . 30
3.3. Conversiones de tipo . . . 31
3.4. Accesores . . . 32
3.5. Transformaciones . . . 34
3.6. Sistema de referencia espacial . . . 35
3.7. Funciones de división . . . 36
3.8. Operaciones de conjuntos . . . 36
3.9. Operaciones de cuadro delimitador . . . 37
3.9.1. Operaciones topológicas . . . 37
3.9.2. Operaciones de posición . . . 38
3.10. Comparaciones . . . 40
3.11. Aggregaciones . . . 41
3.12. Indexación . . . 41

4. Tipos temporales **43**
4.1. Introduction . . . 43
4.2. Ejemplos de tipos temporales . . . 45
4.3. Validez de los tipos temporales . . . 47

5. Tipos temporales: Operaciones genéricas (Parte 1) **48**
5.1. Introducción . . . 48
5.2. Entrada y salida . . . 50
5.3. Constructores . . . 54
5.4. Conversión de tipos . . . 56
5.5. Accesores . . . 57
5.6. Transformaciones . . . 62

6. Tipos temporales: Operaciones genéricas (Parte 2) 66
6.1. Modificaciones . . . 66
6.2. Restricciones . . . 71
6.3. Comparaciones . . . 75
6.3.1. Comparaciones tradicionales . . . 75
6.3.2. Comparaciones alguna vez y siempre . . . 76
6.3.3. Comparaciones temporal . . . 78
6.4. Operadores de cuadro delimitador . . . 79
6.5. Funciones de utilidad . . . 79

7. Tipos temporales: Operaciones alfanuméricas 80
7.1. Operaciones booleanos . . . 80
7.2. Operaciones matemáticas . . . 80
7.3. Operaciones de texto . . . 82

8. Tipos temporales: Operaciones espaciales 83
8.1. Entrada y salida . . . 83
8.2. Sistema de referencia espacial . . . 85
8.3. Operaciones de cuadro delimitador . . . 86
8.4. Accessores . . . 86
8.5. Transformaciones . . . 89
8.6. Operaciones de distancia . . . 91
8.7. Relaciones espaciales . . . 93
8.7.1. Relaciones espaciales alguna vez o siempre . . . 94
8.7.2. Relaciones espaciales temporales . . . 95

9. Tipos temporales: Operaciones de análisis 97
9.1. Simplificación . . . 97
9.2. Reducción . . . 98
9.3. Similaridad . . . 101
9.4. Mosaicos multidimensionales . . . 102
9.4.1. Operaciones de intervalos . . . 103
9.4.2. Operaciones de mosaicos . . . 104
9.4.3. Operaciones de fragmentación . . . 106

10. Tipos temporales: Agregación e indexación 109
10.1. Agregación . . . 109
10.2. Indexación . . . 111
10.3. Estadísticas y selectividad . . . 113
10.3.1. Colecta de estadísticas . . . 113
10.3.2. Estimación de la selectividad . . . 114

11. Puntos de red temporales **115**
11.1. Tipos de red estáticos . . . 115
11.1.1. Constructores . . . 116
11.1.2. Transformaciones . . . 117
11.1.3. Accesores . . . 117
11.1.4. Operaciones espaciales . . . 117
11.1.5. Comparaciones . . . 118
11.2. Puntos de red temporales . . . 119
11.3. Validez de los puntos de red temporal . . . 120
11.4. Constructores . . . 120
11.5. Conversión de tipos . . . 121
11.6. Operaciones para los tipos de puntos de red . . . 122
11.7. Agregacions . . . 127
11.8. Indexación . . . 128

A. Referencia de MobilityDB **129**
A.1. Tipos de MobilityDB . . . 129
A.2. Tipos de conjunto y de rango . . . 129
A.2.1. Entrada y salida . . . 129
A.2.2. Constructores . . . 130
A.2.3. Conversión de tipos . . . 130
A.2.4. Accesores . . . 130
A.2.5. Transformaciones . . . 130
A.2.6. Comparaciones . . . 131
A.2.7. Sistema de referencia espacial . . . 131
A.2.8. Operaciones de conjuntos . . . 131
A.2.9. Operaciones de cuadro delimitador . . . 131
A.2.9.1. Operaciones topológicas . . . 131
A.2.9.2. Operaciones de posición . . . 131
A.2.10. Operaciones de distancia . . . 131
A.2.11. Agregaciones . . . 132
A.3. Tipos de cuadro delimitadores . . . 132
A.3.1. Entrada y salida . . . 132
A.3.2. Constructores . . . 132
A.3.3. Conversión de tipos . . . 132
A.3.4. Accesores . . . 132
A.3.5. Transformaciones . . . 133
A.3.6. Sistema de referencia espacial . . . 133
A.3.7. Funciones de división . . . 133

A.3.8. Operaciones de conjuntos . . . 133
A.3.9. Operaciones de cuadro delimitador . . . 133
A.3.9.1. Operaciones topológicas . . . 133
A.3.9.2. Operaciones de posición . . . 134
A.3.10. Comparaciones . . . 134
A.3.11. Agregaciones . . . 134
A.4. Tipos temporales: Operaciones genéricas . . . 134
A.4.1. Entrada y salida . . . 134
A.4.2. Constructores . . . 134
A.4.3. Conversión de tipos . . . 135
A.4.4. Accesores . . . 135
A.4.5. Transformaciones . . . 136
A.4.6. Modificaciones . . . 136
A.4.7. Restricciones . . . 136
A.4.8. Comparaciones . . . 136
A.4.8.1. Comparaciones tradicionales . . . 136
A.4.8.2. Comparaciones alguna vez y siempre . . . 137
A.4.8.3. Comparaciones temporales . . . 137
A.4.9. Funciones de utilidad . . . 137
A.5. Tipos temporales: Operaciones alfanuméricas . . . 137
A.5.1. Operaciones matemáticas . . . 137
A.5.2. Operaciones booleanas . . . 138
A.5.3. Operaciones de texto . . . 138
A.6. Tipos temporales: Operaciones espaciales . . . 138
A.6.1. Entrada y salida . . . 138
A.6.2. Sistema de referencia espacial . . . 138
A.6.3. Operaciones de cuadro delimitador . . . 139
A.6.4. Accesores . . . 139
A.6.5. Transformaciones . . . 139
A.6.6. Operaciones de distancia . . . 139
A.6.7. Relaciones espaciales . . . 139
A.6.7.1. Relaciones espaciales alguna vez o siempre . . . 139
A.6.7.2. Relaciones espaciales temporales . . . 140
A.7. Tipos temporales: Operaciones de análisis . . . 140
A.7.1. Simplicación . . . 140
A.7.2. Reducción . . . 140
A.7.3. Similaridad . . . 140
A.7.4. Mosaicos multidimensionales . . . 140
A.7.4.1. Operaciones de intervalos . . . 140

A.7.4.2. Operaciones de malla . . . 141
A.7.4.3. Operaciones de fragmentación . . . 141
A.8. Tipos temporales: Agregaciones . . . 141
A.9. Operaciones para puntos de red temporales . . . 141
A.9.1. Tipos de red estáticos . . . 141
A.9.1.1. Constructores . . . 141
A.9.1.2. Accesores . . . 142
A.9.1.3. Transformaciones . . . 142
A.9.1.4. Operaciones espaciales . . . 142
A.9.1.5. Comparaciones . . . 142
A.9.2. Puntos de red temporales . . . 142
A.9.2.1. Constructores . . . 142
A.9.2.2. Conversión de tipos . . . 142
A.9.2.3. Operaciones . . . 143
A.9.2.4. Agregaciones . . . 143

B. Generador de datos sintéticos **144**
B.1. Generador para tipos PostgreSQL . . . 144
B.2. Generador para tipos PostGIS . . . 145
B.3. Generador para tipos de rango, de tiempo y de cuadro delimitador MobilityDB . . . 146
B.4. Generador para tipos temporales MobilityDB . . . 146
B.5. Generación de tablas con valores aleatorios . . . 147
B.6. Generador para tipos de red temporales . . . 152

12. Índice alfabético **154**

Índice de figuras

3.1. Grilla multiresolución sobre datos de Bruselas obtenidos mediante el generador BerlinMOD. Cada celda contiene como máximo 10.000 (izquierda) y 1.000 (derecha) instantes durante todo el período de simulación (cuatro días en este caso). A la izquierda, podemos ver la alta densidad de tráfico en el anillo alrededor de Bruselas, mientras que a la derecha podemos ver otros ejes principales de la ciudad. 37

6.1. Operación de inserción para valores temporales. 66

6.2. Operaciones de actualización y supresión para valores temporales. 67

6.3. Operaciones de modificación para tablas temporales en SQL. 68

8.1. Visualización de la velocidad de un objeto móvil usando una rampa de color en QGIS. 90

9.1. Diferencia entre la distancia espacial y la distancia sincronizada. 98

9.2. Muestreo de flotantes temporales con interpolación discreta, escalonada y lineal. 99

9.3. Cambio de precisión de números flotantes temporales con interpolación discreta, escalonada y lineal. 100

9.4. Mosaicos multidimensionales para números flotantes temporales. 103

Índice de cuadros

1.1. Variables para la documentación del usuario y del desarrollador 6

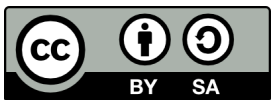
A.1. Instancias actuales de los tipos de plantilla en MobilityDB 129

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).

El manual de MobilityDB tiene una licencia Creative Commons Attribution-Share Alike 3.0 License 3. No dude en utilizar este material como desee, pero le pedimos que atribuya crédito al proyecto MobilityDB y, siempre que sea posible, un enlace a MobilityDB.



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 de la marchar utilizada por un automóvil en movimiento. 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 velocidad de un automóvil. 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`, `integer`, `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 proporciona los tipos de plantilla *set*, *span* y *span set* para representar, respectivamente, un conjunto de valores, un rango de valores y un conjunto de rangos de valores de tipos de base o tipos de tiempo. Ejemplos de valores de tipos de conjunto son `intset`, `floatset` y `tstzset`, donde el último representa un conjunto de valores `timestamptz`. Ejemplos de valores de tipos de rango son `intspan`, `floatspan` y `tstzspan`. Ejemplos de valores de tipos de conjuntos de rangos son `intspanset`, `floatspanset` y `tstzspanset`.

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: 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, copresidente del OGC Moving Feature Standard Working Group (MF-SWG)

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

- Vicky Vergara: integración con pgRouting, enlace con OSGeo
- Esteban Zimányi (chair): cofoundador del proyecto MobilityDB, coordinación general del proyecto, principal contribuidor del código de backend, BerlinMOD generator

1.2. Otros colaboradores del código

- Arthur Lesuisse
- Xinyiang Li
- Maxime Schoemans

1.3. Patrocinadores

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

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 a partir de las fuentes

1.5.1. Versión corta

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

```
git clone https://github.com/MobilityDB/MobilityDB
mkdir MobilityDB/build
cd MobilityDB/build
cmake  ..
make
```

```
sudo make install
```

Los comandos anteriores instalan la rama `master`. Si desea instalar otra rama, por ejemplo, `develop`, puede reemplazar el primer comando anterior de la siguiente manera:

```
git clone --branch develop https://github.com/MobilityDB/MobilityDB
```

También debe configurar lo siguiente en el archivo `postgresql.conf` según la versión de PostGIS que haya instalado (a continuación usamos PostGIS 3):

```
shared_preload_libraries = 'postgis-3'
max_locks_per_transaction = 128
```

Si no carga previamente la biblioteca PostGIS con la configuración anterior, no podrá cargar la biblioteca MobilityDB y obtendrá un mensaje de error como el siguiente:

```
ERROR:  could not load library "/usr/local/pgsql/lib/libMobilityDB-1.1.so":
  undefined symbol: ST_Distance
```

Puede encontrar la ubicación del archivo `postgresql.conf` de la manera siguiente.

```
$ which postgres
/usr/local/pgsql/bin/postgres
$ ls /usr/local/pgsql/data/postgresql.conf
/usr/local/pgsql/data/postgresql.conf
```

Como puede verse, los binarios de PostgreSQL están en el subdirectorio `bin` mientras que el archivo `postgresql.conf` está en el subdirectorio `data`.

Una vez que MobilityDB está instalado, debe habilitarse en cada base de datos en la que desee usarlo. En el siguiente ejemplo, usamos una base de datos llamada `mobility`.

```
createdb mobility
psql mobility -c "CREATE EXTENSION PostGIS"
psql mobility -c "CREATE EXTENSION MobilityDB"
```

Las dos extensiones PostGIS y MobilityDB también se pueden crear en un solo comando.

```
psql mobility -c "CREATE EXTENSION MobilityDB cascade"
```

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:

- Sistema de compilación multiplataforma `CMake`.
- Compilador C `gcc` o `clang`. Se pueden usar otros compiladores ANSI C, pero pueden causar problemas al compilar algunas dependencias.
- 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 12 o superior. PostgreSQL está disponible en http://www.postgresql.org.
- PostGIS versión 3 o superior. PostGIS está disponible en https://postgis.net/.
- Biblioteca científica GNU (GSL). GSL está disponible en https://www.gnu.org/software/gsl/. GSL se utiliza para los generadores de números aleatorios.

Nótese que PostGIS tiene sus propias dependencias, como Proj, GEOS, LibXML2 o JSON-C y estas bibliotecas también se utilizan en MobilityDB. Consulte http://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS para obtener una matriz de compatibilidad de PostGIS con PostgreSQL, GEOS y Proj.

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-14 postgresql-server-dev-14 postgresql-14-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

Nótese que la versión actual de MobilityDB solo se ha probado en sistemas Linux, MacOS y Windows. Puede funcionar en otros sistemas similares a Unix, pero no se ha probado. 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 '021_tbox'
```

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

```
ctest -R '022_*'
```

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. Instalación a partir de binarios

1.6.1. Distribuciones de Linux basadas en Debian

Se está desarrollando soporte para sistemas Linux basados en Debian, como Ubuntu y Arch Linux.

1.6.2. Windows

Desde la versión 3.3.3 de PostGIS, MobilityDB se distribuye en el PostGIS Bundle para Windows, que está disponible en application stackbuilder y en el sitio web de OSGeo. Para más information refiérase a la documentación PostGIS.

1.7. 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.

1.7.1. 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;
```

1.7.2. 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.

1.8. Migración de la versión 1.0 a la versión 1.1

MobilityDB 1.1 es una revisión importante con respecto a la versión inicial 1.0. El cambio más significativo en la version 1.1 fue extraer la funcionalidad central para la gestión de datos temporales y espaciotemporales de MobilityDB en la bibilioteca C Mobility Engine Open Source (MEOS). De esta forma, la misma funcionalidad que proporciona MobilityDB en un entorno de base de datos está disponible en un programa C para ser utilizada, por ejemplo, en un entorno de streaming. La biblioteca MEOS para la gestión de la movilidad proporciona una funcionalidad similar a la biblioteca C/C++ Geometry Engine Open Source (GEOS) para geometría computacional. Además, están disponibles contenedores para la biblioteca MEOS en otros lenguajes de programación, en particular para Python con PyMEOS. Además, contenedores para C#, Java y Javascript, están en desarrollando.

Fueron necesarios varios cambios con respecto a la versión 1.0 de MobilityDB para habilitar lo anterior. Uno importante fue la definición de nuevos tipos de datos `span` y `spanset`, que brindan una funcionalidad similar a los tipos de datos de PostgreSQL `range` y `multirange` pero también se puede usar en varios lenguajes de programación independientemente de PostgreSQL. Estos son *tipos de plantilla*, lo que significa que son contenedores de otros tipos, de forma similar a como las listas y matrices contienen valores de otros tipos. Además, también se agregó un nuevo tipo de datos de plantilla `set`. Por lo tanto, los tipos `timestampset`, `period` y `periodset` en la versión 1.0 se reemplazan por los tipos `tstzset`, `tstzspan` y `tstzspanset` en la versión 1.1. El nombre de las funciones constructoras para estos tipos se modificó en consecuencia.

Finalmente, se simplificó la API de MEOS y MobilityDB para mejorar la usabilidad. Detallamos a continuación los cambios más importantes en la API.

- Las funciones `atTimestamp`, `atTimestampSet`, `atPeriod`, and `atPeriodSet` fueron renombradas a `atTime`.
- Las funciones `minusTimestamp`, `minusTimestampSet`, `minusPeriod` y `minusPeriodSet` fueron renombradas a `minusTime`.

- Las funciones `atValue`, `atValues`, `atRange` y `atRanges` fueron renombradas a `atValues`.
- Las funciones `minusValue`, `minusValues`, `minusRange` y `minusRanges` fueron renombradas a `minusValues`.
- Las funciones `contains`, `disjoint`, `dwithin`, `intersects` y `touches` fueron renombradas, respectivamente, a `eContains`, `eDisjoint`, `eDwithin`, `eIntersects` y `eTouches`.

Capítulo 2

Tipos de conjunto y de rango

MobilityDB proporciona los tipos de *conjunto*, *rango* y *conjunto de rangos* para representar conjuntos de valores de otro tipo, que se denomina *tipo base*. Los tipos de conjunto son similares a los tipos de matrices de PostgreSQL restringidos a una dimensión, pero imponen la restricción de que los conjuntos no tienen duplicados. Los tipos de rango y conjunto de rangos en MobilityDB corresponden a los tipos de rango y multirango en PostgreSQL pero tienen restricciones adicionales. En particular, los tipos de rango en MobilityDB tienen una longitud fija y no permiten rangos vacíos ni límites infinitos. Si bien los tipos de rango en MobilityDB proporcionan una funcionalidad similar a los tipos de rango en PostgreSQL, los tipos de rango en MobilityDB permiten aumentar el rendimiento. En particular, se elimina la sobrecarga del procesamiento de tipos de longitud variable y, además, se pueden utilizar la aritmética de punteros y la búsqueda binaria.

Los tipos de base que se utilizan para construir tipos de conjunto, de rango y de conjunto de rangos son los tipos `integer`, `bigint`, `float`, `text`, `date`, and `timestamptz` (marca de tiempo con zona horaria) proporcionados por PostgreSQL, los tipos `geometry` y `geography` proporcionados por PostGIS, y el tipo `npoint` (*network point* o punto de red) proporcionado por MobilityDB (ver Capítulo 11). MobilityDB proporciona los siguientes tipos de conjunto y de rango:

- `set`: `intset`, `bigintset`, `floatset`, `textset`, `dateset`, `tstzset`, `geomset`, `geogset`, `npointset`.
- `span`: `intspan`, `bigintspan`, `floatspan`, `datespan`, `tstzspan`.
- `spanset`: `intspanset`, `bigintspanset`, `floatspanset`, `datespanset`, `tstzspanset`.

A continuación presentamos las funciones y operadores para tipos de conjunto 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 las funciones y los operadores, utilizamos la siguiente notación:

- `set` representa cualquier tipo de conjunto, como `intset` o `tstzset`.
- `span` representa cualquier tipo de rango, como `intspan` o `tstzspan`.
- `spanset` representa cualquier tipo de conjunto de rangos, como `intspanset` o `tstzspanset`.
- `spans` representa cualquier tipo de rango o conjunto de rangos, como `intspan` o `tstzspanset`.
- `base` representa cualquier tipo de base de un tipo de conjunto o de rango, como `integer` o `timestamptz`.
- `number` representa cualquier tipo de base de un tipo de rango numérico, como `integer` o `float`,
- `numset` representa cualquier tipo de conjunto numérico, como `intset` o `floatset`.
- `numspans` representa cualquier tipo de rango o conjunto de rangos numérico, como `intspan` o `floatspanset`.
- `numbers` representa cualquier tipo de conjunto o de rango numérico, como `integer`, `intset`, `intspan` o `intspanset`,
- `dates` representa cualquier tipo de tiempo con granularidad `date`, a saber `date`, `dateset`, `datespan` o `datespanset`,

- `numset` representa cualquier tipo de conjunto numérico, como `intset` o `floatset`.
- `numspans` representa cualquier tipo de rango o conjunto de rangos numérico, como `intspan` o `floatspanset`.
- `numbers` representa cualquier tipo de conjunto o de rango numérico, como `integer`, `intset`, `intspan` o `intspanset`,
- `dates` representa cualquier tipo de tiempo con granularidad `date`, a saber `date`, `dateset`, `datespan` o `datespanset`,
- `times` representa cualquier tipo de tiempo con granularidad `timestamptz`, a saber `timestamptz`, `tstzset`, `tstzspan` o `tstzspanset`,
- Un conjunto de tipos como `{set,base}` representa cualquiera de los tipos enumerados,
- `type[]` representa una matriz de `type`.

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

```
{set,spans} @> {base,set,spans} → boolean
```

Nótese que la firma anterior es una versión abreviada de la firma más precisa a continuación

```
set @> {base,set} → boolean
spans @> {base,spans} → boolean
```

ya que los conjuntos y los rangos no se pueden mezclar en las operaciones y, por lo tanto, por ejemplo, no se puede preguntar si un rango contiene un conjunto. A continuación, por concisión, utilizamos el estilo abreviado de las firmas anteriores. Además, la parte de tiempo de las marcas de tiempo se omite en la mayoría de los ejemplos. Recuerde que en ese caso PostgreSQL asume el tiempo `00:00:00`.

A continuación, dado que los tipos de rango y conjunto de rangos tienen funciones y operadores similares, cuando hablamos de tipos rango nos referimos a los tipos de rango y conjuntos de rangos, a menos que nos refiramos explícitamente a los tipos de rango *unitarios* y a los tipos de *conjunto* de rangos para distinguirlos. Además, cuando nos referimos a tipos de tiempo, nos referimos a uno de los siguientes tipos: `timestamptz`, `tstzset`, `tstzspan` o `tstzspanset`.

2.1. Entrada y salida

MobilityDB generaliza los formatos de entrada y salida Well-Known Text (WKT) y Well-Known Binary (WKB) del Open Geospatial Consortium (OGC) a para todos sus tipos. De esta forma, las aplicaciones pueden intercambiar datos entre ellas utilizando un formato de intercambio estandarizado. El formato WKT es legible por humanos, mientras que el formato WKB es más compacto y más eficiente que el formato WKT. El formato WKB se puede generar como una cadena binaria o como una cadena de caracteres codificada en ASCII hexadecimal.

Los tipos de conjunto representan un conjunto *ordenado* de valores *diferentes*. Un conjunto debe contener al menos un elemento. Ejemplos de valores de tipos de conjunto son como sigue:

```
SELECT tstzset '{2001-01-01 08:00:00, 2001-01-03 09:30:00}';
-- Conjunto unitario
SELECT textset '{"highway"}';
-- Conjunto erróneo: elementos desordenados
SELECT floatset '{3.5, 1.2}';
-- Conjunto erróneo: elementos duplicados
SELECT geomset '{"Point(1 1)", "Point(1 1)"}';
```

Nótese que los elementos de los conjuntos `textset`, `geomset`, `geogset` y `npointset` deben estar delimitados entre commillas dobles. Nótese también que las geometrías y las geografías utilizan el orden definido por PostGIS.

Un valor de un tipo de rango unitario tiene dos límites, el *límite inferior* y el *límite superior*, que son valores del *tipo de base* subyacente. Por ejemplo, un valor del tipo `tstzspan` tiene dos límites, que son valores de `timestamptz`. Los límites pueden ser inclusivos o exclusivos. Un límite inclusivo significa que el instante límite está incluido en el rango, mientras que un límite exclusivo significa que el instante límite no está incluido en el rango. En el formato textual de un valor de un rango, 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 un rango, el límite inferior debe ser menor o igual que el límite superior. Un valor de rango con límites iguales e inclusivos se llama *rango instantáneo* y corresponde a un valor del tipo de base. Ejemplos de valores de rango son como sigue:

```
SELECT intspan '[1, 3)';
SELECT floatspan '[1.5, 3.5]';
SELECT tstzspan '[2001-01-01 08:00:00, 2001-01-03 09:30:00)';
-- Rangos instantáneos
SELECT intspan '[1, 1]';
SELECT floatspan '[1.5, 1.5]';
SELECT tstzspan '[2001-01-01 08:00:00, 2001-01-01 08:00:00]';
-- Rango erróneo: límites inválidos
SELECT tstzspan '[2001-01-01 08:10:00, 2001-01-01 08:00:00]';
-- Rango erróneo: rango vacío
SELECT tstzspan '[2001-01-01 08:00:00, 2001-01-01 08:00:00)';
```

Los valores de `intspan`, `bigintspan` y `datespan` son convertidos en *forma normal* para que los valores equivalentes tengan representaciones idénticas. En la representación canónica de estos tipos, el límite inferior es inclusivo y el límite superior es exclusivo, como se muestra en los siguientes ejemplos:

```
SELECT intspan '[1, 1]';
-- [1, 2)
SELECT bigintspan '(1, 3]';
--[2, 4)
SELECT datespan '[2001-01-01, 2001-01-03]';
-- [2001-01-01, 2001-01-04)
```

Un valor de un tipo de conjunto de rangos representa un conjunto *ordenado* de valores de rango *disjuntos*. Un valor de conjunto de rangos debe contener al menos un elemento, en cuyo caso corresponde a un único valor de rango. Ejemplos de valores conjunto de rangos son los siguientes:

```
SELECT floatspanset '{[8.1, 8.5],[9.2, 9.4]}';
-- Singleton spanset
SELECT tstzspanset '{[2001-01-01 08:00:00, 2001-01-01 08:10:00]}';
-- Erroneous spanset: unordered elements
SELECT intspanset '{[3,4],[1,2]}';
-- Erroneous spanset: overlapping elements
SELECT tstzspanset '{[2001-01-01 08:00:00, 2001-01-01 08:10:00],
  [2001-01-01 08:05:00, 2001-01-01 08:15:00]}';
```

Los valores de los tipos conjunto de rangos son convertidos en *forma normal* de modo que los valores equivalentes tengan representaciones idénticas. Para ello, los valores de rango consecutivos que son adyacentes se fusionan cuando es posible. Ejemplos de transformación a forma normal son los siguientes:

```
SELECT intspanset '{[1,2],[3,4]}';
-- {[1, 5)}
SELECT floatspanset '{[1.5,2.5],(2.5,4.5]}';
-- {[1.5, 4.5]}
SELECT tstzspanset '{[2001-01-01 08:00:00, 2001-01-01 08:10:00),
  [2001-01-01 08:10:00, 2001-01-01 08:10:00], (2001-01-01 08:10:00, 2001-01-01 08:20:00]}';
-- {[2001-01-01 08:00:00+00,2001-01-01 08:20:00+00]}
```

Damos a continuación las funciones de entrada y salida de tipos de conjunto y de rango en formato textual (Well-Known Text o WKT) y binario (Well-Known Binary o WKB). El formato de salida predeterminado de todos los tipos de conjuntos y de rango es el formato de texto conocido. La función `asText` que se da a continuación permite determinar la salida de valores de punto flotante.

- Obtener la representación textual conocida (Well-Known Text o WKT)

  `asText({floatset,floatspans},maxdecdigits=15)` → `text`

El argumento `maxdecdigits` se puede utilizar para definir el número máximo de decimales para la salida de los valores de coma flotante (por defecto 15).

```
SELECT asText(floatset '{1.123456789,2.123456789}', 3);
-- {1.123, 2.123}
SELECT asText(floatspanset '{[1.55,2.55],[4,5]}',0);
-- {[2, 3], [4, 5]}
```

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

  `asBinary({set,spans},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(dateset '{2001-01-01, 2001-01-03}');
-- \x01050001020000006e01000070010000
SELECT asBinary(intspan '[1, 3)');
-- \x011300010100000003000000
SELECT asBinary(floatspanset '{[1, 2], [4, 5]}', 'XDR');
-- \x00000e00000002033ff0000000000004000000000000000003401000000000000040140000000000000
```

- Obtener la representación hexadecimal binaria conocida (HexWKB) en formato texto

  `asHexWKB({set,spans},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 la codificación de la máquina.

```
SELECT asHexWKB(dateset '{2001-01-01, 2001-01-03}');
-- 01050001020000006E01000070010000
SELECT asHexWKB(intspan '[1, 3)');
-- 011300010100000003000000
SELECT asHexWKB(floatspanset '{[1, 2], [4, 5]}', 'XDR');
-- 00000E00000002033FF0000000000004000000000000000003401000000000000040140000000000000
```

- Obtener a partir de la representación binaria conocida (WKB)

  `settypeFromBinary(bytea) → set`

  `spantypeFromBinary(bytea) → span`

  `spansettypeFromBinary(bytea) → spanset`

  Hay una función por tipo de conjunto o de rango, el nombre de la función tiene como prefijo el nombre del tipo.

```
SELECT datesetFromBinary('\x01050001020000006e01000070010000');
-- {2001-01-01, 2001-01-03}
SELECT intspanFromBinary('\x011300010100000003000000');
-- [1, 3)
SELECT floatspansetFromBinary(
  '\x00000e00000002033ff0000000000004000000000000000003401000000000000040140000000000000');
-- {[1, 2], [4, 5]}
```

- Obtener a partir de la representación hexadecimal binaria conocida (HexWKB)

  `settypeFromHexWKB(text) → set`

  `spantypeFromHexWKB(text) → span`

  `spansettypeFromHexWKB(text) → spanset`

  Hay una función por tipo de conjunto o de rango, el nombre de la función tiene como prefijo el nombre del tipo.

```
SELECT datesetFromHexWKB('01050001020000006E01000070010000');
-- {2001-01-01, 2001-01-03}
SELECT intspanFromHexWKB('011300010100000003000000');
-- [1, 3)
```

```
SELECT floatspansetFromHexWKB(
  '00000E00000002033FF0000000000000400000000000000003401000000000000040140000000000000');
-- {[1, 2], [4, 5]}
```

2.2. Constructores

La función constructora para los tipos de conjunto tiene un único argumento que es una matriz de valores del tipo base correspondiente. Los valores deben estar ordenados y no pueden tener nulos ni duplicados.

- Constructor para tipos de conjunto

  `set(base[]) → set`

```
SELECT set(ARRAY['highway', 'primary', 'secondary']);
-- {"highway", "primary", "secondary"}
SELECT set(ARRAY[timestamptz '2001-01-01 08:00:00', '2001-01-03 09:30:00']);
-- {2001-01-01 08:00:00+00, 2001-01-03 09:30:00+00}
```

Los tipos de rango unitarios tienen una función constructora que acepta cuatro argumentos, donde los dos últimos son opcionales. Los primeros dos argumentos especifican, respectivamente, el límite inferior y el superior, y los dos últimos argumentos son valores booleanos que indican, respectivamente, si los límites inferior y el superior son inclusivos o no. Se supone que los dos últimos argumentos son, respectivamente, verdadero y falso si no se especifican. Nótese que los rangos de enteros se transforman en *forma normal*, es decir, con límite inferior inclusivo y límite superior exclusivo.

- Constructor para tipos de rango

  `span(lower base,upper base,leftInc bool=true,rightInc bool=false) → span`

```
SELECT span(20.5, 25);
-- [20.5, 25)
SELECT span(20, 25, false, true);
-- [21, 26)
SELECT span(timestamptz '2001-01-01 08:00:00', '2001-01-03 09:30:00', false, true);
-- (2001-01-01 08:00:00, 2001-01-03 09:30:00]
```

La función constructora para los tipos de conjuntos de intervalos tiene un solo argumento que es una matriz de valores de intervalos del mismo subtipo.

- Constructor for conjunto de rangos

  `spanset(span[]) → spanset`

```
SELECT spanset(ARRAY[intspan '[10,12]', '[13,15]']);
-- {[10, 16)}
SELECT spanset(ARRAY[floatspan '[10.5,12.5]', '[13.5,15.5]']);
-- {[10.5, 12.5], [13.5, 15.5]}
SELECT spanset(ARRAY[tstzspan '[2001-01-01 08:00, 2001-01-01 08:10]',
  '[2001-01-01 08:20, 2001-01-01 08:40]']);
-- {[2001-01-01 08:00, 2001-01-01 08:10], [2001-01-01 08:20, 2001-01-01 08:40]};
```

2.3. Conversión de tipos

Los valores de los tipos de conjunto y de rango se pueden convertir entre sí o convertirse a tipos de rango de PostgreSQL y desde ellos mediante la función `CAST` o mediante la notación `::`.

- Convertir un valor de base en un valor de conjunto, rango o conjunto de rangos

  `base::{set,span,spanset}`

  `set(base) → set`

  `span(base) → span`

  `spanset(base) → spanset`

```
SELECT CAST(timestamptz '2001-01-01 08:00:00' AS tstzset);
-- {2001-01-01 08:00:00}
SELECT timestamptz '2001-01-01 08:00:00'::tstzspan;
-- [2001-01-01 08:00:00, 2001-01-01 08:00:00]
SELECT spanset(timestamptz '2001-01-01 08:00:00');
-- {[2001-01-01 08:00:00, 2001-01-01 08:00:00]}
```

- Convertir un valor de conjunto en un valor de conjunto de rangos

  `set::spanset`

  `spanset(set) → spanset`

```
SELECT spanset(tstzset '{2001-01-01 08:00:00, 2001-01-01 08:15:00,
  2001-01-01 08:25:00}');
/* {[2001-01-01 08:00:00, 2001-01-01 08:00:00],
   [2001-01-01 08:15:00, 2001-01-01 08:15:00],
   [2001-01-01 08:25:00, 2001-01-01 08:25:00]} */
```

- Convertir un valor de rango a un valor de conjunto de rangos

  `span::spanset`

  `spanset(span) → spanset`

```
SELECT floatspan '[1.5,2.5]'::floatspanset;
-- {[1.5, 2.5]}
SELECT tstzspan '[2001-01-01 08:00:00, 2001-01-01 08:30:00)'::tstzspanset;
-- {[2001-01-01 08:00:00, 2001-01-01 08:30:00)}
```

- Convertir un conjunto de valores o un conjunto de rangos a un rango, ignorando las posibles brechas de tiempo

  `{set,spanset}::span`

  `span({set,spanset}) → span`

```
SELECT span(dateset '{2001-01-01, 2001-01-03, 2001-01-05}');
-- [2001-01-01, 2001-01-06)
SELECT span(tstzspanset '{[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-04)}');
-- [2001-01-01, 2001-01-04)
```

- Convertir un valor de de rango en MobilityDB hacia y desde un valor de rango de PostgreSQL

  `span::range`

  `range::span`

  `range(span) → range`

  `span(range) → span`

  Nótese que los valores de rango en PostgreSQL aceptan rangos vacíos y rangos con límites infinitos, que no están permitidos como valores de rango en MobilityDB

```
SELECT intspan '[10, 20)'::int4range;
-- [10,20)
SELECT tstzspan '[2001-01-01 08:00:00, 2001-01-01 08:30:00)'::tstzrange;
-- ["2001-01-01 08:00:00","2001-01-01 08:30:00")
SELECT int4range '[10, 20)'::intspan;
-- [10,20)
```

```
SELECT int4range 'empty'::intspan;
-- ERROR:  Range cannot be empty
SELECT int4range '[10,)'::intspan;
-- ERROR:  Range bounds cannot be infinite
SELECT tstzrange '[2001-01-01 08:00:00, 2001-01-01 08:30:00)'::tstzspan;
-- [2001-01-01 08:00:00, 2001-01-01 08:30:00)
```

- Convertir un valor de conjunto de rangos de MobilityDB hacia y desde un valor de multirango de PostgreSQL

  `spanset::multirange`

  `multirange::spanset`

  `multirange(spanset) → multirange`

  `spanset(multirange) → spanset`

```
SELECT intspanset '{[1,2],[4,5]}'::int4multirange;
-- {[1,3),[4,6)}
SELECT tstzspanset '{[2001-01-01,2001-01-02],[2001-01-04,2001-01-05]}'::tstzmultirange;
-- {[2001-01-01,2001-01-02],[2001-01-04,2001-01-05]}
SELECT int4multirange '{[1,2],[4,5]}'::intspanset;
-- {[1, 3), [4, 6)}
SELECT tstzmultirange '{[2001-01-01,2001-01-02],[2001-01-04,2001-01-05]}'::tstzspanset;
-- {[2001-01-01, 2001-01-02], [2001-01-04, 2001-01-05]}
```

2.4. Accesores

- Obtener el tamaño de la memoria en bytes

  `memSize({set,spanset}) → integer`

```
SELECT memSize(tstzset '{2001-01-01, 2001-01-02, 2001-01-03}');
-- 48
SELECT memSize(tstzspanset '{[2001-01-01, 2001-01-02], [2001-01-03, 2001-01-04],
  [2001-01-05, 2001-01-06]}');
-- 112
```

- Obtener el límite inferior o superior

  `lower(spans) → base`

  `upper(spans) → base`

```
SELECT lower(tstzspan '[2001-01-01, 2001-01-05)');
-- 2001-01-01
SELECT lower(intspanset '{[1,2],[4,5]}');
--  1
```

```
SELECT lower(tstzspan '[2001-01-01, 2001-01-05)');
-- 2001-01-01
SELECT upper(intspanset '{[1,2],[4,5]}');
--  6
SELECT lower(tstzspan '[2001-01-01, 2001-01-05)');
-- 2001-01-01
SELECT lower(intspanset '{[1,2],[4,5]}');
--  1
```

```
SELECT upper(floatspan '[20.5, 25.3)');
-- 25.3
SELECT upper(tstzspan '[2001-01-01, 2001-01-05)');
-- 2001-01-05
```

- ¿Es el límite inferior or superior inclusivo?

  `lowerInc(spans) → boolean`

  `upperInc(spans) → boolean`

```
SELECT lowerInc(datespan '[2001-01-01, 2001-01-05)');
-- true
SELECT lowerInc(intspanset '{[1,2],[4,5]}');
-- true
```

```
SELECT lowerInc(tstzspan '[2001-01-01, 2001-01-05)');
-- true
SELECT upperInc(intspanset '{[1,2],[4,5]}');
-- false
SELECT upper(floatspan '[20.5, 25.3]');
-- true
SELECT upperInc(tstzspan '[2001-01-01, 2001-01-05)');
-- false
```

- Obtener el ancho del rango como un número de punto flotante

  `width(numspan) → float`

  `width(numspanset,boundspan=false) → float`

  Se puede establecer a verdadero un parámetro adicional para calcular el ancho del rango delimitador, ignorando así las posibles brechas de valors

```
SELECT width(floatspan '[1, 3)');
-- 2
SELECT width(intspanset '{[1,3),[5,7)}');
-- 4
SELECT width(intspanset '{[1,3),[5,7)}', true);
-- 6
```

- Obtener el intervalo de tiempo

  `duration({datespan,tstzspan}) → interval`

  `duration({datespanset,tstzspanset},boundspan bool=false) → interval`

  Se puede establecer a verdadero un parámetro adicional para calcular la duración en el rango delimitador, ignorando así las posibles brechas de tiempo

```
SELECT duration(datespan '[2001-01-01, 2001-01-03)');
-- 2 days
SELECT duration(tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05)}');
-- 3 days
SELECT duration(tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05)}', true);
-- 4 days
```

- Obtener el número de valores

  `numValues(set) → integer`

```
SELECT numValues(intset '{1,3,5,7}');
-- 4
```

- Obtener el valor inicial, final o enésimo

  `startValue(set) → base`

  `endValue(set) → base`

  `valueN(set,integer) → base`

```
SELECT startValue(intset '{1,3,5,7}');
-- 1
SELECT endValue(dateset '{2001-01-01, 2001-01-03, 2001-01-05, 2001-01-07}');
-- 2001-01-07
SELECT valueN(floatset '{1,3,5,7}',2);
-- 3
```

- Obtener los valores

  `getValues(set) → base[]`

```
SELECT getValues(tstzset '{2001-01-01, 2001-01-03, 2001-01-05, 2001-01-07}');
-- {"2001-01-01", "2001-01-03", "2001-01-05", "2001-01-07"}
```

- Obtener el número de rangos

  `numSpans(spanset) → integer`

```
SELECT numSpans(intspanset '{[1,3),[4,5),[6,7)}');
-- 3
SELECT numSpans(datespanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05),
  [2001-01-06, 2001-01-07)}');
-- 3
```

- Obtener el rango inicial, final o enésimo

  `startSpan(spanset) → span`

  `endSpan(spanset) → span`

  `spanN(spanset,integer) → span`

```
SELECT startSpan(intspanset '{[1,3),[4,5),[6,7)}');
-- [1,3)
SELECT startSpan(datespanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05),
  [2001-01-06, 2001-01-07)}');
-- [2001-01-01,2001-01-03)
```

```
SELECT endSpan(floatspanset '{[1,3),[4,4],[6,7)}');
-- [6,7)
SELECT endSpan(tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-04],
  [2001-01-05, 2001-01-06)}');
-- [2001-01-05,2001-01-06)
```

```
SELECT spanN(floatspanset '{[1,3),[4,4],[6,7)}',2);
-- [4,4]
SELECT spanN(tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-04],
  [2001-01-05, 2001-01-06)}', 2);
-- [2001-01-04,2001-01-04]
```

- Obtener los rangos

  `spans(spanset) → span[]`

```
SELECT spans(floatspanset '{[1,3),[4,4],[6,7)}');
-- {"[1,3)","[4,4]","[6,7)"}
SELECT spans(tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-04],
  [2001-01-05, 2001-01-06)}');
-- {"[2001-01-01,2001-01-03)", "[2001-01-04,2001-01-04]", "[2001-01-05,2001-01-06)"}
```

- Obtener el número de fechas o marcas de tiempo diferentes

  `numDates(datespanset) → integer`

  `numTimestamps(tstzspanset) → integer`

```
SELECT numDates(datespanset '{[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-04)}');
-- 4
SELECT numTimestamps(tstzspanset '{[2001-01-01, 2001-01-03), (2001-01-03, 2001-01-05)}');
-- 3
```

- Obtener la fecha o marca de tiempo inicial, final o enésima

  `startDate(datespanset) → date`

  `endDate(datespanset) → date`

  `dateN(datespanset,integer) → date`

  `startTimestamp(tstzspanset) → timestamptz`

  `endTimestamp(tstzspanset) → timestamptz`

  `timestampN(tstzspanset,integer) → timestamptz`

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

```
SELECT startDate(datespanset '{[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-04)}');
-- 2001-01-01
SELECT endTimestamp(tstzspanset '{[2001-01-01, 2001-01-03), (2001-01-03, 2001-01-05)}');
-- 2001-01-05
SELECT timestampN(tstzspanset '{[2001-01-01, 2001-01-03), (2001-01-03, 2001-01-05)}', 3);
-- 2001-01-05
```

- Obtener el conjunto de fechas o marcas de tiempo diferentes

  `dates(datespanset) → dateset`

  `timestamps(tstzspanset) → tstzset`

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

```
SELECT dates(datespanset '{[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-04)}');
-- {2001-01-01, 2001-01-02, 2001-01-03, 2001-01-04}
SELECT timestamps(tstzspanset '{[2001-01-01, 2001-01-03), (2001-01-03, 2001-01-05)}');
-- {"2001-01-01 00:00:00", "2001-01-03 00:00:00", "2001-01-05 00:00:00"}
```

2.5. Transformaciones

- Desplazar un conjunto o un span con un valor o intervalo de tiempo

  `shift(numbers,base) → numbers`

  `shift(dates,integer) → dates`

  `shift(times,interval) → times`

```
SELECT shift(dateset '{2001-01-01, 2001-01-03, 2001-01-05}', 1);
-- {2001-01-02, 2001-01-04, 2001-01-06}
SELECT shift(intspan '[1, 4)', -1);
-- [0, 3)
SELECT shift(tstzspan '[2001-01-01, 2001-01-03]', interval '1 day');
-- [2001-01-02, 2001-01-04]
SELECT shift(floatspanset '{[1, 2], [3, 4]}', -1);
-- {[0, 1], [2, 3]}
SELECT shift(tstzspanset '{[2001-01-01, 2001-01-03], [2001-01-04, 2001-01-05]}',
  interval '1 day');
-- {[2001-01-02, 2001-01-04], [2001-01-05, 2001-01-06]}
```

- Escalear un conjunto o un rango con un valor o intervalo de tiempo

  `scale(numbers,base) → numbers`

  `scale(dates,integer) → dates`

  `scale(times,interval) → times`

  Si el ancho o el lapso de tiempo del valor de entrada es cero (por ejemplo, para un conjunto de marcas de tiempo único), el resultado es el valor de entrada. El valor o intervalo de tiempo dado debe ser estrictamente mayor que cero.

```
SELECT scale(tstzset '{2001-01-01}', interval '1 day');
-- {2001-01-01}
SELECT scale(tstzset '{2001-01-01, 2001-01-03, 2001-01-05}', '2 days');
-- {2001-01-01, 2001-01-02, 2001-01-03}
SELECT scale(intspan '[1, 4)', 4);
-- [1, 6)
SELECT scale(datespan '[2001-01-01, 2001-01-04)', 4);
-- [2001-01-01, 2001-01-06)
SELECT scale(tstzspan '[2001-01-01, 2001-01-03]', '1 day');
-- [2001-01-01, 2001-01-02]
SELECT scale(floatspanset '{[1, 2], [3, 4]}', 6);
-- {[1, 3], [5, 7]}
SELECT scale(tstzspanset '{[2001-01-01, 2001-01-03], [2001-01-04, 2001-01-05]}', '1 day');
-- {[2001-01-01 00:00:00, 2001-01-01 12:00:00],
  [2001-01-01 18:00:00, 2001-01-02 00:00:00]}
SELECT scale(tstzset '{2001-01-01}', '-1 day');
-- ERROR:  The duration must be a positive interval: -1 days
```

- Desplazar y escalear un conjunto o un rango con los valores o intervalos de tiempo

  `shiftScale(numbers,base,base) → numbers`

  `shiftScale(dates,integer,integer) → dates`

  `shiftScale(times,interval,interval) → times`

  Estas funciones combinan las funciones shift y scale.

```
SELECT shiftScale(tstzset '{2001-01-01}', '1 day', '1 day');
-- {2001-01-02}
SELECT shiftScale(tstzset '{2001-01-01, 2001-01-03, 2001-01-05}', '1 day','2 days');
-- {2001-01-02, 2001-01-03, 2001-01-04}
SELECT shiftScale(intspan '[1, 4)', -1, 4);
-- [0, 5)
SELECT shiftScale(datespan '[2001-01-01, 2001-01-04)', -1, 4);
-- [2001-12-31, 2001-01-05)
SELECT shiftScale(tstzspan '[2001-01-01, 2001-01-03]', '1 day', '1 day');
-- [2001-01-02, 2001-01-03]
SELECT shiftScale(floatspanset '{[1, 2], [3, 4]}', -1, 6);
-- {[0, 2], [4, 6]}
SELECT shiftScale(tstzspanset '{[2001-01-01, 2001-01-03], [2001-01-04, 2001-01-05]}',
  '1 day', '1 day');
/* {[2001-01-02 00:00:00, 2001-01-02 12:00:00],
   [2001-01-02 18:00:00, 2001-01-03 00:00:00]} */
```

- Redondear un conjunto o un rango flotante a un número de decimales

  `round({floatset,floatspans},integer=0) → {floatset,floatspans}`

```
SELECT round(floatset '{1.123456789,2.123456789}', 3);
-- {1.123, 2.123}
SELECT round(floatspan '[1.123456789,2.123456789)', 3);
-- [1.123,2.123)
SELECT round(floatspan '[1.123456789, inf)', 3);
-- [1.123,Infinity)
SELECT round(floatspanset '{[1.123456789, 2.123456789],[3.123456789,4.123456789]}', 3);
-- {[1.123, 2.123], [3.123, 4.123]}
```

- Convertir un conjunto flotante a grados o radianes

  `degrees(floatset, normalize=false) → floatset`

  `radians(floatset) → floatset`

  El parámetro adicional en la función `degrees` puede ser utilizado para normalizar los valores entre 0 y 360 grados.

```
SELECT round(degrees(floatset '{0, 0.5, 0.7, 1.0}', true), 3);
-- {0, 28.648, 40.107, 57.296}
SELECT round(radians(floatset '{0, 45, 90}'), 3);
-- {0, 0.785, 1.571}
```

- Transformar en minúsculas, majúsculas o initcap

  `lower(textset) → textset`

  `upper(textset) → textset`

  `initcap(textset) → textset`

```
SELECT lower(textset '{"AAA", "BBB", "CCC"}');
-- {"aaa", "bbb", "ccc"}
SELECT upper(textset '{"aaa", "bbb", "ccc"}');
-- {"AAA", "BBB", "CCC"}
SELECT initcap(textset '{"aaa", "bbb", "ccc"}');
-- {"Aaa", "Bbb", "Ccc"}
```

- Concatenación de texto

  `{text,textset} || {text,textset} → textset`

```
SELECT textset '{aaa, bbb}' || text 'XX';
-- {"aaaXX", "bbbXX"}
SELECT text 'XX' || textset '{aaa, bbb}';
-- {"XXaaa", "XXbbb"}
```

- Establecer la precisión temporal del valor de tiempo al intervalo con respecto al origen

  `tprecision(times,interval,origin timestamptz='2000-01-03') → times`

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

```
SELECT tprecision(timestamptz '2001-12-03', '30 days');
-- 2001-11-23
SELECT tprecision(timestamptz '2001-12-03', '30 days', '2001-12-01');
-- 2001-12-01
SELECT tprecision(tstzset '{2001-01-01 08:00, 2001-01-01 08:10, 2001-01-01 09:00,
  2001-01-01 09:10}', '1 hour');
-- {"2001-01-01 08:00:00+01", "2001-01-01 09:00:00+01"}
SELECT tprecision(tstzspan '[2001-12-01 08:00, 2001-12-01 09:00]', '1 day');
-- [2001-12-01, 2001-12-02)
SELECT tprecision(tstzspan '[2001-12-01 08:00, 2001-12-15 09:00]', '1 day');
-- [2001-12-01, 2001-12-16)
SELECT tprecision(tstzspanset '{[2001-12-01 08:00, 2001-12-01 09:00],
  [2001-12-01 10:00, 2001-12-01 11:00]}', '1 day');
-- {[2001-12-01, 2001-12-02)}
SELECT tprecision(tstzspanset '{[2001-12-01 08:00, 2001-12-01 09:00],
  [2001-12-01 10:00, 2001-12-01 11:00]}', '1 day');
-- {[2001-12-01, 2001-12-02)}
```

2.6. Sistema de referencia espacial

- Obtener el identificador de referencia espacial

  `SRID(geoset) → integer`

```
SELECT SRID(geoset 'STBOX ZT(((1.0,2.0,3.0),(4.0,5.0,6.0)),[2001-01-01,2001-01-02])');
-- 0
SELECT SRID(geoset 'SRID=5676;STBOX XT(((1.0,2.0),(4.0,5.0)),[2001-01-01,2001-01-02])');
-- 5676
SELECT SRID(geoset 'GEODSTBOX T([2001-01-01,2001-01-02))');
-- ERROR:  The box must have space dimension
```

- Especificar el identificador de referencia espacial

  `setSRID(geoset) → geoset`

```
SELECT setSRID(stbox 'STBOX ZT(((1.0,2.0,3.0),(4.0,5.0,6.0)),
  [2001-01-01,2001-01-02])', 5676);
-- SRID=5676;STBOX ZT(((1,2,3),(4,5,6)),[2001-01-01,2001-01-02])
```

- Transformar a una referencia espacial diferente

  `transform(geoset,to_srid integer) → geoset`

  `transform(geoset,pipeline text,to_srid integer,is_forward bool=true) → geoset`

  La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando el conjunto tiene un SRID desconocido (representado por 0).

  La función `transformPipeline` especifica la transformación con una canalización de transformación de coordenadas definida representada con el siguiente formato de cadena: `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. El SRID del conjunto de entrada se ignora y el SRID de la conjunto de salida se establecerá en cero a menos que se proporcione un valor a través del parámetro opcional `to_srid`. Como se indica en el último parámetro, la canalización se ejecuta de forma predeterminada en dirección hacia adelante; al establecer el parámetro en falso, la canalización se ejecuta en la dirección inversa.

```
SELECT round(transform(stbox 'SRID=4326;STBOX XT(((2.340088, 49.400250),
  (6.575317, 51.553167)),[2001-01-01,2001-01-02])', 3812), 6);
/* SRID=3812;STBOX XT(((502773.429981,511805.120402),(803028.908265,751590.742629)),
  [2001-01-01, 2001-01-02]) */
```

```
WITH test(geoset, pipeline) AS (
  SELECT geogset 'SRID=4326;{"Point(4.3525 50.846667 100.0)",
    "Point(-0.1275 51.507222 100.0)"}',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(geoset, pipeline, 4326),
  pipeline, 4326, false), 6)
FROM test;
-- SRID=4326;{"POINT Z (4.3525 50.846667 100)", "POINT Z (-0.1275 51.507222 100)"}
```

2.7. Operaciones de conjuntos

Los tipos de conjunto y de rango tienen operadores de conjuntos asociados, a saber, unión, diferencia e intersección, que se representan, respectivamente, por +, − y * . Los operadores de conjunto para los tipos de rango y tiempo se dan a continuación.

- Unión de valores de rango o de tiempo

  `{set,spans} + {set,spans} → {set,spans}`

```
SELECT dateset '{2001-01-01, 2001-01-03, 2001-01-05}' +
  dateset '{2001-01-03, 2001-01-06}';
-- {2001-01-01, 2001-01-03, 2001-01-05, 2001-01-06}
SELECT intspan '[1, 3)' + intspan '[3, 5)';
-- [1, 5)
SELECT floatspan '[1, 3)' + floatspan '[4, 5)';
```

```
-- {[1, 3), [4, 5)}
SELECT tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05)}' +
  tstzspan '[2001-01-03, 2001-01-04)';
-- {[2001-01-01, 2001-01-05)}
```

- Diferencia de valores de rango o de tiempo

  `{set,spans} - {set,spans} → {set,spans}`

```
SELECT intset '{1, 3, 5}' - intset '{3, 6}';
-- {1, 5}
SELECT datespan '[2001-01-01, 2001-01-05)' - datespan '[2001-01-03, 2001-01-07)';
-- {[2001-01-01, 2001-01-03)}
SELECT floatspan '[1, 5]' - floatspan '[3, 4]';
-- {[1, 3), (4, 5]}
SELECT tstzspanset '{[2001-01-01, 2001-01-06], [2001-01-07, 2001-01-10]}' -
  tstzspanset '{[2001-01-02, 2001-01-03], [2001-01-04, 2001-01-05],
  [2001-01-08, 2001-01-09]}';
/* {[2001-01-01,2001-01-02), (2001-01-03,2001-01-04), (2001-01-05,2001-01-06],
   [2001-01-07,2001-01-08), (2001-01-09,2001-01-10]} */
```

- Intersección de valores de rango o de tiempo

  `{set,spans} * {set,spans} → {set,spans}`

```
SELECT tstzset '{2001-01-01, 2001-01-03}' * tstzset '{2001-01-03, 2001-01-05}';
-- {2001-01-03}
SELECT intspan '[1, 5)' * intspan '[3, 6)';
-- [3, 5)
SELECT floatspanset '{[1, 5),[6, 8)}' * floatspan '[1, 6)';
-- {[3, 5)}
SELECT tstzspan '[2001-01-01, 2001-01-05)' * tstzspan '[2001-01-03, 2001-01-07)';
-- [2001-01-03, 2001-01-05)
```

2.8. Operaciones de cuadro delimitador

2.8.1. Operaciones topológicas

A continuación se presentan los operadores topológicos disponibles para los tipos de conjunto y de rango.

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

  `{set,spans} && {set,spans} → boolean`

```
SELECT intset '{1, 3}' && intset '{2, 3, 4}';
-- true
SELECT floatspan '[1, 3)' && floatspan '[3, 4)';
-- false
SELECT floatspanset '{[1, 5),[6, 8)}' && floatspan '[1, 6)';
-- true
SELECT tstzspan '[2001-01-01, 2001-01-05)' && tstzspan '[2001-01-02, 2001-01-07)';
-- true
```

- ¿Contiene el primer valor el segundo?

  `{set,spans} @> {base,set,span} → boolean`

```
SELECT floatset '{1.5, 2.5}' @> 2.5;
-- true
SELECT tstzspan '[2001-01-01, 2001-05-01)' @> timestamptz '2001-02-01';
```

```
-- true
SELECT floatspanset '{[1, 2),(2, 3)}' @> 2.0;
-- false
```

- ¿Está el primer valor contenido en el segundo?

  `{base,set,spans} <@ {set,spans} → boolean`

```
SELECT timestamptz '2001-01-10' <@ tstzspan '[2001-01-01, 2001-05-01)';
-- true
SELECT floatspan '[2, 5]' <@ floatspan '[1, 5)';
-- false
SELECT floatspanset '{[1,2],[3,4]}' <@ floatspan '[1, 6]';
-- true
SELECT tstzspan '[2001-02-01, 2001-03-01)' <@ tstzspan '[2001-01-01, 2001-05-01)';
-- true
```

- ¿Es el primer valor adyacente al segundo?

  `spans -|- spans → boolean`

```
SELECT intspan '[2, 6)' -|- intspan '[6, 7)';
-- true
SELECT floatspan '[2, 5)' -|- floatspan '(5, 6)';
-- false
SELECT floatspanset '{[2, 3],[4, 5)}' -|- floatspan '(5, 6)';
-- true
SELECT tstzspan '[2001-01-01, 2001-01-05)' -|- tstzset '{2001-01-05, 2001-01-07}';
-- true
SELECT tstzspanset '{[2001-01-01, 2001-01-02]}' -|- tstzspan '[2001-01-02, 2001-01-03)';
-- false
```

2.8.2. Operaciones de posición

Los operadores de posición disponibles para los tipos de conjunto y de rango se dan a continuación. Observe que los operadores para tipos de tiempo tienen un # adicional para distinguirlos de los operadores para tipos de números.

- ¿Está el primer valor estrictamente a la izquierda del segundo?

  `numbers << numbers → boolean`

  `times <<# times → boolean`

```
SELECT intspan '[15, 20)' << 20;
-- true
SELECT intspanset '{[15, 17],[18, 20)}' << 20;
-- true
SELECT floatspan '[15, 20)' << floatspan '(15, 20)';
-- false
SELECT dateset '{2001-01-01, 2001-01-02}' <<# dateset '{2001-01-03, 2001-01-05}';
-- true
```

- ¿Está el primer valor estrictamente a la derecha del segundo?

  `numbers >> numbers → boolean`

  `times #>> times → boolean`

```
SELECT intspan '[15, 20)' >> 10;
-- true
SELECT floatspan '[15, 20)' >> floatspan '[5, 10]';
-- true
SELECT floatspanset '{[15, 17], [18, 20)}' >> floatspan '[5, 10]';
```

```
-- true
SELECT tstzspan '[2001-01-04, 2001-01-05)' #>>
  tstzspanset '{[2001-01-01, 2001-01-04), [2001-01-05, 2001-01-06)}';
-- true
```

- ¿No está el primer valor a la derecha del segundo?

  `numbers &< numbers → boolean`

  `times &<# times → boolean`

```
SELECT intspan '[15, 20)' &< 18;
-- false
SELECT intspanset '{[15, 16],[17, 18)}' &< 18;
-- true
SELECT floatspan '[15, 20)' &< floatspan '[10, 20]';
-- true
SELECT dateset '{2001-01-02, 2001-01-05}' &<# dateset '{2001-01-01, 2001-01-04}';
-- false
```

- ¿No está el primer valor a la izquierda del segundo?

  `numbers &> numbers → boolean`

  `times #&> times → boolean`

```
SELECT intspan '[15, 20)' &> 30;
-- true
SELECT floatspan '[1, 6]' &> floatspan '(1, 3)';
-- false
SELECT floatspanset '{[1, 2],[3, 4]}' &> floatspan '(1, 3)';
-- false
SELECT timestamp '2001-01-01' #&> tstzspan '[2001-01-01, 2001-01-05)';
-- true
```

2.9. Operaciones de distancia

El operador de distancia <-> para los tipos de rango o de tiempo considera el lapso o período delimitador y devuelve la distancia más pequeña entre los dos valores. En el caso de valores de tiempo, el operador devuelve la cantidad de días o la cantidad de segundos entre los dos valores de tiempo. El operador de distancia también se puede usar para búsquedas de vecinos más cercanos utilizando un índice GiST o SP-GiST (ver Sección 2.12).

- Obtener la distancia mínima

  `numbers <-> numbers → base`

  `dates <-> dates → integer`

  `times <-> times → float`

```
SELECT 3 <-> intspan '[6, 8)';
-- 3
SELECT floatspan '[1, 3]' <-> floatspan '(5.5, 7]';
-- 2.5
SELECT floatspan '[1, 3]' <-> floatspanset '{(5.5, 7],[8, 9]}';
-- 2.5
SELECT tstzspan '[2001-01-02, 2001-01-06)' <-> timestamptz '2001-01-07';
-- 86400
SELECT dateset '{2001-01-01, 2001-01-03, 2001-01-05}' <->
  dateset '{2001-01-02, 2001-01-04}';
--  0
```

2.10. Comparaciones

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 rango o de tiempo. Para los valores de rango, 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 conjunto y de rango se dan a continuación. Recuerde que los rangos de enteros siempre se representan por su forma canónica.

- ¿Son iguales los valores?

  `{set,spans} = {set,spans} → boolean`

```
SELECT intspan '[1,3]' = intspan '[1,4)';
-- true
SELECT floatspanset '{[1, 2),[2,3)}' = floatspanset '{[1,3)}';
-- true
SELECT tstzspan '[2001-01-01, 2001-01-04)' = tstzspan '[2001-01-01, 2001-01-04)';
-- true
SELECT tstzset '{2001-01-01, 2001-01-02}' = tstzset '{2001-01-01, 2001-01-04}';
-- false
```

- ¿Son diferentes los valores?

  `{set,spans} <> {set,spans} → boolean`

```
SELECT tstzspan '[2001-01-01, 2001-01-04)' <> tstzspan '[2001-01-03, 2001-01-05)';
-- true
SELECT tstzset '{2001-01-01, 2001-01-04}' <> tstzset '{2001-01-01, 2001-01-05}';
-- false
```

- ¿Es el primer valor menor que el segundo?

  `{set,spans} < {set,spans} → boolean`

```
SELECT intspanset '{[1,2],[3,4]}' < intspanset '{[3, 4]}';
-- true
SELECT floatspan '[3, 4]' < floatspan '(3, 4]';
-- true
SELECT tstzspan '[2001-01-01, 2001-01-04)' < tstzspan '[2001-01-01, 2001-01-04]';
-- true
SELECT tstzset '{2001-01-01, 2001-01-04}' < tstzset '{2001-01-01, 2001-01-05}';
-- true
```

- ¿Es el primer valor mayor que el segundo?

  `{set,spans} > {set,spans} → boolean`

```
SELECT floatspan '[3, 4]' > floatspan '[3, 4)';
-- true
SELECT tstzspan '[2001-01-03, 2001-01-04)' > tstzspan '[2001-01-02, 2001-01-05)';
-- true
```

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

  `{set,spans} <= {set,spans} → boolean`

```
SELECT floatspanset '{[1, 4)}' <= floatspanset '{[1, 5), [6, 7)}';
-- true
SELECT tstzspanset '{[2001-01-01, 2001-01-04)}' <=
  tstzspanset '{[2001-01-01, 2001-01-05), [2001-01-06, 2001-01-07)}';
-- true
```

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

  `{set,spans} >= {set,spans} → boolean`

```
SELECT intspanset '{[1, 4)}' >= intspanset '{[1, 5), [6, 7)}';
-- false
SELECT tstzspan '[2001-01-03, 2001-01-05)' >= tstzspan '[2001-01-03, 2001-01-04)';
-- true
```

2.11. Agregaciones

Hay varias funciones agregadas definidas para los tipos de conjunto y de rango. Estas se describen a continuación.

- La función `extent` devuelve el rango o período delimitador que engloba un conjunto de valores de rango o de tiempo.
- La unión es una operación muy útil para los tipos de conjunto y de rango. Como hemos visto en la Sección 2.7, podemos calcular la unión de dos valores de conjunto o de rango usando el operador +. Sin embargo, también es muy útil tener una versión agregada del operador de unión para combinar un número arbitrario de valores. Las funciones `set_union` y `span_union` se pueden utilizar para este propósito.
- La función `tCount` generaliza la función de agregación 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 4). La función tiene dos parámetros opcionales que especifican la granularidad (un `interval`) y el origen del tiempo (un `timestamptz`). Cuando se dan estos parámetros, el conteo temporal se calcula en intervalos de tiempo de la granularidad dada (ver Sección 9.4)

- Rango o período delimitador

  `extent({set,spans}) → span`

```
WITH spans(r) AS (
  SELECT floatspan '[1, 4)' UNION
  SELECT floatspan '(5, 8)' UNION
  SELECT floatspan '(7, 9)'
)
SELECT extent(r) FROM spans;
-- [1,9)
WITH times(ts) AS (
  SELECT tstzset '{2001-01-01, 2001-01-03, 2001-01-05}' UNION
  SELECT tstzset '{2001-01-02, 2001-01-04, 2001-01-06}' UNION
  SELECT tstzset '{2001-01-01, 2001-01-02}'
)
SELECT extent(ts) FROM times;
-- [2001-01-01, 2001-01-06]
WITH periods(ps) AS (
  SELECT tstzspanset '{[2001-01-01, 2001-01-02], [2001-01-03, 2001-01-04]}' UNION
  SELECT tstzspanset '{[2001-01-01, 2001-01-04], [2001-01-05, 2001-01-06]}' UNION
  SELECT tstzspanset '{[2001-01-02, 2001-01-06]}'
)
SELECT extent(ps) FROM periods;
-- [2001-01-01, 2001-01-06]
```

- Unión agregada

  `setUnion({value,set}) → set`

  `spanUnion(spans) → spanset`

```
WITH times(ts) AS (
  SELECT tstzset '{2001-01-01, 2001-01-03, 2001-01-05}' UNION
  SELECT tstzset '{2001-01-02, 2001-01-04, 2001-01-06}' UNION
  SELECT tstzset '{2001-01-01, 2001-01-02}'
)
SELECT setUnion(ts) FROM times;
-- {2001-01-01, 2001-01-02, 2001-01-03, 2001-01-04, 2001-01-05, 2001-01-06}
WITH periods(ps) AS (
  SELECT tstzspanset '{[2001-01-01, 2001-01-02], [2001-01-03, 2001-01-04]}' UNION
  SELECT tstzspanset '{[2001-01-02, 2001-01-03], [2001-01-05, 2001-01-06]}' UNION
  SELECT tstzspanset '{[2001-01-07, 2001-01-08]}'
)
SELECT spanUnion(ps) FROM periods;
-- {[2001-01-01, 2001-01-04], [2001-01-05, 2001-01-06], [2001-01-07, 2001-01-08]}
```

- Conteo temporal

  `tCount(times)` → `{tintSeq,tintSeqSet}`

```
WITH times(ts) AS (
  SELECT tstzset '{2001-01-01, 2001-01-03, 2001-01-05}' UNION
  SELECT tstzset '{2001-01-02, 2001-01-04, 2001-01-06}' UNION
  SELECT tstzset '{2001-01-01, 2001-01-02}'
)
SELECT tCount(ts) FROM times;
-- {2@2001-01-01, 2@2001-01-02, 1@2001-01-03, 1@2001-01-04, 1@2001-01-05, 1@2001-01-06}
WITH periods(ps) AS (
  SELECT tstzspanset '{[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-04)}' UNION
  SELECT tstzspanset '{[2001-01-01, 2001-01-04), [2001-01-05, 2001-01-06)}' UNION
  SELECT tstzspanset '{[2001-01-02, 2001-01-06)}'
)
SELECT tCount(ps) FROM periods;
-- {[2@2001-01-01, 3@2001-01-03, 1@2001-01-04, 2@2001-01-05, 2@2001-01-06)}
```

2.12. Indexación

Se pueden crear índices GiST y SP-GiST en columnas de tablas de los tipos de conjunto y de rango. El índice GiST implementa un árbol R, mientras que el índice SP-GiST implementa un árbol cuádruple. Un ejemplo de creación de un índice GiST en una columna `During` de tipo `tstzspan` en una tabla `Reservation` es como sigue:

```
CREATE TABLE Reservation (ReservationID integer PRIMARY KEY, RoomID integer,
  During tstzspan);
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 rango o 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 rango o 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.

Finalmente, se pueden crear índices hash para columnas de tabla de un tipo de rango o de tiempo. Para estos tipos de índices, la única operación definida es la igualdad.

Capítulo 3

Tipos de 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`.

3.1. Entrada y salida

MobilityDB generaliza los formatos de entrada y salida Well-Known Text (WKT) y Well-Known Binary (WKB) del Open Geospatial Consortium para todos los tipos temporales. Presentamos a continuación las funciones de entrada y salida para los tipos de cuadro delimitador.

Un `tbox` se compone de dimensiónes de valor numérico y/o de tiempo. Para cada dimensión, se proporciona un rango, es decir, ya sea un `intspan` o un `floatspan` para la dimensión de valor y un `tstzspan` para la dimensión de tiempo. Ejemplos de entrada de valores `tbox` son los siguientes:

```
-- Dimensiones de valor y tiempo
SELECT tbox 'TBOXINT XT([1,3),[2001-01-01,2001-01-02])';
SELECT tbox 'TBOXFLOAT XT([1.5,2.5],[2001-01-01,2001-01-02])';
-- Sólo dimensión de valor
SELECT tbox 'TBOXINT X([1,3))';
SELECT tbox 'TBOXFLOAT X((1.5,2.5))';
-- Sólo dimensión de tiempo
SELECT tbox 'TBOX T((2001-01-01,2001-01-02))';
```

Un `stbox` se compone de dimensiónes espacial y/o temporal, donde las coordenadas de la dimensión espacial pueden ser 2D o 3D. Para la dimensión de tiempo se da un `tstzspan`, y para la dimensión espacial se dan los valores mínimos y máximos de las coordenadas, donde estas últimas 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. Los cuadros geodésicos siempre tienen una dimensión Z para tener en cuenta la curvatura de la esfera o esferoide subyacente. Ejemplos de entrada de valores `stbox` son los siguientes:

```
-- Sólo dimensión de valor con coordenadas X e Y
SELECT stbox 'STBOX X((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 XT(((1.0,2.0),(1.0,2.0)),[2001-01-03,2001-01-03])';
-- Dimensiones de valor (con coordenadas X, Y y Z) y de tiempo
SELECT stbox 'STBOX ZT(((1.0,2.0,3.0),(1.0,2.0,3.0)),[2001-01-01,2001-01-03])';
-- 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 Z((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 ZT(((1.0,2.0,3.0),(1.0,2.0,3.0)),[2001-01-04,2001-01-04])';
-- 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 XT(((1.0,2.0),(1.0,2.0)),[2001-01-04,2001-01-04])';
SELECT stbox 'SRID=4326;GEODSTBOX Z((1.0,2.0,3.0),(1.0,2.0,3.0))';
```

Damos a continuación las funciones de entrada y salida de tipos de cuadro delimitador en formato textual (Well-Known Text o WKT) y binario (Well-Known Binary o WKB).

- Obtener la representación textual conocida (Well-Known Text o WKT)

  `asText(box,maxdecdigits=15) → text`

  El argumento `maxdecdigits` se puede utilizar para definir el número máximo de decimales para la salida de los valores de coma flotante (por defecto 15).

```
SELECT asText(tbox 'TBOXFLOAT XT([1.123456789,2.123456789),[2001-01-01,2001-01-02))', 3);
-- TBOXFLOAT XT([1.123, 2.123),[2001-01-01 00:00:00+01, 2001-01-02 00:00:00+01))
SELECT asText(stbox 'STBOX Z((1.55,1.55,1.55),(2.55,2.55,2.55))', 0);
-- STBOX Z((2,2,2),(3,3,3))
```

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

  `asBinary(box,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(tbox 'TBOXFLOAT XT([1,2),[2001-01-01,2001-01-02))');
-- \x0103270001009c57d3c11c000000fc2ef1d51c00000d0001000000000000f03f0000000000000040
SELECT asBinary(tbox 'TBOXFLOAT XT([1,2),[2001-01-01,2001-01-02))', 'XDR');
-- \x000300270100001cc1d3579c0000001cd5f12efc00000d013ff0000000000000400000000000000
SELECT asBinary(stbox 'STBOX X((1,1),(2,2))');
-- \x0101000000000000f03f0000000000000040000000000000f03f0000000000000040
```

- Obtener la representación hexadecimal binaria conocida (HexWKB) en formato texto

  `asHexWKB(box,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 la codificación de la máquina.

```
SELECT asHexWKB(tbox 'TBOXFLOAT XT([1,2),[2001-01-01,2001-01-02))');
-- 0103270001009C57D3C11C000000FC2EF1D51C00000D0001000000000000F03F0000000000000040
SELECT asHexWKB(tbox 'TBOXFLOAT XT([1,2)[2001-01-01,2001-01-02))', 'XDR');
-- 000300270100001CC1D3579C0000001CD5F12EFC00000D013FF0000000000000400000000000000
SELECT asHexWKB(stbox 'STBOX X((1,1),(2,2))');
-- 0101000000000000F03F0000000000000040000000000000F03F0000000000000040
```

- Entrar un cuadro delimitador en una representación binaria conocida (WKB)

  `tboxFromBinary(bytea) → tbox`

  `stboxFromBinary(bytea) → stbox`

```
SELECT tboxFromBinary(
  '\x0103270001009c57d3c11c000000fc2ef1d51c00000d0001000000000000f03f0000000000000040');
-- TBOXFLOAT XT([1,2),[2001-01-01,2001-01-02))
SELECT stboxFromBinary(
  '\x0101000000000000f03f0000000000000040000000000000f03f0000000000000040');
--  STBOX X((1,1),(2,2))
```

- Entrar un cuadro delimitador temporal en una representación hexadecimal binaria conocida (HexWKB)

  `tboxFromHexWKB(text) → tbox`

  `stboxFromHexWKB(text) → stbox`

```
SELECT tboxFromHexWKB(
  '0103270001009C57D3C11C000000FC2EF1D51C00000D0001000000000000F03F0000000000000040');
-- TBOXFLOAT XT([1,2),[2001-01-01,2001-01-02)))
SELECT stboxFromHexWKB(
  '0101000000000000F03F0000000000000040000000000000F03F0000000000000040');
-- STBOX X((1,1),(2,2))
```

3.2. Constructores

El tipo `tbox` tienen varias funciones constructoras dependiendo de si se da la extensión de valor y/o de tiempo. La extensión de valor se puede especificar mediante un número o un intervalo, mientras que la extensión de tiempo se puede especificar mediante un tipo de tiempo.

- Constructor para `tbox`

  `tbox({number,numspan}) → tbox`

  `tbox({timestamptz,tstzspan}) → tbox`

  `tbox({number,numspan},{timestamptz,tstzspan}) → tbox`

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

El tipo `stbox` tiene varias funciones constructoras dependiendo de si se da la extensión espacial y/o de tiempo. Las coordenadas de la extension espacial pueden ser 2D o 3D y pueden ser cartesianas o geodésicas. La extensión espacial se puede especificar mediante los valores de coordenadas mínimo y máximo. El SRID se puede especificar en un último argumento opcional. Si no se proporciona, se asume un valor 0 (respectivamente 4326) por defecto para los cuadros planos (respectivamente geodésicos). La extensión espacial también se puede especificar mediante una geometría o una geografía. La extensión temporal se puede especificar mediante un tipo de tiempo.

- Constructor para `stbox`

  `stboxX(float,float,float,float,srid=0) → stbox`

  `stboxZ(float,float,float,float,float,float,srid=0) → stbox`

  `stboxT({timestamptz,tstzspan}) → stbox`

  `stboxXT(float,float,float,float,{timestamptz,tstzspan},srid=0) → stbox`

  `stboxZT(float,float,float,float,float,float,{timestamptz,tstzspan},srid=0) → stbox`

  `geodstboxZ(float,float,float,float,float,float,srid=4326) → stbox`

  `geodstboxT({timestamptz,tstzspan}) → stbox`

  `geodstboxZT(float,float,float,float,float,float,{timestamptz,tstzspan},srid=4326) → stbox`

  `stbox(geo) → stbox`

  `stbox(geo,{timestamptz,tstzspan}) → stbox`

```
-- Sólo dimensión de valores con coordenadas X e Y
SELECT stboxX(1.0,2.0,1.0,2.0);
-- Sólo dimensión de valores con coordenadas X, Y y Z
SELECT stboxZ(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 stboxZ(1.0,2.0,3.0,1.0,2.0,3.0,5676);
-- Sólo dimensión de tiempo
SELECT stboxT(tstzspan '[2001-01-03,2001-01-03]');
-- Dimensiones de valor (con coordenadas X e Y) y de tiempo
SELECT stboxXT(1.0,2.0, 1.0,2.0, tstzspan '[2001-01-03,2001-01-03]');
-- Dimensiones de valor (con coordenadas X, Y y Z) y de tiempo
SELECT stboxZT(1.0,2.0,3.0, 1.0,2.0,3.0, tstzspan '[2001-01-03,2001-01-03]');
-- Sólo dimensión de valores con coordenadas geodéticas X, Y y Z
SELECT geodstboxZ(1.0,2.0,3.0,1.0,2.0,3.0);
-- Sólo dimensión de tiempo para cuadro geodético
SELECT geodstboxT(tstzspan '[2001-01-03,2001-01-03]');
-- Dimensiones de valor (con coordenadas geodéticas X, Y y Z) y de tiempo
SELECT geodstboxZT(1.0,2.0,3.0, 1.0,2.0,3.0, tstzspan '[2001-01-03,2001-01-04]');
-- Geometry and time dimensión
SELECT stbox(geometry 'Linestring(1 1 1,2 2 2)', tstzspan '[2001-01-03, 2001-01-05]');
-- Geography and time dimensión
SELECT stbox(geography 'Linestring(1 1 1,2 2 2)', tstzspan '[2001-01-03, 2001-01-05]');
```

3.3. Conversiones de tipo

- Convertir un `tbox` a otro tipo

  `tbox::{intspan,floatspan,tstzspan}`

  `intspan(tbox)` → `tbox`

  `floatspan(tbox)` → `tbox`

  `tstzspan(tbox)` → `tbox`

```
SELECT tbox 'TBOXINT XT([1,4),[2001-01-01,2001-01-02))'::intspan;
-- [1,4)
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02))'::floatspan;
-- (1, 2)
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02))'::tstzspan;
-- [2001-01-01, 2001-01-02)
```

- Convertir otro tipo a un `tbox`

  `{numbers,times,tnumber}::tbox`

  `tbox({numbers,times,tnumber})` → `tbox`

```
SELECT intset '{1,2}'::tbox;
-- TBOXINT X([1, 3))
SELECT intspan '[1,3)'::tbox;
-- TBOXINT X([1, 3))
SELECT floatspan '(1.0,2.0)'::tbox;
-- TBOXFLOAT X((1, 2))
SELECT tstzspanset '{(2001-01-01,2001-01-02),(2001-01-03,2001-01-04)}'::tbox;
-- TBOX T((2001-01-01,2001-01-04))
```

- Convertir un `stbox` a otro tipo

  `stbox::{box2d,box2d,geo,tstzspan}`

  `box2d(stbox)` → `box2d`

box3d(stbox) → box2d

geometry(stbox) → geometry

geography(stbox) → geography

tstzspan(stbox) → tstzspan

```
SELECT stbox 'STBOX XT(((1.0,2.0),(3.0,4.0)),[2001-01-01,2001-01-03])'::box2d;
-- BOX(1 2,3 4)
SELECT ST_AsEWKT(stbox 'SRID=4326;STBOX XT(((1,1),(5,5)),[2001-01-01,2001-01-05])'::
  geometry);
-- SRID=4326;POLYGON((1 1,1 5,5 5,5 1,1 1))
SELECT ST_AsEWKT(stbox 'STBOX XT(((1,1),(1,5)),[2001-01-01,2001-01-05])'::geometry);
-- LINESTRING(1 1,1 5)
SELECT ST_AsEWKT(stbox 'GEODSTBOX XT(((1,1),(1,1)),[2001-01-01,2001-01-05])'::geography);
-- SRID=4326;POINT(1 1)
SELECT ST_AsEWKT(stbox 'STBOX ZT(((1,1,1),(5,5,5)),[2001-01-01,2001-01-05])'::
  geometry);
/* POLYHEDRALSURFACE(((1 1 1,1 5 1,5 5 1,5 1 1,1 1 1)),
   ((1 1 5,5 1 5,5 5 5,1 5 5,1 1 5)),((1 1 1,1 1 5,1 5 5,1 5 1,1 1 1)),
   ((5 1 1,5 5 1,5 5 5,5 1 5,5 1 1)),((1 1 1,5 1 1,5 1 5,1 1 5,1 1 1)),
   ((1 5 1,1 5 5,5 5 5,5 5 1,1 5 1))) */
SELECT stbox 'STBOX XT(((1.0,2.0),(3.0,4.0)),[2001-01-01,2001-01-03])'::tstzspan;
-- [2001-01-01, 2001-01-03]
```

- Convertir otro tipo a un stbox

  {box2d,box3d,geo,times,tpoint}::stbox

  stbox({box2d,box3d,geo,times,tpoint}) → stbox

```
SELECT geometry 'Linestring(1 1 1,2 2 2)'::box3d::stbox;
-- STBOX Z((1,1,1),(2,2,2))
SELECT geography 'Linestring(1 1,2 2)'::stbox;
-- SRID=4326;GEODSTBOX X((1,1),(2,2))
SELECT tstzspanset '{(2001-01-01,2001-01-02),(2001-01-03,2001-01-04)}'::stbox;
-- STBOX T((2001-01-01,2001-01-04))
```

3.4. Accesores

- ¿Tiene dimensión X/Z/T?

  hasX(box) → boolean

  hasZ(stbox) → boolean

  hasT(box) → boolean

```
SELECT hasX(tbox 'TBOX T([2001-01-01,2001-01-03))');
-- false
SELECT hasX(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- true
SELECT hasZ(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- false
SELECT hasT(tbox 'TBOXFLOAT XT((1.0,3.0),[2001-01-01,2001-01-03])');
-- true
SELECT hasT(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- false
```

- ¿Es geodética?

  isGeodetic(stbox) → boolean

```
SELECT isGeodetic(stbox 'GEODSTBOX Z((1.0,1.0,0.0),(3.0,3.0,1.0))');
-- true
SELECT isGeodetic(stbox 'STBOX XT(((1.0,2.0),(3.0,4.0)),[2001-01-01,2001-01-02])');
-- false
```

- Obtener el valor mínimo de X/Y/Z/T

  `xMin(box) → float`

  `yMin(stbox) → float`

  `zMin(stbox) → float`

  `tMin(box) → timestamptz`

```
SELECT xMin(tbox 'TBOXFLOAT XT((1.0,3.0),[2001-01-01,2001-01-03))');
-- 1
SELECT yMin(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- 2
SELECT zMin(stbox 'STBOX Z((1.0,2.0,3.0),(4.0,5.0,6.0))');
-- 3
SELECT tMin(stbox 'GEODSTBOX T([2001-01-01,2001-01-03))');
-- 2001-01-01
```

- Obtener el valor máximo de X/Y/Z/T

  `xMax(box) → float`

  `yMax(stbox) → float`

  `zMax(stbox) → float`

  `tMax(box) → timestamptz`

```
SELECT xMax(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- 3
SELECT yMax(stbox 'STBOX X((1.0,2.0),(3.0,4.0))');
-- 4
SELECT zMax(stbox 'STBOX Z((1.0,2.0,3.0),(4.0,5.0,6.0))');
-- 6
SELECT tMax(stbox 'GEODSTBOX T([2001-01-01,2001-01-03))');
-- 2001-01-03
```

- Es el mínimo valor X/T inclusivo?

  `xMinInc(tbox) → bool`

  `tMinInc(box) → bool`

```
SELECT xMinInc(tbox 'TBOXFLOAT XT((1.0,3.0),[2001-01-01,2001-01-03))');
-- false
SELECT tMinInc(stbox 'GEODSTBOX T([2001-01-01,2001-01-03))');
-- true
```

- Es el máximo valor X/T inclusivo?

  `xMaxInc(tbox) → bool`

  `tMaxInc(box) → bool`

```
SELECT xMaxInc(tbox 'TBOXFLOAT XT((1.0,3.0),[2001-01-01,2001-01-03))');
-- false
SELECT tMaxInc(stbox 'GEODSTBOX T([2001-01-01,2001-01-03))');
-- true
```

3.5. Transformaciones

- Desplazar el rango de valores o el período del cuadro delimitador con un valor o intervalo

  `shiftValue(tbox,{integer,float}) → tbox`

  `shiftTime(box,interval) → box`

```
SELECT shiftValue(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])', 1.0);
-- TBOXFLOAT XT([2.5, 3.5],[2001-01-01, 2001-01-02])
SELECT shiftTime(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])',
  interval '1 day');
-- TBOXFLOAT XT([1.5, 2.5],[2001-01-02, 2001-01-03])
SELECT shiftTime(stbox 'STBOX T([2001-01-01,2001-01-02])', interval '-1 day');
-- STBOX T([2001-12-31, 2001-01-01])
SELECT shiftTime(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])',
  interval '1 day');
-- STBOX ZT(((1,1,1),(2,2,2)),[2001-01-02, 2001-01-03])
```

- Escalar el rango de valores o el período del cuadro delimitador a un valor o intervalo. Si el ancho o el lapso de tiempo del valor de tiempo es cero (es decir, los límites inferior y superior son iguales), el resultado es el cuadro delimitador. El valor o intervalo dado debe ser estrictamente mayor que cero.

  `scaleValue(tbox,{integer,float}) → tbox`

  `scaleTime(box,interval) → box`

```
SELECT scaleValue(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])', 2.0);
-- TBOXFLOAT XT([1.5, 3.5],[2001-01-01, 2001-01-02])
SELECT scaleTime(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])',
  interval '2 days');
-- TBOXFLOAT XT([1.5, 2.5],[2001-01-01, 2001-01-03])
SELECT scaleTime(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])',
  interval '1 hour');
-- STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01 00:00:00, 2001-01-01 01:00:00])
SELECT scaleTime(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])',
  interval '-1 day');
-- ERROR:  The interval must be positive: -1 days
```

- Desplazar y escalar el rango de valores o el período del cuadro delimitador con los valores o intervalos

  `shiftScaleValue(tbox,{integer,float},{integer,float}) → tbox`

  `shiftScaleTime(box,interval,interval) → box`

  Estas funciones combinan las funciones shiftValue y scaleValue o shiftTime y scaleTime

```
SELECT shiftScaleValue(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])', 2.0, 3.0);
-- TBOXFLOAT XT([3.5, 6.5],[2001-01-01, 2001-01-02])
SELECT shiftScaleTime(tbox 'TBOXFLOAT XT([1.5, 2.5],[2001-01-01,2001-01-02])',
  interval '1 day', interval '3 days');
-- TBOXFLOAT XT([1.5, 2.5],[2001-01-02, 2001-01-05])
SELECT shiftScaleTime(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])',
  interval '1 hour', interval '3 hours');
-- STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01 01:00:00, 2001-01-01 04:00:00])
```

- Obtener la dimensión espacial del cuadro delimitador, eliminando la dimensión temporal, si existe

  `getSpace(stbox) → stbox`

```
SELECT getSpace(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-03])');
-- STBOX Z((1,1,1),(2,2,2))
```

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 numérica del cuadro delimitador con un valor flotante

  `expandValue(tbox,{integer,float}) → tbox`

```
SELECT expandValue(tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-03])', 1.0);
-- TBOXFLOAT XT((0,3),[2001-01-01, 2001-01-03])
SELECT expandValue(tbox 'TBOX T([2001-01-01,2001-01-03))', 1);
-- The box must have value dimension
```

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

  `expandSpace(stbox,float) → stbox`

```
SELECT expandSpace(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-03])', 1);
-- STBOX ZT(((0,0,0),(3,3,3)),[2001-01-01, 2001-01-03])
SELECT expandSpace(stbox 'STBOX T([2001-01-01,2001-01-03))', 1);
-- The box must have space dimension
```

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

  `expandTime(box,interval) → box`

```
SELECT expandTime(tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-03])', interval '1 day');
-- TBOXFLOAT XT((1,2),[2011-12-31, 2001-01-04])
SELECT expandTime(stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-03])',
  interval '1 day');
-- STBOX ZT(((1,1,1),(2,2,2)),[2011-12-31, 2001-01-04])
```

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

  `round(box,integer=0) → box`

```
SELECT round(tbox 'TBOXFLOAT XT((1.12345,2.12345),[2001-01-01,2001-01-02])', 2);
-- TBOXFLOAT XT((1.12, 2.12),[2001-01-01,2001-01-02])
SELECT round(stbox 'STBOX XT(((1.12345,1.12345),(2.12345,2.12345)),
  [2001-01-01,2001-01-02])', 2);
-- STBOX XT(((1.12,1.12),(2.12,2.12)),[2001-01-01,2001-01-02])
```

3.6. Sistema de referencia espacial

- Obtener el identificador de referencia espacial

  `SRID(stbox) → integer`

```
SELECT SRID(stbox 'STBOX ZT(((1.0,2.0,3.0),(4.0,5.0,6.0)),[2001-01-01,2001-01-02])');
-- 0
SELECT SRID(stbox 'SRID=5676;STBOX XT(((1.0,2.0),(4.0,5.0)),[2001-01-01,2001-01-02])');
-- 5676
SELECT SRID(stbox 'GEODSTBOX T([2001-01-01,2001-01-02))');
-- ERROR:  The box must have space dimension
```

- Especificar el identificador de referencia espacial

  `setSRID(stbox) → stbox`

```
SELECT setSRID(stbox 'STBOX ZT(((1.0,2.0,3.0),(4.0,5.0,6.0)),
  [2001-01-01,2001-01-02])', 5676);
-- SRID=5676;STBOX ZT(((1,2,3),(4,5,6)),[2001-01-01,2001-01-02])
```

- Transformar a una referencia espacial diferente

  `transform(stbox,to_srid integer) → stbox`

  `transform(stbox,pipeline text,to_srid integer,is_forward bool=true) → stbox`

  La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando el cuadro de entrada tiene un SRID desconocido (representado por 0).

  La función `transformPipeline` especifica la transformación con una canalización de transformación de coordenadas definida representada con el siguiente formato de cadena: `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. El SRID del cuadro de entrada se ignora y el SRID del cuadro de salida se establecerá en cero a menos que se proporcione un valor a través del parámetro opcional `to_srid`. Como se indica en el último parámetro, la canalización se ejecuta de forma predeterminada en dirección hacia adelante; al establecer el parámetro en falso, la canalización se ejecuta en la dirección inversa.

```
SELECT round(transform(stbox 'SRID=4326;STBOX XT(((2.340088, 49.400250),
  (6.575317, 51.553167)),[2001-01-01,2001-01-02])', 3812), 6);
/* SRID=3812;STBOX XT(((502773.429981,511805.120402),(803028.908265,751590.742629)),
  [2001-01-01, 2001-01-02]) */
```

```
WITH test(box, pipeline) AS (
  SELECT stbox 'SRID=4326;GEODSTBOX Z((-0.1275,50.846667,100),(4.3525,51.507222,100))',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(box, pipeline, 4326),
  pipeline, 4326, false), 6)
FROM test;
-- SRID=4326;GEODSTBOX Z((-0.1275,50.846667,100),(4.3525,51.507222,100))
```

3.7. Funciones de división

- Dividir el cuadro delimitador en cuadrantes u octantes {}

  `quadSplit(stbox) → {stbox}`

  Como lo indica el símbolo {}, esta función es una *función de retorno de conjuntos* (también conocida como *función de tabla*) ya que normalmente devuelve más de un valor.

```
SELECT quadSplit(stbox 'STBOX XT(((0,0),(4,4)),[2001-01-01,2001-01-05])');
/* {"STBOX XT(((0,0),(2,2)),[2001-01-01, 2001-01-05])",
   "STBOX XT(((2,0),(4,2)),[2001-01-01, 2001-01-05])",
   "STBOX XT(((0,2),(2,4)),[2001-01-01, 2001-01-05])",
   "STBOX XT(((2,2),(4,4)),[2001-01-01, 2001-01-05])"} */
SELECT quadSplit(stbox 'STBOX Z((0,0,0),(4,4,4))');
/* {"STBOX Z((0,0,0),(2,2,2))","STBOX Z((2,0,0),(4,2,2))","STBOX Z((0,2,0),(2,4,2))",
   "STBOX Z((2,2,0),(4,4,2))","STBOX Z((0,0,2),(2,2,4))","STBOX Z((2,0,2),(4,2,4))",
   "STBOX Z((0,2,2),(2,4,4))","STBOX Z((2,2,2),(4,4,4))"} */
```

Tenga en cuenta que la función anterior es una *función de retorno de conjuntos* (también conocida como *función de tabla*) ya que normalmente devuelve más de un valor. Por lo tanto, la función está marcada con el símbolo {}. Esta función se usa normalmente para cuadrículas de resolución múltiple, donde el espacio se divide en celdas de modo que las celdas tengan un número máximo de elementos. Figura 3.1 muestra un ejemplo del resultado de usar esta función usando trayectorias sintéticas en Bruselas.

3.8. Operaciones de conjuntos

Los operadores de conjuntos para los tipos de cuadro delimitador son la unión (+) y la intersección (*). En el caso de la unión, los operandos deben tener exactamente las mismas dimensiónes, de lo contrario se genera un error. Además, si los operandos no se superponen en todas las dimensiónes se genera un error, ya que esto daría como resultado una caja con valores disjuntos, que

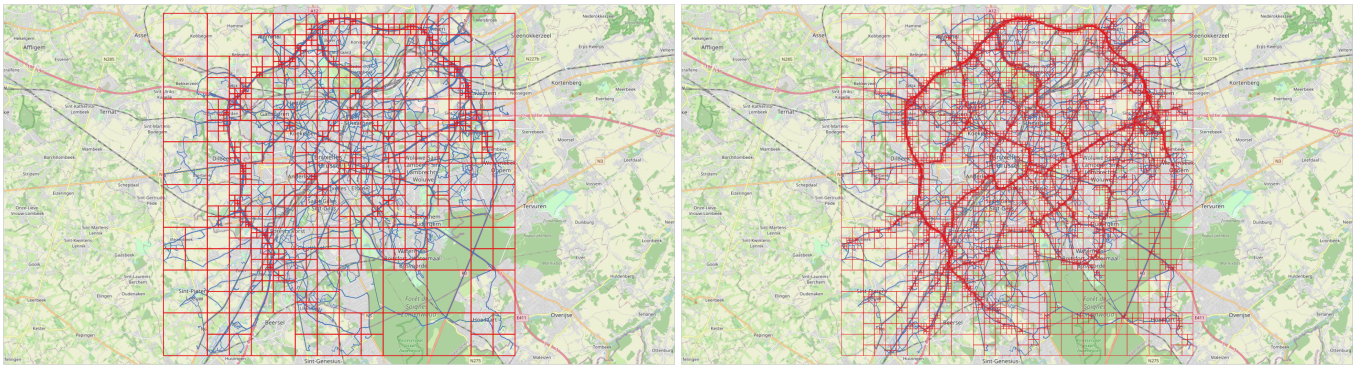


Figura 3.1: Grilla multiresolución sobre datos de Bruselas obtenidos mediante el generador BerlinMOD. Cada celda contiene como máximo 10.000 (izquierda) y 1.000 (derecha) instantes durante todo el período de simulación (cuatro días en este caso). A la izquierda, podemos ver la alta densidad de tráfico en el anillo alrededor de Bruselas, mientras que a la derecha podemos ver otros ejes principales de la ciudad.

no se puede representar. El operador calcula la unión en todas las dimensiónes 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 dimensiónes que están presentes en ambos argumentos.

- Unión de los cuadros delimitadores

  `box + box → box`

```
SELECT tbox 'TBOXINT XT([1,3),[2001-01-01,2001-01-03])' +
  tbox 'TBOXINT XT([2,4),[2001-01-02,2001-01-04])';
-- TBOXINT XT([1,4),[2001-01-01, 2001-01-04])
SELECT stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])' +
  stbox 'STBOX XT(((2,2),(3,3)),[2001-01-01,2001-01-03])';
-- ERROR:  The arguments must be of the same dimensionality
SELECT tbox 'TBOXFLOAT XT((1,3),[2001-01-01,2001-01-02])' +
  tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])';
-- ERROR:  Result of box union would not be contiguous
```

- Intersección de los cuadros delimitadores

  `box * box → box`

```
SELECT tbox 'TBOXINT XT([1,3),[2001-01-01,2001-01-03])' *
  tbox 'TBOX T([2001-01-02,2001-01-04))';
-- TBOX T([2001-01-02,2001-01-03))
SELECT stbox 'STBOX ZT(((1,1,1),(3,3,3)),[2001-01-01,2001-01-02])' *
  stbox 'STBOX X((2,2),(4,4))';
-- STBOX X((2,2),(3,3))
```

3.9. Operaciones de cuadro delimitador

3.9.1. Operaciones topológicas

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 dimensiónes que estén presentes en ambos argumentos.

- ¿Se superponen los cuadros delimitadores?

  `box && box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,3),[2001-01-01,2001-01-03])' &&
  tbox 'TBOXFLOAT XT((2,4),[2001-01-02,2001-01-04])';
-- true
SELECT stbox 'STBOX XT(((1,1),(2,2)),[2001-01-01,2001-01-02])' &&
  stbox 'STBOX XT([2001-01-02,2001-01-02])';
-- true
```

- ¿Contiene el primer cuadro delimitador el segundo?

  `box @> box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,4),[2001-01-01,2001-01-04])' @>
  tbox 'TBOXFLOAT XT((2,3),[2001-01-01,2001-01-02])';
-- true
SELECT stbox 'STBOX Z((1,1,1),(3,3,3))' @>
  stbox 'STBOX XT(((1,1),(2,2)),[2001-01-01,2001-01-02])';
-- true
```

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

  `box <@ box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' <@
  tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])';
-- true
SELECT stbox 'STBOX XT(((1,1),(2,2)),[2001-01-01,2001-01-02)' <@
  stbox 'STBOX ZT(((1,1,1),(2,2,2)),[2001-01-01,2001-01-02])';
-- true
```

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

  `box ~= box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' ~=
  tbox 'TBOXFLOAT XT([2001-01-01,2001-01-02])';
-- true
SELECT stbox 'STBOX XT(((1,1),(3,3)),[2001-01-01,2001-01-03])' ~=
  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* dimensiónes y si su intersección tiene como máximo *n*-1 dimensiónes.

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

3.9.2. Operaciones de posición

Los operadores de posición 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 numérica del tipo `tbox` se dan a continuación.

- ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente menores que los del segundo?

  `box << box → boolean`

  `stbox <<| stbox → boolean`

  `stbox <<| stbox → boolean`

  `box <<# box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' <<
  tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])';
-- true
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' <<
  tbox 'TBOXFLOAT XT([2001-01-03,2001-01-04])';
-- ERROR:  The box must have value dimension
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' << stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' <<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT stbox 'STBOX Z((1,1,1),(2,2,2))' <<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' <<#
  tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])';
-- true
```

- ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente mayores que los del segundo?

  `box >> box → boolean`

  `stbox |>> stbox → boolean`

  `stbox |>> stbox → boolean`

  `box #>> box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])' >>
  tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])';
-- true
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' >> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' |>> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' |>> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
SELECT stbox 'STBOX XT((4,4),[2001-01-03,2001-01-04],((3,3)))'  #>>
  stbox 'STBOX XT(((1,1),(2,2)),[2001-01-01,2001-01-02])';
-- true
```

- ¿No son los valores X/Y/Z/T del primer cuadro delimitador mayores que los del segundo?

  `box &< box → boolean`

  `stbox &<| stbox → boolean`

  `stbox &<| stbox → boolean`

  `box &<# box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,4),[2001-01-01,2001-01-04])' &<
  tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])';
-- true
```

```
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &< stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &<| stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT stbox 'STBOX Z((1,1,1),(4,4,4))' &</ stbox 'STBOX Z((3,3,3),(4,4,4))';
-- true
SELECT tbox 'TBOXFLOAT XT((1,4),[2001-01-01,2001-01-04])' &<#
  tbox 'TBOXFLOAT XT((3,4),[2001-01-03,2001-01-04])';
-- true
```

- ¿No son los valores X/Y/Z/T del primer cuadro delimitador menores que los del segundo?

  `box &> box → boolean`

  `stbox |&> stbox → boolean`

  `stbox /&> stbox → boolean`

  `box #&> box → boolean`

```
SELECT tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-02])' &>
  tbox 'TBOXFLOAT XT((1,4),[2001-01-01,2001-01-04])';
-- true
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' &> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' |&> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- false
SELECT stbox 'STBOX Z((3,3,3),(4,4,4))' /&> stbox 'STBOX Z((1,1,1),(2,2,2))';
-- true
SELECT stbox 'STBOX XT(((1,1),(2,2)),[2001-01-01,2001-01-02])' #&>
  stbox 'STBOX XT(((1,1),(4,4)),[2001-01-01,2001-01-04])';
-- true
```

3.10. Comparaciones

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 'TBOXINT XT([1,1],[2001-01-01,2001-01-04])' =
  tbox 'TBOXINT XT([2,2],[2001-01-03,2001-01-05])';
-- false
```

- ¿Son diferentes los cuadros delimitadores?

  `box <> box → boolean`

```
SELECT tbox 'TBOXFLOAT XT([1,1],[2001-01-01,2001-01-04])' <>
  tbox 'TBOXFLOAT XT([2,2],[2001-01-03,2001-01-05])';
-- true
```

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

  `box < box → boolean`

```
SELECT tbox 'TBOXINT XT([1,1],[2001-01-01,2001-01-04])' <
  tbox 'TBOXINT XT([1,2],[2001-01-03,2001-01-05])';
-- true
```

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

  box > box → boolean

```
SELECT tbox 'TBOXFLOAT XT([1,1],[2001-01-03,2001-01-04])' >
  tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-05])';
-- true
```

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

  box <= box → boolean

```
SELECT tbox 'TBOXINT XT([1,1],[2001-01-01,2001-01-04])' <=
  tbox 'TBOXINT XT([2,2],[2001-01-03,2001-01-05])';
-- true
```

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

  box >= box → boolean

```
SELECT tbox 'TBOXFLOAT XT([1,1],[2001-01-01,2001-01-04])' >=
  tbox 'TBOXFLOAT XT([2,2],[2001-01-03,2001-01-05])';
-- false
```

3.11. Aggregaciones

- Extensión del cuadro delimitador

  extent(box) → box

```
WITH boxes(b) AS (
  SELECT tbox 'TBOXFLOAT XT((1,3),[2001-01-01,2001-01-03])' UNION
  SELECT tbox 'TBOXFLOAT XT((5,7),[2001-01-05,2001-01-07])' UNION
  SELECT tbox 'TBOXFLOAT XT((6,8),[2001-01-06,2001-01-08])'
)
SELECT extent(b) FROM boxes;
-- TBOXFLOAT XT((1,8),[2001-01-01,2001-01-08])
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))
```

3.12. Indexación

Se pueden crear índices GiST y SP-GiST para columnas de tablas de los tipos `tbox` y `stbox`. El índice GiST implementa un árbol R, mientras que el índice SP-GiST implementa un árbol cuádruple n-dimensiónal. 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 4

Tipos temporales

4.1. Introduction

MobilityDB proporciona seis tipos temporales nativos, a saber, `tbool`, `tint`, `tfloat`, `ttext`, `tgeompoint` y `tgeogpoint`, que se basan, respectivamente, en los tipos de base `bool`, `integer`, `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 *discreta* cuando se desconoce el valor entre dos instantes sucesivos. Pueden representar, por ejemplo, entradas/salidas cuando se utiliza un lector de tarjetas RFID para entrar o salir de un edificio. La interpolación es *escalonada* cuando el valor permanece constante entre dos instantes sucesivos. Por ejemplo, la marcha que utiliza un automóvil en movimiento 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 velocidad de un automóvil puede representarse con un flotante temporal, lo que indica que los valores se conocen en los 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 tres subtipos, explicados a continuación.

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 *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 discreta, escalonada, o lineal (ver arriba). Un ejemplo es el siguiente:

```
-- Interpolación discreta
SELECT tfloat '{17@2018-01-01 08:00:00, 17.5@2018-01-01 08:05:00, 18@2018-01-01 08:10:00}';
-- Interpolación escalonada
SELECT tfloat 'Interp=Step;(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00,
  15@2018-01-01 08:10:00]';
-- Interpolación lineal
SELECT tfloat '(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 '`)`'). Por definición, ambos límites deben ser inclusivos cuando la interpolación es discreta o cuando la secuencia tiene un solo instante (que se llama *secuencia instantánea*), como el ejemplo suiguiente.

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

Los valores de secuencia deben ser *uniformes*, es decir, deben construirse a partir de valores de instante del mismo tipo de base. Los valores de secuencia con interpolación escalonada o lineal se denominan *secuencias continuas*.

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]}';
```

Como se muestra en los ejemplos anteriores, los valores de conjunto de secuencias solo pueden ser de interpolación escalonada o lineal. Además, todas las secuencias que componen un valor de conjunto de secuencias deben ser del mismo tipo de base y de la misma interpolación.

Los valores de secuencia continuos se convierten en *forma normal* de modo que los valores equivalentes tengan representaciones idénticas. Para ello, los valores de instante 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 asText(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 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 asText(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 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 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 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 asText(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 asText(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 asText(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`, `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 `tstzspan` 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 y de tiempo se define, respectivamente, por un rango numérico y un rango de tiempo.
- 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 de timpo se define en un rango de tiempo.

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

4.2. 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@2001-01-01, 12@2001-04-01, 12@2001-08-01)'),
  (20, 'Human Resources', tint '[4@2001-02-01, 6@2001-06-01, 6@2001-10-01)');
CREATE TABLE Temperature(RoomNo integer, Temp tfloat);
INSERT INTO Temperature VALUES
  (1001, tfloat '{18.5@2001-01-01 08:00:00, 20.0@2001-01-01 08:10:00}'),
  (2001, tfloat '{19.0@2001-01-01 08:00:00, 22.5@2001-01-01 08:10:00}');

-- Valor en una marca de tiempo
SELECT RoomNo, valueAtTimestamp(Temp, '2001-01-01 08:10:00')
FROM temperature;
-- 1001 | 20
```

```
-- 2001 | 22.5

-- Restricción a un valor
SELECT DeptNo, atValues(NoEmps, 10)
FROM Department;
-- 10 | [10@2001-01-01, 10@2001-04-01)
-- 20 |

-- Restricción a un período
SELECT DeptNo, atTime(NoEmps, tstzspan '[2001-01-01, 2001-04-01]')
FROM Department;
-- 10 | [10@2001-01-01, 12@2001-04-01]
-- 20 | [4@2001-02-01, 4@2001-04-01]

-- Comparación temporal
SELECT DeptNo, NoEmps #<= 10
FROM Department;
-- 10 | [t@2001-01-01, f@2001-04-01, f@2001-08-01)
-- 20 | [t@2001-02-01, t@2001-10-01)

-- Agregación temporal
SELECT tsum(NoEmps)
FROM Department;
/* {[10@2001-01-01, 14@2001-02-01, 16@2001-04-01,
   18@2001-06-01, 6@2001-08-01, 6@2001-10-01)} */
```

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)@2001-01-01 08:00:00, Point(2 0)@2001-01-01 08:10:00,
Point(2 1)@2001-01-01 08:15:00)}'),
  (20, 1, tgeompoint '{[Point(0 0)@2001-01-01 08:05:00, Point(1 1)@2001-01-01 08:10:00,
  Point(3 3)@2001-01-01 08:20:00)}');

-- Valor en una marca de tiempo
SELECT CarId, ST_AsText(valueAtTimestamp(Trip, timestamptz '2001-01-01 08:10:00'))
FROM Trips;
-- 10 | POINT(2 0)
-- 20 | POINT(1 1)

-- Restricción a un valor
SELECT CarId, asText(atValues(Trip, geometry 'Point(2 0)'))
FROM Trips;
-- 10 | {"[POINT(2 0)@2001-01-01 08:10:00+00]"}
-- 20 |

-- Restricción a un período
SELECT CarId, asText(atTime(Trip, tstzspan '[2001-01-01 08:05:00,2001-01-01 08:10:00]'))
FROM Trips;
-- 10 | {[POINT(1 0)@2001-01-01 08:05:00+00, POINT(2 0)@2001-01-01 08:10:00+00]}
-- 20 | {[POINT(0 0)@2001-01-01 08:05:00+00, POINT(1 1)@2001-01-01 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@2001-01-01 08:05:00+00, 1.4142135623731@2001-01-01 08:10:00+00,
   1@2001-01-01 08:15:00+00)} */
```

4.3. 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 o con interpolación discreta debe tener límites inferior y superior 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 de secuencias debe estar compuesto por al menos un valor de secuencia.
- En un valor de conjunto de secuencias, los valores de las secuencias componentes no deben superponerse y deben estar ordenados.

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 para interpolación escalonada
SELECT tint '[1@2001-01-01 08:00:00, 2@2001-01-01 08:10:00)';
-- Conjunto de secuencias vacío
SELECT tint '{[]}';
-- Marcas de tiempo duplicadas
SELECT tint '{1@2001-01-01 08:00:00, 2@2001-01-01 08:00:00}';
-- Períodos superpuestos
SELECT 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 5

Tipos temporales: Operaciones genéricas (Parte 1)

5.1. Introducción

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:

- `time` representa cualquier tipo de tiempo, es decir, `timestamptz`, `tstzspan`, `tstzset` o `tstzspanset`,
- `ttype` representa cualquier tipo temporal,
- `tdisc` representa cualquier tipo temporal con tipo de base discreto, es decir, `tbool`, `tint` o `ttext`,
- `tcont` representa cualquier tipo temporal con tipo de base continuo, es decir, `tfloat`, `tgeompoint` o `tgeogpoint`,
- `torder` representa cualquier tipo temporal cuyo tipo de base tiene definido un orden total, es decir, `tint`, `tfloat` o `ttext`,
- `talpha` representa cualquier tipo temporal alfanumérico, por ejemplo, `tint` or `ttext`,
- `tnumber` representa cualquier tipo de número temporal, es decir, `tint` o `tfloat`,
- `tpoint` representa un tipo de punto temporal, es decir, `tgeompoint` o `tgeogpoint`,
- `ttypeInst` representa cualquier tipo temporal con subtipo instante,
- `ttypeDiscSeq` representa cualquier tipo temporal con subtipo secuencia e interpolación discreta,
- `ttypeContSeq` representa cualquier tipo temporal con subtipo secuencia e interpolación contínua,
- `ttypeSeqSet` 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`,
- `values` representa cualquier conjunto de valores de un tipo base de un tipo temporal, por ejemplo, `integer`, `intset`, `intspan` y `intspanset` para el tipo base `integer`
- `number` representa cualquier tipo de base numérico, es decir, `integer` o `float`,
- `numspan` representa cualquier tipo de rango numérico, es decir, `intspan` o `floatspan`,
- `geo` representa los tipos `geometry` o `geography`,
- `geompoint` representa el tipo `geometry` restringido a un punto.

- `point` representa los tipos `geometry` o `geography` restringidos a un punto.
- `type[]` representa una matriz de `type`.
- `<type>` en el nombre de una función representa las funciones obtenidas al remplazar `<type>` por un `type` específico. Por ejemplo, `tintDiscSeq` o `tfloatDiscSeq` son representadas por `ttypeDiscSeq`.

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 tfloat '[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 o 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 eIntersects(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 aIntersects(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 ejemplo, la función `eIntersects` 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.

5.2. Entrada y salida

MobilityDB generaliza los formatos de entrada y salida Well-Known Text (WKT), Moving Features JSON (MF-JSON) y Well-Known Binary (WKB) del Open Geospatial Consortium para todos los tipos temporales. Presentamos a continuación las funciones de entrada y salida para los tipos temporales. Empezamos describiendo formato WKT.

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`. Ejemplos de entrada de valores de instante 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';
```

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 '`)`'). Un valor de secuencia compuesto por una sola pareja `v@t` se denomina *secuencia instantánea*. Los valores de secuencia tienen una *función de interpolación* asociada que puede ser discreta, lineal o escalonada. Por definición, los límites inferior y superior de una secuencia instantánea o de un valor de secuencia con interpolación discreta son inclusivos. La extensión temporal de un valor de secuencia con interpolación discreta es un conjunto de marcas de tiempo. Ejemplos de valores de secuencia con interpolación discreta 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 tint '{1@2001-01-01 08:00:00}'; -- Instantaneous sequence
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}';
```

La extensión temporal de un valor de secuencia con interpolación lineal o escalonada es un período definido por el primer y el últimpo instante, así como por los límites inferior y superior. Ejemplos de valores de secuencia con interpolación lineal son los siguientes:

```
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]'; -- Instantaneous 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)';
```

Los valores de secuencia cuyo tipo base es continuo pueden especificar que la interpolación es escalonada con el prefijo `Interp=Step`. Si no se especifica, se supone que la interpolación es lineal por defecto. A continuación se dan ejemplos de valores de secuencia con interpolación escalonada:

```
SELECT tfloat 'Interp=Step;[2.5@2001-01-01 08:00:00, 3@2001-01-01 08:10:00]';
SELECT tgeompoint 'Interp=Step;[Point(0 0)@2017-01-01 08:00:00,
  Point(1 1)@2017-01-01 08:05:00, Point(1 1)@2017-01-01 08:10:00)';
SELECT tgeompoint 'Interp=Step;[Point(0 0)@2017-01-01 08:00:00,
 Point(1 1)@2017-01-01 08:05:00, Point(0 0)@2017-01-01 08:10:00)';
ERROR:  Invalid end value for temporal sequence with step interpolation
SELECT tgeogpoint 'Interp=Step;[Point(0 0)@2017-01-01 08:00:00,
  Point(1 1)@2017-01-01 08:10:00]';
```

Los dos últimos instantes de un valor de secuencia con interpolación discreta y límite superior exclusivo deben tener el mismo valor base, como se muestra en el segundo y tercer ejemplo anteriores.

Un *valor de conjunto de secuencias* es un conjunto `{v1,...,vn}` donde cada `vi` es un valor de secuencia. La interpolación de los valores conjunto de secuencias solo puede ser lineal o escalonada, no discreta. Todas las secuencias que componen un valor de conjunto de secuencias deben tener la misma interpolación. La extensión temporal de un valor de conjunto de secuencias es un conjunto de períodos. Ejemplos de valores de conjunto de secuencias con interpolación lineal son los siguientes:

```
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)}';
```

Los valores de conjunto de secuencias cuyo tipo base es continuo pueden especificar que la interpolación es escalonada con el prefijo `Interp=Step`. Si no se especifica, se supone que la interpolación es lineal por defecto. A continuación se dan ejemplos de valores de conjunto de secuencias con interpolación escalonada:

```
SELECT tfloat 'Interp=Step;{[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 tgeompoint 'Interp=Step;{[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(0 1)@2017-01-01 08:15:00)}';
SELECT tgeogpoint 'Interp=Step;{[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(0 1)@2017-01-01 08:15:00)}';
```

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)@2001-01-01,Point(0 1)@2001-01-02]'
```

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)@2001-01-01,SRID=5435;Point(0 1)@2001-01-02]'
```

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)@2001-01-01,SRID=4326;Point(0 1)@2001-01-02]';
-- ERROR: Geometry SRID (4326) does not match temporal type SRID (5435)
SELECT tgeompoint 'SRID=5435;[SRID=4326;Point(0 0)@2001-01-01,
  SRID=4326;Point(0 1)@2001-01-02]'
-- ERROR: Geometry SRID (4326) does not match temporal type SRID (5435)
```

Damos a continuación las funciones de entrada y salida en formato textual (Well-Known Text o WKT), binario (Well-Known Binary o WKB) y Moving Features JSON (MF-JSON) para los tipos alfanuméricos temporales. Las funciones correspondientes para los puntos temporales se detallan en la Sección 8.1. El formato de salida predeterminado de todos los tipos alfanuméricos temporales es el formato de texto conocido. La función `asText` que se da a continuación permite determinar la salida de valores temporales de punto flotante.

- Obtener la representación textual conocida (Well-Known Text o WKT)

  `asText(tfloat,maxdecdigits=15)` → `text`

  El argumento `maxdecdigits` puede usarse para establecer el número máximo de decimales en la salida de los valores en punto flotante (por defecto 15).

```
SELECT asText(tfloat '[10.55@2001-01-01, 25.55@2001-01-02]', 0);
-- [11@2001-01-01, 26@2001-01-02]
```

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

  `asBinary(ttype,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(tbool 'true@2001-01-01');
-- \x011a000101009c57d3c11c0000
SELECT asBinary(tint '1@2001-01-01', 'XDR');
-- \x00002301000000100001cc1d3579c00
SELECT asBinary(tfloat '1.5@2001-01-01');
-- \x0121000100000000000f83f009c57d3c11c0000
SELECT asBinary(ttext 'AAA@2001-01-01');
-- \x0129000104000000000000041414100009c57d3c11c0000
```

- Obtener la representación hexadecimal binaria conocida (HexWKB) en formato texto

  `asHexWKB(ttype,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 la codificación de la máquina.

```
SELECT asHexWKB(tbool 'true@2001-01-01');
-- 011A000101009C57D3C11C0000
SELECT asHexWKB(tint '1@2001-01-01', 'XDR');
-- 00002301000000100001CC1D3579C00
SELECT asHexWKB(tfloat '1.5@2001-01-01');
-- 0121000100000000000F83F009C57D3C11C0000
SELECT asHexWKB(ttext 'AAA@2001-01-01');
-- 0129000104000000000000041414100009C57D3C11C0000
```

- Obtener la representación JSON de características móviles (Moving Features JSON o MF-JSON)

  `asMFJSON(ttype,options=0,flags=0,maxdecdigits=15) → bytea`

  El argumento `options` puede usarse para agregar un cuadro delimitador en la salida MFJSON:

  - 0: significa que no hay opción (valor por defecto)
  - 1: cuadro delimitador MFJSON

  El argumento `flags` puede usarse para personalizar la salida JSON, por ejemplo, para producir una salida JSON fácil de leer (para lectores humanos). Consulte la documentación de la biblioteca `json-c` para conocer los valores possible. Los valores típicos son los siguientes:

  - 0: means no option (default value)
  - 1: JSON_C_TO_STRING_SPACED
  - 2: JSON_C_TO_STRING_PRETTY

  El argumento `maxdecdigits` puede usarse para establecer el número máximo de decimales en la salida de los valores en punto flotante (por defecto 15).

```
SELECT asMFJSON(tbool 't@2001-01-01 18:00:00', 1);
/* {"type":"MovingBoolean","period":{"begin":"2001-01-01T18:00:00+01",
  "end":"2001-01-01T18:00:00+01","lowerInc":true,"upperInc":true},
  "values":[true],"datetimes":["2001-01-01T18:00:00+01"],"interpolation":"None"} */
SELECT asMFJSON(tint '{10@2001-01-01 18:00:00, 25@2001-01-01 18:10:00}', 1);
/* {"type":"MovingInteger","bbox":[10,25],"period":{"begin":"2001-01-01T18:00:00+01",
   "end":"2001-01-01T18:10:00+01"},"values":[10,25],"datetimes":["2001-01-01T18:00:00+01",
   "2001-01-01T18:10:00+01"],"lowerInc":true,"upperInc":true,
```

```
   "interpolation":"Discrete"} */
SELECT asMFJSON(tfloat '[10.5@2001-01-01 18:00:00+02, 25.5@2001-01-01 18:10:00+02]');
/* {"type":"MovingFloat","values":[10.5,25.5],"datetimes":["2001-01-01T17:00:00+01",
  "2001-01-01T17:10:00+01"],"lowerInc":true,"upperInc":true,"interpolation":"Linear"} */
SELECT asMFJSON(ttext '{[walking@2001-01-01 18:00:00+02,
  driving@2001-01-01 18:10:00+02]}');
/* {"type":"MovingText","sequences":[{"values":["walking","driving"],
  "datetimes":["2001-01-01T17:00:00+01","2001-01-01T17:10:00+01"],
  "lowerInc":true,"upperInc":true}],"interpolation":"Step"} */
```

- Entrar un valor temporal de su representación binaria conocida (WKB)

  `ttypeFromBinary(bytea) → ttype`

  Hay una función por tipo temporal, el nombre de la función tiene como prefijo el nombre del tipo, que son `tbool` o `tint` en los ejemplos a continuación.

```
SELECT tboolFromBinary('\x010d000101009c57d3c11c0000');
-- t@2001-01-01
SELECT tintFromBinary('\x00001601000000100001cc1d3579c00');
-- 1@2001-01-01
SELECT tfloatFromBinary('\x0113008100000000000f83f009c57d3c11c0000');
-- 1.5@2001-01-01
SELECT ttextFromBinary('\x0117000104000000000000041414100009c57d3c11c0000');
-- "AAA"@2001-01-01
```

- Entrar un valor temporal de su representación hexadecimal binaria conocida (HexWKB)

  `ttypeFromHexWKB(text) → ttype`

  Hay una función por tipo temporal, el nombre de esta función tiene como prefijo el nombre del tipo, que son `tbool` o `tint` en los ejemplos a continuación

```
SELECT tboolFromHexWKB('010D000101009C57D3C11C0000');
-- t@2001-01-01
SELECT tintFromHexWKB('00001601000000100001CC1D3579C00');
-- 1@2001-01-01
SELECT tfloatFromHexWKB('0113008100000000000F83F009C57D3C11C0000');
-- 1.5@2001-01-01
SELECT ttextFromHexWKB('0117000104000000000000041414100009C57D3C11C0000');
-- "AAA"@2001-01-01
```

- Entrar un valor temporal de su representación Moving Features JSON (MF-JSON)

  `ttypeFromMFJSON(bytea) → ttype`

  Hay una función por tipo temporal, el nombre de esta función tiene como prefijo el nombre del tipo, que son `tbool` o `tint` en los ejemplos a continuación

```
SELECT tboolFromMFJSON(text
  '{"type":"MovingBoolean","period":{"begin":"2001-01-01T18:00:00+01",
  "end":"2001-01-01T18:00:00+01","lowerInc":true,"upperInc":true},
  "values":[true],"datetimes":["2001-01-01T18:00:00+01"],"interpolation":"None"}');
-- t@2001-01-01 18:00:00
SELECT tintFromMFJSON(text
  '{"type":"MovingInteger","bbox":[10,25],"period":{"begin":"2001-01-01T18:00:00+01",
   "end":"2001-01-01T18:10:00+01"},"values":[10,25],"datetimes":["2001-01-01T18:00:00+01",
   "2001-01-01T18:10:00+01"],"lowerInc":true,"upperInc":true,
   "interpolation":"Discrete"}');
-- {10@2001-01-01 18:00:00, 25@2001-01-01 18:10:00}
SELECT tfloatFromMFJSON(text
  '{"type":"MovingFloat","values":[10.5,25.5],"datetimes":["2001-01-01T17:00:00+01",
   "2001-01-01T17:10:00+01"],"lowerInc":true,"upperInc":true,
   "interpolation":"Linear"}');
```

```
-- [10.5@2001-01-01 18:00:00, 25.5@2001-01-01 18:10:00]'
SELECT ttextFromMFJSON(text
  '{"type":"MovingText","sequences":[{"values":["walking","driving"],
  "datetimes":["2001-01-01T17:00:00+01","2001-01-01T17:10:00+01"],
  "lowerInc":true,"upperInc":true}],"interpolation":"Step"}');
-- {["walking"@2001-01-01 18:00:00, "driving"@2001-01-01 18:10:00]}');
```

5.3. Constructores

A continuación, damos las funciones de constructor para los distintos subtipos. El uso de la función de constructor suele ser más conveniente que escribir una constante literal.

- Constructores para tipos temporales que tienen un valor constante

  Estos constructores tienen dos argumentos, un tipo base y un tipo de tiempo, donde el último es un valor de `timestamptz`, `tstzset`, `tstzspan` o `tstzspanset` para construir, respectivamente, un valor de subtipo instante, una secuencia con interpolación discreta, una secuencia con interpolación linear o escalonada o un conjunto de secuencias. Las funciones para valores de secuencia o de conjunto de secuencias con tipo de base continuo tienen además un tercer argumento opcional que indica si el valor temporal resultante tiene interpolación lineal o escalonada. Se asume por defecto una interpolación lineal si el argumento no se especifica.

  `ttype(base,timestamptz) → ttypeInst`

  `ttype(base,tstzset) → ttypeDiscSeq`

  `ttype(base,tstzspan,interp='linear') → ttypeContSeq`

  `ttype(base,tstzspanset,interp='linear') → ttypeSeqSet`

```
SELECT tbool(true, timestamptz '2001-01-01');
SELECT tint(1, timestamptz '2001-01-01');
SELECT tfloat(1.5, tstzset '{2001-01-01, 2001-01-02}');
SELECT ttext('AAA', tstzset '{2001-01-01, 2001-01-02}');
SELECT tfloat(1.5, tstzspan '[2001-01-01, 2001-01-02]');
SELECT tfloat(1.5, tstzspan '[2001-01-01, 2001-01-02]', 'step');
SELECT tgeompoint('Point(0 0)', tstzspan '[2001-01-01, 2001-01-02]');
SELECT tgeogpoint('SRID=7844;Point(0 0)', tstzspanset '{[2001-01-01, 2001-01-02],
  [2001-01-03, 2001-01-04]}', 'step');
```

- Constructores para tipos temporales de subtipo secuencia

  Estos constructores tienen un primer argumento obligatorio, que es una matriz de valores de los valores instantáneos correspondientes y argumentos opcionales adicionales. El segundo argumento establece la interpolación. Si no se proporciona el argumento, es por defecto escalonada para los tipos de base base discretos como los enteros y es lineal para tipos de base continuous como los flotantes. Se genera un error cuando se establece la interpolación lineal para valores temporales con tipos de base discretos. Para secuencias continuas, el tercero y el cuarto argumentos son valores booleanos que indican, respectivamente, si los límites izquierdo y derecho son inclusivos o exclusivos. Se supone que estos argumentos son verdaderos de forma predeterminada si no se especifican.

  `ttypeSeq(ttypeInst[],interp={'step','linear'},leftInc bool=true,`
  `rightInc bool=true) → ttypeSeq`

```
SELECT tboolSeq(ARRAY[tbool 'true@2001-01-01 08:00:00','false@2001-01-01 08:05:00'],
  'discrete');
SELECT tintSeq(ARRAY[tint '1@2001-01-01 08:00:00', '2@2001-01-01 08:05:00']);
SELECT tintSeq(ARRAY[tint '1@2001-01-01 08:00:00', '2@2001-01-01 08:05:00'], 'linear');
-- ERROR:  The temporal type cannot have linear interpolation
SELECT tfloatSeq(ARRAY[tfloat '1.0@2001-01-01 08:00:00', '2.0@2001-01-01 08:05:00'],
  'step', false, true);
SELECT ttextSeq(ARRAY[ttext 'AAA@2001-01-01 08:00:00', 'BBB@2001-01-01 08:05:00']);
SELECT tgeompointSeq(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'], 'discrete');
SELECT tgeogpointSeq(ARRAY[tgeogpoint 'Point(1 1)@2001-01-01 08:00:00',
  'Point(2 2)@2001-01-01 08:05:00']);
```

- Constructores para tipos temporales de subtipo conjunto de secuencias

  `ttypeSeqSet(ttypeContSeq[])` → `ttypeSeqSet`

  `ttypeSeqSetGaps(ttypeInst[],maxt=NULL,maxdist=NULL,interp='linear')` → `ttypeSeqSet`

  Un primer conjunto de constructores tiene un solo argumento, que es una matriz de valores de los valores de *secuencia* correspondientes. La interpolación del valor temporal resultante depende de la interpolación de las secuencias que lo componen. Se genera un error si las secuencias que componen la matriz tienen una interpolación diferente.

```
SELECT tboolSeqSet(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 tintSeqSet(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 tfloatSeqSet(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 tfloatSeqSet(ARRAY[tfloat 'Interp=Step;[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=Step;[3.0@2001-01-01 08:15:00, 3.0@2001-01-01 08:20:00)']);
SELECT ttextSeqSet(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 tgeompointSeqSet(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 tgeogpointSeqSet(ARRAY[tgeogpoint
  'Interp=Step;[Point(0 0)@2001-01-01 08:00:00, Point(0 0)@2001-01-01 08:05:00)',
  'Interp=Step;[Point(1 1)@2001-01-01 08:10:00, Point(1 1)@2001-01-01 08:15:00)']);
SELECT tfloatSeqSet(ARRAY[tfloat 'Interp=Step;[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:  The temporal values must have the same interpolation
```

  Otro conjunto de constructores para valores de conjunto de secuencias tiene como primer argumento una matriz de valores de *instante* correspondientes, y dos argumentos que establecen un intervalo de tiempo máximo y una distancia máxima tal que se introduce un espacio entre secuencias del resultado siempre que dos instantes de entrada consecutivos tengan un intervalo de tiempo o una distancia mayor que estos valores. Para puntos temporales, la distancia se especifica en unidades del sistema de coordenadas. Estos dos argumentos de brechas son opcionales, pero al menos uno de ellos debe especificarse. Además, cuando el tipo base es continuo, un último argumento adicional establece si la interpolación es escalonada o lineal. Si no se especifica este argumento, se asume que es lineal por defecto.

  Los parámetros de la función dependen del tipo temporal. Por ejemplo, el parámetro de interpolación no está permitido para tipos temporales con subtipos discretos como `tint`. De manera similar, el parámetro `maxdist` no está permitido para tipos escalares como `ttext` que no tienen una función de distancia estándar.

```
SELECT tintSeqSetGaps(ARRAY[tint '1@2001-01-01', '3@2001-01-02', '4@2001-01-03',
  '5@2001-01-05'], '1 day', 1);
-- {[1@2001-01-01], [3@2001-01-02, 4@2001-01-03], [5@2001-01-05]}
SELECT ttextSeqSetGaps(ARRAY[ttext 'AA@2001-01-01', 'BB@2001-01-02', 'AA@2001-01-03',
  'CC@2001-01-05'], '1 day');
-- {["AA"@2001-01-01, "BB"@2001-01-02, "AA"@2001-01-03], ["CC"@2001-01-05]}
SELECT asText(tgeompointSeqSetGaps(ARRAY[tgeompoint 'Point(1 1)@2001-01-01',
  'Point(2 2)@2001-01-02', 'Point(3 2)@2001-01-03', 'Point(3 2)@2001-01-05'],
  '1 day', 1, 'step'));
/* Interp=Step;{[POINT(1 1)@2001-01-01], [POINT(2 2)@2001-01-02, POINT(3 2)@2001-01-03],
   [POINT(3 2)@2001-01-05]} */
```

5.4. 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 cuadro delimitador

  `ttype::tstzspan`

  `tnumber::{span,tbox}`

  `tpoint::stbox`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::tstzspan;
-- [2001-01-01, 2001-01-03]
SELECT ttext '(A@2001-01-01, B@2001-01-03, C@2001-01-05]'::tstzspan;
-- (2001-01-01, 2001-01-05]
SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::tstzspan;
-- [2001-01-01, 2001-01-03]
```

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::intspan;
-- [1, 3)
SELECT tfloat '(1@2001-01-01, 3@2001-01-03, 2@2001-01-05]'::floatspan;
-- (1, 3]
SELECT tfloat 'Interp=Step;(1@2001-01-01, 3@2001-01-03, 2@2001-01-05]'::floatspan;
-- [1, 3]
```

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::tbox;
-- TBOXINT XT((1,2),[2001-01-01,2001-01-03])
SELECT tfloat '(1@2001-01-01, 3@2001-01-03, 2@2001-01-05]'::tbox;
-- TBOXFLOAT XT((1,3),[2001-01-01,2001-01-05])
SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::stbox;
-- STBOX XT(((1,1),(3,3)),[2001-01-01, 2001-01-03])
SELECT tgeogpoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::stbox;
-- SRID=4326;GEODSTBOX XT(((1,1),(3,3)),[2001-01-01, 2001-01-03 00:00:00+01])
SELECT tgeogpoint '[Point(1 1 1)@2001-01-01, Point(3 3 3)@2001-01-03]'::stbox;
-- SRID=4326;GEODSTBOX ZT(((1,1,1),(3,3,3)),[2001-01-01, 2001-01-03])
```

- Convertir entre un entero temporal y un flotante temporal

  `tint::tfloat`

  `tfloat::tint`

```
SELECT tint '[1@2001-01-01, 2@2001-01-03]'::tfloat;
-- Interp=Step;[1@2001-01-01 00:00:00+00, 2@2001-01-03 00:00:00+00]
SELECT tint '[1@2001-01-01, 2@2001-01-03, 3@2001-01-05]'::tfloat;
-- Interp=Step;[1@2001-01-01, 2@2001-01-03, 3@2001-01-05]
SELECT tfloat 'Interp=Step;[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 entre un punto geométrico temporal y un punto geográfico temporal

  `tgeompoint::tgeogpoint`

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

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 entre un punto temporal y una trayectoria PostGIS

  `tpoint::geo`

  `geo::tpoint`

```
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)) */
```

```
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]} */
```

5.5. Accesores

- Obtener el tamaño de la memoria en bytes

  `memSize(ttype) → integer`

```
SELECT memSize(tint '{1@2001-01-01, 2@2001-01-02, 3@2001-01-03}');
-- 176
```

- Obtener el subtipo temporal

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

```
SELECT tempSubtype(tint '[1@2001-01-01, 2@2001-01-02, 3@2001-01-03]');
-- Sequence
```

- Obtener la interpolación

  `interp(ttype) → {'Discrete','Step','Linear'}`

```
SELECT interp(tfloat '{1@2001-01-01, 2@2001-01-02, 3@2001-01-03}');
-- Discrete
SELECT interp(tint '[1@2001-01-01, 2@2001-01-02, 3@2001-01-03]');
-- Step
SELECT interp(tfloat '[1@2001-01-01, 2@2001-01-02, 3@2001-01-03]');
-- Linear
SELECT interp(tfloat 'Interp=Step;[1@2001-01-01, 2@2001-01-02, 3@2001-01-03]');
-- Step
SELECT interp(tgeompoint 'Interp=Step;[Point(1 1)@2001-01-01,
  Point(2 2)@2001-01-02, Point(3 3)@2001-01-03]');
-- Step
```

- Obtener el valor o la marca de tiempo de un instantes

  `getValue(ttypeInst) → base`

  `getTimestamp(ttypeInst) → timestamptz`

```
SELECT getValue(tint '1@2001-01-01');
-- 1
SELECT getTimestamp(tfloat '1@2001-01-01');
-- 2001-01-01
```

- Obtener los valores o el tiempo en el que se define el valor temporal

  `getValues(talpha) → {bool[],spanset,textset}`

  `getTime(ttype) → tstzspanset`

```
SELECT getValues(tbool '[false@2001-01-01, true@2001-01-02, false@2001-01-03]');
-- {f,t}
SELECT getValues(tint '[1@2001-01-01, 3@2001-01-02, 1@2001-01-03]');
-- {[1, 2), [3, 4)}
SELECT getValues(tint '{[1@2001-01-01, 2@2001-01-02, 1@2001-01-03],
  [4@2001-01-04, 4@2001-01-05]}');
-- {[1, 3), [4, 5)}
SELECT getValues(tfloat '{1@2001-01-01, 2@2001-01-02, 1@2001-01-03}');
-- {[1, 1], [2, 2]}
SELECT getValues(tfloat 'Interp=Step;{[1@2001-01-01, 2@2001-01-02, 1@2001-01-03],
  [3@2001-01-04, 3@2001-01-05]}');
-- {[1, 1], [2, 2], [3, 3]}
SELECT getValues(tfloat '[1@2001-01-01, 2@2001-01-02, 1@2001-01-03]');
-- {[1, 2]}
SELECT getValues(tfloat '{[1@2001-01-01, 2@2001-01-02, 1@2001-01-03],
  [3@2001-01-04, 3@2001-01-05]}');
-- {[1, 2], [3, 3]}
```

```
SELECT getTime(ttext 'walking@2001-01-01');
-- {[2001-01-01, 2001-01-01]}
SELECT getTime(tfloat '{1@2001-01-01, 2@2001-01-02, 1@2001-01-03}');
-- {[2001-01-01, 2001-01-01], [2001-01-02, 2001-01-02], [2001-01-03, 2001-01-03]}
SELECT getTime(tint '[1@2001-01-01, 1@2001-01-15)');
-- {[2001-01-01, 2001-01-15)}
SELECT getTime(tfloat '{[1@2001-01-01, 1@2001-01-10), [12@2001-01-12, 12@2001-01-15]}');
-- {[2001-01-01, 2001-01-10), [2001-01-12, 2001-01-15]}
```

- Obtener el rango de valores o el rango de tiempo ignorando las posibles brechas

  `valueSpan(tnumber) → numspan`

  `timeSpan(ttype) → tstzspan`

```
SELECT valueSpan(tint '{[1@2001-01-01, 1@2001-01-03), [4@2001-01-03, 6@2001-01-05]}');
-- [1,7)
SELECT valueSpan(tfloat '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- [1,3])
```

```
SELECT timeSpan(tint '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- [2001-01-01, 2001-01-05]
SELECT timeSpan(tfloat '{[1@2001-01-01, 1@2001-01-02), [2@2001-01-03, 3@2001-01-04)}');
-- [2001-01-01, 2001-01-04)
```

- Obtener los valores del número o punto temporal como un conjunto

  `valueSet(tnumber,tpoint) → {numset,geoset}`

```
SELECT valueSet(tint '[1@2001-01-01, 2@2001-01-03]');
-- {1, 2}
SELECT valueSet(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 4@2001-01-05)}');
-- {1, 2, 3, 4}
SELECT asText(valueSet(tgeompoint '{[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02),
  [Point(0 1)@2001-01-03, Point(1 1)@2001-01-04)}'));
-- {"POINT(0 0)", "POINT(1 1)", "POINT(0 1)"}
```

- Obtener el valor inicial o final

  `startValue(ttype) → base`

  `endValue(ttype) → base`

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

```
SELECT startValue(tfloat '(1@2001-01-01, 2@2001-01-03)');
-- 1
SELECT endValue(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- 5
```

- Obtener el valor mínimo o máximo

  `minValue(torder) → base`

  `maxValue(torder) → base`

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

```
SELECT minValue(tfloat '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- 1
SELECT maxValue(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- 5
```

- Obtener el instante con el valor mínimo o máximo

  `minInstant(torder) → base`

  `maxInstant(torder) → base`

  La función no tiene en cuenta si los límites son inclusivos o no. Si varios instantes tienen el valor mínimo, se devuelve el primero.

```
SELECT minInstant(tfloat '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- 1@2001-01-01
SELECT maxInstant(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- 5@2001-01-05
```

- Obtener el valor en una marca de tiempo

  `valueAtTimestamp(ttype,timestamptz) → base`

```
SELECT valueAtTimestamp(tfloat '[1@2001-01-01, 4@2001-01-04)', '2001-01-02');
-- 2
```

- Obtener el intervalo de tiempo

  `duration(ttype,boundspan=false) → interval`

  Se puede poner en verdadero un parámetro adicional para calcular la duración del período limitador, ignorando así los posibles intervalos de tiempo

```
SELECT duration(tfloat '{1@2001-01-01, 2@2001-01-03, 2@2001-01-05}');
-- 00:00:00
SELECT duration(tfloat '{1@2001-01-01, 2@2001-01-03, 2@2001-01-05}', true);
-- 4 days
SELECT duration(tfloat '[1@2001-01-01, 2@2001-01-03, 2@2001-01-05)');
-- 4 days
SELECT duration(tfloat '{[1@2001-01-01, 2@2001-01-03), [2@2001-01-04, 2@2001-01-05)}');
-- 3 days
SELECT duration(tfloat '{[1@2001-01-01, 2@2001-01-03), [2@2001-01-04, 2@2001-01-05)}',
  true);
-- 4 days
```

- ¿Es el instante inicial/final inclusivo?

  `lowerInc(ttype) → bool`

  `upperInc(ttype) → bool`

```
SELECT lowerInc(tint '[1@2001-01-01, 2@2001-01-02)');
-- true
SELECT upperInc(tfloat '{[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03)}');
-- false
```

- Obtener el número de instantes diferentes

  `numInstants(ttype) → integer`

```
SELECT numInstants(tfloat '{[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03)}');
-- 3
```

- Obtener el instante inicial, final o enésimo

  `startInstant(ttype) → ttypeInst`

  `endInstant(ttype) → ttypeInst`

  `instantN(ttype,integer) → ttypeInst`

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

```
SELECT startInstant(tfloat '{[1@2001-01-01, 2@2001-01-02),
  (2@2001-01-02, 3@2001-01-03)}');
-- 1@2001-01-01
SELECT endInstant(tfloat '{[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03)}');
-- 3@2001-01-03
SELECT instantN(tfloat '{[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03)}', 3);
-- 3@2001-01-03
```

- Obtener los instantes diferentes

  `instants(ttype) → ttypeInst[]`

```
SELECT instants(tfloat '{[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03)}');
-- {"1@2001-01-01","2@2001-01-02","3@2001-01-03"}
```

- Obtener el número de marcas de tiempo diferentes

  `numTimestamps(ttype) → integer`

```
SELECT numTimestamps(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}');
-- 3
```

- Obtener la marca de tiempo inicial, final o enésima

  `startTimestamp(ttype) → timestamptz`

  `endTimestamp(ttype) → timestamptz`

  `timestampN(ttype,integer) → timestamptz`

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

```
SELECT startTimestamp(tfloat '[1@2001-01-01, 2@2001-01-03)');
-- 2001-01-01
SELECT endTimestamp(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}');
-- 2001-01-05
SELECT timestampN(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}', 3);
-- 2001-01-05
```

- Obtener las marcas de tiempo diferentes

  `timestamps(ttype) → timestamptz[]`

```
SELECT timestamps(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- {"2001-01-01", "2001-01-03", "2001-01-05"}
```

- Obtener el número de secuencias

  `numSequences({ttypeContSeq,ttypeSeqSet}) → integer`

```
SELECT numSequences(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}');
-- 2
```

- Obtener la secuencia inicial, final, o enésima

  `startSequence({ttypeContSeq,ttypeSeqSet}) → ttypeContSeq`

  `endSequence({ttypeContSeq,ttypeSeqSet}) → ttypeContSeq`

  `sequenceN({ttypeContSeq,ttypeSeqSet},integer) → ttypeContSeq`

```
SELECT startSequence(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}');
-- [1@2001-01-01, 2@2001-01-03)
SELECT endSequence(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- [3@2001-01-03, 5@2001-01-05)
SELECT sequenceN(tfloat '{[1@2001-01-01, 2@2001-01-03),
  [3@2001-01-03, 5@2001-01-05)}', 2);
-- [3@2001-01-03, 5@2001-01-05)
```

- Obtener las secuencias

  `sequences({ttypeContSeq,ttypeSeqSet}) → ttypeContSeq[]`

```
SELECT sequences(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- {"[1@2001-01-01, 2@2001-01-03)", "[3@2001-01-03, 5@2001-01-05)"}
```

- Obtener los segmentos

  `segments({ttypeContSeq,ttypeSeqSet}) → ttypeContSeq[]`

```
SELECT segments(tint '{[1@2001-01-01, 3@2001-01-02, 2@2001-01-03],
  (3@2001-01-03, 5@2001-01-05]}');
/* {"[1@2001-01-01, 1@2001-01-02)","[3@2001-01-02, 3@2001-01-03)","[2@2001-01-03]",
  "(3@2001-01-03, 3@2001-01-05)","[5@2001-01-05]"} */
SELECT segments(tfloat '{[1@2001-01-01, 3@2001-01-02, 2@2001-01-03],
  (3@2001-01-03, 5@2001-01-05]}');
/* {"[1@2001-01-01, 3@2001-01-02)","[3@2001-01-02, 2@2001-01-03]",
   "(3@2001-01-03, 5@2001-01-05]"} */
```

- Obtener el área bajo la curva

  `integral(tnumber) → float`

```
SELECT  integral(tint '[1@2001-01-01,2@2001-01-02]') / (24 * 3600 * 1e6);
-- 1
SELECT integral(tfloat '[1@2001-01-01,2@2001-01-02]') / (24 * 3600 * 1e6);
-- 1.5
```

- Obtener el promedio ponderado en el tiempo

  `twAvg(tnumber) → float`

```
SELECT twAvg(tfloat '{[1@2001-01-01, 2@2001-01-03), [2@2001-01-04, 2@2001-01-06)}');
-- 1.75
```

5.6. Transformaciones

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

  `ttypeInst(ttype) → ttypeInst`

  `ttypeSeq(ttype) → ttypeSeq`

  `ttypeSeqSet(ttype) → ttypeSeqSet`

```
SELECT tboolInst(tbool '{[true@2001-01-01]}');
-- t@2001-01-01
SELECT tboolInst(tbool '{[true@2001-01-01, true@2001-01-02]}');
-- ERROR: Cannot transform input value to a temporal instant
SELECT tintSeq(tint '1@2001-01-01');
-- [1@2001-01-01]
SELECT tfloatSeqSet(tfloat '2.5@2001-01-01');
-- {[2.5@2001-01-01]}
SELECT tfloatSeqSet(tfloat '{2.5@2001-01-01, 1.5@2001-01-02, 3.5@2001-01-03}');
-- {[2.5@2001-01-01],[1.5@2001-01-02],[3.5@2001-01-03]}
```

- Transformar un valor temporal a otra interpolación

  `setInterp(ttype, interp) → ttype`

```
SELECT setInterp(tbool 'true@2001-01-01','discrete');
-- {t@2001-01-01}
SELECT setInterp(tfloat '{[1@2001-01-01], [2@2001-01-02], [1@2001-01-03]}', 'discrete');
-- {1@2001-01-01, 2@2001-01-02, 1@2001-01-03}
SELECT setInterp(tfloat 'Interp=Step;[1@2001-01-01, 2@2001-01-02,
  1@2001-01-03, 2@2001-01-04]', 'linear');
/* {[1@2001-01-01, 1@2001-01-02), [2@2001-01-02, 2@2001-01-03),
```

```
   [1@2001-01-03, 1@2001-01-04), [2@2001-01-04]}" */
SELECT asText(setInterp(tgeompoint 'Interp=Step;{[Point(1 1)@2001-01-01,
  Point(2 2)@2001-01-02], [Point(3 3)@2001-01-05, Point(4 4)@2001-01-06]}', 'linear'));
/* {[POINT(1 1)@2001-01-01, POINT(1 1)@2001-01-02), [POINT(2 2)@2001-01-02],
   [POINT(3 3)@2001-01-05, POINT(3 3)@2001-01-06), [POINT(4 4)@2001-01-06]} */
```

- Desplazar el rango de valores or el intervalo de tiempo de un valor temporal con un valor or intervalo

  `shiftValue(tnumber,base) → tnumber`

  `shiftTime(ttype,interval) → ttype`

```
SELECT shiftValue(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}', 1);
-- {2@2001-01-02, 3@2001-01-04, 2@2001-01-06}
SELECT shiftValue(tfloat '{[1@2001-01-01,2@2001-01-02],[3@2001-01-03,4@2001-01-04]}', -1);
--  {[0@2001-01-01, 1@2001-01-02], [2@2001-01-03, 3@2001-01-04]}
```

```
SELECT shiftTime(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 shiftTime(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day');
-- [1@2001-01-02, 2@2001-01-04]
SELECT asText(shiftTime(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 rango de valores o el intervalo de tiempo de un valor temporal a un valor o intervalo

  `scaleValue(tnumber,width) → tnumber`

  `scaleTime(ttype,interval) → ttype`

  Si el rango de valores o el intervalo de tiempo del valor temporal es cero (por ejemplo, para un instante temporal), el resultado es el valor temporal. El ancho o intervalo dado debe ser estrictamente mayor que cero.

```
SELECT scaleValue(tint '1@2001-01-01', 1);
-- 1@2001-01-01
SELECT scaleValue(tfloat '{[1@2001-01-01,2@2001-01-02], [3@2001-01-03,4@2001-01-04]}', 6);
-- {[1@2001-01-01, 3@2001-01-03], [5@2001-01-05, 7@2001-01-07]}
SELECT scaleValue(tint '1@2001-01-01', -1);
-- ERROR:   The value must be strictly positive: -1
```

```
SELECT scaleTime(tint '1@2001-01-01', '1 day');
-- 1@2001-01-01
SELECT scaleTime(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 scaleTime(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day');
-- [1@2001-01-01, 2@2001-01-02]
SELECT asText(scaleTime(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 scaleTime(tint '1@2001-01-01', '-1 day');
-- ERROR:  The interval must be positive: -1 days
```

- Desplazar y escalear el rango de valores o intervalo de tiempo de un valor temporal a los dos valores o intervalos

  `shiftScaleValue(tnumber,base,base) → tnumber`

  `shiftScaleTime(ttype,interval,interval) → ttype`

  Estas funciones combinan las funciones shiftValue y scaleValue o shiftTime y scaleTime.

```
SELECT shiftScaleValue(tint '1@2001-01-01', 1, 1);
-- 2@2001-01-01
SELECT shiftScaleValue(tfloat '{[1@2001-01-01,2@2001-01-02],[3@2001-01-03,4@2001-01-04]}',
  -1, 6);
--  {[0@2001-01-01, 2@2001-01-02], [4@2001-01-03, 6@2001-01-04]}
```

```
SELECT shiftScaleTime(tint '1@2001-01-01', '1 day', '1 day');
-- 1@2001-01-02
SELECT shiftScaleTime(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 shiftScaleTime(tfloat '[1@2001-01-01, 2@2001-01-03]', '1 day', '1 day');
-- [1@2001-01-02, 2@2001-01-03]
SELECT asText(shiftScaleTime(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]} */
```

- Extraer de un sequencia o un conjunto de secuencias temporales con interpolación linear los segmentos cuya distancia es menor o igual que el argumento dados y/o cuya duración es mayor o igual que el argumento dado

  `stops({ttypeContSeq,ttypeSeqSet},maxDist=0.0,minDuration='0 minutes')` → `ttypeSeqSet`

  Para los tipos espaciales, el tamaño del área se calcula como la diagonal del rectángulo mínimamente girado de los puntos de la subsecuencia. Para los tipos escalares, el tamaño se calcula como el ancho del intervalo circundante. Si no se proporciona `maxDist`, se supone que es 0.0 y, por lo tanto, la función extrae los segmentos constantes del valor temporal dado. Para puntos temporales, la distancia se calcula en las unidades del sistema de coordenadas. Nótese que aunque la función acepta geometrías 3D, el cálculo siempre se realiza en 2D.

```
SELECT stops(tfloat '[1@2001-01-01, 1@2001-01-02, 2@2001-01-03]');
-- {[1@2001-01-01, 1@2001-01-02)}
SELECT stops(tfloat '[1@2001-01-01, 1@2001-01-02, 2@2001-01-03]', 0.0, '2 days');
-- NULL
SELECT asText(stops(tgeompoint '[Point(1 1)@2001-01-01, Point(1 1)@2001-01-02,
  Point(2 2)@2001-01-03, Point(2 2)@2001-01-04]'));
/* {[POINT(1 1)@2001-01-01, POINT(1 1)@2001-01-02), [POINT(2 2)@2001-01-03,
   POINT(2 2)@2001-01-04]} */
SELECT asText(stops(tgeompoint '[Point(1 1 1)@2001-01-01, Point(1 1 1)@2001-01-02,
  Point(2 2 2)@2001-01-03, Point(2 2 2)@2001-01-04]', 1.75));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (1 1 1)@2001-01-02, POINT Z (2 2 2)@2001-01-03,
   POINT Z (2 2 2)@2001-01-04]} */
```

- Transformar un valor temporal no lineal en un conjunto de filas, cada una es una pareja compuesta de un valor de base y un conjunto de períodos durante el cual el valor temporal tiene el valor de base {}

  `unnest(ttype)` → `{(value,time)}`

```
SELECT (un).value, (un).time
FROM (SELECT unnest(tfloat '{1@2001-01-01, 2@2001-01-02, 1@2001-01-03}') AS un) t;
-- 1 | {[2001-01-01, 2001-01-01], [2001-01-03, 2001-01-03]}
-- 2 | {[2001-01-02, 2001-01-02]}
SELECT (un).value, (un).time
FROM (SELECT unnest(tint '[1@2001-01-01, 2@2001-01-02, 1@2001-01-03]') AS un) t;
-- 1 | {[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-03]}
-- 2 | {[2001-01-02, 2001-01-03)}
SELECT (un).value, (un).time
FROM (SELECT unnest(tfloat '[1@2001-01-01, 2@2001-01-02, 1@2001-01-03]') AS un) t;
-- ERROR:  The temporal value cannot have linear interpolation
SELECT ST_AsText((un).value), (un).time
FROM (SELECT unnest(tgeompoint 'Interp=Step;[Point(1 1)@2001-01-01,
```

```
  Point(2 2)@2001-01-02, Point(1 1)@2001-01-03]') AS un) t;
--  POINT(1 1) | {[2001-01-01, 2001-01-02), [2001-01-03, 2001-01-03]}
--  POINT(2 2) | {[2001-01-02, 2001-01-03)}
```

Capítulo 6

Tipos temporales: Operaciones genéricas (Parte 2)

6.1. Modificaciones

A continuación, explicamos la semántica de las operaciones de modificación (es decir, `insert`, `update` y `delete`) para tipos temporales. Estas operaciones tienen una semántica similar a las operaciones correspondientes para tablas temporales de tiempo de aplicación (*application-time temporal tables*) introducidas en el estándar SQL:2011. La principal diferencia es que SQL usa marcas de tiempo de tuplas (donde las marcas de tiempo se adjuntan a las tuplas), mientras que los valores temporales en MobilityDB usan marcas de tiempo de atributos (donde las marcas de tiempo se adjuntan a los valores de los atributos).

La operación `insert` agrega a un valor temporal los instantes de otro sin modificar los instantes existentes, como se ilustra en Figura 6.1.

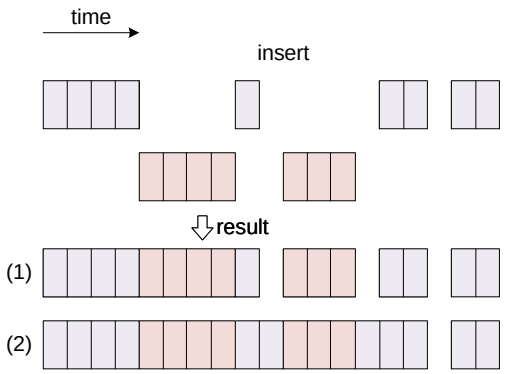


Figura 6.1: Operación de inserción para valores temporales.

Como se muestra en la figura, los valores temporales solo pueden intersectarse en su límite y, en ese caso, deben tener el mismo valor de base en sus marcas de tiempo comunes; de lo contrario, se genera un error. El resultado de la operación es la unión de los instantes para ambos valores temporales, como se muestra en el primer resultado de la figura. Esto es equivalente a una operación `merge` que se explica a continuación. Alternativamente, como se muestra en el segundo resultado de la figura, los fragmentos insertados que son disjuntos con el valor original se conectan al último instante anterior y al primer instante posterior al fragmento. Se utiliza un parámetro booleano `connect` para elegir entre los dos resultados, y el parámetro es verdadero por defecto. Nótese que esto solo se aplica a valores temporales continuos.

La operación `update` reemplaza los instantes en un primer valor temporal con los de un segundo valor como se ilustra en la Figura 6.2.

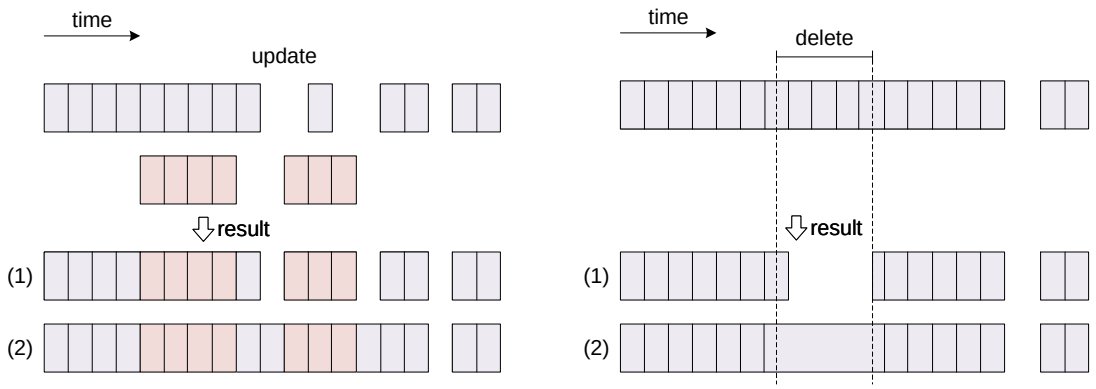


Figura 6.2: Operaciones de actualización y supresión para valores temporales.

Como se muestra en la figura, el valor resultante contiene los instantes del segundo valor, independientemente de los instantes anteriores que tenía en el valor temporal original. Como en el caso de una operación `insert`, un parámetro booleano adicional determina si los nuevos fragmentos desconectados están conectados en el valor resultante, como se muestra en los dos posibles resultados de la figura. Cuando los dos valores temporales son disjuntos o solo se intersectan en su límite, esto corresponde a una operación `insert` como se explicó anteriormente. En este caso, la operación `update` se comporta como una operación `upsert` en SQL.

La operación `deleteTime` elimina los instantes de un valor temporal que intersectan un valor de tiempo. Esta operación se puede utilizar en dos situaciones diferentes, ilustradas en la Figura 6.2.

1. En el primer caso, que se muestra como el resultado superior de la figura, el significado de la operación es introducir brechas de tiempo después de eliminar los instantes del valor temporal que intersectan el valor de tiempo. Esto es equivalente a las operaciones de restricción (Sección 6.2), que restringen un valor temporal al complemento del valor de tiempo.

2. El segundo caso, que se muestra como el resultado inferior de la figura, se usa para eliminar valores erróneos (por ejemplo, detectados como valores atípicos) sin introducir una brecha de tiempo, o para eliminar una brecha de tiempo. En este caso, los valores en el fragmento del valor temporal se eliminan y el último instante anterior y el primer instante posterior a une fragmento suprimido se conectan. Este comportamiento se especifica estableciendo un parámetro booleano adicional de la operación. Nótese que esto solo se aplica a valores temporales continuos.

La Figura 6.3 muestra las operaciones de modificación equivalentes para tablas temporales en el estándar SQL. Intuitivamente, estas figuras se obtienen girando 90 grados en el sentido de las agujas del reloj las figuras correspondientes para los valores temporales (Figura 6.1 y Figura 6.2). Esto se debe al hecho de que en SQL, las tuplas consecutivas ordenadas por tiempo generalmente se conectan a través de las funciones de ventana `LEAD` y `LAG`.

- Insertar un valor temporal en otro

  `insert(ttype,ttype,connect=true) → ttype`

```
SELECT insert(tint '{1@2001-01-01, 3@2001-01-03, 5@2001-01-05}',
  tint '{3@2001-01-03, 7@2001-01-07}');
-- {1@2001-01-01, 3@2001-01-03, 5@2001-01-05, 7@2001-01-07}
SELECT insert(tint '{1@2001-01-01, 3@2001-01-03, 5@2001-01-05}',
  tint '{5@2001-01-03, 7@2001-01-07}');
-- ERROR: The temporal values have different value at their overlapping instant 2001-01-03
SELECT insert(tfloat '[1@2001-01-01, 2@2001-01-02]',
  tfloat '[1@2001-01-03, 1@2001-01-05]');
-- [1@2001-01-01, 2@2001-01-02, 1@2001-01-03, 1@2001-01-05]
SELECT insert(tfloat '[1@2001-01-01, 2@2001-01-02]',
  tfloat '[1@2001-01-03, 1@2001-01-05]', false);
-- {[1@2001-01-01, 2@2001-01-02], [1@2001-01-03, 1@2001-01-05]}
SELECT asText(insert(tgeompoint '{[Point(1 1 1)@2001-01-01, Point(2 2 2)@2001-01-02],
  [Point(3 3 3)@2001-01-04],[Point(1 1 1)@2001-01-05]}',
```

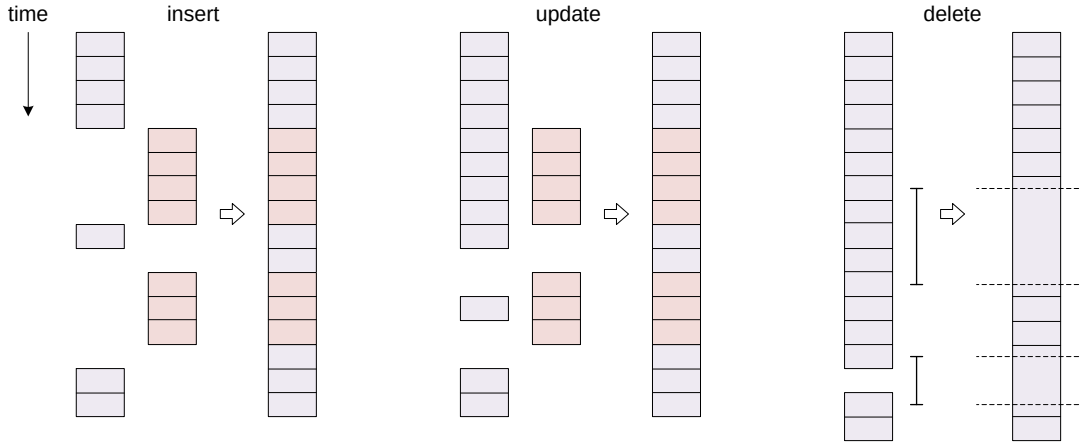


Figura 6.3: Operaciones de modificación para tablas temporales en SQL.

```
  tgeompoint 'Point(1 1 1)@2001-01-03'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02, POINT Z (1 1 1)@2001-01-03,
 POINT Z (3 3 3)@2001-01-04], [POINT Z (1 1 1)@2001-01-05]} */
```

- Actualizar un valor temporal con otro

  `update(ttype,ttype,connect=true) → ttype`

```
SELECT update(tint '{1@2001-01-01, 3@2001-01-03, 5@2001-01-05}',
  tint '{5@2001-01-03, 7@2001-01-07}');
-- {1@2001-01-01, 5@2001-01-03, 5@2001-01-05, 7@2001-01-07}
SELECT update(tfloat '[1@2001-01-01, 1@2001-01-05]',
  tfloat '[1@2001-01-02, 3@2001-01-03, 1@2001-01-04]');
-- {[1@2001-01-01, 1@2001-01-02, 3@2001-01-03, 1@2001-01-04, 1@2001-01-05]}
SELECT asText(update(tgeompoint '{[Point(1 1 1)@2001-01-01, Point(3 3 3)@2001-01-03,
  Point(1 1 1)@2001-01-05], [Point(1 1 1)@2001-01-07]}',
  tgeompoint '[Point(2 2 2)@2001-01-02, Point(2 2 2)@2001-01-04]'));
/*  {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02, POINT Z (2 2 2)@2001-01-04,
  POINT Z (1 1 1)@2001-01-05], [POINT Z (1 1 1)@2001-01-07]} */
```

- Eliminar de un valor temporal un valor de tiempo

  `deleteTime(ttype,time,connect=true) → ttype`

```
SELECT deleteTime(tint '[1@2001-01-01, 1@2001-01-03]', timestamptz '2001-01-02', false);
-- {[1@2001-01-01, 1@2001-01-02), (1@2001-01-02, 1@2001-01-03]}
SELECT deleteTime(tint '[1@2001-01-01, 1@2001-01-03]', timestamptz '2001-01-02');
-- [1@2001-01-01, 1@2001-01-03]
SELECT deleteTime(tfloat '[1@2001-01-01, 4@2001-01-02, 2@2001-01-04, 5@2001-01-05]',
  tstzspan '[2001-01-02, 2001-01-04]');
-- [1@2001-01-01, 5@2001-01-05]
SELECT asText(deleteTime(tgeompoint '{[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02], [Point(3 3 3)@2001-01-04, Point(3 3 3)@2001-01-05]}',
  tstzspan '[2001-01-02, 2001-01-04]'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02, POINT Z (3 3 3)@2001-01-04,
  POINT Z (3 3 3)@2001-01-05]} */
```

- Anexar un instante temporal a un valor temporal

  `appendInstant(ttype,ttypeInst) → ttype`

  `appendInstant(ttypeInst,maxdist=NULL,maxt=NULL) → ttypeSeq`

La primera versión de la función devuelve el resultado de agregar el segundo argumento al primero. Si cualquiera de las entradas es NULL, se devuelve NULL.

La segunda versión de la función anterior es una función de *agregación* que devuelve el resultado de agregar sucesivamente un conjunto de filas de valores temporales. Esto significa que funciona de la misma manera que las funciones `SUM()` y `AVG()` y, como la mayoría de los agregados, también ignora los valores NULL. Dos argumentos opcionales establecen una distancia máxima y un intervalo de tiempo máximo de modo que se introduce una brecha de tiempo cada vez que dos instantes consecutivos tienen una distancia o un intervalo de tiempo mayor que estos valores. Para puntos temporales, la distancia se especifica en unidades del sistema de coordenadas. Si uno de estos argumentos no se dan, no se tiene en cuenta para determinar las brechas.

```
SELECT appendInstant(tint '1@2001-01-01', tint '1@2001-01-02');
-- {1@2001-01-01, 1@2001-01-02}
SELECT appendInstant(tint '[1@2001-01-01]', tint '1@2001-01-02');
-- [1@2001-01-01, 1@2001-01-02]
SELECT asText(appendInstant(tgeompoint '{[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02], [Point(3 3 3)@2001-01-04, Point(3 3 3)@2001-01-05]}',
  tgeompoint 'Point(1 1 1)@2001-01-06'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02],
   [POINT Z (3 3 3)@2001-01-04, POINT Z (3 3 3)@2001-01-05,
   POINT Z (1 1 1)@2001-01-06]} */
```

```
WITH temp(inst) AS (
  SELECT tfloat '1@2001-01-01' UNION
  SELECT tfloat '2@2001-01-02' UNION
  SELECT tfloat '3@2001-01-03' UNION
  SELECT tfloat '4@2001-01-04' UNION
  SELECT tfloat '5@2001-01-05' )
SELECT appendInstant(inst ORDER BY inst) FROM temp;
-- [1@2001-01-01, 5@2001-01-05]
WITH temp(inst) AS (
  SELECT tgeogpoint 'Point(1 1)@2001-01-01' UNION
  SELECT tgeogpoint 'Point(2 2)@2001-01-02' UNION
  SELECT tgeogpoint 'Point(3 3)@2001-01-03' UNION
  SELECT tgeogpoint 'Point(4 4)@2001-01-04' UNION
  SELECT tgeogpoint 'Point(5 5)@2001-01-05' )
SELECT asText(appendInstant(inst ORDER BY inst)) FROM temp;
/* [POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02, POINT(3 3)@2001-01-03,
   POINT(4 4)@2001-01-04, POINT(5 5)@2001-01-05] */
```

Observe que en la primera consulta con `tfloat`, las observaciones intermedias fueron eliminadas por el proceso de normalización ya que eran redundantes debido a la interpolación lineal. Este no es el caso de la segunda consulta con `tgeogpoint` ya que se utilizan coordenadas geodésicas.

```
WITH temp(inst) AS (
  SELECT tfloat '1@2001-01-01' UNION
  SELECT tfloat '2@2001-01-02' UNION
  SELECT tfloat '4@2001-01-04' UNION
  SELECT tfloat '5@2001-01-05' UNION
  SELECT tfloat '7@2001-01-07' )
SELECT appendInstant(inst, 0.0, '1 day' ORDER BY inst) FROM temp;
-- {[1@2001-01-01, 2@2001-01-02], [4@2001-01-04, 5@2001-01-05], [7@2001-01-07]}
WITH temp(inst) AS (
  SELECT tgeompoint 'Point(1 1)@2001-01-01' UNION
  SELECT tgeompoint 'Point(2 2)@2001-01-02' UNION
  SELECT tgeompoint 'Point(4 4)@2001-01-04' UNION
  SELECT tgeompoint 'Point(5 5)@2001-01-05' UNION
  SELECT tgeompoint 'Point(7 7)@2001-01-07' )
SELECT asText(appendInstant(inst, sqrt(2), '1 day' ORDER BY inst)) FROM temp;
/* {[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02],
   [POINT(4 4)@2001-01-04, POINT(5 5)@2001-01-05], [POINT(7 7)@2001-01-07]} */
```

- Anexar una secuencia temporal a un valor temporal

  `appendSequence(ttype,ttypeSeq) → {ttypeSeq, ttypeSeqSet}`

  `appendSequence(ttypeSeq) → {ttypeSeq,ttypeSeqSet}`

  La primera versión de la función devuelve el resultado de agregar el segundo argumento al primero. Si cualquiera de las entradas es NULL, se devuelve NULL.

  La segunda versión de la función anterior es una función de *agregación* que devuelve el resultado de agregar sucesivamente un conjunto de filas de valores temporales. Esto significa que funciona de la misma manera que las funciones `SUM()` y `AVG()` y, como la mayoría de los agregados, también ignora los valores NULL.

```
SELECT appendSequence(tint '1@2001-01-01', tint '{2@2001-01-02, 3@2001-01-03}');
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03}
SELECT appendSequence(tint '[1@2001-01-01, 2@2001-01-02]',
  tint '[2@2001-01-02, 3@2001-01-03]');
-- [1@2001-01-01, 2@2001-01-02, 3@2001-01-03]
SELECT asText(appendSequence(tgeompoint '{[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02], [Point(3 3 3)@2001-01-04, Point(3 3 3)@2001-01-05]}',
  tgeompoint '[Point(3 3 3)@2001-01-05, Point(1 1 1)@2001-01-06]'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02],
   [POINT Z (3 3 3)@2001-01-04, POINT Z (3 3 3)@2001-01-05,
   POINT Z (1 1 1)@2001-01-06]} */
```

- Fusionar los valores temporales

  `merge(ttype,ttype) → ttype`

  `merge(ttype[]) → ttype`

  Los valores temporales solo pueden intersectar en su límite y en ese caso, los valores de base en las marcas de tiempo comunes deben ser los mismos; de lo contrario, se genera un error.

```
SELECT merge(tint '1@2001-01-01', tint '1@2001-01-02');
-- {1@2001-01-01, 1@2001-01-02}
SELECT merge(tint '[1@2001-01-01, 2@2001-01-02]', tint '[2@2001-01-02, 1@2001-01-03]');
-- [1@2001-01-01, 2@2001-01-02, 1@2001-01-03]
SELECT merge(tint '[1@2001-01-01, 2@2001-01-02]', tint '[3@2001-01-03, 1@2001-01-04]');
-- {[1@2001-01-01, 2@2001-01-02], [3@2001-01-03, 1@2001-01-04]}
SELECT merge(tint '[1@2001-01-01, 2@2001-01-02]', tint '[1@2001-01-02, 2@2001-01-03]');
-- ERROR:  The temporal values have different value at their common timestamp 2001-01-02
SELECT asText(merge(tgeompoint '{[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02], [Point(3 3 3)@2001-01-04, Point(3 3 3)@2001-01-05]}',
  tgeompoint '{[Point(3 3 3)@2001-01-05, Point(1 1 1)@2001-01-06]}'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02],
   [POINT Z (3 3 3)@2001-01-04, POINT Z (3 3 3)@2001-01-05,
   POINT Z (1 1 1)@2001-01-06]} */
```

```
SELECT merge(ARRAY[tint '1@2001-01-01', '1@2001-01-02']);
-- {1@2001-01-01, 1@2001-01-02}
SELECT merge(ARRAY[tint '{1@2001-01-01, 2@2001-01-02}', '{2@2001-01-02, 3@2001-01-03}']);
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03}
SELECT merge(ARRAY[tint '{1@2001-01-01, 2@2001-01-02}', '{3@2001-01-03, 4@2001-01-04}']);
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03, 4@2001-01-04}
SELECT merge(ARRAY[tint '[1@2001-01-01, 2@2001-01-02]', '[2@2001-01-02, 1@2001-01-03]']);
-- [1@2001-01-01, 2@2001-01-02, 1@2001-01-03]
SELECT merge(ARRAY[tint '[1@2001-01-01, 2@2001-01-02]', '[3@2001-01-03, 4@2001-01-04]']);
-- {[1@2001-01-01, 2@2001-01-02], [3@2001-01-03, 4@2001-01-04]}
SELECT asText(merge(ARRAY[tgeompoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02],
  [Point(3 3)@2001-01-03, Point(4 4)@2001-01-04]}', '{[Point(4 4)@2001-01-04,
  Point(3 3)@2001-01-05], [Point(6 6)@2001-01-06, Point(7 7)@2001-01-07]}']));
/* {[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02], [Point(3 3)@2001-01-03,
   Point(4 4)@2001-01-04, Point(3 3)@2001-01-05],
   [Point(6 6)@2001-01-06, Point(7 7)@2001-01-07]} */
```

```
SELECT asText(merge(ARRAY[tgeompoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02]}',
  '{[Point(2 2)@2001-01-02, Point(1 1)@2001-01-03]}']));
-- [Point(1 1)@2001-01-01, Point(2 2)@2001-01-02, Point(1 1)@2001-01-03]
```

6.2. Restricciones

Hay dos conjuntos complementarios de funciones de restricción. El primer conjunto de funciones restringe el valor temporal con respecto a un valor o una extensión de tiempo. Dos ejemplos son `atValues` o `atTime`. El segundo conjunto de funciones restringe el valor temporal con respecto al *complement* de un valor o una extensión de tiempo. Dos ejemplos son `minusValues` o `minusTime`

- Restringir a (al complemento de) un valor

  `atValues(ttype,values)` → `ttype`

  `minusValues(ttype,values)` → `ttype`

```
SELECT atValues(tint '[1@2001-01-01, 1@2001-01-15)', 1);
-- [1@2001-01-01, 1@2001-01-15)
SELECT asText(atValues(tgeompoint '[Point(0 0 0)@2001-01-01, Point(2 2 2)@2001-01-03)',
  geometry 'Point(1 1 1)'));
-- {[POINT Z (1 1 1)@2001-01-02]}
SELECT atValues(tfloat '[1@2001-01-01, 4@2001-01-4)', floatset '{1, 3, 5}');
-- {[1@2001-01-01], [3@2001-01-03]}
SELECT asText(atValues(tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-03)',
  geomset '{"Point(0 0)", "Point(1 1)"}'));
-- {[POINT(0 0)@2001-01-01], [POINT(1 1)@2001-01-02]}
SELECT atValues(tfloat '[1@2001-01-01, 4@2001-01-4)', floatspan '[1,3]');
-- [1@2001-01-01, 3@2001-01-03]
SELECT atValues(tfloat '[1@2001-01-01, 5@2001-01-05)',
  floatspanset '{[1,2], [3,4]}');
-- {[1@2001-01-01, 2@2001-01-02],[3@2001-01-03, 4@2001-01-04]}
```

```
SELECT minusValue(tint '[1@2001-01-01, 2@2001-01-02, 2@2001-01-03)', 1);
-- {[2@2001-01-02, 2@2001-01-03)}
SELECT asText(minusValue(tgeompoint '[Point(0 0 0)@2001-01-01, Point(2 2 2)@2001-01-03)',
  geometry 'Point(1 1 1)'));
/* {[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02),
   (POINT Z (1 1 1)@2001-01-02, POINT Z (2 2 2)@2001-01-03)} */
SELECT minusValues(tfloat '[1@2001-01-01, 4@2001-01-4)', floatset '{2, 3}');
/* {[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 3@2001-01-03),
   (3@2001-01-03, 4@2001-01-04)} */
SELECT asText(minusValues(tgeompoint '[Point(0 0 0)@2001-01-01, Point(3 3 3)@2001-01-04)',
  geomset '{"Point(1 1 1)", "Point(2 2 2)"}'));
/* {[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02),
   (POINT Z (1 1 1)@2001-01-02, POINT Z (2 2 2)@2001-01-03),
   (POINT Z (2 2 2)@2001-01-03, POINT Z (3 3 3)@2001-01-04)} */
SELECT minusValues(tfloat '[1@2001-01-01, 4@2001-01-4)', floatspan '[2,3]');
-- {[1@2001-01-01, 2@2001-01-02), (3@2001-01-03, 4@2001-01-04)}
SELECT minusValues(tfloat '[1@2001-01-01, 5@2001-01-05)',
  floatspanset '{[1,2], [3,4]}');
-- {(2@2001-01-02, 3@2001-01-03), (4@2001-01-04, 5@2001-01-05)}
```

- Restringir al (al complemento del) valor mínimo

  `atMin(torder)` → `torder`

  `minusMin(torder)` → `torder`

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

```
SELECT atMin(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}');
-- {1@2001-01-01, 1@2001-01-05}
SELECT atMin(tint '(1@2001-01-01, 3@2001-01-03]');
-- {(1@2001-01-01, 1@2001-01-03)}
SELECT atMin(tfloat '(1@2001-01-01, 3@2001-01-03]');
-- NULL
SELECT atMin(ttext '{(AA@2001-01-01, AA@2001-01-03), (BB@2001-01-03, AA@2001-01-05]}');
-- {(AA@2001-01-01, AA@2001-01-03), [AA@2001-01-05]}
```

```
SELECT minusMin(tint '{1@2001-01-01, 2@2001-01-03, 1@2001-01-05}');
-- {2@2001-01-03}
SELECT minusMin(tfloat '[1@2001-01-01, 3@2001-01-03]');
-- {(1@2001-01-01, 3@2001-01-03]}
SELECT minusMin(tfloat '(1@2001-01-01, 3@2001-01-03)');
-- {(1@2001-01-01, 3@2001-01-03)}
SELECT minusMin(tint '{[1@2001-01-01, 1@2001-01-03), (1@2001-01-03, 1@2001-01-05)}');
-- NULL
```

- Restringir al (al complemento del) valor máximo

  `atMax(torder) → torder`

  `minusMax(torder) → torder`

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

```
SELECT atMax(tint '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- {3@2001-01-05}
SELECT atMax(tfloat '(1@2001-01-01, 3@2001-01-03)');
-- NULL
SELECT atMax(tfloat '{(2@2001-01-01, 1@2001-01-03), [2@2001-01-03, 2@2001-01-05)}');
-- {[2@2001-01-03, 2@2001-01-05]}
SELECT atMax(ttext '{(AA@2001-01-01, AA@2001-01-03), (BB@2001-01-03, AA@2001-01-05]}');
-- {("BB"@2001-01-03, "BB"@2001-01-05)}
```

```
SELECT minusMax(tint '{1@2001-01-01, 2@2001-01-03, 3@2001-01-05}');
-- {1@2001-01-01, 2@2001-01-03}
SELECT minusMax(tfloat '[1@2001-01-01, 3@2001-01-03]');
-- {[1@2001-01-01, 3@2001-01-03)}
SELECT minusMax(tfloat '(1@2001-01-01, 3@2001-01-03)');
-- {(1@2001-01-01, 3@2001-01-03)}
SELECT minusMax(tfloat '{[2@2001-01-01, 1@2001-01-03), [2@2001-01-03, 2@2001-01-05)}');
-- {(2@2001-01-01, 1@2001-01-03)}
SELECT minusMax(tfloat '{[1@2001-01-01, 3@2001-01-03), (3@2001-01-03, 1@2001-01-05)}');
-- {[1@2001-01-01, 3@2001-01-03), (3@2001-01-03, 1@2001-01-05)}
```

- Restringir a (al complemento de) una geometría, un rango de Z y/o un período

  `atGeometry(tgeompoint,geometry[,zspan]) → tgeompoint`

  `minusGeometry(tgeompoint,geometry[,zspan]) → tgeompoint`

  `atGeometryTime(tgeompoint,geometry[,zspan],period) → tgeompoint`

  `minusGeometryTime(tgeompoint,geometry[,zspan],period) → tgeompoint`

  La geometría debe estar en 2D y el cálculo con respecto a ella se realiza en 2D. El resultado conserva la dimensión Z del punto temporal, si existe.

```
SELECT asText(atGeometry(tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-04)',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
-- {"[POINT(1 1)@2001-01-02, POINT(2 2)@2001-01-03]"}
SELECT astext(atGeometry(tgeompoint '[Point(0 0 0)@2001-01-01, Point(4 4 4)@2001-01-05]',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
```

```
-- {[POINT Z (1 1 1)@2001-01-02, POINT Z (2 2 2)@2001-01-03]}
SELECT asText(atGeometry(tgeompoint '[Point(1 1 1)@2001-01-01, Point(3 1 1)@2001-01-03,
  Point(3 1 3)@2001-01-05]', 'Polygon((2 0,2 2,2 4,4 0,2 0))', '[0,2]'));
-- {[POINT Z (2 1 1)@2001-01-02, POINT Z (3 1 1)@2001-01-03, POINT Z (3 1 2)@2001-01-04]}
```

```
SELECT asText(atGeometryTime(tgeompoint '[Point(1 1)@2001-01-01,
  Point(5 1)@2001-01-05, Point(1 1)@2001-01-09]',
  'Polygon((2 0,2 2,4 2,4 0,2 0))',
  tstzspan '[2001-01-03, 2001-01-05]'));
-- {[POINT(3 1)@2001-01-03, POINT(4 1)@2001-01-04]}
SELECT asText(atGeometryTime(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(5 1 5)@2001-01-05, Point(1 1 9)@2001-01-09]',
  geometry 'Polygon((2 0,2 2,4 2,4 0,2 0))', floatspan '[0,5]',
  tstzspan '[2001-01-03, 2001-01-06]'));
-- {[POINT Z (3 1 3)@2001-01-03, POINT Z (4 1 4)@2001-01-04]}
```

```
SELECT asText(minusGeometry(tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-04)',
  geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
/* {[POINT(0 0)@2001-01-01, POINT(1 1)@2001-01-02), (POINT(2 2)@2001-01-03,
  POINT(3 3)@2001-01-04)} */
SELECT astext(minusGeometry(tgeompoint '[Point(0 0 0)@2001-01-01,
  Point(4 4 4)@2001-01-05]', geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))'));
/* {[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02),
   (POINT Z (2 2 2)@2001-01-03, POINT Z (4 4 4)@2001-01-05]} */
SELECT asText(minusGeometry(tgeompoint '[Point(1 1 1)@2001-01-01, Point(3 1 1)@2001-01-03,
  Point(3 1 3)@2001-01-05]', 'Polygon((2 0,2 2,2 4,4 0,2 0))', '[0,2]'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 1 1)@2001-01-02),
    (POINT Z (3 1 2)@2001-01-04, POINT Z (3 1 3)@2001-01-05]} */
```

```
SELECT asText(minusGeometryTime(tgeompoint '[Point(1 1)@2001-01-01,
  Point(5 1)@2001-01-05, Point(1 1)@2001-01-09]',
  geometry 'Polygon((2 0,2 2,4 2,4 0,2 0))',
  tstzspan '[2001-01-03, 2001-01-05]'));
/* {[POINT(1 1)@2001-01-01, POINT(3 1)@2001-01-03), (POINT(4 1)@2001-01-04,
     POINT(5 1)@2001-01-05, POINT(1 1)@2001-01-09]} */
SELECT asText(minusGeometryTime(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(5 1 5)@2001-01-05, Point(1 1 9)@2001-01-09]',
  geometry 'Polygon((2 0,2 2,4 2,4 0,2 0))', floatspan '[0,5]',
  tstzspan '[2001-01-03, 2001-01-06]'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (3 1 3)@2001-01-03),
    (POINT Z (4 1 4)@2001-01-04, POINT Z (5 1 5)@2001-01-05,
     POINT Z (1 1 9)@2001-01-09]} */
```

- Restringir a (al complemento de) un valor de tiempo

  `atTime(ttype,times)` → `ttype`

  `minusTime(ttype,times)` → `ttype`

```
SELECT atTime(tfloat '[1@2001-01-01, 5@2001-01-05)', timestamptz '2001-01-02');
-- 2@2001-01-02
SELECT atTime(tint '[1@2001-01-01, 1@2001-01-15)',
  tstzset '{2001-01-01, 2001-01-03}');
-- {1@2001-01-01, 1@2001-01-03}
SELECT atTime(tfloat '{[1@2001-01-01, 3@2001-01-03), [3@2001-01-04, 1@2001-01-06)}',
  tstzspan '[2001-01-02,2001-01-05)');
-- {[2@2001-01-02, 3@2001-01-03), [3@2001-01-04, 2@2001-01-05)}
SELECT atTime(tint '[1@2001-01-01, 1@2001-01-15)',
  tstzspanset '{[2001-01-01, 2001-01-03), [2001-01-04, 2001-01-05)}');
-- {[1@2001-01-01, 1@2001-01-03),[1@2001-01-04, 1@2001-01-05)}
```

```
SELECT minusTime(tfloat '[1@2001-01-01, 5@2001-01-05)', timestamptz '2001-01-02');
-- {[1@2001-01-01, 2@2001-01-02), (2@2001-01-02, 5@2001-01-05)}
SELECT minusTime(tint '[1@2001-01-01, 1@2001-01-15)',
  tstzset '{2001-01-02, 2001-01-03}');
/* {[1@2001-01-01, 1@2001-01-02), (1@2001-01-02, 1@2001-01-03),
   (1@2001-01-03, 1@2001-01-15)} */
SELECT minusTime(tfloat '{[1@2001-01-01, 3@2001-01-03), [3@2001-01-04, 1@2001-01-06)}',
  tstzspan '[2001-01-02,2001-01-05)');
-- {[1@2001-01-01, 2@2001-01-02), [2@2001-01-05, 1@2001-01-06)}
SELECT minusTime(tint '[1@2001-01-01, 1@2001-01-15)',
  tstzspanset '{[2001-01-02, 2001-01-03), [2001-01-04, 2001-01-05)}');
/* {[1@2001-01-01, 1@2001-01-02), [1@2001-01-03, 1@2001-01-04),
   [1@2001-01-05, 1@2001-01-15)} */
```

- Restringir a (al complemento de) un `tbox`

  `atTbox(tnumber,tbox)` → `tnumber`

  `minusTbox(tnumber,tbox)` → `tnumber`

```
SELECT atTbox(tfloat '[0@2001-01-01, 3@2001-01-04)',
  tbox 'TBOXFLOAT XT((0,2),[2001-01-02, 2001-01-04])');
-- {[1@2001-01-02, 2@2001-01-03]}
```

```
SELECT minusTbox(tfloat '[1@2001-01-01, 4@2001-01-04)',
  'TBOXFLOAT XT((1,4),[2001-01-03, 2001-01-04])');
-- {[1@2001-01-01, 3@2001-01-03)}
WITH temp(temp, box) AS (
  SELECT tfloat '[1@2001-01-01, 4@2001-01-04)',
    tbox 'TBOXFLOAT XT((1,2),[2001-01-03, 2001-01-04])' )
SELECT minusValues(minusTime(temp, box::tstzspan), box::floatspan) FROM temp;
-- {[1@2001-01-01], [2@2001-01-02, 3@2001-01-03)}
```

Observe que cuando el cuadro delimitador tiene dimensiones de valor y de tiempo, la diferencia se calcula restringiendo el número temporal al cuadro usando la función `atTbox`, calculando la extensión de tiempo de esta restricción, y restringiendo el número temporal a la diferencia de la extensión del tiempo. Esto es necesario para garantizar que `tnumber = merge(atTbox(tnumber,tbox), minusTbox(tnumber,tbox))` (ver la función merge). En otras palabras, mientras que la función `atTbox` restringe el número temporal con respecto al rango *y* al período que definen el cuadro delimitador, la función `minusTbox` restringe el número temporal con respecto al rango *o* al período. Para obtener la restricción usando una semántica *y*, deben aplicarse las dos funciones `minusValues` y `minusTime`.

- Restringir a (al complemento de) un `stbox`

  `atStbox(tgeompoint,stbox)` → `tgeompoint`

  `minusStbox(tgeompoint,stbox)` → `tgeompoint`

```
SELECT asText(atStbox(tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-04)',
  stbox 'STBOX XT(((0,0),(2,2)),[2001-01-02, 2001-01-04])'));
-- {[POINT(1 1)@2001-01-02, POINT(2 2)@2001-01-03]}
SELECT asText(atStbox(tgeompoint '[Point(1 1 1)@2001-01-01, Point(3 3 3)@2001-01-03,
  Point(3 3 2)@2001-01-04, Point(3 3 7)@2001-01-09]', stbox 'STBOX Z((2,2,2),(3,3,3))'));
/* {[POINT Z (2 2 2)@2001-01-02, POINT Z (3 3 3)@2001-01-03, POINT Z (3 3 2)@2001-01-04,
  POINT Z (3 3 3)@2001-01-05]} */
```

```
SELECT asText(minusStbox(tgeompoint '[Point(1 1)@2001-01-01, Point(4 4)@2001-01-04)',
  stbox 'STBOX XT(((1,1),(4,4)),[2001-01-03,2001-01-04])'));
-- {[POINT(1 1)@2001-01-01), POINT(3 3)@2001-01-03)}
SELECT asText(minusStbox(tgeompoint '[Point(1 1 1)@2001-01-01, Point(3 3 3)@2001-01-03,
  Point(3 3 2)@2001-01-04, Point(3 3 7)@2001-01-09]', stbox 'STBOX Z((2,2,2),(3,3,3))'));
/* {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02), (POINT Z (3 3 3)@2001-01-05,
  POINT Z (3 3 7)@2001-01-09]} */
```

```
WITH temp(temp, box) AS (
  SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(4 4)@2001-01-04)',
    stbox 'STBOX XT(((1,1),(2,2)),[2001-01-03,2001-01-04])')
SELECT asText(minusGeometry(minusTime(temp, box::tstzspan), box::geometry)) FROM temp;
-- {(POINT(2 2)@2001-01-02, POINT(3 3)@2001-01-03)}
```

De manera similar a la función minusTbox, cuando el cuadro delimitador tiene dimensiones de espacio y tiempo, la función `minusStbox` restringe el punto temporal con respecto a las extensiones de espacio *o* de tiempo del cuadro. Para obtener la restricción utilizando una semántica *y*, deben aplicarse las dos funciones `minusGeometry` y `minusTime`.

6.3. Comparaciones

6.3.1. Comparaciones 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 2.10), después los cuadros delimitadores (ver la Sección 3.10) 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, 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@2001-01-01, 1@2001-01-04)' = tint '[2@2001-01-03, 2@2001-01-05)';
-- false
```

- ¿Son diferentes los valores temporales?

  `ttype <> ttype → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' <> tint '[2@2001-01-03, 2@2001-01-05)';
-- true
```

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

  `ttype < ttype → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' < tint '[2@2001-01-03, 2@2001-01-05)';
-- true
```

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

  `ttype > ttype → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' > tint '[2@2001-01-03, 2@2001-01-05)';
-- false
```

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

  `ttype <= ttype → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' <= tint '[2@2001-01-03, 2@2001-01-05)';
-- true
```

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

  `ttype >= ttype → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' >= tint '[2@2001-01-03, 2@2001-01-05)';
-- false
```

6.3.2. Comparaciones 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 or dos valores temporales 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 o siempre están disponibles para todos los tipos temporales, mientras que las desigualdades alguna vez o 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 o siempre igual al valor?

  `{base,ttype} ?= {base,ttype} → boolean`

  `{base,ttype} %= {base,ttype} → boolean`

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

```
SELECT tint '[1@2001-01-01, 3@2001-01-04]' ?= 2;
-- false
SELECT tfloat '[1@2001-01-01, 3@2001-01-04)' ?= 2;
-- true
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-04]' ?=
  geometry 'Point(2 2)';
-- true
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-04)' ?=
  geometry 'Point(2 2)';
-- false
```

```
SELECT tfloat '[1@2001-01-01, 1@2001-01-04)' %= 1;
-- true
SELECT tfloat '[1@2001-01-01, 3@2001-01-04)' %= 2;
-- false
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-04)' %=
  geometry 'Point(1 1)';
-- false
```

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

  `{base,ttype} ?<> {base,ttype} → boolean`

  `{base,ttype} %<> {base,ttype} → boolean`

```
SELECT tfloat '[1@2001-01-01, 3@2001-01-04)' ?<> 2;
-- true
SELECT tfloat '[2@2001-01-01, 2@2001-01-04)' ?<> 2;
-- false
SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(1 1)@2001-01-04)' ?<>
  geometry 'Point(1 1)';
-- false
```

```
SELECT tfloat '[1@2001-01-01, 3@2001-01-04)' %<> 2;
-- false
SELECT tfloat '[2@2001-01-01, 2@2001-01-04)' %<> 3;
-- true
SELECT tgeogpoint '[Point(1 1)@2001-01-01, Point(1 1)@2001-01-04)' %<>
  geography 'Point(2 2)';
-- true
```

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

  `{base,torder} ?< {base,torder} → boolean`

  `{base,torder}%< {base,torder} → boolean`

```
SELECT tint '[1@2001-01-01, 4@2001-01-04]' ?< 2;
-- true
```

```
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' %< 2;
-- false
```

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

  `{base,torder} ?> {base,torder} → boolean`

  `{base,torder}%> {base,torder} → boolean`

```
SELECT tint '[1@2001-01-03, 1@2001-01-05]' ?> 0;
-- true
```

```
SELECT tfloat '[1@2001-01-03, 1@2001-01-05)' %> 1;
-- false
```

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

  `{base,torder} ?<= {base,torder} → boolean`

  `{base,torder}%<= {base,torder} → boolean`

```
SELECT tint '[1@2001-01-01, 1@2001-01-05]' ?<= 2;
-- true
```

```
SELECT tfloat '[1@2001-01-01, 1@2001-01-05)' %<= 4;
-- true
```

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

  `{base,torder} ?>= {base,torder} → boolean`

  `{base,torder}%>= {base,torder} → boolean`

```
SELECT ttext '{[AAA@2001-01-01, AAA@2001-01-03), [BBB@2001-01-04, BBB@2001-01-05)}'
   ?> 'AAA'::text;
-- true
```

```
SELECT ttext '{[AAA@2001-01-01, AAA@2001-01-03), [BBB@2001-01-04, BBB@2001-01-05)}'
   %> 'AAA'::text;
-- false
```

6.3.3. Comparaciones 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@2001-01-01, 2@2001-01-04)' #= 3;
-- {[f@2001-01-01, f@2001-01-04)}
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' #= tfloat '[1@2001-01-01, 1@2001-01-04)';
-- {[t@2001-01-01], (f@2001-01-01, f@2001-01-04)}
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' #= tfloat '[4@2001-01-02, 1@2001-01-05)';
-- {[f@2001-01-02, t@2001-01-03], (f@2001-01-03, f@2001-01-04)}
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-03)' #=
  geometry 'Point(1 1)';
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-03)' #=
  tgeompoint '{[Point(0 2)@2001-01-01], (Point(0 0)@2001-01-01,
  Point(2 2)@2001-01-03)}';
-- {[f@2001-01-01], (t@2001-01-01, t@2001-01-03)}
```

- Diferente temporal

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

```
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' #<> 2;
-- {[t@2001-01-01, f@2001-01-02], (t@2001-01-02, 2001-01-04)}
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' #<> tfloat '[2@2001-01-02, 2@2001-01-05)';
-- {[f@2001-01-02], (t@2001-01-02, t@2001-01-04)}
```

- Menor que temporal

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

```
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' #< 2;
-- {[t@2001-01-01, f@2001-01-02, f@2001-01-04)}
SELECT tfloat '[2@2001-01-01, 2@2001-01-05)' #< tfloat '[1@2001-01-03, 3@2001-01-05)';
-- {[f@2001-01-03, f@2001-01-04], (t@2001-01-04, t@2001-01-05)}
```

- Mayor que temporal

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

```
SELECT 1 #> tint '[1@2001-01-03, 1@2001-01-05)';
-- [f@2001-01-03, f@2001-01-05)
```

- Menor o igual que temporal

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

```
SELECT tfloat '[1@2001-01-01, 1@2001-01-05)' #<= tfloat '{2@2001-01-03, 3@2001-01-04}';
-- {t@2001-01-03, t@2001-01-04}
```

- Mayor o igual que temporal

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

```
SELECT 'AAA'::text #> ttext '{[AAA@2001-01-01, AAA@2001-01-03),
  [BBB@2001-01-04, BBB@2001-01-05)}';
-- {[f@2001-01-01, f@2001-01-03), [t@2001-01-04, t@2001-01-05)}
```

6.4. 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 4, el cuadro delimitador asociado a un tipo temporal depende del tipo base: es el tipo `tstzspan` 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 3.3, 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: superposición (`&&`), 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 3.9 para los operadores de cuadro delimitador.

6.5. Funciones de utilidad

- Versión de la extensión MobilityDB

  `mobilitydb_version() → text`

```
SELECT mobilitydb_version();
-- MobilityDB 1.1.0
```

- Versión de la extensión MobilityDB y de sus dependencias

  `mobilitydb_full_version() → text`

```
SELECT mobilitydb_full_version();
-- MobilityDB 1.1.0, PostgreSQL 16.1, PostGIS 3.4.0, GEOS 3.12.0-CAPI-1.18.0, PROJ 9.2.0
```

Capítulo 7

Tipos temporales: Operaciones alfanuméricas

7.1. Operaciones booleanos

- Y temporal

  `{boolean,tbool} & {boolean,tbool} → tbool`

```
SELECT tbool '[true@2001-01-03, true@2001-01-05)' &
  tbool '[false@2001-01-03, false@2001-01-05)';
-- [f@2001-01-03, f@2001-01-05)
SELECT tbool '[true@2001-01-03, true@2001-01-05)' &
  tbool '{[false@2001-01-03, false@2001-01-04),
  [true@2001-01-04, true@2001-01-05)}';
-- {[f@2001-01-03, t@2001-01-04, t@2001-01-05)}
```

- O temporal

  `{boolean,tbool} | {boolean,tbool} → tbool`

```
SELECT tbool '[true@2001-01-03, true@2001-01-05)' |
  tbool '[false@2001-01-03, false@2001-01-05)';
-- [t@2001-01-03, t@2001-01-05)
```

- No temporal

  `~tbool → tbool`

```
SELECT ~tbool '[true@2001-01-03, true@2001-01-05)';
-- [f@2001-01-03, f@2001-01-05)
```

- Obtener el tiempo cuando el booleano temporal toma el valor verdadero

  `whenTrue(tbool) → tstzspanset`

```
SELECT whenTrue(tfloat '[1@2001-01-01, 4@2001-01-04, 1@2001-01-07]' #> 2);
-- {(2001-01-02, 2001-01-06)}
SELECT whenTrue(tdwithin(tgeompoint '[Point(1 1)@2001-01-01, Point(4 4)@2001-01-04,
  Point(1 1)@2001-01-07]', geometry 'Point(1 1)', sqrt(2)));
-- {[2001-01-01, 2001-01-02], [2001-01-06, 2001-01-07]}
```

7.2. Operaciones matemáticas

- Adición temporal

  `{number,tnumber} + {number,tnumber} → tnumber`

```
SELECT tint '[2@2001-01-01, 2@2001-01-04)' + 1;
-- [3@2001-01-01, 3@2001-01-04)
SELECT tfloat '[2@2001-01-01, 2@2001-01-04)' + tfloat '[1@2001-01-01, 4@2001-01-04)';
-- [3@2001-01-01, 6@2001-01-04)
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' +
  tfloat '{[1@2001-01-01, 2@2001-01-02), [1@2001-01-02, 2@2001-01-04)}';
-- {[2@2001-01-01, 4@2001-01-04), [3@2001-01-02, 6@2001-01-04)}
```

- Resta temporal

  `{number,tnumber} - {number,tnumber} → tnumber`

```
SELECT tint '[1@2001-01-01, 1@2001-01-04)' - tint '[2@2001-01-03, 2@2001-01-05)';
-- [-1@2001-01-03, -1@2001-01-04)
SELECT tfloat '[3@2001-01-01, 6@2001-01-04)' - tfloat '[2@2001-01-01, 2@2001-01-04)';
-- [1@2001-01-01, 4@2001-01-04)
```

- Multiplicación temporal

  `{number,tnumber} * {number,tnumber} → tnumber`

```
SELECT tint '[1@2001-01-01, 4@2001-01-04]' * 2;
-- [2@2001-01-01, 8@2001-01-04]
SELECT tfloat '[1@2001-01-01, 4@2001-01-04)' * tfloat '[2@2001-01-01, 2@2001-01-04)';
-- [2@2001-01-01, 8@2001-01-04)
SELECT tfloat '[1@2001-01-01, 3@2001-01-03)' * '[3@2001-01-01, 1@2001-01-03)';
-- {[3@2001-01-01, 4@2001-01-02, 3@2001-01-03)}
```

- 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@2001-01-01, 3@2001-01-04)';
-- [2@2001-01-01, 0.666666666666667@2001-01-04)
SELECT tfloat '[1@2001-01-01, 5@2001-01-05)' / tfloat '[5@2001-01-01, 1@2001-01-05)';
-- {[0.2@2001-01-01, 1@2001-01-03,2001-01-03, 5@2001-01-03,2001-01-05)}
SELECT 2 / tfloat '[-1@2001-01-01, 1@2001-01-02]';
-- ERROR:  Division by zero
SELECT tfloat '[-1@2001-01-04, 1@2001-01-05]' / tfloat '[-1@2001-01-01, 1@2001-01-05]';
-- [-2@2001-01-04, 1@2001-01-05]
```

- Obtener el valor absoluto del número temporal

  `abs(tnumber) → tnumber`

```
SELECT abs(tfloat '[1@2001-01-01, -1@2001-01-03, 1@2001-01-05]');
-- [1@2001-01-01, 0@2001-01-02, 1@2001-01-03, 0@2001-01-04, 1@2001-01-05],
SELECT abs(tint '[1@2001-01-01, -1@2001-01-03, 1@2001-01-05]');
-- [1@2001-01-01, 1@2001-01-05]
```

- Obtener la diferencia de valor entre instantes consectivos del número temporal

  `deltaValue(tnumber) → tnumber`

```
SELECT deltaValue(tint '[1@2001-01-01, 2@2001-01-02, 1@2001-01-03]');
-- [1@2001-01-01, -1@2001-01-02, -1@2001-01-03)
SELECT deltaValue(tfloat '{[1.5@2001-01-01, 2@2001-01-02, 1@2001-01-03],
  [2@2001-01-04, 2@2001-01-05]}');
/* Interp=Step;{[0.5@2001-01-01, -1@2001-01-02, -1@2001-01-03),
  [0@2001-01-04, 0@2001-01-05)} */
```

- Redondear los valores a un número de posiciones decimales

  `round(tfloat,integer=0) → tfloat`

```
SELECT round(tfloat '[0.785398163397448@2001-01-01, 2.356194490192345@2001-01-02]', 2);
-- [0.79@2001-01-01, 2.36@2001-01-02]
```

- Convertir a grados o radianes

  `degrees({float,tfloat},normalize=false) → tfloat`

  `radians(tfloat) → tfloat`

  El parámetro adicional en la función `degrees` puede ser utilizado para normalizar los valores entre 0 y 360 grados.

```
SELECT degrees(pi() * 5);
-- 900
SELECT degrees(pi() * 5, true);
-- 180
SELECT round(degrees(tfloat '[0.785398163397448@2001-01-01, 2.356194490192345@2001-01-02]' ↩
    ));
-- [45@2001-01-01, 135@2001-01-02]
SELECT radians(tfloat '[45@2001-01-01, 135@2001-01-02]');
-- [0.785398163397448@2001-01-01, 2.356194490192345@2001-01-02]
```

- 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@2001-01-01, 10@2001-01-02, 5@2001-01-03],
  [1@2001-01-04, 0@2001-01-05]}') * 3600 * 24;
/* Interp=Step;{[-10@2001-01-01, 5@2001-01-02, 5@2001-01-03],
  [1@2001-01-04, 1@2001-01-05]} */
SELECT derivative(tfloat 'Interp=Step;[0@2001-01-01, 10@2001-01-02, 5@2001-01-03]');
-- ERROR:  The temporal value must have linear interpolation
```

7.3. Operaciones de texto

- Concatenación de texto

  `{text,ttext} || {text,ttext} → ttext`

```
SELECT ttext '[AA@2001-01-01, AA@2001-01-04)' || text 'B';
-- ["AAB"@2001-01-01, "AAB"@2001-01-04)
SELECT ttext '[AA@2001-01-01, AA@2001-01-04)' || ttext '[BB@2001-01-02, BB@2001-01-05)';
-- ["AABB"@2001-01-02, "AABB"@2001-01-04)
SELECT ttext '[A@2001-01-01, B@2001-01-03, C@2001-01-04]' ||
  ttext '{[D@2001-01-01, D@2001-01-02), [E@2001-01-02, E@2001-01-04)}';
-- {["AD"@2001-01-01, "AE"@2001-01-02, "BE"@2001-01-03, "BE"@2001-01-04)}
```

- Transformar en minúsculas, mayúsculas o initcap

  `lower(ttext) → ttext`

  `upper(ttext) → ttext`

  `initcap(ttext) → ttext`

```
SELECT upper(ttext '[AA@2001-01-01, bb@2001-01-02]');
-- ["AA"@2001-01-01, "BB"@2001-01-02]
SELECT lower(ttext '[AA@2001-01-01, bb@2001-01-02]');
-- ["aa"@2001-01-01, "bb"@2001-01-02]
SELECT initcap(ttext '[AA@2001-01-01, bb@2001-01-02]');
-- ["aa"@2001-01-01, "bb"@2001-01-02]
```

Capítulo 8

Tipos temporales: Operaciones espaciales

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

8.1. Entrada y 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)@2001-01-01, Point(1 1 1)@2001-01-02)');
-- [POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02)
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)@2001-01-01, Point(1 1 1)@2001-01-02)');
-- SRID=4326;[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02)
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 características móviles (Moving Features JSON o MF-JSON) ⬚ 🌐

  `asMFJSON(tpoint,options=0,flags=0,maxdecdigits=15) → text`

  El 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)

  El argumento `flags` puede usarse para personalizar la salida JSON, por ejemplo, para producir una salida JSON fácil de leer (para lectores humanos). Consulte la documentación de la biblioteca `json-c` para conocer los valores posibles. Los valores típicos son los siguientes:

  - 0: means no option (default value)
  - 1: JSON_C_TO_STRING_SPACED
  - 2: JSON_C_TO_STRING_PRETTY

El argumento `maxdecdigits` puede usarse para establecer el número máximo de decimales en la salida de los valores en punto flotante (por defecto 15).

```
SELECT asMFJSON(tgeompoint 'Point(1 2)@2001-01-01 18:00:00.15+02');
/* {"type":"MovingPoint","coordinates":[[1,2]],"datetimes":["2001-01-01T17:00:00.15+01"],
   "interpolation":"None"} */
SELECT asMFJSON(tgeompoint 'SRID=4326;
  Point(50.813810 4.384260)@2001-01-01 18:00:00.15+02', 3, 2);
/* {"type":"MovingPoint","crs":{"type":"name","properties":{"name":"EPSG:4326"}},
   "stBoundedBy":{"bbox":[50.81,4.38,50.81,4.38],
   "period":{"begin":"2001-01-01 17:00:00.15+01","end":"2001-01-01 17:00:00.15+01"}},
   "coordinates":[[50.81,4.38]],"datetimes":["2001-01-01T17:00:00.15+01"],
   "interpolation":"None"} */
```

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

  `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)@2001-01-01');
-- \x012e0011000000000000f03f0000000000000040000000000000000840009c57d3c11c0000
```

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

  `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)@2001-01-01');
-- \x012f0071a41e000000000000000f03f0000000000000040000000000000000840009c57d3c11c0000
```

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

  `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 la codificación de la máquina.

```
SELECT asHexEWKB(tgeompoint 'SRID=3812;Point(1 2 3)@2001-01-01');
-- 012E0051E40E000000000000000F03F0000000000000040000000000000000840009C57D3C11C0000
```

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

  `tgeompointFromText(text) → tgeompoint`

  `tgeogpointFromText(text) → tgeogpoint`

```
SELECT asEWKT(tgeompointFromText(text '[POINT(1 2)@2001-01-01, POINT(3 4)@2001-01-02]'));
-- [POINT(1 2)@2001-01-01, POINT(3 4)@2001-01-02]
SELECT asEWKT(tgeogpointFromText(text '[POINT(1 2)@2001-01-01, POINT(3 4)@2001-01-02]'));
-- SRID=4326;[POINT(1 2)@2001-01-01, POINT(3 4)@2001-01-02]
```

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

  `tgeompointFromEWKT(text) → tgeompoint`

  `tgeogpointFromEWKT(text) → tgeogpoint`

```
SELECT asEWKT(tgeompointFromEWKT(text 'SRID=3812;[POINT(1 2)@2001-01-01,
  POINT(3 4)@2001-01-02]'));
-- SRID=3812;[POINT(1 2)@2001-01-01 00:00:00+01, POINT(3 4)@2001-01-02 00:00:00+01]
SELECT asEWKT(tgeogpointFromEWKT(text 'SRID=7844;[POINT(1 2)@2001-01-01,
  POINT(3 4)@2001-01-02]'));
-- SRID=7844;[POINT(1 2)@2001-01-01, POINT(3 4)@2001-01-02]
```

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

  `tgeompointFromMFJSON(text)` → `tgeompoint`

  `tgeogpointFromMFJSON(text)` → `tgeogpoint`

```
SELECT asEWKT(tgeompointFromMFJSON(text '{"type":"MovingGeomPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[[50.81,4.38]],
  "datetimes":["2001-01-01T17:00:00.15+01"],"interpolation":"None"}'));
-- SRID=4326;POINT(50.81 4.38)@2001-01-01 17:00:00.15+01
SELECT asEWKT(tgeogpointFromMFJSON(text '{"type":"MovingGeogPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[[50.81,4.38]],
  "datetimes":["2001-01-01T17:00:00.15+01"],"interpolation":"None"}'));
-- SRID=4326;POINT(50.81 4.38)@2001-01-01 17:00:00.15+01
```

- Entrar un punto temporal en una representación binaria conocida (WKB)

  `tgeompointFromBinary(bytea)` → `tgeompoint`

  `tgeogpointFromBinary(bytea)` → `tgeogpoint`

```
SELECT asEWKT(tgeompointFromBinary(
  '\x012e0011000000000000f03f0000000000000040000000000000000840009c57d3c11c0000'));
-- POINT Z (1 2 3)@2001-01-01
```

- Entrar un punto temporal en una representación extendida binaria conocida (EWKB)

  `tgeompointFromEWKB(bytea)` → `tgeompoint`

  `tgeogpointFromEWKB(bytea)` → `tgeogpoint`

```
SELECT asEWKT(tgeogpointFromEWKB(
  '\x012f0071a41e0000000000000000f03f0000000000000040000000000000000840009c57d3c11c0000'));
-- SRID=7844;POINT Z (1 1 1)@2001-01-01
```

- Entrar un punto temporal geométrico en una representación hexadecimal extendida binaria conocida (HexEWKB)

  `tgeompointFromHexEWKB(text)` → `tgeompoint`

  `tgeogpointFromHexEWKB(text)` → `tgeogpoint`

```
SELECT asEWKT(tgeompointFromHexEWKB(
  '012E0051E40E0000000000000000F03F0000000000000040000000000000000840009C57D3C11C0000'));
-- SRID=3812;POINT(1 2 3)@2001-01-01
```

8.2. Sistema de referencia espacial

- Obtener el identificador de referencia espacial

  `SRID(tpoint)` → `integer`

```
SELECT SRID(tgeompoint 'Point(0 0)@2001-01-01');
-- 0
```

- Establecer el identificador de referencia espacial

  `setSRID(tpoint) → tpoint`

```
SELECT asEWKT(setSRID(tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02)', 4326));
-- SRID=4326;[POINT(0 0)@2001-01-01 00:00:00+00, POINT(1 1)@2001-01-02 00:00:00+00)
```

- Transformar a una referencia espacial diferente

  `transform(tpoint,to_srid integer) → tpoint`

  `transform(tpoint,pipeline text,to_srid integer,is_forward bool=true) → tpoint`

  La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando el punto temporal tiene un SRID desconocido (representado por 0).

  La función `transformPipeline` especifica la transformación con una canalización de transformación de coordenadas definida representada con el siguiente formato de cadena: `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. El SRID del punto temporal de entrada se ignora y el SRID del punto temporal de salida se establecerá en cero a menos que se proporcione un valor a través del parámetro opcional `to_srid`. Como se indica en el último parámetro, la canalización se ejecuta de forma predeterminada en dirección hacia adelante; al establecer el parámetro en falso, la canalización se ejecuta en la dirección inversa.

```
SELECT asEWKT(transform(tgeompoint 'SRID=4326;Point(4.35 50.85)@2001-01-01', 3812));
-- SRID=3812;POINT(648679.018035303 671067.055638114)@2001-01-01 00:00:00+00
```

```
WITH test(tpoint, pipeline) AS (
  SELECT tgeogpoint 'SRID=4326;{Point(4.3525 50.846667 100.0)@2001-01-01,
    Point(-0.1275 51.507222 100.0)@2001-01-02}',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(tpoint, pipeline, 4326),
  pipeline, 4326, false), 6)
FROM test;
/* SRID=4326;{POINT Z (4.3525 50.846667 100)@2001-01-01,
   POINT Z (-0.1275 51.507222 100)@2001-01-02} */
```

8.3. Operaciones de cuadro delimitador

- Obtener el cuadro delimitador espaciotemporal expandido en la dimensión espacial por un valor flotante

  `expandSpace({geo,tpoint},float) → stbox`

```
SELECT expandSpace(geography 'Linestring(0 0,1 1)', 2);
-- SRID=4326;GEODSTBOX X((-2,-2),(3,3))
SELECT expandSpace(tgeompoint 'Point(0 0)@2001-01-01', 2);
-- STBOX XT(((-2,-2),(2,2)),[2001-01-01,2001-01-01])
```

8.4. Accessores

- Obtenir la trayectoria

  `trajectory(tpoint) → geo`

  Esta función es equivalente a getValues para los valores temporales alphanuméricos

```
SELECT ST_AsText(trajectory(tgeompoint '{[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02),
  [Point(0 1)@2001-01-03, Point(1 1)@2001-01-04)}'));
-- MULTILINESTRING((0 0,0 1),(0 1,1 1))
SELECT ST_AsText(trajectory(tgeompoint 'Interp=Step;{[Point(0 0)@2001-01-01,
```

```
  Point(0 1)@2001-01-02], [Point(0 1)@2001-01-03, Point(1 1)@2001-01-04]}'));
-- MULTIPOINT((0 0),(0 1),(0 1),(1 1))
SELECT ST_AsText(trajectory(tgeompoint '{Point(0 0)@2001-01-01, Point(0 1)@2001-01-02}'));
-- MULTIPOINT((0 0),(0 1))
SELECT ST_AsText(trajectory(tgeompoint '{[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02),
  [Point(1 1)@2001-01-03, Point(1 1)@2001-01-04),
  [Point(2 1)@2001-01-05, Point(2 2)@2001-01-06)}'));
-- GEOMETRYCOLLECTION(POINT(1 1),MULTILINESTRING((0 0,0 1),(2 1,2 2)))
```

- Obtener los valores de las coordenadas X/Y/Z como un número flotante temporal

  `getX(tpoint) → tfloat`

  `getY(tpoint) → tfloat`

  `getZ(tpoint) → tfloat`

```
SELECT getX(tgeompoint '{Point(1 2)@2001-01-01, Point(3 4)@2001-01-02,
  Point(5 6)@2001-01-03}');
-- {1@2001-01-01, 3@2001-01-02, 5@2001-01-03}
SELECT getX(tgeogpoint 'Interp=Step;[Point(1 2 3)@2001-01-01, Point(4 5 6)@2001-01-02,
  Point(7 8 9)@2001-01-03]');
-- Interp=Step;[1@2001-01-01, 4@2001-01-02, 7@2001-01-03]
SELECT getY(tgeompoint '{Point(1 2)@2001-01-01, Point(3 4)@2001-01-02,
  Point(5 6)@2001-01-03}');
-- {2@2001-01-01, 4@2001-01-02, 6@2001-01-03}
SELECT getY(tgeogpoint 'Interp=Step;[Point(1 2 3)@2001-01-01, Point(4 5 6)@2001-01-02,
  Point(7 8 9)@2001-01-03]');
-- Interp=Step;[2@2001-01-01, 5@2001-01-02, 8@2001-01-03]
SELECT getZ(tgeompoint '{Point(1 2)@2001-01-01, Point(3 4)@2001-01-02,
  Point(5 6)@2001-01-03}');
-- The temporal point must have Z dimension
SELECT getZ(tgeogpoint 'Interp=Step;[Point(1 2 3)@2001-01-01, Point(4 5 6)@2001-01-02,
  Point(7 8 9)@2001-01-03]');
-- Interp=Step;[3@2001-01-01, 6@2001-01-02, 9@2001-01-03]
```

- Obtener 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)@2001-01-01, Point(1 1)@2001-01-02,
  Point(0 0)@2001-01-03]');
-- false
SELECT isSimple(tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02,
  Point(2 0 2)@2001-01-03, Point(0 0 0)@2001-01-04]');
-- false
SELECT isSimple(tgeompoint '{[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02],
  [Point(1 1 1)@2001-01-03, Point(0 0 0)@2001-01-04]}');
-- true
```

- Obtener la longitud atravesada por el punto temporal

  `length(tpoint) → float`

```
SELECT length(tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02]');
-- 1.73205080756888
SELECT length(tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02,
  Point(0 0 0)@2001-01-03]');
-- 3.46410161513775
SELECT length(tgeompoint 'Interp=Step;[Point(0 0 0)@2001-01-01,
  Point(1 1 1)@2001-01-02, Point(0 0 0)@2001-01-03]');
-- 0
```

- Obtener la longitud acumulada atravesada por el punto temporal

  `cumulativeLength(tpoint) → tfloatSeq`

```
SELECT round(cumulativeLength(tgeompoint '{[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(1 0)@2001-01-03], [Point(1 0)@2001-01-04, Point(0 0)@2001-01-05]}'), 6);
-- {[0@2001-01-01, 1.414214@2001-01-02, 2.414214@2001-01-03],
  [2.414214@2001-01-04, 3.414214@2001-01-05]}
SELECT cumulativeLength(tgeompoint 'Interp=Step;[Point(0 0 0)@2001-01-01,
  Point(1 1 1)@2001-01-02, Point(0 0 0)@2001-01-03]');
-- Interp=Step;[0@2001-01-01, 0@2001-01-03]
```

- Obtener la velocidad del punto temporal en unidades por segundo

  `speed(tpoint) → tfloatSeqSet`

  El punto temporal debe tener interpolación linear

```
SELECT speed(tgeompoint '{[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(1 0)@2001-01-03], [Point(1 0)@2001-01-04, Point(0 0)@2001-01-05]}') * 3600 * 24;
/* Interp=Step;{[1.4142135623731@2001-01-01, 1@2001-01-02, 1@2001-01-03],
   [1@2001-01-04, 1@2001-01-05]} */
SELECT speed(tgeompoint 'Interp=Step;[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(1 0)@2001-01-03]');
-- 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)@2001-01-01,
  Point(0 1 1)@2001-01-02, Point(0 1 1)@2001-01-03, Point(0 0 0)@2001-01-04)}'));
-- POINT Z (0 0.666666666666667 0.666666666666667)
```

- Obtener la dirección, es decir, el acimut entre las ubicaciones inicial y final

  `direction(tpoint) → float`

  El resultado se expresa en radianes. Es NULL si solo hay una ubicación o si las ubicaciones inicial y final son iguales.

```
SELECT round(degrees(direction(tgeompoint '[Point(0 0)@2001-01-01,
  Point(-1 -1)@2001-01-02, Point(1 1)@2001-01-03]'))::numeric, 6);
-- 45.000000
SELECT direction(tgeompoint '{[Point(0 0 0)@2001-01-01,
  Point(0 1 1)@2001-01-02, Point(0 1 1)@2001-01-03, Point(0 0 0)@2001-01-04)}');
-- NULL
```

- Obtener el acimut temporal

  `azimuth(tpoint) → tfloat`

  El resultado se expresa en radianes. El azimut es indefinido cuando dos localizaciones sucesivas son iguales y en este caso se añade una brecha de tempo.

```
SELECT round(degrees(azimuth(tgeompoint '[Point(0 0 0)@2001-01-01,
  Point(1 1 1)@2001-01-02, Point(1 1 1)@2001-01-03, Point(0 0 0)@2001-01-04)')));
-- Interp=Step;{[45@2001-01-01, 45@2001-01-02], [225@2001-01-03, 225@2001-01-04)}
```

- Obtener la diferencia angular temporal

  `angularDifference(tpoint) → tfloat`

  El resultado se expresa en grados.

```
SELECT round(angularDifference(tgeompoint '[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02,
  Point(1 1)@2001-01-03]'), 3);
-- {0@2001-01-01, 180@2001-01-02, 0@2001-01-03}
SELECT round(degrees(angularDifference(tgeompoint '{[Point(1 1)@2001-01-01,
  Point(2 2)@2001-01-02], [Point(2 2)@2001-01-03, Point(1 1)@2001-01-04]}')), 3);
-- {0@2001-01-01, 0@2001-01-02, 0@2001-01-03, 0@2001-01-04}
```

- Obtener el rumbo temporal

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

  Nótese que esta función no acepta dos puntos geográficos temporales.

```
SELECT degrees(bearing(tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]',
  geometry 'Point(2 2)'));
-- [45@2001-01-01, 0@2001-01-02, 225@2001-01-03]
SELECT round(degrees(bearing(tgeompoint '[Point(0 0)@2001-01-01, Point(2 0)@2001-01-03]',
  tgeompoint '[Point(2 1)@2001-01-01, Point(0 1)@2001-01-03]')), 3);
--  [63.435@2001-01-01, 0@2001-01-02, 296.565@2001-01-03]
SELECT round(degrees(bearing(tgeompoint '[Point(2 1)@2001-01-01, Point(0 1)@2001-01-03]',
  tgeompoint '[Point(0 0)@2001-01-01, Point(2 0)@2001-01-03]')), 3);
-- [243.435@2001-01-01, 116.565@2001-01-03]
```

8.5. Transformaciones

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

  `round(tpoint,integer=0) → tpoint`

```
SELECT asText(round(tgeompoint '{Point(1.12345 1.12345 1.12345)@2001-01-01,
  Point(2 2 2)@2001-01-02, Point(1.12345 1.12345 1.12345)@2001-01-03}', 2));
/* {POINT Z (1.12 1.12 1.12)@2001-01-01, POINT Z (2 2 2)@2001-01-02,
   POINT Z (1.12 1.12 1.12)@2001-01-03} */
SELECT asText(round(tgeogpoint 'Point(1.12345 1.12345)@2001-01-01', 2));
-- POINT(1.12 1.12)@2001-01-01
```

- Obtener una matriz de fragmentos del punto temporal que son simples

  `makeSimple(tpoint) → tgeompoint[]`

```
SELECT asText(makeSimple(tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(0 0)@2001-01-03]'));
/* {"[POINT(0 0)@2001-01-01, POINT(1 1)@2001-01-02)",
   "[POINT(1 1)@2001-01-02, POINT(0 0)@2001-01-03]"} */
SELECT asText(makeSimple(tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02,
  Point(2 0 2)@2001-01-03, Point(0 0 0)@2001-01-04]'));
/* {"[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02, POINT Z (2 0 2)@2001-01-03,
   POINT Z (0 0 0)@2001-01-04]"} */
SELECT asText(makeSimple(tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02,
  Point(0 1)@2001-01-03, Point(1 0)@2001-01-04]'));
/* {"[POINT(0 0)@2001-01-01, POINT(1 1)@2001-01-02, POINT(0 1)@2001-01-03)",
   "[POINT(0 1)@2001-01-03, POINT(1 0)@2001-01-04]"} */
SELECT asText(makeSimple(tgeompoint '{[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-02],
  [Point(1 1 1)@2001-01-03, Point(0 0 0)@2001-01-04]}'));
/* {"{[POINT Z (0 0 0)@2001-01-01, POINT Z (1 1 1)@2001-01-02],
   [POINT Z (1 1 1)@2001-01-03, POINT Z (0 0 0)@2001-01-04]}"} */
```

- 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)@2001-01-01,
  Point(2 2 2)@2001-01-02}', '{5@2001-01-01, 5@2001-01-02}'));
-- MULTIPOINT ZM (1 1 1 5,2 2 2 5)
SELECT st_astext(geoMeasure(tgeogpoint '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02],
  [Point(1 1)@2001-01-03, Point(1 1)@2001-01-04]}',
  '{[5@2001-01-01, 5@2001-01-02],[7@2001-01-03, 7@2001-01-04]}'));
-- GEOMETRYCOLLECTION M (POINT M (1 1 7),LINESTRING M (1 1 5,2 2 5))
SELECT st_astext(geoMeasure(tgeompoint '[Point(1 1)@2001-01-01,
  Point(2 2)@2001-01-02, Point(1 1)@2001-01-03]',
  '[5@2001-01-01, 7@2001-01-02, 5@2001-01-03]', 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 8.1 muestra el resultado de la consulta a continuación usando una rampa de color en QGIS.

```
WITH Temp(t) AS (
  SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-05,
    Point(2 0)@2001-01-08, Point(3 1)@2001-01-10, Point(4 0)@2001-01-11]'
)
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)
  )
```

Figura 8.1: Visualización de la velocidad de un objeto móvil usando una rampa de color en QGIS.

- Transformar un punto geométrico temporal en el espacio de coordenadas de un Mapbox Vector Tile

  `asMVTGeom(tpoint,bounds,extent=4096,buffer=256,clip=true) → (geom,times)`

  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. Los parámetros son los siguientes:

  - `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)@2001-01-01, Point(100 100)@2001-01-02]',
  stbox 'STBOX X((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)@2001-01-01, Point(100 100)@2001-01-02]',
  stbox 'STBOX X((40,40),(60,60))', clip:=false) AS mvt ) AS t;
-- LINESTRING(-8192 12288,12288 -8192) | {946681200,946767600}
```

8.6. Operaciones de distancia

- Obtener la distancia más pequeña que haya existido

  `{geo,tpoint} |=| {geo,tpoint} → float`

```
SELECT tgeompoint '[Point(0 0)@2001-01-02, Point(1 1)@2001-01-04, Point(0 0)@2001-01-06)'
  |=| geometry 'Linestring(2 2,2 1,3 1)';
-- 1
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-03, Point(0 0)@2001-01-05)'
  |=| tgeompoint '[Point(2 0)@2001-01-02, Point(1 1)@2001-01-04, Point(2 2)@2001-01-06)';
-- 0.5
SELECT tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-03,
  Point(0 0 0)@2001-01-05)' |=| tgeompoint '[Point(2 0 0)@2001-01-02,
  Point(1 1 1)@2001-01-04, Point(2 2 2)@2001-01-06)';
-- 0.5
SELECT tgeompoint 'Interp=Step;(Point(1 1)@2001-01-01, Point(3 1)@2001-01-03]' |=|
  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 o SP-GIST (ver la Sección 10.2). 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)@2001-01-01, Point(1 1 1)@2001-01-03,
    Point(0 0 0)@2001-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04,
    Point(2 2 2)@2001-01-06)'::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)@2001-01-01,
  Point(3 1)@2001-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- POINT(2 1)@2001-01-02
SELECT asText(NearestApproachInstant(tgeompoint 'Interp=Step;(Point(1 1)@2001-01-01,
  Point(3 1)@2001-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- POINT(1 1)@2001-01-01
SELECT asText(NearestApproachInstant(tgeompoint '(Point(1 1)@2001-01-01,
  Point(2 2)@2001-01-03]', tgeompoint '(Point(1 1)@2001-01-01, Point(4 1)@2001-01-03]'));
-- POINT(1 1)@2001-01-01
SELECT asText(nearestApproachInstant(tgeompoint '[Point(0 0 0)@2001-01-01,
  Point(1 1 1)@2001-01-03, Point(0 0 0)@2001-01-05)', tgeompoint
```

```
  '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04, Point(2 2 2)@2001-01-06)'));
-- POINT Z (0.75 0.75 0.75)@2001-01-03 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)@2001-01-01, Point(1 1 1)@2001-01-03,
    Point(0 0 0)@2001-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04,
    Point(2 2 2)@2001-01-06)'::geometry));
-- 2001-01-03 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)@2001-01-01,
  Point(3 1)@2001-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- LINESTRING(2 1,2 2)
SELECT ST_AsText(shortestLine(tgeompoint 'Interp=Step;(Point(1 1)@2001-01-01,
  Point(3 1)@2001-01-03]', geometry 'Linestring(1 3,2 2,3 3)'));
-- LINESTRING(1 1,2 2)
SELECT ST_AsText(shortestLine(
  tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-03,
    Point(0 0 0)@2001-01-05)',
  tgeompoint '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04,
    Point(2 2 2)@2001-01-06)'));
-- 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)@2001-01-01, Point(1 1 1)@2001-01-03,
    Point(0 0 0)@2001-01-05)',
  tgeompoint '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04,
    Point(2 2 2)@2001-01-06)')) <= 0.5;
-- true
SELECT ST_CPAWithin(
  tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 1)@2001-01-03,
    Point(0 0 0)@2001-01-05)'::geometry,
  tgeompoint '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04,
    Point(2 2 2)@2001-01-06)'::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)@2001-01-01, Point(1 1)@2001-01-03)' <->
  geometry 'Point(0 1)';
-- [1@2001-01-01, 0.707106781186548@2001-01-02, 1@2001-01-03)
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-03)' <->
  tgeompoint '[Point(0 1)@2001-01-01, Point(1 0)@2001-01-03)';
-- [1@2001-01-01, 0@2001-01-02, 1@2001-01-03)
SELECT tgeompoint '[Point(0 1)@2001-01-01, Point(0 0)@2001-01-03)' <->
  tgeompoint '[Point(0 0)@2001-01-01, Point(1 0)@2001-01-03)';
-- [1@2001-01-01, 0.707106781186548@2001-01-02, 1@2001-01-03)
SELECT tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-02)' <->
  tgeompoint '[Point(0 1)@2001-01-01, Point(1 2)@2001-01-02)';
-- [1@2001-01-01,1@2001-01-02)
```

8.7. 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 tres versiones de las relaciones espaciales:

- Las relaciones *alguna vez* determinan determinan si la relación topológica o de distancia se satisface alguna vez (ver Sección 6.3.2) y resultan en un `boolean`. Ejemplos son las funciones `eIntersects` y `eDwithin`.
- Las relaciones *siempre* determinan determinan si la relación topológica o de distancia se satisface siempre (ver Sección 6.3.2) y resultan en un `boolean`. Ejemplos son las funciones `aIntersects` y `aDwithin`.
- Las relaciones *temporales* calculan la función topológica o de distancia en cada instante y dan como resultado un `tbool`. Ejemplos son las funciones `tIntersects` y `tDwithin`.

Por ejemplo, la siguiente consulta

```
SELECT eIntersects(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  tgeompoint '[Point(0 2)@2001-01-01, Point(4 2)@2001-01-05)');
-- t
```

determina si el punto temporal se cruza alguna vez con la geometría. En este caso, la consulta es equivalente a la siguiente

```
SELECT ST_Intersects(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  geometry 'Linestring(0 2,4 2)');
```

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

```
SELECT tIntersects(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  tgeompoint '[Point(0 2)@2001-01-01, Point(4 2)@2001-01-05)');
-- {[f@2001-01-01, t@2001-01-02, t@2001-01-04], (f@2001-01-04, f@2001-01-05)}
```

calcula en cada instante si el punto temporal se cruza con la geometría. Del mismo modo, la siguiente consulta

```
SELECT eDwithin(tgeompoint '[Point(3 1)@2001-01-01, Point(5 1)@2001-01-03)',
  tgeompoint '[Point(3 1)@2001-01-01, Point(1 1)@2001-01-03)', 2);
-- t
```

determina si la distancia entre los puntos temporales es alguna vez menor o igual a 2, mientras que la siguiente consulta

```
SELECT tDwithin(tgeompoint '[Point(3 1)@2001-01-01, Point(5 1)@2001-01-03)',
  tgeompoint '[Point(3 1)@2001-01-01, Point(1 1)@2001-01-03)', 2);
-- {[t@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
```

calcula en cada instante si la distancia entre los puntos temporales es menor o igual a 2.

Las relaciones alguna vez o siempre 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 eIntersects(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: `eIntersects`, `eDisjoint`, `eDwithin`, `eContains`, y `eTouches`, 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 `eContains`, `aContains` 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.

8.7.1. Relaciones espaciales alguna vez o siempre

Presentamos a continuación las relaciones espaciales alguna vez o siempre. Estas relaciones incluyen automáticamente una comparación de cuadro delimitador que hace uso de cualquier índice espacial que esté disponible en los argumentos.

- Contiene alguna vez

  `eContains(geo,tgeompoint) → boolean`

  `aContains(geo,tgeompoint) → boolean`

  Esta función devuelve verdadero si el punto temporal está alguna vez en el interior de la geometría. Recuerde que una geometría no contiene cosas en su borde y, por lo tanto, los polígonos y las líneas no contienen líneas y puntos que se encuentran en su borde. Consulte la documentación de la función ST_Contains en PostGIS.

```
SELECT eContains(geometry 'Linestring(1 1,3 3)',
  tgeompoint '[Point(4 2)@2001-01-01, Point(2 4)@2001-01-02]');
-- false
SELECT eContains(geometry 'Linestring(1 1,3 3,1 1)',
  tgeompoint '[Point(4 2)@2001-01-01, Point(2 4)@2001-01-03]');
-- true
SELECT eContains(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
 tgeompoint '[Point(0 1)@2001-01-01, Point(4 1)@2001-01-02]');
-- false
SELECT eContains(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  tgeompoint '[Point(1 4)@2001-01-01, Point(4 1)@2001-01-04]');
-- true
```

- Está disjunto alguna vez

  `eDisjoint({geo,tgeompoint},{geo,tgeompoint}) → boolean`

  `aDisjoint({geo,tpoint},{geo,tpoint}) → boolean`

```
SELECT eDisjoint(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-03)');
-- false
SELECT eDisjoint(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint '[Point(0 0 1)@2001-01-01, Point(1 1 0)@2001-01-03)');
-- true
```

- Está alguna vez a distancia de

  `eDwithin({geo,tpoint},{geo,tpoint},float) → boolean`

  `aDwithin({geo,tgeompoint},{geo,tgeompoint},float) → boolean`

```
SELECT eDwithin(geometry 'Point(1 1 1)',
  tgeompoint '[Point(0 0 0)@2001-01-01, Point(1 1 0)@2001-01-02]', 1);
-- true
SELECT eDwithin(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint '[Point(0 2 2)@2001-01-01,Point(2 2 2)@2001-01-02]', 1);
-- false
```

- Intersecta alguna vez

  `eIntersects({geo,tpoint},{geo,tpoint}) → boolean`

  `aIntersects({geo,tgeompoint},{geo,tgeompoint}) → boolean`

```
SELECT eIntersects(geometry 'Polygon((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))',
  tgeompoint '[Point(0 0 1)@2001-01-01, Point(1 1 1)@2001-01-03)');
-- false
SELECT eIntersects(geometry 'Polygon((0 0 0,0 1 1,1 1 1,1 0 0,0 0 0))',
  tgeompoint '[Point(0 0 1)@2001-01-01, Point(1 1 1)@2001-01-03)');
-- true
```

- Toca alguna vez

  `eTouches({geo,tgeompoint},{geo,tgeompoint}) → boolean`

  `aTouches({geo,tgeompoint},{geo,tgeompoint}) → boolean`

```
SELECT eTouches(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))',
  tgeompoint '[Point(0 0)@2001-01-01, Point(0 1)@2001-01-03)');
-- true
```

8.7.2. Relaciones espaciales temporales

Presentamos a continuación las relaciones espaciales temporales. Un requisito común con respecto ellas es restringir el resultado de la relación a los instantes en que el valor del resultado es verdadero o falso. Por ejemplo, la siguiente consulta calcula para cada viaje el tiempo dedicado viajando en el municipio de Bruselas.

```
SELECT TripId, duration(atValue(tIntersects(T.trip, M.geom), True))
FROM Trips T, Municipality M
WHERE M.Name = "Brussels" AND atValue(tIntersects(T.trip, M.geom), True) IS NOT NULL;
```

Para simplificar la escritura de consultas, las relaciones espaciales temporales tienen un último parámetro opcional, que si se proporciona aplica la función `atValue` (ver Sección 6.2) al resultado de la relación. De esta forma, la consulta anterior se puede escribir de la siguiente manera.

```
SELECT TripId, duration(tIntersects(T.trip, M.geom, True))
FROM Trips T, Municipality M
WHERE M.Name = "Brussels" AND tIntersects(T.trip, M.geom, True) IS NOT NULL;
```

- Contiene temporal

  `tContains(geometry,tgeompoint,atValue boolean=NULL) → tbool`

```
SELECT tContains(geometry 'Linestring(1 1,3 3)',
  tgeompoint '[Point(4 2)@2001-01-01, Point(2 4)@2001-01-02]');
-- {[f@2001-01-01, f@2001-01-02]}
SELECT tContains(geometry 'Linestring(1 1,3 3,1 1)',
  tgeompoint '[Point(4 2)@2001-01-01, Point(2 4)@2001-01-03]');
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03]}
SELECT tContains(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  tgeompoint '[Point(0 1)@2001-01-01, Point(4 1)@2001-01-02]');
-- {[f@2001-01-01, f@2001-01-02]}
SELECT tContains(geometry 'Polygon((1 1,1 3,3 3,3 1,1 1))',
  tgeompoint '[Point(1 4)@2001-01-01, Point(4 1)@2001-01-04]');
-- {[f@2001-01-01, f@2001-01-02], (t@2001-01-02, f@2001-01-03, f@2001-01-04]}
```

- Disjunto temporal

  `tDisjoint({geo,tpoint},{geo,tpoint},atValue boolean=NULL) → tbool`

  La función solo admite 3D o geografías para dos puntos temporales

```
SELECT tDisjoint(geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))',
  tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-04)');
-- {[t@2001-01-01, f@2001-01-02, f@2001-01-03], (t@2001-01-03, t@2001-01-04]}
SELECT tDisjoint(tgeompoint '[Point(0 3)@2001-01-01, Point(3 0)@2001-01-05)',
  tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-05)');
-- {[t@2001-01-01, f@2001-01-03], (t@2001-01-03, t@2001-01-05)}
```

- Está a distancia de temporal

  `tDwithin({geompoint,tgeompoint},{geompoint,tgeompoint},float,atValue boolean=NULL) → tbool`

```
SELECT tDwithin(geometry 'Point(1 1)',
  tgeompoint '[Point(0 0)@2001-01-01, Point(2 2)@2001-01-03)', sqrt(2));
--  {[t@2001-01-01, t@2001-01-03)}
SELECT tDwithin(tgeompoint '[Point(1 0)@2001-01-01, Point(1 4)@2001-01-05]',
  tgeompoint 'Interp=Step;[Point(1 2)@2001-01-01, Point(1 3)@2001-01-05]', 1);
-- {[f@2001-01-01, t@2001-01-02, t@2001-01-04], (f@2001-01-04, t@2001-01-05]}
```

- Intersección temporal

  `tIntersects({geo,tpoint},{geo,tpoint},atValue boolean=NULL) → tbool`

  La función solo admite 3D o geografías para dos puntos temporales

```
SELECT tIntersects(geometry 'MultiPoint(1 1,2 2)',
  tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-04)');
/* {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, t@2001-01-03],
   (f@2001-01-03, f@2001-01-04]} */
SELECT tIntersects(tgeompoint '[Point(0 3)@2001-01-01, Point(3 0)@2001-01-05)',
  tgeompoint '[Point(0 0)@2001-01-01, Point(3 3)@2001-01-05)');
-- {[f@2001-01-01, t@2001-01-03], (f@2001-01-03, f@2001-01-05)}
```

- Toca temporal

  `tTouches({geo,tgeompoint},{geo,tgeompoint},atValue boolean=NULL) → tbool`

```
SELECT tTouches(geometry 'Polygon((1 0,1 2,2 2,2 0,1 0))',
  tgeompoint '[Point(0 0)@2001-01-01, Point(3 0)@2001-01-04)');
-- {[f@2001-01-01, t@2001-01-02, t@2001-01-03], (f@2001-01-03, f@2001-01-04]}
```

Capítulo 9

Tipos temporales: Operaciones de análisis

9.1. Simplificación

- Simplificar un flotante o un punto temporal asegurándose de que los valores consecutivos estén al menos separados por una cierta distancia o intervalo de tiempo

  `minDistSimplify({tfloat,tpoint},mindist float) → {tfloat,tpoint}`

  `minTimeDeltaSimplify({tfloat,tpoint},mint interval) → {tfloat,tpoint}`

  En el caso de puntos temporales la distancia se especifica en las unidades del sistema de coordenadas. 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.

```
SELECT minDistSimplify(tfloat '[1@2001-01-01,2@2001-01-02,3@2001-01-04,4@2001-01-05]', 1);
-- [1@2001-01-01, 3@2001-01-04, 4@2001-01-05]
SELECT asText(minDistSimplify(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02, Point(3 3 3)@2001-01-04, Point(5 5 5)@2001-01-05)', sqrt(3)));
-- [POINT Z (1 1 1)@2001-01-01, POINT Z (3 3 3)@2001-01-04, POINT Z (5 5 5)@2001-01-05)
SELECT asText(minDistSimplify(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02, Point(3 3 3)@2001-01-04, Point(4 4 4)@2001-01-05)', sqrt(3)));
-- [POINT Z (1 1 1)@2001-01-01, POINT Z (3 3 3)@2001-01-04, POINT Z (4 4 4)@2001-01-05]
```

```
SELECT minTimeDeltaSimplify(tfloat '[1@2001-01-01, 2@2001-01-02, 3@2001-01-04,
  4@2001-01-05]', '1 day');
-- [1@2001-01-01, 3@2001-01-04, 4@2001-01-05]
SELECT asText(minTimeDeltaSimplify(tgeogpoint '[Point(1 1 1)@2001-01-01,
  Point(2 2 2)@2001-01-02, Point(3 3 3)@2001-01-04, Point(5 5 5)@2001-01-05)', '1 day'));
-- [POINT Z (1 1 1)@2001-01-01, POINT Z (3 3 3)@2001-01-04, POINT Z (5 5 5)@2001-01-05]
```

- Simplificar un flotante o un punto temporal usando el algoritmo de Douglas-Peucker

  `maxDistSimplify({tfloat,tgeompoint},maxdist float,syncdist=true) → {tfloat,tgeompoint}`

  `douglasPeuckerSimplify({tfloat,tgeompoint},maxdist float,syncdist=true) → {tfloat,tgeompoint}`

  La diferencia entre las dos funciones es que `maxDistSimplify` usa una versión del algoritmo de un solo recorrido, mientras que `douglasPeuckerSimplify` usa el algoritmo recursivo estándar.

  La función elimina los valores or los puntos cuya distancia es menor que la distancia pasada como segundo argumento. En el caso de puntos temporales la distancia se especifica en las unidades del sistema de coordenadas. El tercer argumento se aplica solo a puntos temporales y especifica si se utiliza la distancia espacial o la distancia sincronizada. 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.

```
-- Only synchronous distance for temporal floats
SELECT maxDistSimplify(tfloat '[1@2001-01-01, 2@2001-01-02, 1@2001-01-03, 3@2001-01-04,
  1@2001-01-05]', 1, false);
-- [1@2001-01-01, 1@2001-01-03, 3@2001-01-04, 1@2001-01-05]
-- Synchronous distance by default for temporal points
SELECT asText(maxDistSimplify(tgeompoint '[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02,
  Point(3 1)@2001-01-03, Point(3 3)@2001-01-05, Point(5 1)@2001-01-06]', 2));
-- [POINT(1 1)@2001-01-01, POINT(3 3)@2001-01-05, POINT(5 1)@2001-01-06]
-- Spatial distance
SELECT asText(maxDistSimplify(tgeompoint '[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02,
  Point(3 1)@2001-01-03, Point(3 3)@2001-01-05, Point(5 1)@2001-01-06]', 2, false));
-- [POINT(1 1)@2001-01-01, POINT(5 1)@2001-01-06]
```

```
-- Spatial vs synchronized distance
SELECT asText(douglasPeuckerSimplify(tgeompoint '[Point(1 1)@2001-01-01, Point(6 1)@2001 ↩
    -01-06,
  Point(7 4)@2001-01-07]', 2.3, false));
-- [POINT(1 1)@2001-01-01, POINT(7 4)@2001-01-07]
SELECT asText(douglasPeuckerSimplify(tgeompoint '[Point(1 1)@2001-01-01, Point(6 1)@2001 ↩
    -01-06,
  Point(7 4)@2001-01-07]', 2.3, true));
-- [POINT(1 1)@2001-01-01, POINT(6 1)@2001-01-06, POINT(7 4)@2001-01-07]
```

La diferencia entre la distancia espacial y la distancia sincronizada se ilustra en los dos últimos ejemplos anteriores y en la Figura 9.1. En el primer ejemplo, que usa la distancia espacial, se elimina el segundo instante ya que la distancia perpendicular entre `POINT(2 2)` y la línea definida por `POINT(1 1)` y `POINT(7 4)` es igual a 2.23. Por el contrario, en el segundo ejemplo se mantiene el segundo instante dado que la proyección de `Point(6 2)` en la marca de tiempo `2001-01-06` sobre el segmento de línea temporal da como resultado `Point(6 3.5)` y la distancia entre el punto original y su proyección es 2.5.

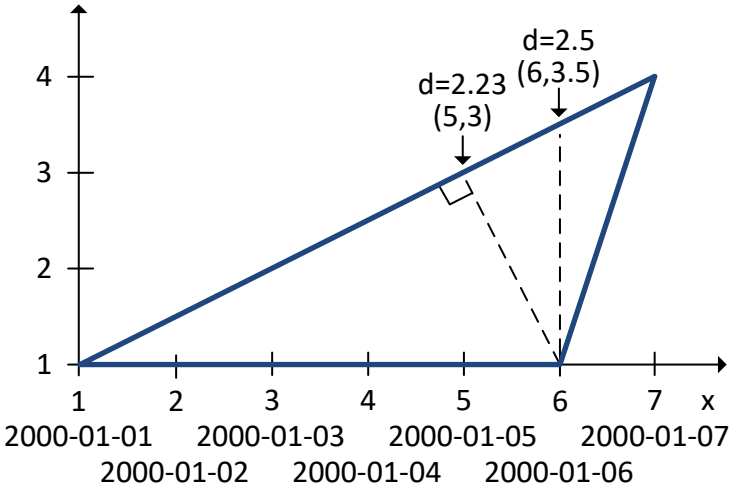


Figura 9.1: Diferencia entre la distancia espacial y la distancia sincronizada.

Un uso típico de la función `douglasPeuckerSimplify` es reducir el tamaño de un conjunto de datos, en particular con fines de visualización. Si la visualización es estática, se debe preferir la distancia espacial; si la visualización es dinámica o animada, se debe preferir la distancia sincronizada.

9.2. Reducción

- Muestrear un valor temporal con respecto a un intervalo

`tsample({tnumber,tgeompoint},duration interval,torigin timestamptz='2000-01-03') → {tnumber,tgeompoint}`

Si el origen no se especifica, su valor se establece por defecto en lunes 3 de enero de 2000. El intervalo dado debe ser estrictamente mayor que cero. Nótese que el resultado es un instante temporal o una secuencia temporal con interpolación discreta.

```
SELECT tsample(tint '{1@2001-01-01,5@2001-01-05}', '3 days', '2001-01-01');
-- {1@2001-01-01}
SELECT tsample(tfloat '[1@2001-01-01,5@2001-01-05]', '1 day', '2001-01-01');
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03, 4@2001-01-04, 5@2001-01-05}
SELECT tsample(tfloat '[1@2001-01-01,5@2001-01-05]', '3 days', '2001-01-01');
-- {1@2001-01-01, 4@2001-01-04}
```

```
SELECT asText(tsample(tgeompoint '[Point(1 1)@2001-01-01, Point(5 5)@2001-01-05]',
  '2 days', '2001-01-01'));
-- {POINT(1 1)@2001-01-01, POINT(3 3)@2001-01-03, POINT(5 5)@2001-01-05}
SELECT asText(tsample(tgeompoint '{[Point(1 1)@2001-01-01, Point(5 5)@2001-01-05],
  [Point(1 1)@2001-01-06, Point(5 5)@2001-01-08]}', '3 days', '2001-01-01'));
-- {POINT(1 1)@2001-01-01, POINT(4 4)@2001-01-04, POINT(3 3)@2001-01-07}
```

La Figura 9.2 ilustra el muestreo para números flotantes temporales con varias interpolaciones. Como ilustra la figura, la operación de muestreo es más adecuada para valores temporales con interpolación continua.

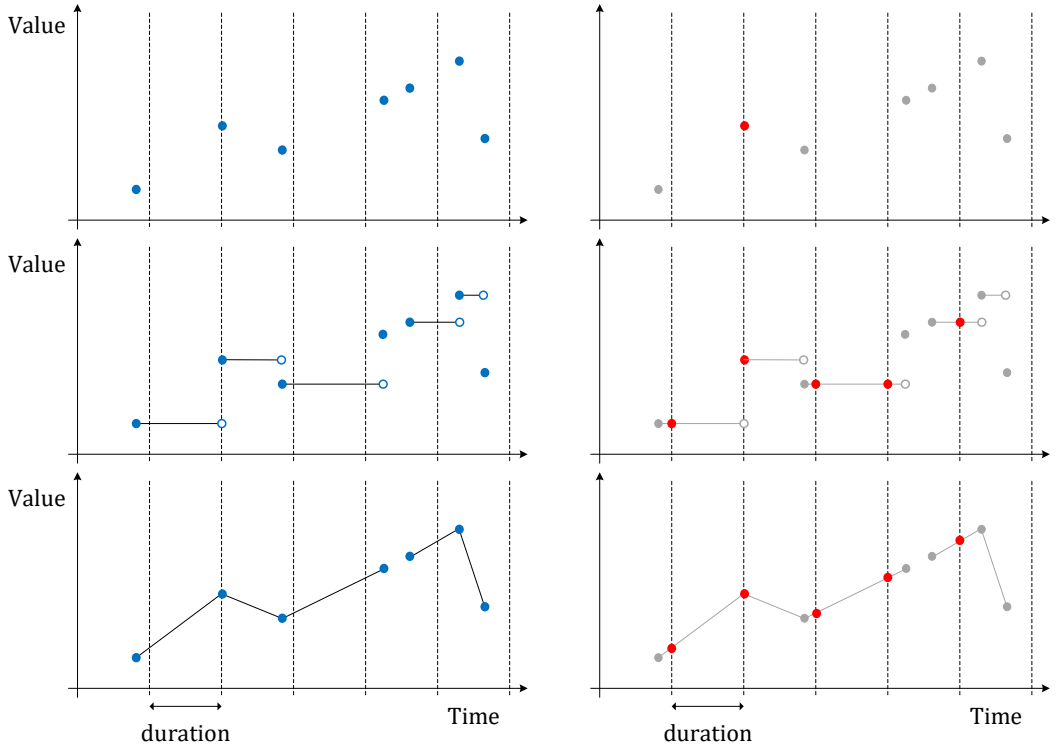


Figura 9.2: Muestreo de flotantes temporales con interpolación discreta, escalonada y lineal.

- Reducir la precisión temporal de un valor temporal con respecto a un intervalo calculando el promedio/centroide ponderado por el tiempo en cada intervalo de tiempo

  `tprecision({tnumber,tgeompoint},duration interval,torigin timestamptz='2000-01-03') → {tnumber,tgeompoint}`

  Si el origen no se especifica, su valor se establece por defecto en lunes 3 de enero de 2000. El intervalo dado debe ser estrictamente mayor que cero.

```
SELECT tprecision(tint '[1@2001-01-01,5@2001-01-05,1@2001-01-09]','1 day',
  '2001-01-01');
-- Interp=Step;[1@2001-01-01, 5@2001-01-05, 1@2001-01-09]
SELECT tprecision(tfloat '[1@2001-01-01,5@2001-01-05,1@2001-01-09)','1 day',
  '2001-01-01');
-- [1.5@2001-01-01, 4.5@2001-01-04, 4.5@2001-01-05, 1.5@2001-01-08]
SELECT tprecision(tfloat '[1@2001-01-01,5@2001-01-05,1@2001-01-09]','1 day',
  '2001-01-01');
-- [1.5@2001-01-01, 4.5@2001-01-04, 4.5@2001-01-05, 1.5@2001-01-08, 1@2001-01-09]
SELECT tprecision(tfloat '[1@2001-01-01,5@2001-01-05,1@2001-01-09)','2 days',
  '2001-01-01');
-- [2@2001-01-01, 4@2001-01-03, 4@2001-01-05, 2@2001-01-07]
```

```
SELECT asText(tprecision(tgeompoint '[Point(1 1)@2001-01-01, Point(5 5)@2001-01-05,
  Point(1 1)@2001-01-09)', '1 day', '2001-01-01'));
/* [POINT(1.5 1.5)@2001-01-01, POINT(4.5 4.5)@2001-01-04, POINT(4.5 4.5)@2001-01-05,
   POINT(1.5 1.5)@2001-01-08] */
SELECT asText(tprecision(tgeompoint '[Point(1 1)@2001-01-01, Point(5 5)@2001-01-05,
  Point(1 1)@2001-01-09)', '2 days', '2001-01-01'));
/* [POINT(2 2)@2001-01-01, POINT(4 4)@2001-01-03, POINT(4 4)@2001-01-05,
   POINT(2 2)@2001-01-07] */
SELECT asText(tprecision(tgeompoint '[Point(1 1)@2001-01-01, Point(5 5)@2001-01-05,
  Point(1 1)@2001-01-09)', '4 days', '2001-01-01'));
-- [POINT(3 3)@2001-01-01, POINT(3 3)@2001-01-05]
```

Cambiar la precisión de un valor temporal es similar a cambiar su *granularidad temporal*, por ejemplo, de marcas de tiempo a horas o días, aunque la precisión se puede establecer en un intervalo arbitrario, como 2 horas y 15 minutos. La Figura 9.3 ilustra un cambio de precisión temporal para números flotantes temporales con varias interpolaciones.

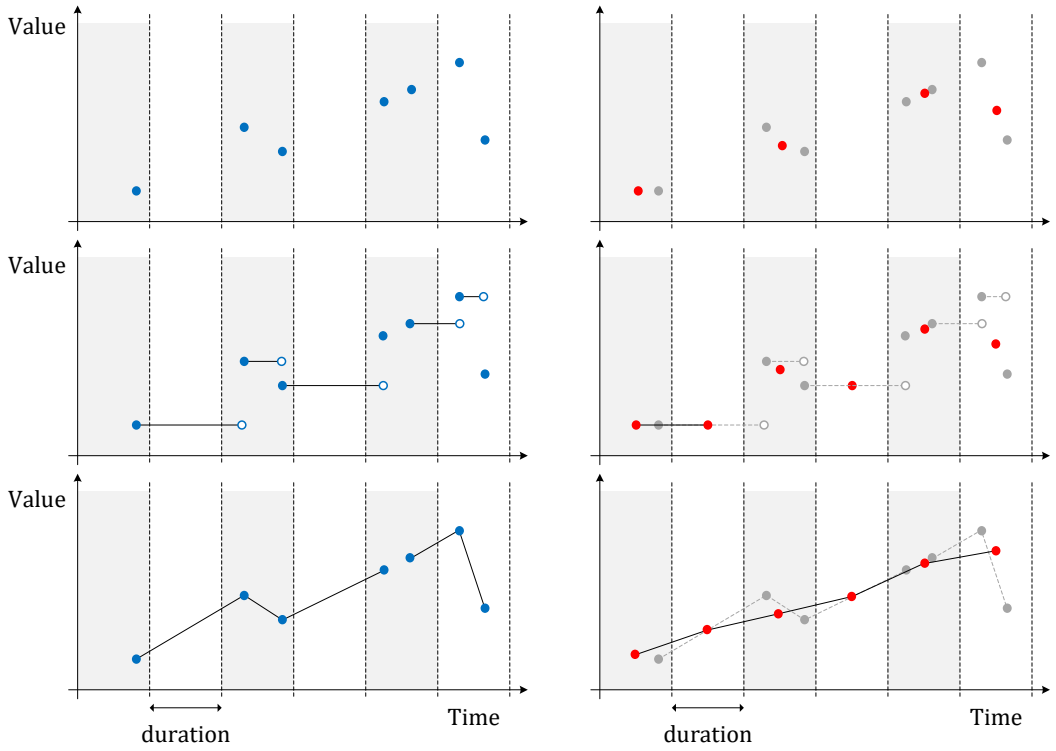


Figura 9.3: Cambio de precisión de números flotantes temporales con interpolación discreta, escalonada y lineal.

9.3. Similaridad

- Obtener la distancia de Hausdorff discreta entre dos valores temporales

  `hausdorffDistance({tnumber, tgeo}, {tnumber, tgeo}) → float`

  Esta función tiene una complejidad 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 hausdorffDistance(tfloat '[1@2001-01-01, 3@2001-01-03, 1@2001-01-06]',
 tfloat '[1@2001-01-01, 1.5@2001-01-02, 2.5@2001-01-03, 1.5@2001-01-04, 1.5@2001-01-05]');
-- 0.5
SELECT round(hausdorffDistance(tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03,
  Point(1 1)@2001-01-05]', tgeompoint '[Point(1.1 1.1)@2001-01-01,
  Point(2.5 2.5)@2001-01-02, Point(4 4)@2001-01-03, Point(3 3)@2001-01-04,
  Point(1.5 2)@2001-01-05]')::numeric, 6);
-- 1.414214
```

- 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@2001-01-01, 3@2001-01-03, 1@2001-01-06]',
 tfloat '[1@2001-01-01, 1.5@2001-01-02, 2.5@2001-01-03, 1.5@2001-01-04, 1.5@2001-01-05]');
-- 0.5
SELECT round(frechetDistance(tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03,
  Point(1 1)@2001-01-05]', tgeompoint '[Point(1.1 1.1)@2001-01-01,
  Point(2.5 2.5)@2001-01-02, Point(4 4)@2001-01-03, Point(3 3)@2001-01-04,
  Point(1.5 2)@2001-01-05]')::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}) → {(i,j)}`

  El resultado es un conjunto de pares `(i,j)`. 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@2001-01-01, 3@2001-01-03, 1@2001-01-06]',
 tfloat '[1@2001-01-01, 1.5@2001-01-02, 2.5@2001-01-03, 1.5@2001-01-04, 1.5@2001-01-05]');
-- (0,0)
-- (1,0)
-- (2,1)
-- (3,2)
-- (4,2)
SELECT frechetDistancePath(tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03,
  Point(1 1)@2001-01-05]', tgeompoint '[Point(1.1 1.1)@2001-01-01,
  Point(2.5 2.5)@2001-01-02, Point(4 4)@2001-01-03, Point(3 3)@2001-01-04,
  Point(1.5 2)@2001-01-05]');
-- (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

  `dynTimeWarpDistance({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 dynTimeWarpDistance(tfloat '[1@2001-01-01, 3@2001-01-03, 1@2001-01-06]',
 tfloat '[1@2001-01-01, 1.5@2001-01-02, 2.5@2001-01-03, 1.5@2001-01-04, 1.5@2001-01-05]');
-- 2
SELECT round(dynTimeWarpDistance(tgeompoint '[Point(1 1)@2001-01-01,
  Point(3 3)@2001-01-03, Point(1 1)@2001-01-05]',
  tgeompoint '[Point(1.1 1.1)@2001-01-01, Point(2.5 2.5)@2001-01-02,
  Point(4 4)@2001-01-03, Point(3 3)@2001-01-04, Point(1.5 2)@2001-01-05]')::numeric, 6);
-- 3.380776
```

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

  `dynTimeWarpPath({tnumber, tgeo}, {tnumber, tgeo}) → {(i,j)}`

  El resultado es un conjunto de pares `(i,j)`. 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 dynTimeWarpPath(tfloat '[1@2001-01-01, 3@2001-01-03, 1@2001-01-06]',
 tfloat '[1@2001-01-01, 1.5@2001-01-02, 2.5@2001-01-03, 1.5@2001-01-04, 1.5@2001-01-05]');
-- (0,0)
-- (1,0)
-- (2,1)
-- (3,2)
-- (4,2)
SELECT dynTimeWarpPath(tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03,
  Point(1 1)@2001-01-05]', tgeompoint '[Point(1.1 1.1)@2001-01-01,
  Point(2.5 2.5)@2001-01-02, Point(4 4)@2001-01-03, Point(3 3)@2001-01-04,
  Point(1.5 2)@2001-01-05]');
-- (0,0)
-- (1,1)
-- (2,1)
-- (3,1)
-- (4,2)
```

9.4. 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.

La Figura 9.4 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

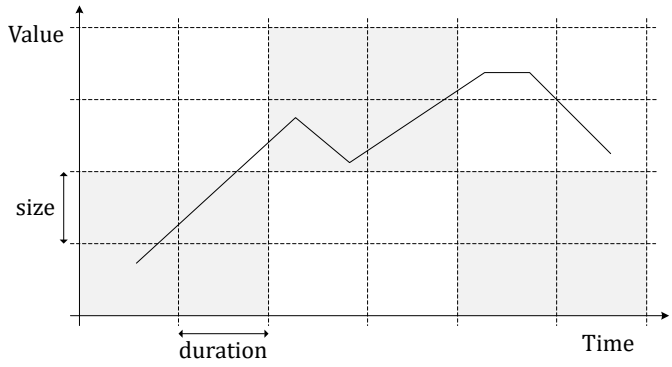


Figura 9.4: Mosaicos multidimensionales para números flotantes temporales.

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 {}.

9.4.1. Operaciones de intervalos

- Obtener un conjunto de intervalos que cubre el rango o el período con intervalos de la misma amplitud o duración alineados con el origen {}

  `bucketList(bounds unitspan,width number,origin number=0) → {(index,span)}`

  `bucketList(bounds tstzspan,duration interval,origin timestamptz='2000-01-03') → {(index,span)}`

  El resultado es un conjunto de pares `(index,span)`. 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.

```
SELECT (bl).index, (bl).span
FROM (SELECT bucketList(intspan '[15, 25]', 2) AS bl) t;
-- 1 | [14,16)
-- 2 | [16,18)
-- 3 | [18,20)
-- ...
SELECT bucketList(floatspan '[-10, -1]', 2.5, -7);
-- (1,"[-12, -9.5)")
-- (2,"[-9.5, -7)")
-- (3,"[-7, -4.5)")
-- ...
SELECT (bl).index, (bl).span
FROM (SELECT bucketList(tstzspan '[2001-01-15, 2001-01-25]','2 days') AS bl) t;
-- 1 | [2001-01-15, 2001-01-17)
-- 2 | [2001-01-17, 2001-01-19)
-- 3 | [2001-01-19, 2001-01-21)
-- ...
SELECT bucketList(tstzspan '[2001-01-15, 2001-01-25]', '2 days', '2001-01-02');
-- (1,"[2001-01-14, 2001-01-16)")
-- (2,"[2001-01-16, 2001-01-18)")
-- (3,"[2001-01-18, 2001-01-20)")
-- ...
```

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

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

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

```
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.

  `spanBucket(value number,width number,origin number=0) → span`

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

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

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

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

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

```
SELECT timeBucket(timestamptz '2001-05-01', interval '2 days');
-- 2001-04-29 01:00:00
SELECT timeBucket(timestamptz '2001-05-01', interval '2 days', timestamptz '2001-01-01');
-- 2001-04-30 01:00:00
```

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

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

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

```
SELECT periodBucket('2001-01-04', interval '1 week');
-- [2001-01-03, 2001-01-10)
SELECT periodBucket('2001-01-04', interval '1 week', '2001-01-07');
-- [2000-12-31, 2001-01-07)
```

9.4.2. Operaciones de mosaicos

- Obtener un conjunto de mosaicos que cubre el cuadro delimitador con mosaicos multidimensionales del mismo tamaño y duración ⧈{}

  `tileList(bounds tbox,size float,duration interval,vorigin float=0, torigin timestamptz='2000-01-03') → {(index,tile)}`

  `tileList(bounds stbox,xsize float,[ysize float,zsize float,] sorigin geompoint='Point(0 0 0)') → {(index,tile)}`

  `tileList(bounds stbox,xsize float,[ysize float,zsize float,]duration interval, sorigin geompoint='Point(0 0 0)',torigin='2000-01-03') → {(index,tile)}`

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.

En el caso de una malla espacio-temporal, `ysize` y `zsize` son opcionales, se supone que el tamaño de las dimensiones faltantes es igual a `xsize`. 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).tile
FROM (SELECT tileList(tfloat '[15@2001-01-15, 25@2001-01-25]'::tbox, 2.0, '2 days')
  AS gr) t;
-- 1 | TBOX XT([14,16),[2001-01-15,2001-01-17))
-- 2 | TBOX XT([16,18),[2001-01-15,2001-01-17))
-- 3 | TBOX XT([18,20),[2001-01-15,2001-01-17),)
-- ...
SELECT tileList(tfloat '[15@2001-01-15, 25@2001-01-25]'::tbox, 2.0, '2 days', 11.5);
-- (1,"TBOX XT([13.5,15.5),[2001-01-15,2001-01-17))")
-- (2,"TBOX XT([15.5,17.5),[2001-01-15,2001-01-17))")
-- (3,"TBOX XT([17.5,19.5),[2001-01-15,2001-01-17))")
-- ...
SELECT tileList(tgeompoint '[Point(3 3)@2001-01-15,
  Point(15 15)@2001-01-25]'::stbox, 2.0);
-- (1,"STBOX X((2,2),(4,4))")
-- (2,"STBOX X((4,2),(6,4))")
-- (3,"STBOX X((6,2),(8,4))")
-- ...
SELECT tileList(tgeompoint 'SRID=3812;[Point(3 3)@2001-01-15,
  Point(15 15)@2001-01-25]'::stbox, 2.0, geometry 'Point(3 3)');
-- (1,"SRID=3812;STBOX X((3,3),(5,5))")
-- (2,"SRID=3812;STBOX X((5,3),(7,5))")
-- (3,"SRID=3812;STBOX X((7,3),(9,5))")
-- ...
SELECT tileList(tgeompoint '[Point(3 3 3)@2001-01-15,
  Point(15 15 15)@2001-01-25]'::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 tileList(tgeompoint '[Point(3 3)@2001-01-15,
  Point(15 15)@2001-01-25]'::stbox, 2.0, interval '2 days');
-- (1,"STBOX XT(((2,2),(4,4)),[2001-01-15,2001-01-17))")
-- (2,"STBOX XT(((4,2),(6,4)),[2001-01-15,2001-01-17))")
-- (3,"STBOX XT(((6,2),(8,4)),[2001-01-15,2001-01-17))")
-- ...
SELECT tileList(tgeompoint '[Point(3 3 3)@2001-01-15,
  Point(15 15 15)@2001-01-25]'::stbox, 2.0, interval '2 days',
  'Point(3 3 3)', '2001-01-15');
-- (1,"STBOX ZT(((3,3,3),(5,5,5)),[2001-01-15,2001-01-17))")
-- (2,"STBOX ZT(((5,3,3),(7,5,5)),[2001-01-15,2001-01-17))")
-- (3,"STBOX ZT(((7,3,3),(9,5,5)),[2001-01-15,2001-01-17))")
-- ...
```

- Obtener el mosaico de la malla multidimensional que contiene el valor y la marca de tiempo

  `tile(value float,time timestamptz,size float,duration interval,`
  `vorigin float=0.0,torigin timestamptz='2000-01-03') → tbox`
  `tile(point geometry,xsize float,[ysize float,zsize float],`
  `sorigin geompoint='Point(0 0 0)') → stbox`
  `tile(point geometry,time timestamptz,xsize float,[ysize float,zsize float,]duration`

`interval,sorigin geompoint='Point(0 0 0)',torigin timestamptz='2000-01-03') → stbox`

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.

En el caso de una malla espacio-temporal, `ysize` y `zsize` son opcionales, se supone que el tamaño de las dimensiones faltantes es igual a `xsize`. 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 tile(15, '2001-01-15', 2, interval '2 days');
-- TBOX XT([14,16),[2001-01-15,2001-01-17))
SELECT tile(15, '2001-01-15', 2, interval '2 days', 1, '2001-01-02');
-- TBOX XT([15,17),[2001-01-14,2001-01-16))
SELECT tile(geometry 'Point(1 1 1)', 2.0);
-- STBOX Z((0,0,0),(2,2,2))
SELECT tile(geometry 'Point(1 1)', '2001-01-01', 2.0, interval '2 days');
-- STBOX XT((0,0),(2,2),[2001-01-01,2001-01-03))
SELECT tile(geometry 'Point(1 1)', '2001-01-01', 2.0, interval '2 days',
  'Point(1 1)', '2001-01-02');
-- STBOX XT(((1,1),(3,3)),[2000-12-31,2001-01-02))
```

9.4.3. Operaciones de fragmentación

Estas operaciones fragmentan un valor temporal con respecto a una secuencia de intervalos (ver la Sección 9.4.1) o una mosaico multidimensional (ver la Sección 9.4.2).

- Fragmentar el número temporal con respecto a intervalos de valores {}

  `valueSplit(tnumber,width number,origin number=0) → {(number,tnumber)}`

  El resultado es un conjunto de pares `(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@2001-01-01, 2@2001-01-02, 5@2001-01-05, 10@2001-01-10]',
  2) AS sp) t;
--  0 | {[1@2001-01-01 00:00:00+01, 1@2001-01-02 00:00:00+01)}
--  2 | {[2@2001-01-02 00:00:00+01, 2@2001-01-05 00:00:00+01)}
--  4 | {[5@2001-01-05 00:00:00+01, 5@2001-01-10 00:00:00+01)}
-- 10 | {[10@2001-01-10 00:00:00+01]}
SELECT valueSplit(tfloat '[1@2001-01-01, 10@2001-01-10)', 2.0, 1.0);
-- (1,"{[1@2001-01-01 00:00:00+01, 3@2001-01-03 00:00:00+01)}")
-- (3,"{[3@2001-01-03 00:00:00+01, 5@2001-01-05 00:00:00+01)}")
-- (5,"{[5@2001-01-05 00:00:00+01, 7@2001-01-07 00:00:00+01)}")
-- (7,"{[7@2001-01-07 00:00:00+01, 9@2001-01-09 00:00:00+01)}")
-- (9,"{[9@2001-01-09 00:00:00+01, 10@2001-01-10 00:00:00+01)}")
```

- Fragmentar el valor temporal con respecto a intervalos de tiempo {}

  `timeSplit(ttype,duration interval,origin timestamptz='2000-01-03') → {(time,temp)}`

  El resultado es un conjunto de pares `(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@2001-02-01, 10@2001-02-10)', '2 days') AS ts) t;
-- 2001-01-31 | [1@2001-02-01, 2@2001-02-02)
-- 2001-02-02 | [2@2001-02-02, 4@2001-02-04)
-- 2001-02-04 | [4@2001-02-04, 6@2001-02-06)
-- ...
```

```
SELECT (ts).time, astext((ts).temp) AS temp
FROM (SELECT timeSplit(tgeompoint '[Point(1 1)@2001-02-01, Point(10 10)@2001-02-10]',
  '2 days', '2001-02-01') AS ts) AS t;
-- 2001-02-01 | [POINT(1 1)@2001-02-01, POINT(3 3)@2001-02-03)
-- 2001-02-03 | [POINT(3 3)@2001-02-03, POINT(5 5)@2001-02-05)
-- 2001-02-05 | [POINT(5 5)@2001-02-05, POINT(7 7)@2001-02-07)
-- ...
```

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@2001-01-01, 10@2001-01-03]', '1 hour') AS ts) t
GROUP BY hour ORDER BY hour;
/* 00:00:00 | {[1@2001-01-01 00:00:00+01, 1.1875@2001-01-01 01:00:00+01),
               [5.5@2001-01-02 00:00:00+01, 5.6875@2001-01-02 01:00:00+01)} */
/* 01:00:00 | {[1.1875@2001-01-01 01:00:00+01, 1.375@2001-01-01 02:00:00+01),
               [5.6875@2001-01-02 01:00:00+01, 5.875@2001-01-02 02:00:00+01)} */
/* 02:00:00 | {[1.375@2001-01-01 02:00:00+01, 1.5625@2001-01-01 03:00:00+01),
               [5.875@2001-01-02 02:00:00+01, 6.0625@2001-01-02 03:00:00+01)} */
/* 03:00:00 | {[1.5625@2001-01-01 03:00:00+01, 1.75@2001-01-01 04:00:00+01),
               [6.0625@2001-01-02 03:00:00+01, 6.25@2001-01-02 04:00:00+01)} */
/* ... */
SELECT EXTRACT(DOW FROM (ts).time) AS dow_no, TO_CHAR((ts).time, 'Dy') AS dow,
  asText(round(merge((ts).temp), 2)) AS temp
FROM (SELECT timeSplit(tgeompoint '[Point(1 1)@2001-01-01, Point(10 10)@2001-01-14)',
  '1 hour') AS ts) t
GROUP BY dow, dow_no ORDER BY dow_no;
/* 0 | Sun | {[POINT(1 1)@2001-01-01, POINT(1.69 1.69)@2001-01-02),
              [POINT(5.85 5.85)@2001-01-08, POINT(6.54 6.54)@2001-01-09)} */
/* 1 | Mon | {[POINT(1.69 1.69)@2001-01-02, POINT(2.38 2.38)@2001-01-03),
              [POINT(6.54 6.54)@2001-01-09, POINT(7.23 7.23)@2001-01-10)} */
/* 2 | Tue | {[POINT(2.38 2.38)@2001-01-03, POINT(3.08 3.08)@2001-01-04),
              [POINT(7.23 7.23)@2001-01-10, POINT(7.92 7.92)@2001-01-11)} */
/* ... */
```

- Fragmentar el número temporal con respecto a los mosaicos de una malla de valores y de tiempo {}

  `valueTimeSplit(tumber,width number,duration interval,vorigin number=0, torigin timestamptz='2000-01-03') → {(number,time,tnumber)}`

  El resultado es un conjunto de triples `(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@2001-02-01, 2@2001-02-02, 5@2001-02-05,
  10@2001-02-10]', 5, '5 days') AS sp) t;
--  0 | 2001-02-01 | {[1@2001-02-01, 2@2001-02-02, 2@2001-02-05)}
--  5 | 2001-02-01 | {[5@2001-02-05, 5@2001-02-06)}
--  5 | 2001-02-06 | {[5@2001-02-06, 5@2001-02-10)}
-- 10 | 2001-02-06 | {[10@2001-02-10]}
SELECT (sp).number, (sp).time, (sp).tnumber
FROM (SELECT valueTimeSplit(tfloat '[1@2001-02-01, 10@2001-02-10)', 5.0, '5 days', 1.0,
  '2001-02-01') AS sp) t;
-- 1 | 2001-01-01 | [1@2001-01-01, 6@2001-01-06)
-- 6 | 2001-01-06 | [6@2001-01-06, 10@2001-01-10)
```

- Fragmentar el punto temporal con respecto a los mosaicos de una malla espacial {}

  `spaceSplit(tgeompoint,xsize float,[ysize float,zsize float,] origin geompoint='Point(0 0 0)',bitmatrix boolean=true) → {(point,tpoint)}`

El resultado es un conjunto de pares `(point,tpoint)`. Si el origen del espacio no se especifica, su valor se establece por defecto en `'Point(0 0 0)'`. Los argumentos `ysize` y `zsize` son opcionales, se supone que el tamaño de las dimensiones faltantes es igual a `xsize`. 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)@2001-03-01, Point(10 10)@2001-03-10]',
  2.0) AS sp) t;
-- POINT(0 0) | {[POINT(1 1)@2001-03-01, POINT(2 2)@2001-03-02)}
-- POINT(2 2) | {[POINT(2 2)@2001-03-02, POINT(4 4)@2001-03-04)}
-- POINT(4 4) | {[POINT(4 4)@2001-03-04, POINT(6 6)@2001-03-06)}
-- ...
SELECT ST_AsText((sp).point) AS point, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceSplit(tgeompoint '[Point(1 1 1)@2001-03-01,
  Point(10 10 10)@2001-03-10]', 2.0, geometry 'Point(1 1 1)') AS sp) t;
-- POINT Z(1 1 1) | {[POINT Z (1 1 1)@2001-03-01, POINT Z (3 3 3)@2001-03-03)}
-- POINT Z(3 3 3) | {[POINT Z (3 3 3)@2001-03-03, POINT Z (5 5 5)@2001-03-05)}
-- POINT Z(5 5 5) | {[POINT Z (5 5 5)@2001-03-05, POINT Z (7 7 7)@2001-03-07)}
-- ...
```

- Fragmentar el punto temporal con respecto a los mosaicos de una malla espacio-temporal {}

  `spaceTimeSplit(tgeompoint,xsize float,[ysize float,ysize float,]`
  `duration interval,sorigin geompoint='Point(0 0 0)',`
  `torigin timestamptz='2000-01-03',bitmatrix boolean=true) → {(point,time,tpoint)}`

  El resultado es un conjunto de triples `(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. Los argumentos `ysize` y `zsize` son opcionales, se supone que el tamaño de las dimensiones faltantes es igual a `xsize`. 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)@2001-02-01, Point(10 10)@2001-02-10]',
  2.0, interval '2 days') AS sp) t;
-- POINT(0 0) | 2001-01-31 | {[POINT(1 1)@2001-02-01, POINT(2 2)@2001-02-02)}
-- POINT(2 2) | 2001-01-31 | {[POINT(2 2)@2001-02-02]}
-- POINT(2 2) | 2001-02-02 | {[POINT(2 2)@2001-02-02, POINT(4 4)@2001-02-04)}
-- ...
SELECT ST_AsText((sp).point) AS point, (sp).time, astext((sp).tpoint) AS tpoint
FROM (SELECT spaceTimeSplit(tgeompoint '[Point(1 1 1)@2001-02-01,
  Point(10 10 10)@2001-02-10]', 2.0, interval '2 days', 'Point(1 1 1)',
  '2001-03-01') AS sp) t;
-- POINT Z(1 1 1) | 2001-02-01 | {[POINT Z(1 1 1)@2001-02-01, POINT Z(3 3 3)@2001-02-03)}
-- POINT Z(3 3 3) | 2001-02-01 | {[POINT Z(3 3 3)@2001-02-03]}
-- POINT Z(3 3 3) | 2001-02-03 | {[POINT Z(3 3 3)@2001-02-03, POINT Z (5 5 5)@2001-02-05)}
-- ...
```

Capítulo 10

Tipos temporales: Agregación e indexación

10.1. Agregación

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 `tstzspan`, 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 4.2.

- Conteo temporal

  `tCount(ttype) → {tintSeq,tintSeqSet}`

```
SELECT tCount(NoEmps) FROM Department;
-- {[1@2001-01-01, 2@2001-02-01, 1@2001-08-01, 1@2001-10-01)}
```

- Extensión del cuadro delimitador

  `extent(temp) → {tstzspan,tbox,stbox}`

```
SELECT extent(noEmps) FROM Department;
-- TBOX XT((4,12),[2001-01-01,2001-10-01])
SELECT extent(Trip) FROM Trips;
-- STBOX XT(((0,0),(3,3),[2001-01-01 08:00:00+01, 2001-01-01 08:20:00+01)))
```

- Y temporal

  `tAnd(tbool) → tbool`

```
SELECT tAnd(NoEmps #> 6) FROM Department;
-- {[t@2001-01-01, f@2001-04-01, f@2001-10-01)}
```

- O temporal

  `tOr(tbool) → tbool`

```
SELECT tOr(NoEmps #> 6) FROM Department;
-- {[t@2001-01-01, f@2001-08-01, f@2001-10-01)}
```

- Mínimo temporal

  `tMin(ttype) → ttype`

```
SELECT tMin(NoEmps) FROM Department;
-- {[10@2001-01-01, 4@2001-02-01, 6@2001-06-01, 6@2001-10-01)}
```

- Máximo temporal

  `tMax(ttype) → ttype`

```
SELECT tMax(NoEmps) FROM Department;
-- {[10@2001-01-01, 12@2001-04-01, 6@2001-08-01, 6@2001-10-01)}
```

- Suma temporal

  `tSum(tnumber) → {tnumberSeq,tnumberSeqSet}`

```
SELECT tSum(NoEmps) FROM Department;
/* {[10@2001-01-01, 14@2001-02-01, 16@2001-04-01, 18@2001-06-01, 6@2001-08-01,
   6@2001-10-01)} */
```

- Promedio temporal

  `tAvg(tnumber) → {tfloatSeq,tfloatSeqSet}`

```
SELECT tAvg(NoEmps) FROM Department;
/* {[10@2001-01-01, 10@2001-02-01), [7@2001-02-01, 7@2001-04-01),
   [8@2001-04-01, 8@2001-06-01), [9@2001-06-01, 9@2001-08-01),
   [6@2001-08-01, 6@2001-10-01) */
```

- Mínimo de ventana

  `wMin(tnumber,interval) → {tnumberSeq,tnumberSeqSet}`

```
SELECT wMin(NoEmps, interval '2 days') FROM Department;
-- {[10@2001-01-01, 4@2001-04-01, 6@2001-06-03, 6@2001-10-03)}
```

- Máximo de ventana

  `wMax(tnumber,interval) → {tnumberSeq,tnumberSeqSet}`

```
SELECT wMax(NoEmps, interval '2 days') FROM Department;
-- {[10@2001-01-01, 12@2001-04-01, 6@2001-08-03, 6@2001-10-03)}
```

- Conteo de ventana

  `wCount(tnumber,interval) → {tintSeq,tintSeqSet}`

```
SELECT wCount(NoEmps, interval '2 days') FROM Department;
/* {[1@2001-01-01, 2@2001-02-01, 3@2001-04-01, 2@2001-04-03, 3@2001-06-01, 2@2001-06-03,
   1@2001-08-03, 1@2001-10-03)} */
```

- Suma de ventana

  `wSum(tint,interval) → {tintSeq,tintSeqSet}`

```
SELECT wSum(NoEmps, interval '2 days') FROM Department;
/* {[10@2001-01-01, 14@2001-02-01, 26@2001-04-01, 16@2001-04-03, 22@2001-06-01,
   18@2001-06-03, 6@2001-08-03, 6@2001-10-03)} */
```

- Promedio de ventana

  `wAvg(tint,interval) → {tfloatDiscSeq,tfloatSeqSet}`

```
SELECT wAvg(NoEmps, interval '2 days') FROM Department;
/* {[10@2001-01-01, 10@2001-02-01), [7@2001-02-01, 7@2001-04-01),
   [8.66666666666667@2001-04-01, 8.66666666666667@2001-04-03),
   [8@2001-04-03, 8@2001-06-01),
   [7.33333333333333@2001-06-01, 7.33333333333333@2001-06-03),
   [9@2001-06-03, 9@2001-08-03), [6@2001-08-03, 6@2001-10-03)} */
```

- Centroide temporal

  `tCentroid(tgeompoint) → tgeompoint`

```
SELECT tCentroid(Trip) FROM Trips;
/* {[POINT(0 0)@2001-01-01 08:00:00+00, POINT(1 0)@2001-01-01 08:05:00+00),
   [POINT(0.5 0)@2001-01-01 08:05:00+00, POINT(1.5 0.5)@2001-01-01 08:10:00+00,
   POINT(2 1.5)@2001-01-01 08:15:00+00),
   [POINT(2 2)@2001-01-01 08:15:00+00, POINT(3 3)@2001-01-01 08:20:00+00)} */
```

10.2. Indexación

Se pueden crear índices GiST y SP-GiST para columnas de tabla de tipos temporales. El índice GiST implementa un árbol R, mientras que el índice SP-GiST implementa un árbol cuádruple n-dimensional. 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 4, estos son

- el tipo `tstzspan` 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 6.4 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, `tstzspan`, `tbox` o `stbox`), no los valores completos.

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 && intspan '[1, 5)';
SELECT * FROM Department WHERE NoEmps && tstzspan '[2001-04-01, 2001-05-01)';
SELECT * FROM Department WHERE NoEmps &&
  tbox(intspan '[1, 5)', tstzspan '[2001-04-01, 2001-05-01)');
SELECT * FROM Department WHERE NoEmps &&
  tfloat '{[1@2001-01-01, 1@2001-02-01), [5@2001-04-01, 5@2001-05-01)}';
```

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 && tstzspan '[2001-01-01, 2001-01-05)';
SELECT * FROM Trips WHERE Trip &&
  stbox(geometry 'Polygon((0 0,0 1,1 1,1 0,0 0))', tstzspan '[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 algunas de las funciones de los tipos temporales, 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 atTime(T.Trip, tstzspan '[2001-01-01, 2001-01-02)')
FROM Trips T
-- Filtro de índice con cuadro delimitador
WHERE T.Trip && tstzspan '[2001-01-01, 2001-01-02)';
```

En el caso de los puntos temporales, todas las relaciones espaciales con la semántica posible (ver la Sección 8.7) 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);
```

10.3. Estadísticas y selectividad

10.3.1. 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 `tstzspan`, 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 tipos de tiempo y de rango `tstzset`, `tstzspan`, `tstzspanset`, `intspan` y `floatspan` replican las recopiladas por PostgreSQL para `tstzrange`. Esto es claro para los tipos de rango en MobilityDB, que son versiones más eficientes de los tipos de rango en PostgreSQL. Para los tipos `tstzset` y `tstzspanset`, un valor se convierte en su período delimitador y luego se recopilan las estadísticas del tipo `tstzspan`.

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.

10.3.2. Estimación de la selectividad

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 tipo MobilityDB (como `tstzspan`, `tint`, 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.

MobilityDB estima la selectividad de restricción y de combinación para tipos de tiempo, de rango y temporales.

Capítulo 11

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`.

11.1. 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
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.

```
-- Valor rid incorrecto
SELECT npoint 'Npoint(87.5, 1.0)';
-- Valor de posición incorrecto
SELECT npoint 'Npoint(87, 2.0)';
-- Valor rid inexistente en la table ways
SELECT npoint 'Npoint(99999999, 1.0)';
```

Damos a continuación las funciones y operadores para los tipos de redes estáticas.

11.1.1. Constructores

- Constructor de puntos de red

  `npoint(bigint,double precision)` $\rightarrow$ `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);
```

11.1.2. Transformaciones

- Redondear la(s) posición(es) del punto de red or el segmento de red en el número de posiciones decimales

  `round({npoint,nsegment},integer=0) → {npoint,nsegment}`

```
SELECT round(npoint(76, 0.123456789), 6);
--  NPoint(76,0.123457)
SELECT round(nsegment(76, 0.123456789, 0.223456789), 6);
--  NSegment(76,0.123457,0.223457)
```

11.1.3. Accesores

- 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/final

  `startPosition(nsegment) → float`

  `endPosition(nsegment) → float`

```
SELECT startPosition(nsegment 'Nsegment(76, 0.3, 0.5)');
-- 0.3
SELECT endPosition(nsegment 'Nsegment(76, 0.3, 0.5)');
-- 0.5
```

11.1.4. Operaciones espaciales

- Obtener el identificador de referencia espacial

  `SRID({npoint,nsegment}) → integer`

```
SELECT SRID(npoint '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
```

11.1.5. Comparaciones

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
```

11.2. 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 tres subtipos, a saber, instante, secuencia y conjunto de secuencias. A continuación se dan ejemplos de valores de `tnpoint` en estos subtipos.

```
SELECT tnpoint 'Npoint(1, 0.5)@2001-01-01';
SELECT tnpoint '{Npoint(1, 0.3)@2001-01-01, Npoint(1, 0.5)@2001-01-02,
  Npoint(1, 0.5)@2001-01-03}';
SELECT tnpoint '[Npoint(1, 0.2)@2001-01-01, Npoint(1, 0.4)@2001-01-02,
  Npoint(1, 0.5)@2001-01-03]';
SELECT tnpoint '{[Npoint(1, 0.2)@2001-01-01, Npoint(1, 0.4)@2001-01-02,
  Npoint(1, 0.5)@2001-01-03], [Npoint(2, 0.6)@2001-01-04, Npoint(2, 0.6)@2001-01-05]}';
```

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`, `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)@2001-01-01, Npoint(1, 0.4)@2001-01-02,
  Npoint(1, 0.5)@2001-01-03]';
SELECT tnpoint(Sequence) 'Npoint(1, 0.2)@2001-01-01';
-- ERROR: Temporal type (Instant) does not match column type (Sequence)
```

Los valores de puntos de red temporales del subtipo de secuencia et interpolación linear o escalonada 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)}
```

11.3. Validez de los puntos de red temporal

Los valores de los puntos de red temporal deben satisfacer las restricciones especificadas en la Sección 4.3 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)';
```

11.4. Constructores

- Constructor para puntos de red temporales con valor constante

  `tnpoint(npoint,timestamptz)` → `tnpointInst`

  `tnpoint(npoint,tstzset)` → `tnpointDiscSeq`

  `tnpoint(npoint,tstzspan,interp='linear')` → `tnpointContSeq`

  `tnpoint(npoint,tstzspanset,interp='linear')` → `tnpointSeqSet`

```
SELECT tnpoint('Npoint(1, 0.5)', timestamptz '2001-01-01');
-- NPoint(1,0.5)@2001-01-01
SELECT tnpoint('Npoint(1, 0.3)', tstzset '{2001-01-01, 2001-01-03, 2001-01-05}');
-- {NPoint(1,0.3)@2001-01-01, NPoint(1,0.3)@2001-01-03, NPoint(1,0.3)@2001-01-05}
SELECT tnpoint('Npoint(1, 0.5)', tstzspan '[2001-01-01, 2001-01-02]');
-- [NPoint(1,0.5)@2001-01-01, NPoint(1,0.5)@2001-01-02]
SELECT tnpoint('Npoint(1, 0.2)', tstzspanset '{[2001-01-01, 2001-01-03]}', 'step');
-- Interp=Step;{[NPoint(1,0.2)@2001-01-01, NPoint(1,0.2)@2001-01-03]}
```

- Constructor para puntos de red temporal de subtipo secuencia

  `tnpointSeq(tnpointInst[],interp={'step','linear'},leftInc bool=true, rightInc bool=true) → tnpointSeq`

```
SELECT tnpointSeq(ARRAY[tnpoint 'Npoint(1, 0.3)@2001-01-01',
  'Npoint(1, 0.5)@2001-01-02', 'Npoint(1, 0.5)@2001-01-03']);
-- {NPoint(1,0.3)@2001-01-01, NPoint(1,0.5)@2001-01-02, NPoint(1,0.5)@2001-01-03}
SELECT tnpointSeq(ARRAY[tnpoint 'Npoint(1, 0.2)@2001-01-01',
  'Npoint(1, 0.4)@2001-01-02', 'Npoint(1, 0.5)@2001-01-03']);
-- [NPoint(1,0.2)@2001-01-01, NPoint(1,0.4)@2001-01-02, NPoint(1,0.5)@2001-01-03]
```

- Constructor para puntos de red temporal de subtipo conjunto de secuencias

  `tnpointSeqSet(tnpoint[]) → tnpointSeqSet`

  `tnpointSeqSetGaps(tnpointInst[],maxt=NULL,maxdist=NULL,interp='linear') → tnpointSeqSet`

```
SELECT tnpointSeqSet(ARRAY[tnpoint '[Npoint(1,0.2)@2001-01-01, Npoint(1,0.4)@2001-01-02,
  Npoint(1,0.5)@2001-01-03]', '[Npoint(2,0.6)@2001-01-04, Npoint(2,0.6)@2001-01-05]']);
/* {[NPoint(1,0.2)@2001-01-01, NPoint(1,0.4)@2001-01-02, NPoint(1,0.5)@2001-01-03],
   [NPoint(2,0.6)@2001-01-04, NPoint(2,0.6)@2001-01-05]} */
SELECT tnpointSeqSetGaps(ARRAY[tnpoint 'NPoint(1,0.1)@2001-01-01',
  'NPoint(1,0.3)@2001-01-03', 'NPoint(1,0.5)@2001-01-05'], '1 day');
-- {[NPoint(1,0.1)@2001-01-01], [NPoint(1,0.3)@2001-01-03], [NPoint(1,0.5)@2001-01-05]}
```

11.5. Conversión de tipos

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.

11.6. Operaciones para los tipos de puntos de red

La mayoría de las funciones para tipos temporales descritas en los capítulos precendentes 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

  `tnpointInst(tnpoint) → tnpointInst`

  `tnpointSeq(tnpoint,interp) → tnpointSeq`

  `tnpointSeqSet(tnpoint) → tnpointSeqSet`

```
SELECT tnpointSeq(tnpoint 'NPoint(1, 0.5)@2001-01-01', 'discrete');
-- {NPoint(1,0.5)@2001-01-01}
SELECT tnpointSeq(tnpoint 'NPoint(1, 0.5)@2001-01-01');
-- [NPoint(1,0.5)@2001-01-01]
SELECT tnpointSeqSet(tnpoint 'NPoint(1, 0.5)@2001-01-01');
-- {[NPoint(1,0.5)@2001-01-01]}
```

- Transformar un punto de red temporal en otra interpolación

  `setInterp(tnpoint, interp) → tnpoint`

```
SELECT setInterp(tnpoint 'NPoint(1,0.2)@2001-01-01','linear');
-- [NPoint(1,0.2)@2001-01-01]
SELECT setInterp(tnpoint '{[NPoint(1,0.1)@2001-01-01], [NPoint(1,0.2)@2001-01-02]}',
  'discrete');
-- {NPoint(1,0.1)@2001-01-01, NPoint(1,0.2)@2001-01-02}
```

- Redondear la fracción del punto de red temporal en el número de lugares decimales

  `round(tnpoint,integer=0) → tnpoint`

```
SELECT round(tnpoint '{[NPoint(1,0.123456789)@2001-01-01, NPoint(1,0.5)@2001-01-02)}', 6);
-- {[NPoint(1,0.123457)@2001-01-01 00:00:00+01, NPoint(1,0.5)@2001-01-02 00:00:00+01)}
```

- Obtener los valores

  `getValues(tnpoint) → npoint[]`

```
SELECT getValues(tnpoint '{[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02)}');
-- {"NPoint(1,0.3)","NPoint(1,0.5)"}
SELECT getValues(tnpoint '{[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.3)@2001-01-02)}');
-- {"NPoint(1,0.3)"}
```

- Obtener los identificadores de ruta

  `routes(tnpoint) → bigintset`

```
SELECT routes(tnpoint '{NPoint(3, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02}');
-- {1, 3}
```

- Obtener el valor en una marca de tiempo

  `valueAtTimestamp(tnpoint,timestamptz) → npoint`

```
SELECT valueAtTimestamp(tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03)',
  '2001-01-02');
-- NPoint(1,0.4)
```

- Obtener la longitud atravesada por el punto de red temporal

  `length(tnpoint) → float`

```
SELECT length(tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]');
-- 54.3757408468784
```

- Obtener la longitud acumulada atravesada por el punto de red temporal

  `cumulativeLength(tnpoint) → tfloat`

```
SELECT cumulativeLength(tnpoint '{[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02,
  NPoint(1, 0.5)@2001-01-03], [NPoint(1, 0.6)@2001-01-04, NPoint(1, 0.7)@2001-01-05]}');
/* {[0@2001-01-01, 54.3757408468784@2001-01-02, 54.3757408468784@2001-01-03],
   [54.3757408468784@2001-01-04, 81.5636112703177@2001-01-05]} */
```

- Obtener la velocidad del punto de red temporal en unidades por segundo

  `speed({tnpointSeq, tnpointSeqSet}) → tfloatSeqSet`

```
SELECT speed(tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.4)@2001-01-02,
  NPoint(1, 0.6)@2001-01-03]') * 3600 * 24;
/* Interp=Step;[21.4016800272077@2001-01-01, 14.2677866848051@2001-01-02,
   14.2677866848051@2001-01-03] */
```

- 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 X((48.711754,20.92568),(48.711758,20.925682))
SELECT stbox(npoint 'NPoint(1,0.3)', timestamptz '2001-01-01');
-- STBOX XT((62.786633,80.143555),(62.786636,80.143562),[2001-01-01,2001-01-01])
SELECT stbox(npoint 'NPoint(1,0.3)', period '[2001-01-01,2001-01-02]');
-- STBOX XT((62.786633,80.143555),(62.786636,80.143562),[2001-01-01,2001-01-02])
```

- Obtener el centroide ponderado en el tiempo

  `twCentroid(tnpoint) → geometry(Point)`

```
SELECT st_astext(twCentroid(tnpoint '{[NPoint(1, 0.3)@2001-01-01,
  NPoint(1, 0.5)@2001-01-02, NPoint(1, 0.5)@2001-01-03, NPoint(1, 0.7)@2001-01-04)}'));
-- POINT(79.9787466444847 46.2385558051041)
```

- Obtener el acimut temporal

  `azimuth(tnpoint) → tfloat`

```
SELECT azimuth(tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-02]');
/* {[0.974681063778863@2001-01-01 00:00:00+01,
   0.974681063778863@2001-01-01 23:54:36.721091+01),
   [3.68970843029227@2001-01-01 23:54:36.721091+01,
   3.68970843029227@2001-01-02 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)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)');
-- NPoint(2,0.349928)@2001-01-01 02:59:44.402905+01
SELECT nearestApproachInstant(tnpoint '[NPoint(2, 0.3)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', npoint 'NPoint(1, 0.5)');
-- NPoint(2,0.592181)@2001-01-01 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)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)');
-- 1.41793220500979
SELECT nearestApproachDistance(tnpoint '[NPoint(2, 0.3)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', npoint 'NPoint(1, 0.5)');
-- NPoint(2,0.592181)@2001-01-01 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 10.2).

- 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)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)'));
-- LINESTRING(50.7960725266492 48.8266286733015,50 50)
SELECT st_astext(shortestLine(tnpoint '[NPoint(2, 0.3)@2001-01-01,
  NPoint(2, 0.7)@2001-01-02]', npoint 'NPoint(1, 0.5)'));
-- LINESTRING(77.0902838115125 66.6659083092593,90.8134936900394 46.4385792121146)
```

- Restringir a un valor

  `atValue(tnpoint,base) → tnpoint`

  `minusValue(tnpoint,base) → tnpoint`

```
SELECT atValue(tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-03]',
  'NPoint(2, 0.5)');
-- {[NPoint(2,0.5)@2001-01-02]}
SELECT minusValue(tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-03]',
  'NPoint(2, 0.5)');
/* {[NPoint(2,0.3)@2001-01-01, NPoint(2,0.5)@2001-01-02),
   (NPoint(2,0.5)@2001-01-02, NPoint(2,0.7)@2001-01-03]} */
```

- Restringir a una geometría

  `atGeometry(tnpoint,geometry) → tnpoint`

  `minusGeometry(tnpoint,geometry) → tnpoint`

```
SELECT atGeometry(tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
SELECT minusGeometry(tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
/* {(NPoint(2,0.342593)@2001-01-01 05:06:40.364673+01,
   NPoint(2,0.7)@2001-01-03 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)@2001-01-01, NPoint(1, 0.4)@2001-01-03)' #=
  npoint 'NPoint(1, 0.3)';
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
SELECT tnpoint '[NPoint(1, 0.2)@2001-01-01, NPoint(1, 0.8)@2001-01-03)' #<>
  tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.7)@2001-01-03)';
-- {[t@2001-01-01, f@2001-01-02], (t@2001-01-02, t@2001-01-03)}
```

- Operadores de igualdad alguna vez y siempre

  `tnpoint ?= tnpoint → boolean`

  `tnpoint %= tnpoint → boolean`

```
SELECT tnpoint '[Npoint(1, 0.2)@2001-01-01, Npoint(1, 0.4)@2001-01-04)' ?= Npoint(1, 0.3);
-- true
SELECT tnpoint '[Npoint(1, 0.2)@2001-01-01, Npoint(1, 0.2)@2001-01-04)' &= Npoint(1, 0.2);
-- true
```

- Operadores de posición

  `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)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' <<
  npoint 'NPoint(1, 0.2)'
-- false
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' <<|
  stbox(npoint 'NPoint(1, 0.5)')
-- false
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' &>
  npoint 'NPoint(1, 0.3)'::geometry
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' >>#
  tnpoint '[NPoint(1, 0.3)@2001-01-03, NPoint(1, 0.5)@2001-01-05]'
-- true
```

- Operadores topológicos

  `tnpoint && tnpoint → boolean`

  `tnpoint <@ tnpoint → boolean`

  `tnpoint @> tnpoint → boolean`

  `tnpoint ~= tnpoint → boolean`

  `tnpoint -|- tnpoint → boolean`

```
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' &&
  npoint 'NPoint(1, 0.5)'
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' @>
  stbox(npoint 'NPoint(1, 0.5)')
-- true
SELECT npoint 'NPoint(1, 0.5)'::geometry <@
  tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]'
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' ~=
  tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.35)@2001-01-02,
  NPoint(1, 0.5)@2001-01-03]'
-- true
```

- Obtener la distancia más pequeña entre los dos argumentos

  `tgeompoint |=| tnpoint → float`

```
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' |=|
  npoint 'NPoint(1, 0.2)';
-- 2.34988300875063
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' |=|
  geometry 'Linestring(2 2,2 1,3 1)';
-- 82.2059262761477
```

- Obtener la distancia temporal

  `tgeompoint <-> tnpoint → tfloat`

```
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' <->
  npoint 'NPoint(1, 0.2)';
-- [2.34988300875063@2001-01-02 00:00:00+01, 2.34988300875063@2001-01-03 00:00:00+01]
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' <->
  geometry 'Point(50 50)';
-- [25.0496666945044@2001-01-01 00:00:00+01, 26.4085688426232@2001-01-03 00:00:00+01]
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03]' <->
  tnpoint '[NPoint(1, 0.3)@2001-01-02, NPoint(1, 0.5)@2001-01-04]'
-- [2.34988300875063@2001-01-02 00:00:00+01, 2.34988300875063@2001-01-03 00:00:00+01]
```

- Relaciones espaciales alguna vez

  `eContains(geometry,tnpoint) → boolean`

  `aContains(geometry,tnpoint) → boolean`

  `eDisjoint({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `aDisjoint({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `eIntersects({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `aIntersects({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `eTouches({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `aTouches({geometry,npoint,tnpoint},{geometry,npoint,tnpoint}) → boolean`

  `eDwithin({geometry,npoint,tnpoint},{geometry,npoint,tnpoint},float) → boolean`

  `aDwithin({geometry,npoint,tnpoint},{geometry,npoint,tnpoint},float) → boolean`

```
SELECT eContains(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)');
-- false
SELECT eDisjoint(npoint 'NPoint(2, 0.0)',
  tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)');
-- true
SELECT eIntersects(tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)',
  tnpoint '[NPoint(2, 0.0)@2001-01-01, NPoint(2, 1)@2001-01-03)');
-- 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)@2001-01-01, NPoint(1, 0.3)@2001-01-03)');
-- {[t@2001-01-01 00:00:00+01, t@2001-01-03 00:00:00+01)}
SELECT tDwithin(tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03)',
  tnpoint '[NPoint(1, 0.5)@2001-01-01, NPoint(1, 0.3)@2001-01-03)', 1);
/* {[t@2001-01-01 00:00:00+01, t@2001-01-01 22:35:55.379053+01],
   (f@2001-01-01 22:35:55.379053+01, t@2001-01-02 01:24:04.620946+01,
   t@2001-01-03 00:00:00+01)} */
```

11.7. Agregacions

Las tres funciones agregadas para puntos de red temporales se ilustran a continuación.

- Conteo temporal

  `tCount(tnpoint) → {tintSeq,tintSeqSet}`

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2001-01-02, NPoint(1, 0.4)@2001-01-04)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-03, NPoint(1, 0.5)@2001-01-05)' )
SELECT tCount(Temp)
FROM Temp
-- {[1@2001-01-01, 2@2001-01-02, 1@2001-01-04, 1@2001-01-05)}
```

- Conteo de ventana

  `wCount(tnpoint) → {tintSeq,tintSeqSet}`

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2001-01-02, NPoint(1, 0.4)@2001-01-04)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-03, NPoint(1, 0.5)@2001-01-05)' )
SELECT wCount(Temp, '1 day')
FROM Temp
/* {[1@2001-01-01, 2@2001-01-02, 3@2001-01-03, 2@2001-01-04, 1@2001-01-05,
   1@2001-01-06)} */
```

- Centroide temporal

  `tCentroid(tnpoint) → tgeompoint`

```
WITH Temp(temp) AS (
SELECT tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.2)@2001-01-01, NPoint(1, 0.4)@2001-01-03)' UNION
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-03)' )
SELECT astext(tCentroid(Temp))
FROM Temp
/* {[POINT(72.451531682218 76.5231414472853)@2001-01-01,
   POINT(55.7001249027598 72.9552602410653)@2001-01-03)} */
```

11.8. Indexación

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,
- `&&`, `@>`, `<@`, `~=`, `-|-` y `|=|`, 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.

Apéndice A

Referencia de MobilityDB

A.1. Tipos de MobilityDB

MobilityDB define cuatro *tipos de plantilla* (o *template types*) que actúan como constructores de tipos sobre *tipos de base*. Estos tipos de plantilla son `set`, `span`, `spanset` y `temporal`. Tabla A.1 muestra los tipos definidos sobre estos tipos de plantilla. Además, MobilityDB define dos tipos de cuadros delimitadores, a saber, `tbox` y `stbox`.

Tipos de base	**Tipos de plantilla**			
	set	**span**	**spanset**	**temporal**
bool				tbool
text	textset			ttext
integer	intset	intspan	intspanset	tint
bigint	bigintset	bigintspan	bigintspanset	
float	floatset	floatspan	floatspanset	tfloat
date	dateset	datespan	datespanset	
timestamptz	tstzset	tstzspan	tstzspanset	
geometry	geomset			tgeompoint
geography	geogset			tgeogpoint
npoint	npointset			tnpoint

Cuadro A.1: Instancias actuales de los tipos de plantilla en MobilityDB

A.2. Tipos de conjunto y de rango

A.2.1. Entrada y salida

- asText: Obtener la representación textual conocida (Well-Known Text o WKT)
- asBinary: Obtener la representación binaria conocida (Well-Known Binary o WKB)
- asHexWKB: Obtener la representación hexadecimal binaria conocida (HexWKB)
- settypeFromBinary, spantypeFromBinary, spansettypeFromBinary: Obtener a partir de la representación binaria conocida (WKB)
- settypeFromHexWKB, spantypeFromHexWKB, spansettypeFromHexWKB: Obtener a partir de la representación hexadecimal binaria conocida (HexWKB)

A.2.2. Constructores

- set: Constructor para valores de conjunto
- span: Constructores para valores de rango
- spanset: Constructores para valores de conjunto de rangos

A.2.3. Conversión de tipos

- ::, set(base), span(base), spanset(base): Convertir un valor de base en un valor de conjunto, de rango o de conjunto de rangos
- ::, spanset(set): Convertir un valor de conjunto en un valor de conjunto de rangos
- ::, spanset(span): Convertir un rango a un conjunto de rangos
- ::, span(range), range(span): Convertir un rango de MobilityDB hacia y desde un rango PostgreSQL
- ::, multirange(spanset), spanset(multirange): Convertir un rango PostgreSQL a un rango MobilityDB

A.2.4. Accesores

- memSize: Obtener el tamaño de la memoria en bytes
- lower, upper: Obtener el límite inferior o superior
- lowerInc, upperInc: ¿Es el límite inferior o superior inclusivo?
- width: Obtener el ancho del rango como un número de punto flotante
- duration: Obtener el intervalo de tiempo
- span: Obtener el lapso de tiempo delimitador ignorando las posibles brechas de tiempo
- numValues: Obtener el número de valores
- startValue, endValue, valueN: Obtener el valor inicial, final o enésimp
- getValue: Obtener los valores
- numSpans: Obtener el número de rangos o períodos
- startSpan, endSpan, spanN: Obtener el rango inicial, final o enésimo
- spans: Obtener los rangos
- numDates, numTimestamps: Obtener el número de fechas o marcas de tiempo
- startDate, endDate, dateN, startTimestamp, endValue, valueN: Obtener la fecha or marca de tiempo inicial, final o enésima
- dates, timestamps: Obtener los fechas o marcas de tiempo diferentes

A.2.5. Transformaciones

- shift: Desplazar un conjunto o un rango con un valor o intervalo de tiempo
- scale: Escalar un conjunto o un rango con un valor o intervalo de tiempo
- shiftScale: Desplazar y escalar un conjunto o un rango con los valores o intervalos de tiempo
- round: Redondear un conjunto o un rango flotante a un número de decimales
- degrees, radians: Convertir un conjunto flotante a grados o radianes
- lower, upper, initcap: Transformar en minúsculas, majúsculas o initcap
- concat: Concatenación de texto
- tprecision: Establecer la precisión temporal del valor de tiempo al intervalo con respecto al origen

A.2.6. Comparaciones

- =: ¿Son iguales los valores?
- <>: ¿Son diferentes los valores?
- <: ¿Es el primer valor menor que el segundo?
- >: ¿Es el primer valor mayor que el segundo?
- <=: ¿Es el primer valor menor o igual que el segundo?
- >=: ¿Es el primer valor mayor o igual que el segundo?

A.2.7. Sistema de referencia espacial

- SRID: Obtener el identificador de referencia espacial
- setSRID: Especificar el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.2.8. Operaciones de conjuntos

- +: Unión de valores
- -: Diferencia de valores
- *: Intersección de valores

A.2.9. Operaciones de cuadro delimitador

A.2.9.1. Operaciones topológicas

- &&: ¿Se superponen los valores (tienen valores en común)?
- @>: ¿Contiene el primer valor el segundo?
- <@: ¿Está el primer valor contenido en el segundo?
- -|-: ¿Es el primer valor adyacente al segundo?

A.2.9.2. Operaciones de posición

- <<, <<#: ¿Está el primer valor estrictamente a la izquierda del segundo?
- >>, #>>: ¿Está el primer valor de rango estrictamente a la izquierda del segundo?
- &<, &<#: ¿No está el primer valor a la derecha del segundo?
- &>, #&>: ¿No está el primer valor a la izquierda del segundo?

A.2.10. Operaciones de distancia

- <->: Obtener la distancia mínima

A.2.11. Agregaciones

- tCount: Conteo temporal
- extent: rango o período delimitador
- setUnion, spanUnion: Unión agregada

A.3. Tipos de cuadro delimitadores

A.3.1. Entrada y salida

- asText: Obtener la representación textual conocida (Well-Known Text o WKT)
- asBinary: Obtener la representación binaria conocida (Well-Known Binary o WKB)
- asHexWKB: Obtener la representación hexadecimal binaria conocida (HexWKB)
- tboxFromBinary, stboxFromBinary: Obtener a partir de la representación binaria conocida (WKB)
- tboxFromHexWKB, stboxFromHexWKB: Obtener a partir de la representación hexadecimal binaria conocida (HexWKB)

A.3.2. Constructores

- tbox: Constructor para `tbox`
- stboxX, stboxZ, stboxT, stboxXT, stboxZY, geodstboxZ, geodstboxT, geodstboxZT: Constructor para `stbox`

A.3.3. 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.3.4. Accesores

- hasX, hasZ, hasT: ¿Tiene dimension X/Z/T?
- isGeodetic: ¿Es geodética?
- xMin, yMin, zMin, tMin: Obtener el valor mínimo de X/Y/Z/T
- xMax, yMax, zMax, tMax: Obtener el valor máximo de X/Y/Z/T
- xMinInc, tMinInc: Es el mínimo valor X/T inclusivo?
- xMaxInc, tMaxInc: Es el máximo valor X/T inclusivo?

A.3.5. Transformaciones

- shiftValue, shiftTime: Desplazar el rango de valores o el período de un cuadro delimitador con un valor o intervalo
- scaleValue, scaleTime: Escalar el rango de valores o el período de un cuadro delimitador a un valor o intervalo
- shiftScaleValue, shiftScaleTime: Desplazar y escalar el rango de valores o el período del cuadro delimitador con los valores o intervalos
- getSpace: Obtener la dimensión espacial de un cuadro delimitador, eliminando la dimensión temporal, si existe
- expandValue: Extender la dimensión numérica de un cuadro delimitador con un valor flotante
- expandSpace: Extender la dimensión espacial de un cuadro delimitador con un valor flotante
- expandTime: Extender la dimensión temporal de un cuadro delimitador con un intervalo de tiempo
- round: Redondear el valor o las coordenadas de un cuadro delimitador a un número de posiciones decimales

A.3.6. Sistema de referencia espacial

- SRID: Obtener el identificador de referencia espacial
- setSRID: Especificar el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.3.7. Funciones de división

- quadSplit: Dividir el cuadro delimitador en cuadrantes u octantes

A.3.8. Operaciones de conjuntos

- +: Union de los cuadros delimitadores
- *: Intersección de los cuadros delimitadores

A.3.9. Operaciones de cuadro delimitador

A.3.9.1. Operaciones topológicas

- &&: ¿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?
- <<, <<|, <<\/, <<#: ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente mayores que los del segundo?
- >>, |>>, />>, #>>: ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente mayores que los del segundo?
- &<, &<|, &<\/, &<#: ¿No son los valores X/Y/Z/T del primer cuadro delimitador mayores que los del segundo?
- &>, |&>, /&>, #&>: ¿No son los valores X/Y/Z/T del primer cuadro delimitador menores que los del segundo?

A.3.9.2. Operaciones de posición

- <<, <<|, <</, <<#: ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente mayores que los del segundo?
- >>, |>>, />>, #>>: ¿Son los valores X/Y/Z/T del primer cuadro delimitador estrictamente mayores que los del segundo?
- &<, &<|, &</, &<#: ¿No son los valores X/Y/Z/T del primer cuadro delimitador mayores que los del segundo?
- &>, |&>, /&>, #&>: ¿No son los valores X/Y/Z/T del primer cuadro delimitador menores que los del segundo?

A.3.10. Comparaciones

- =: ¿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.3.11. Agregaciones

- extent: Extensión del cuadro delimitador

A.4. Tipos temporales: Operaciones genéricas

A.4.1. Entrada y salida

- asText: Obtener la representación Well-Known Text (WKT)
- asBinary: Obtener la representación Well-Known Binary (WKB)
- asHexWKB: Obtener la representación Hexadecimal Well-Known Binary (HexWKB) como texto
- asMFJSON: Obtener la representación Moving Features JSON (MF-JSON)
- ttypeFromBinary: Obtener a partir de la representación Well-Known Binary (WKB)
- ttypeFromHexWKB: Obtener a partir de la representación Hexadecimal Well-Known Binary (HexWKB)
- ttypeFromMFJSON: Obtener a partir de la representación Moving Features JSON (MF-JSON)

A.4.2. Constructores

- ttype: Constructor de tipos temporales a partir de un valor base y un valor de tiempo
- ttypeSeq: Constructor para tipos temporales de subtipo secuencia
- ttypeSeqSet, ttypeSeqSetGaps: Constructor para tipos temporales de subtipo conjunto de secuencias

A.4.3. Conversión de tipos

- ttype::tstzspan, tnumber::{numspan,tbox}, tpoint::stbox: Convertir un valor temporal a un cuadro delimitador
- tint::tfloat, tfloat::tint: Convertir entre un entero temporal y un flotante temporal
- tgeompoint::tgeogpoint, tgeogpoint::tgeompoint: Convertir entre un punto geométrico temporal y un punto geográfico temporal
- tgeompoint::geometry, tgeogpoint::geography, geometry::tgeompoint, geography::tgeogpoint: Convertir entre un punto temporal y una trayectoria PostGIS

A.4.4. Accesores

- memSize: Obtener el tamaño de la memoria en bytes
- tempSubtype: Obtener el subtipo temporal
- interp: Obtener la interpolación
- getValue, getTimestamp: Obtener el valor o el tiempo del instante
- getValues, getTime: Obtener los valores o el tiempo en el que se define el valor temporal
- valueSpan, timeSpan: Obtener el rango de valores o de tiempo
- valueSet: Obtener los valores del número o punto temporal number como un conjunto
- startValue, endValue: Obtener el valor inicial o final
- minValue, maxValue: Obtener el valor mínimo o máximo
- minInstant, maxInstant: Obtener el instante con el valor mínimo o máximo
- valueAtTimestamp: Obtener el valor en una marca de tiempo
- duration: Obtener el intervalo de tiempo
- lowerInc, upperInc: ¿Es el instante inicial/final inclusivo?
- numInstants: Obtener el número de instantes diferentes
- startInstant, endInstant, instantN: Obtener el instante inicial, final o enésimo
- instants: Obtener los instantes diferentes
- numTimestamps: Obtener el número de marcas de tiempo diferentes
- startTimestamp, endTimestamp, timestampN: Obtener la marca de tiempo inicial, final o enésima
- timestamps: Obtener las marcas de tiempo diferentes
- numSequences: Obtener el número de secuencias
- startSequence, endSequence, sequenceN: Obtener la secuencia inicial, final o enésima
- sequences: Obtener las secuencias
- segments: Obtener los segmentos
- integral: Obtener el área bajo la curva
- twAvg: Obtener el promedio ponderado en el tiempo

A.4.5. Transformaciones

- ttypeInst, ttypeSeq, ttypeSeqSet: Transformar un valor temporal en otro subtipo
- setInterp: Transformar un valor temporal a otra interpolación
- shiftValue, shiftTime: Desplazar el rango de valores o el intervalo de tiempo del valor temporal con un valor or intervalo
- scaleValue, scaleTime: Escalear el rango de valores o el intervalo de tiempo del valor temporal a un valor or intervalo
- shiftScaleValue, shiftScaleTime: Desplazar y escalear el rango de valores o el intervalo de tiempo del valor temporal a los dos valores or intervalos
- stops: Extraer de un sequencia o un conjunto de secuencias temporales con interpolación linear los segmentos cuya distancia es menor o igual que el argumento dados y/o cuya duración es mayor o igual que el argumento dado
- unnest: Transformar un valor temporal no lineal en un conjunto de filas, cada una compuesta de un valor base y un conjunto de períodos durante el cual el valor temporal tiene el valor de base

A.4.6. Modificaciones

- insert: Insertar un valor temporal en otro
- update: Actualizar un valor temporal con otro
- deleteTime: Eliminar de un valor temporal los instantes que intersectan un valor de tiempo
- appendInstant: Anexar un instante temporal a un valor temporal
- appendSequence: Anexar una secuencia temporal a un valor temporal
- merge: Fusionar los valores temporales

A.4.7. Restricciones

- atValues, minusValues: Restringir a (al complemento de) un conjunto de valores
- atMin, minusMin: Restringir al (complemento del) valor mínimo
- atMax, minusMax: Restringir al (complemento del) valor máximo
- atGeometry,minusGeometry, atGeometryTime, minusGeometryTime: Restringir a (al complemento de) una geometría
- atTime, minusTime: Restringir a (al complemento de) un valor de tiempo
- atTbox, minusTbox: Restringir a (al complemento de) un `tbox`
- atStbox, minusStbox: Restringir a (al complemento de) un `stbox`

A.4.8. Comparaciones

A.4.8.1. Comparaciones 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.4.8.2. Comparaciones alguna vez y siempre

- ?=, %=: ¿Es el valor temporal alguna vez o siempre igual al valor?
- ?<>, %<>: ¿Es el valor temporal alguna vez o siempre diferente del valor?
- ?<, %<: ¿Es el valor temporal alguna vez o siempre menor que el valor?
- ?>, %>: ¿Es el valor temporal alguna vez o siempre mayor que el valor?
- ?<=, %<=: ¿Es el valor temporal alguna vez o siempre menor o igual que el valor?
- ?>=, %>=: ¿Es el valor temporal alguna vez o siempre mayor o igual que el valor?

A.4.8.3. Comparaciones temporales

- #=: Igual temporal
- #<>: Diferente temporal
- #<: Menor que temporal
- #>: Mayor que temporal
- #<=: Menor o igual que temporal
- #>=: Mayor o igual que temporal

A.4.9. Funciones de utilidad

- mobilitydb_version: Versión de la extensión MobilityDB
- mobilitydb_full_version: Versión de la extensión MobilityDB y de sus dependencias

A.5. Tipos temporales: Operaciones alfanuméricas

A.5.1. Operaciones matemáticas

- +: Adición temporal
- -: Resta temporal
- *: Multiplicación temporal
- /: División temporal
- abs: Obtener el valor absoluto del número temporal
- deltaValue: Obtener la diferencia de valor entre instantes consectivos del número temporal
- round: Redondear los valores a un número de posiciones decimales
- degrees, radians: Convertir a grados o radianes
- derivative: Obtener la derivada sobre el tiempo del número temporal en unidades por segundo

A.5.2. Operaciones booleanas

- &: Y temporal
- |: O temporal
- ~: No temporal
- whenTrue: Obtener el tiempo cuando el booleano temporal toma el valor verdadero

A.5.3. Operaciones de texto

- ||: Concatenación de texto
- lower, upper, initcap: Transformar en minúsculas, mayúsculas o initcap

A.6. Tipos temporales: Operaciones espaciales

A.6.1. Entrada y 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 JSON o MF-JSON)
- 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, tgeogpointFromText: Entrar un punto temporal en representación de texto conocido (Well-Known Text o WKT)
- tgeompointFromEWKT, tgeogpointFromEWKT: Entrar un punto temporal en representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- tgeompointFromMFJSON, tgeogpointFromMFJSON: Entrar un punto temporal geométrico en una representación JSON de características móviles (Moving Features)
- tgeompointFromBinary, tgeogpointFromBinary: Entrar un punto temporal en representación binaria conocida (Well-Known Binary o WKB)
- tgeompointFromEWKB, tgeogpointFromEWKB: Entrar un punto temporal en una representación extendida binaria conocida (Extended Well-Known Binary o EWKB)
- tgeompointFromHexEWKB, tgeogpointFromHexEWKB: Entrar un punto temporal en representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB) en formato texto

A.6.2. Sistema de referencia espacial

- SRID: Obtener el identificador de referencia espacial
- setSRID: Establecer el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.6.3. Operaciones de cuadro delimitador

- expandSpace: Obtener el cuadro delimitador espaciotemporal expandido en la dimensión espacial por un valor flotante

A.6.4. Accesores

- trajectory: Obtener la trayectoria
- getX, getY, getZ: Obtener los valores de las coordenadas X/Y/Z como un número flotante temporal
- isSimple: Obtener 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
- direction: Obtener la dirección
- azimuth: Obtener el acimut temporal
- angularDifference: Obtener la diferencia angular temporal
- bearing: Obtener el rumbo temporal

A.6.5. Transformaciones

- round: Redondear los valores de las coordenadas a un número de decimales
- makeSimple: Obtener una matriz de fragmentos del punto temporal que son simples
- 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.6.6. Operaciones 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.6.7. Relaciones espaciales

A.6.7.1. Relaciones espaciales alguna vez o siempre

- eContains, aContains: Contiene alguna vez o siempre
- eDisjoint, aDisjoint: Es disjunto alguna vez o siempre
- eDwithin, aDwithin: Está alguna vez o siempre a distancia de
- eIntersects, aIntersects: Cruza alguna vez o siempre
- eTouches, aTouches: Toca alguna vez o siempre

A.6.7.2. Relaciones espaciales temporales

- tContains: Contiene temporal
- tDisjoint: Disjunto temporal
- tDwithin: Estar a distancia de temporal
- tIntersects: Intersección temporal
- tTouches: Toca temporal

A.7. Tipos temporales: Operaciones de análisis

A.7.1. Simplicación

- minDistSimplify, minTimeDeltaSimplify: Simplificar un flotante o un punto temporal asegurándose de que los valores consecutivos estén al menos separados por una cierta distancia o intervalo de tiempo
- maxDistSimplify, douglasPeuckerSimplify: Simplificar un flotante o un punto temporal usando el algoritmo de Douglas-Peucker

A.7.2. Reducción

- tsample: Muestrear un valor temporal con respecto a un intervalo
- tprecision: Reducir la precisión temporal de un valor temporal con respecto a un intervalo calculando el promedio/centroide ponderado por el tiempo en cada intervalo de tiempo

A.7.3. Similaridad

- hausdorffDistance: Obtener la distancia de Hausdorff discreta entre dos valores temporales
- 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
- dynTimeWarpDistance: Obtener la distancia de distorsión de tiempo dinámica (Dynamic Time Warp) entre dos valores temporales
- dynTimeWarpPath: Obtener las parejas de correspondencia entre dos valores temporales con respecto a la distancia de distorsión de tiempo dinámica (Dynamic Time Warp)

A.7.4. Mosaicos multidimensionales

A.7.4.1. Operaciones de intervalos

- bucketList: Obtener un conjunto de intervalos 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
- spanBucket: 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.7.4.2. Operaciones de malla

- tileList: Obtener un conjunto de mosaicos que cubre el cuadro delimitador con mosaicos multidimensionales del mismo tamaño y duración
- tile: Obtener el mosaico de la malla multidimensional que contiene el valor y la marca de tiempo

A.7.4.3. 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.8. Tipos temporales: Agregaciones

- 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.9. Operaciones para puntos de red temporales

A.9.1. Tipos de red estáticos

A.9.1.1. Constructores

- npoint: Constructor para puntos de red
- nsegment: Constructor para segmentos de red

A.9.1.2. Accesores

- route: Obtener el identificador de ruta
- getPosition: Obtener la fracción de posición
- startPosition, endPosition: Obtener la posición inicial o final
- SRID: Obtener el identificador de referencia espacial

A.9.1.3. Transformaciones

- round: Redondear la(s) posición(es) del punto de red or el segmento de red en el número de posiciones decimales

A.9.1.4. Operaciones 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.9.1.5. Comparaciones

- =: ¿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.9.2. Puntos de red temporales

A.9.2.1. Constructores

- tnpoint: Constructor para puntos de red temporal a partir de un valor base y un valor de tiempo
- tnpointSeq: Constructor para puntos de red temporal de subtipo secuencia
- tnpointSeqSet: Constructor para puntos de red temporal de subtipo conjunto de secuencias

A.9.2.2. 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.9.2.3. Operaciones

- tnpointInst, tnpointSeq, tnpointSeqSet: Transformar un punto de red temporal en otro subtipo
- setInterp: Transformar un punto de red temporal en otra interpolación
- round: Redondear la fracción del punto de red temporal en el número de posiciones decimales
- getValues: Obtener los valores
- routes: Obtener los identificadores de ruta
- 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, minusValue: Restringir a (al complemento de) un valor
- atGeometry, minusGeometry: Restringir a (al complemento de) 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
- eContains, aContains, eDisjoint, aDisjoint, eIntersects, aIntersects, eTouches, aTouches, eDwithin, aDwithin: Relaciones espaciales alguna vez o siempre
- tContains, tDisjoint, tIntersects, tTouches, tDwithin: Relaciones espaciales temporales

A.9.2.4. Agregaciones

- 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_int4range`: Generar un rango aleatorio de enteros
- `random_float`: Generar un número flotante aleatorio
- `random_float_array`: Generar una matriz de números flotantes aleatorios
- `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_minutes`: Generar un intervalo de minutos aleatorio
- `random_tstzrange`: Generar rango aleatorio de marcas de tiempo con zona horaria
- `random_tstzrange_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 rango, de tiempo y de cuadro delimitador MobilityDB

- `random_intspan`: Generar un rango aleatorio de enteros
- `random_floatspan`: Generar un rango aleatorio de números flotantes
- `random_tstzspan`: Generar un `tstzspan` aleatorio
- `random_tstzspan_array`: Generar una matriz de valores `tstzspan` aleatorios
- `random_tstzset`: Generar un `tstzset` aleatorio
- `random_tstzspanset`: Generar un `tstzspanset` 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 aleatorio de subtipo instante
- `random_tint_inst`: Generar un entero temporal aleatorio de subtipo instante
- `random_tfloat_inst`: Generar un flotante temporal aleatorio de subtipo instante
- `random_ttext_inst`: Generar un texto temporal aleatorio de subtipo instante
- `random_tgeompoint_inst`: Generar un punto geométrico temporal 2D aleatorio de subtipo instante
- `random_tgeompoint3D_inst`: Generar un punto geométrico temporal 3D aleatorio de subtipo instante
- `random_tgeogpoint_inst`: Generar un punto geográfico temporal 2D aleatorio de subtipo instante
- `random_tgeogpoint3D_inst`: Generar un punto geográfico temporal 3D aleatorio de subtipo instante
- `random_tbool_discseq`: Generar un booleano temporal aleatorio de subtipo secuencia con interpolación discreta
- `random_tint_discseq`: Generar un entero temporal aleatorio de subtipo secuencia con interpolación discreta
- `random_tfloat_discseq`: Generar un flotante temporal aleatorio de subtipo secuencia con interpolación discreta
- `random_ttext_discseq`: Generar un texto temporal aleatorio de subtipo secuencia con interpolación discreta
- `random_tgeompoint_discseq`: Generar un punto temporal geométrico 2D aleatorio de subtipo secuencia con interpolación discreta
- `random_tgeompoint3D_discseq`: Generar un punto geométrico temporal 3D aleatorio de subtipo secuencia con interpolación discreta
- `random_tgeogpoint_discseq`: Generar un punto geográfico temporal 2D aleatorio de subtipo secuencia con interpolación discreta
- `random_tgeogpoint3D_discseq`: Generar un punto geográfico temporal 3D aleatorio de subtipo secuencia con interpolación discreta
- `random_tbool_seq`: Generar un booleano temporal aleatorio de subtipo secuencia

- `random_tint_seq`: Generar un entero temporal aleatorio de subtipo secuencia
- `random_tfloat_seq`: Generar un flotante temporal aleatorio de subtipo secuencia
- `random_ttext_seq`: Generar un texto temporal aleatorio de subtipo secuencia
- `random_tgeompoint_seq`: Generar un punto geométrico temporal 2D aleatorio de subtipo secuencia
- `random_tgeompoint3D_seq`: Generar un punto geométrico temporal 3D aleatorio de subtipo secuencia
- `random_tgeogpoint_seq`: Generar un punto geográfico temporal 2D aleatorio de subtipo secuencia
- `random_tgeogpoint3D_seq`: Generar un punto geográfico temporal 3D aleatorio de subtipo secuencia
- `random_tbool_seqset`: Generar un booleano temporal aleatorio de subtipo conjunto de secuencias
- `random_tint_seqset`: Generar un entero temporal aleatorio de subtipo conjunto de secuencias
- `random_tfloat_seqset`: Generar un flotante temporal aleatorio de subtipo conjunto de secuencias
- `random_ttext_seqset`: Generar un texto temporal aleatorio de subtipo conjunto de secuencias
- `random_tgeompoint_seqset`: Generar un punto geométrico temporal 2D aleatorio de subtipo conjunto de secuencias
- `random_tgeompoint3D_seqset`: Generar un punto geométrico temporal 3D aleatorio de subtipo conjunto de secuencias
- `random_tgeogpoint_seqset`: Generar un punto geográfico temporal 2D aleatorio de subtipo conjunto de secuencias
- `random_tgeogpoint3D_seqset`: Generar un punto geográfico temporal 3D aleatorio de subtipo conjunto de secuencias

B.5. Generación de tablas con valores aleatorios

Los archivos `create_test_tables_temporal.sql` y `create_test_tables_tpoint.sql` 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 integer DEFAULT 100)
RETURNS text AS $$
DECLARE
  perc integer;
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_floatspan` que contiene valores aleatorios `floatspan` 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_floatspan AS
/* Add perc NULL valores */
SELECT k, NULL AS f
FROM generate_series(1, perc) AS k UNION
SELECT k, random_floatspan(0, 100, 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_tstzset` que contiene valores aleatorios `tstzset` 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_tstzset AS
/* Add perc NULL valores */
SELECT k, NULL AS ts
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tstzset('2001-01-01', '2001-12-31', 10, 5, 10)
FROM generate_series(perc+1, size) AS k;
```

La creación de una tabla `tbl_tstzspan` que contiene valores aleatorios `tstzspan` 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_tstzspan AS
/* Add perc NULL valores */
SELECT k, NULL AS p
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tstzspan('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;
```

Nótese 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_discseq` que contiene valores aleatorios `tint` valores de subtipo secuencia con interpolación discreta 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_discseq AS
/* Add perc NULL valores */
SELECT k, NULL AS ti
FROM generate_series(1, perc) AS k UNION
SELECT k, random_tint_discseq(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_discseq t1
SET ti = (SELECT ti FROM tbl_tint_discseq 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_discseq t1
SET ti = (SELECT ti + random_int(1, 2) FROM tbl_tint_discseq 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_discseq t1
SET ti = (SELECT shift(ti, endTimestamp(ti)-startTimestamp(ti))
  FROM tbl_tint_discseq 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_discseq t1
SET ti = (SELECT shift(ti, date_trunc('minute',(endTimestamp(ti)-startTimestamp(ti))/2))
  FROM tbl_tint_discseq 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 los valores `float` 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_contseq(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_discseq` que contiene valores aleatorios `tgeompoint` 2D valores de subtipo secuencia con interpolación discreta 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_discseq AS
SELECT k, random_tgeompoint_discseq(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_discseq t1
SET ti = (SELECT ti FROM tbl_tgeompoint_discseq 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_discseq t1
SET ti = (SELECT round(ti,6) FROM tbl_tgeompoint_discseq 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_discseq t1
SET ti = (SELECT shift(ti, endTimestamp(ti)-startTimestamp(ti))
  FROM tbl_tgeompoint_discseq 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_discseq t1
SET ti = (SELECT shift(ti, date_trunc('minute',(endTimestamp(ti)-startTimestamp(ti))/2))
  FROM tbl_tgeompoint_discseq 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 round(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 aleatorio de subtipo instant
- `random_tnpoint_discseq`: Generar un punto de red temporal aleatorio de subtipo secuencia e interpolación discreta

- `random_tnpoint_seq`: Generar un punto de red temporal aleatorio de subtipo secuencia e interpolación linear o escalonada
- `random_tnpoint_seqset`: Generar un punto de red temporal aleatorio de subtipo conjunto de secuencias

El archivo `/datagen/npoint/create_test_tables_tnpoint.sql` da ejemplos de utilización de las funciones que generan valores aleatorios listadas arriba.

Capítulo 12

Índice alfabético

_
*, 22, 37, 81
+, 21, 37, 80
-, 22, 81
-|-, 23, 38, 126
/, 81
/>>, 39
/&>, 40
::, 14, 15, 31, 32, 56, 57, 118, 121, 125
<, 25, 40, 75, 119
<->, 24, 93, 126
<<, 23, 39
<</, 39
<<#, 23, 39
<<|, 39
<=, 25, 41, 76, 119
<>, 25, 40, 75, 119
<@, 23, 38, 126
=, 25, 40, 75, 118
>, 25, 41, 76, 119
>=, 26, 41, 76, 119
>>, 23, 39
?<, 77
?<=, 77
?<>, 76
?=, 76, 125
?>, 77
?>=, 77
@>, 22, 38, 126
#<, 78
#<=, 78
#<>, 78
#=, 78
#>, 78
#>=, 78
#>>, 23, 39
#&>, 24, 40
%<, 77
%<=, 77
%<>, 76
%=, 76
%>, 77
%>=, 77
&, 80
&<, 24, 39
&</, 39
&<#, 24, 39
&<|, 39
&=, 125
&>, 24, 40
&&, 22, 37, 126
|, 80
|=|, 91, 126
|>>, 39
|&>, 40
||, 20, 82
~, 80
~=, 38, 126

A
abs, 81
aContains, 94, 127
aDisjoint, 94, 127
aDwithin, 95, 127
aIntersects, 95, 127
angularDifference, 88
appendInstant, 68
appendSequence, 70
asBinary, 12, 29, 52, 84
asEWKB, 84
asEWKT, 83
asHexEWKB, 84
asHexWKB, 12, 29, 52
asMFJSON, 52, 83
asMVTGeom, 90
asText, 11, 29, 51, 83
atGeometry, 72, 124
atMax, 72
atMin, 71
aTouches, 95, 127
atStbox, 74
atTbox, 74
atTime, 73
atValue, 124
atValues, 71
azimuth, 88, 123

B
bearing, 89
bucketList, 103

C
cumulativeLength, 88, 123

D
dateN, 18
dates, 18
degrees, 20, 82
deltaValue, 81
derivative, 82
direction, 88
douglasPeuckerSimplify, 97
duration, 16, 60
dynTimeWarpDistance, 101
dynTimeWarpPath, 102

E
eContains, 94, 127
eDisjoint, 94, 127
eDwithin, 95, 127
eIntersects, 95, 127
endDate, 18
endInstant, 60
endPosition, 117
endSequence, 61
endSpan, 17
endTimestamp, 18, 61
endValue, 16, 59
eTouches, 95, 127
expandSpace, 35, 86
expandTime, 35
expandValue, 35
extent, 26, 41, 110

F
floatset, 13
frechetDistance, 101
frechetDistancePath, 101

G
geodstboxT, 30
geodstboxZ, 30
geodstboxZT, 30
geoMeasure, 90
getPosition, 117
getSpace, 34
getTimestamp, 58
getValue, 58
getValues, 17, 58, 122
getX, 87
getY, 87
getZ, 87

H
hasT, 32
hasX, 32
hasZ, 32
hausdorffDistance, 101

I
initcap, 20, 82
instantN, 60
instants, 60
integral, 62
interp, 57
intset, 13
isGeodetic, 32
isSimple, 87

L
length, 87, 123
lower, 15, 20, 82
lowerInc, 16, 60

M
makeSimple, 89
maxDistSimplify, 97
maxInstant, 59
maxValue, 59
memSize, 15, 57
merge, 70
minDistSimplify, 97
minInstant, 59
minTimeDeltaSimplify, 97
minusGeometry, 72, 124
minusMax, 72
minusMin, 71
minusStbox, 74
minusTbox, 74
minusTime, 73
minusValue, 124
minusValues, 71
minValue, 59
mobilitydb_full_version, 79
mobilitydb_version, 79
multirange, 15

N
nearestApproachDistance, 124
nearestApproachInstant, 91, 123
npoint, 116
nsegment, 117
numDates, 17
numInstants, 60
numSequences, 61
numSpans, 17
numTimestamps, 17, 61
numValues, 16

P
periodBucket, 104

Q

quadSplit, 36

R
radians, 20, 82
range, 14
round, 19, 35, 82, 89, 117, 122
route, 117
routes, 122

S
scale, 19
scaleTime, 34, 63
scaleValue, 34, 63
segments, 62
sequenceN, 61
sequences, 61
set, 14
setInterp, 62, 122
setSRID, 21, 35, 86
settypeFromBinary, 12
settypeFromHexWKB, 12
setUnion, 26
shift, 18
shiftScale, 19
shiftScaleTime, 34, 63
shiftScaleValue, 34, 63
shiftTime, 34, 63
shiftValue, 34, 63
shortestLine, 92, 124
spaceSplit, 107
spaceTimeSplit, 108
span, 13, 14
spanBucket, 104
spanN, 17
spans, 17
spanset, 13–15
spansettypeFromBinary, 12
spansettypeFromHexWKB, 12
spantypeFromBinary, 12
spantypeFromHexWKB, 12
spanUnion, 26
speed, 88, 123
SRID, 20, 35, 85, 117
startDate, 18
startInstant, 60
startPosition, 117
startSequence, 61
startSpan, 17
startTimestamp, 18, 61
startValue, 16, 59
stbox, 123
stboxFromBinary, 29
stboxFromHexWKB, 30
stboxT, 30
stboxX, 30
stboxXT, 30
stboxZ, 30
stboxZT, 30
stops, 64

T
tAnd, 110
tAvg, 110
tbox, 30
tboxFromBinary, 29
tboxFromHexWKB, 30
tCentroid, 111, 128
tContains, 96, 127
tCount, 27, 109, 127
tDisjoint, 96, 127
tDwithin, 96, 127
tempSubtype, 57
textset, 13
tgeogpointFromBinary, 85
tgeogpointFromEWKB, 85
tgeogpointFromEWKT, 84
tgeogpointFromHexEWKB, 85
tgeogpointFromMFJSON, 85
tgeogpointFromText, 84
tgeompointFromBinary, 85
tgeompointFromEWKB, 85
tgeompointFromEWKT, 84
tgeompointFromHexEWKB, 85
tgeompointFromMFJSON, 85
tgeompointFromText, 84
tile, 105
tileList, 104
timeBucket, 104
timeSpan, 58
timeSplit, 106
timestampN, 18, 61
timestamps, 18, 61
tIntersects, 96, 127
tMax, 33, 110
tMaxInc, 33
tMin, 33, 110
tMinInc, 33
tnpoint, 120
tnpointInst, 122
tnpointSeq, 121, 122
tnpointSeqSet, 121, 122
tOr, 110
tprecision, 20, 99
trajectory, 86
transform, 21, 36, 86
transformPipeline, 21, 36, 86
tsample, 98
tstzset, 13
tSum, 110
tTouches, 96, 127
ttype, 54
ttypeFromBinary, 53
ttypeFromHexWKB, 53
ttypeFromMFJSON, 53

ttypeInst, 62
ttypeSeq, 54, 62
ttypeSeqSet, 55, 62
ttypeSeqSetGaps, 55
twAvg, 62
twCentroid, 88, 123

U
unnest, 64
upper, 15, 20, 82
upperInc, 16, 60

V
valueAtTimestamp, 59, 122
valueBucket, 104
valueN, 16
valueSet, 59
valueSpan, 58
valueSplit, 106
valueTimeSplit, 107

W
wAvg, 111
wCount, 111, 128
whenTrue, 80
width, 16
wMax, 110
wMin, 110
wSum, 111

X
xMax, 33
xMaxInc, 33
xMin, 33
xMinInc, 33

Y
yMax, 33
yMin, 33

Z
zMax, 33
zMin, 33