MobilityDB 1.3 Manual de usuario

Esteban Zimányi

COLABORADORES			
	TÍTULO : MobilityDB 1.3 Manual de usuario		
ACCIÓN	*NOMBRE*	*FECHA*	*FIRMA*
ESCRITO POR	Esteban Zimányi	6 de agosto de 2026	

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 . . . 6

1.5.8. Documentación . . . 6

1.6. Instalación a partir de binarios . . . 7

1.6.1. Distribuciones de Linux basadas en Debian . . . 7

1.6.2. Windows . . . 7

1.7. Soporte . . . 7

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 . . . 8

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 . . . 21

2.7. Operaciones de conjuntos . . . 22

2.8. Operaciones de cuadro delimitador . . . 23
2.8.1. Operaciones topológicas . . . 23
2.8.2. Operaciones de posición . . . 24
2.8.3. Operaciones de división . . . 25
2.9. Operaciones de distancia . . . 26
2.10. Comparaciones . . . 26
2.11. Agregaciones . . . 27
2.12. Indexación . . . 28

3. Tipos de cuadro delimitador **29**
3.1. Entrada y salida . . . 29
3.2. Constructores . . . 31
3.3. Conversión de tipos . . . 32
3.4. Accesores . . . 33
3.5. Transformaciones . . . 35
3.6. Sistema de referencia espacial . . . 36
3.7. Operaciones de división . . . 37
3.8. Operaciones de conjuntos . . . 38
3.9. Operaciones de cuadro delimitador . . . 39
3.9.1. Operaciones topológicas . . . 39
3.9.2. Operaciones de posición . . . 40
3.10. Comparaciones . . . 41
3.11. Aggregaciones . . . 42
3.12. Indexación . . . 42

4. Tipos temporales (Parte 1) **43**
4.1. Introduction . . . 43
4.2. Ejemplos de tipos temporales . . . 45
4.3. Validez de los tipos temporales . . . 47
4.4. Temporalización de operaciones . . . 47
4.5. Notación . . . 48
4.6. Entrada y salida . . . 49
4.7. Constructores . . . 53
4.8. Conversión de tipos . . . 55
4.9. Accesores . . . 56
4.10. Transformaciones . . . 60

5. Tipos temporales (Parte 2) 63
5.1. Modificaciones . . . 63
5.2. Restricciones . . . 68
5.3. Operadores de cuadro delimitador . . . 70
5.4. Comparaciones . . . 70
5.4.1. Comparaciones tradicionales . . . 70
5.4.2. Comparaciones alguna vez y siempre . . . 71
5.4.3. Comparaciones temporales . . . 72
5.5. Funciones de utilidad . . . 73

6. Tipos alfanuméricos temporales 74
6.1. Notación . . . 74
6.2. Conversión de tipos . . . 74
6.3. Accessores . . . 75
6.4. Transformaciones . . . 76
6.5. Restrictions . . . 77
6.6. Operaciones booleanas . . . 78
6.7. Operaciones matemáticas . . . 78
6.8. Operaciones de texto . . . 81

7. Tipos temporales geométricos (Parte 1) 82
7.1. Tipos espaciotemporales . . . 82
7.2. Notación . . . 82
7.3. Entrada y salida . . . 84
7.4. Conversión de tipos . . . 86
7.5. Accessores . . . 87
7.6. Transformaciones . . . 90

8. Tipos temporales geométricos (Parte 2) 94
8.1. Restricciones . . . 94
8.2. Sistema de referencia espacial . . . 95
8.3. Operaciones de cuadro delimitador . . . 96
8.4. Operaciones de distancia . . . 96
8.5. Relaciones espaciales . . . 100
8.5.1. Relaciones alguna vez o siempre . . . 101
8.5.2. Relaciones espaciotemporales . . . 102

9. Tipos temporales: Operaciones de análisis **104**
9.1. Simplificación . . . 104
9.2. Reducción . . . 105
9.3. Similaridad . . . 108
9.4. Operaciones de división . . . 110
9.5. Mosaicos multidimensionales . . . 114
9.5.1. Operaciones de intervalos . . . 115
9.5.2. Operaciones de mosaicos . . . 115
9.5.3. Operaciones de cuadro delimitador . . . 118
9.5.4. Operaciones de división . . . 120

10. Tipos temporales: Agregación e indexación **123**
10.1. Agregación . . . 123
10.2. Indexación . . . 125
10.3. Estadísticas y selectividad . . . 126
10.3.1. Colecta de estadísticas . . . 126
10.3.2. Estimación de la selectividad . . . 128

11. Poses temporales **129**
11.1. Poses estáticas . . . 129
11.1.1. Entrada y salida . . . 130
11.1.2. Constructores . . . 131
11.1.3. Conversión de tipos . . . 131
11.1.4. Accesores . . . 132
11.1.5. Transformaciones . . . 132
11.1.6. Sistema de referencia espacial . . . 132
11.1.7. Comparaciones . . . 133
11.1.8. Soporte de OGC GeoPose v1.0 . . . 133
11.1.8.1. Entrada y salida JSON . . . 133
11.1.8.2. Renormalización de cuaterniones . . . 134
11.1.8.3. Accesores de ángulos de Euler . . . 134
11.1.8.4. Registro de metadatos de marcos . . . 135
11.1.8.5. Transformación rígida cuerpo↔mundo . . . 135
11.1.8.6. Transformaciones de marco entre SRID . . . 136
11.1.8.7. E/S JSON temporal . . . 136
11.1.8.8. Velocidad y velocidad angular . . . 137
11.2. Poses temporales . . . 137
11.3. Validez de las poses temporales . . . 138
11.4. Entrada y salida . . . 138

11.5. Constructores . . . 140
11.6. Conversión de tipos . . . 141
11.7. Accesores . . . 141
11.8. Transformaciones . . . 142
11.9. Modificaciones . . . 143
11.10Restricciones . . . 143
11.11Sistema de referencia espacial . . . 144
11.12Operaciones de cuadro delimitador . . . 145
11.13Operaciones de distancia . . . 145
11.14Relaciones espaciales . . . 146
11.15Comparaciones . . . 147
11.16Agregaciones . . . 148
11.17Indexación . . . 148

12. Puntos de red temporales **149**
12.1. Tipos de red estáticos . . . 149
12.1.1. Constructores . . . 151
12.1.2. Conversión de tipos . . . 151
12.1.3. Accesores . . . 151
12.1.4. Transformaciones . . . 152
12.1.5. Operaciones espaciales . . . 152
12.1.6. Comparaciones . . . 152
12.2. Puntos de red temporales . . . 153
12.3. Validez de los puntos de red temporal . . . 154
12.4. Constructores . . . 154
12.5. Conversión de tipos . . . 155
12.6. Entrada y Salida MF-JSON . . . 155
12.7. Accessores . . . 155
12.8. Transformaciones . . . 156
12.9. Modificaciones . . . 157
12.10Restricciones . . . 157
12.11Operaciones de distancia . . . 158
12.12Operaciones espaciales . . . 159
12.13Operaciones de cuadro delimitador . . . 159
12.14Relaciones espaciales . . . 160
12.15Comparaciones . . . 160
12.16Agregacions . . . 161
12.17Indexación . . . 162
12.18Agregación . . . 162

12.19 Guía de producción . . . 162
12.19.1 Cuándo usar tnpoint frente a tgeompoint . . . 163
12.19.2 Herencia del SRID desde el registro de la red . . . 163
12.19.3 Peligros numéricos . . . 163
12.19.4 Durabilidad y almacenamiento . . . 164

13. Búferes circulares temporales **165**
13.1. Búferes circulares estáticos . . . 166
13.1.1. Constructores . . . 166
13.1.2. Conversión de tipos . . . 166
13.1.3. Accesores . . . 167
13.1.4. Transformaciones . . . 167
13.1.5. Operaciones espaciales . . . 167
13.1.6. Comparaciones . . . 168
13.2. Búferes circulares temporales . . . 168
13.3. Validez de los búferes circulares temporales . . . 169
13.4. Entrada y salida . . . 169
13.5. Constructores . . . 171
13.6. Conversión de tipos . . . 172
13.7. Accesores . . . 172
13.8. Transformaciones . . . 173
13.9. Modificaciones . . . 174
13.10 Operaciones espaciales . . . 174
13.11 Operaciones de distancia . . . 176
13.12 Restricciones . . . 177
13.13 Comparaciones . . . 177
13.14 Operaciones de cuadro delimitador . . . 178
13.15 Relaciones espaciales . . . 179
13.16 Agregaciones . . . 180
13.17 Simplificación . . . 181
13.18 Indexación . . . 181

14. Geometrías rígidas temporales **182**
14.1. Entrada y salida . . . 182
14.2. Constructores . . . 184
14.3. Conversión de tipos . . . 184
14.4. Accesores . . . 185
14.5. Área recorrida . . . 186
14.6. Funciones espaciales . . . 186

14.7. Métricas de movimiento . . . 187
14.8. Modificaciones . . . 187
14.9. Restricciones espaciales . . . 188
14.10 Operaciones de distancia . . . 189
14.11 Similitud . . . 190
14.12 Comparaciones . . . 191
14.13 Operadores de caja delimitadora . . . 191
14.14 Mosaicos . . . 192
14.15 Cajas delimitadoras . . . 192
14.16 Agregaciones . . . 193
14.17 Indexación . . . 193
14.18 Guía de producción . . . 194
14.18.1. Cuándo usar trgeometry frente a tpose frente a tgeometry . . . 194
14.18.2. Convenciones de marco . . . 194
14.18.3. Riesgos numéricos . . . 195
14.18.4. Durabilidad y almacenamiento . . . 195

15. Tipos JSON en MobilityDB 196

15.1. Operaciones sobre conjuntos JSONB . . . 196
15.2. Valores JSONB temporales . . . 202
15.3. Validez de los valores JSONB temporales . . . 203
15.4. Entrada y salida . . . 204
15.5. Constructores . . . 205
15.6. Conversiones . . . 206
15.7. Accesores . . . 206
15.8. Transformaciones . . . 207
15.9. Operaciones JSON temporales . . . 207
15.10 Restricciones . . . 214
15.11 Comparaciones . . . 214
15.12 Operaciones de caja delimitadora . . . 215
15.13 Agregaciones . . . 215
15.14 Indexación . . . 216

16. Índices de Celdas H3 Temporales 217

16.1. Entrada y Salida . . . 217
16.2. Conversiones . . . 218
16.3. Cobertura de Geometrías Estáticas . . . 218
16.4. Funciones de Acceso . . . 219
16.5. Inspección de Índices . . . 219

16.6. Jerarquía . . . 220
16.7. Conversiones de Latitud/Longitud . . . 221
16.8. Aristas Dirigidas . . . 221
16.9. Vértices . . . 222
16.10 Recorrido de la Cuadrícula . . . 223
16.11 Métricas . . . 223
16.12 Comparaciones . . . 224
16.12.1. Comparaciones Siempre y Alguna Vez . . . 224
16.12.2. Comparaciones Temporales . . . 225
16.13 Operadores de Caja Envolvente . . . 225
16.14 Funciones que devuelven conjuntos . . . 226
16.15 Relaciones Espaciales . . . 227
16.16 Indexación . . . 228
16.17 Notas operativas . . . 228
16.17.1. Cuándo usar th3index . . . 228
16.17.2. Peligros específicos de H3 . . . 228
16.17.3. Durabilidad y almacenamiento . . . 229

17. Índices de Celdas QUADBIN Temporales **231**
17.1. Entrada y Salida . . . 231
17.2. Conversiones . . . 232
17.3. Celdas Estáticas . . . 232
17.4. Accesores . . . 233
17.5. Inspección del Índice . . . 233
17.6. Jerarquía . . . 234
17.7. Conversiones de Geometría . . . 235
17.8. Teselas y Quadkeys . . . 235
17.9. Métricas . . . 236
17.10 Comparaciones . . . 236
17.11 Indexación . . . 236
17.12 Notas operativas . . . 237
17.12.1. Cuándo usar tquadbin . . . 237
17.12.2. Peligros específicos de QUADBIN . . . 237
17.12.3. Durabilidad y almacenamiento . . . 238

18. Nubes de Puntos Temporales 239
18.1. Inicio rápido . . . 239
18.1.1. Registrar un esquema . . . 239
18.1.2. Construir valores base y temporales . . . 240
18.1.3. Consultar con la superficie bbox . . . 240
18.1.4. Índice para búsquedas impulsadas por bbox . . . 241
18.1.5. Agregación . . . 241
18.1.6. Acceso por punto . . . 241
18.1.7. ¿Qué sigue? . . . 241
18.2. Tipos estáticos `pcpoint` y `pcpatch` . . . 241
18.2.1. Constructores ergonómicos . . . 242
18.2.2. Funciones de acceso . . . 242
18.3. Tipos de conjunto `pcpointset` y `pcpatchset` . . . 243
18.3.1. Constructores . . . 243
18.4. Caja Delimitadora `tpcbox` . . . 243
18.4.1. Constructores . . . 243
18.4.2. Conversiones . . . 243
18.4.3. Accesores . . . 244
18.4.4. Transformaciones . . . 245
18.4.5. Operaciones de conjuntos . . . 245
18.4.6. Operadores Topológicos . . . 245
18.4.7. Operadores de Posición . . . 246
18.4.8. Comparaciones . . . 247
18.5. Tipo Temporal `tpcpoint` . . . 247
18.5.1. Constructores . . . 248
18.5.2. Funciones de Acceso . . . 248
18.5.3. Proyecciones por Dimensión . . . 248
18.5.4. Conversiones . . . 249
18.5.5. Transformaciones . . . 249
18.5.6. Comparaciones . . . 250
18.5.7. Restricciones . . . 251
18.5.8. Modificaciones . . . 252
18.5.9. Fragmentación . . . 253
18.5.10.Operadores de caja delimitadora . . . 253
18.5.11.Ordenación B-tree — qué significa `ORDER BY tpcpoint` . . . 254
18.5.12.Relaciones espaciales . . . 254
18.6. Tipo Temporal `tpcpatch` . . . 255
18.6.1. Constructores . . . 255
18.6.2. Accesores . . . 256

18.6.3. Transformaciones . . . 257
18.6.4. Restricciones . . . 257
18.6.5. Modificaciones . . . 259
18.6.6. Fragmentación . . . 260
18.6.7. Relaciones espaciales . . . 260
18.6.8. Operadores de caja delimitadora . . . 261
18.6.9. Comparaciones . . . 261
18.6.10. Ordenación B-tree — qué significa `ORDER BY tpcpatch` . . . 262
18.6.11. Operaciones por punto: qué está y qué no está disponible . . . 262
18.7. Índices . . . 263
18.8. Agregaciones . . . 263
18.9. Representación binaria y portabilidad . . . 265
18.10 Salida de texto . . . 265
18.11 Salida MF-JSON . . . 266
18.12 Integración con PDAL: `readers.tpcpatch` y `writers.tpcpatch` . . . 266
18.12.1. `readers.tpcpatch` . . . 266
18.12.2. `writers.tpcpatch` . . . 267
18.12.3. Esquemas de compresión soportados . . . 267
18.12.4. Limitaciones . . . 268
18.13 Puente PCL: filtros invocables desde SQL y registro . . . 268
18.14 Advertencias . . . 268

19. Muestreo de Ráster **270**
19.1. Operador de Muestreo . . . 270
19.2. Muestreo Raquet / QUADBIN . . . 271
19.3. Accesores y Comparación de Raquet . . . 273
19.4. Operadores de Restricción y Predicado . . . 275
19.5. Compilación e Instalación . . . 276
19.6. Hoja de Ruta de Producción . . . 276

20. Dialecto Portable de Funciones con Nombre **277**

A. Referencia de MobilityDB **280**
A.1. Tipos de MobilityDB . . . 280
A.2. Tipos de conjunto y de rango . . . 280
A.2.1. Entrada y salida . . . 280
A.2.2. Constructores . . . 281
A.2.3. Conversión de tipos . . . 281
A.2.4. Accesores . . . 281
A.2.5. Transformaciones . . . 282

A.2.6. Sistema de referencia espacial . . . 282
A.2.7. Operaciones de conjuntos . . . 282
A.2.8. Operaciones de cuadro delimitador . . . 282
A.2.8.1. Operaciones topológicas . . . 282
A.2.8.2. Operaciones de posición . . . 282
A.2.8.3. Operaciones de división . . . 282
A.2.9. Operaciones de distancia . . . 283
A.2.10. Comparaciones . . . 283
A.2.11. Agregaciones . . . 283
A.3. Tipos de cuadro delimitadores . . . 283
A.3.1. Entrada y salida . . . 283
A.3.2. Constructores . . . 283
A.3.3. Conversión de tipos . . . 283
A.3.4. Accesores . . . 283
A.3.5. Transformaciones . . . 284
A.3.6. Sistema de referencia espacial . . . 284
A.3.7. Operaciones de división . . . 284
A.3.8. Operaciones de conjuntos . . . 284
A.3.9. Operaciones de cuadro delimitador . . . 284
A.3.9.1. Operaciones topológicas . . . 284
A.3.9.2. Operaciones de posición . . . 284
A.3.10. Comparaciones . . . 285
A.3.11. Agregaciones . . . 285
A.4. Tipos temporales . . . 285
A.4.1. Entrada y salida . . . 285
A.4.2. Constructores . . . 285
A.4.3. Conversión de tipos . . . 285
A.4.4. Accesores . . . 285
A.4.5. Transformaciones . . . 286
A.4.6. Modificaciones . . . 286
A.4.7. Restricciones . . . 286
A.4.8. Comparaciones . . . 286
A.4.9. Funciones de utilidad . . . 286
A.5. Tipos temporales alfanuméricos . . . 287
A.5.1. Conversión de tipos . . . 287
A.5.2. Accessores . . . 287
A.5.3. Transformaciones . . . 287
A.5.4. Restricciones . . . 287
A.5.5. Operaciones booleanas . . . 287

A.5.6. Operaciones matemáticas . . . 287
A.5.7. Operaciones de texto . . . 288
A.6. Tipos temporales geométricos . . . 288
A.6.1. Entrada y salida . . . 288
A.6.2. Conversión de tipos . . . 288
A.6.3. Sistema de referencia espacial . . . 288
A.6.4. Accesores . . . 289
A.6.5. Transformaciones . . . 289
A.6.6. Restricciones . . . 289
A.6.7. Operaciones de distancia . . . 289
A.6.8. Relaciones espaciales . . . 290
A.6.8.1. Relaciones alguna vez o siempre . . . 290
A.6.8.2. Relaciones espaciotemporales . . . 290
A.7. Tipos temporales: Operaciones de análisis . . . 290
A.7.1. Simplicación . . . 290
A.7.2. Reducción . . . 290
A.7.3. Similaridad . . . 290
A.7.4. Operaciones de división de cuadro delimitador . . . 291
A.7.5. Mosaicos multidimensionales . . . 291
A.7.5.1. Operaciones de intervalos . . . 291
A.7.5.2. Operaciones de mosaico . . . 291
A.7.5.3. Operaciones de cuadro delimitador . . . 291
A.7.5.4. Operaciones de división . . . 291
A.7.6. Agregaciones . . . 292
A.8. Poses temporales . . . 292
A.8.1. Poses estáticas . . . 292
A.8.1.1. Entrada y salida . . . 292
A.8.1.2. Constructores . . . 292
A.8.1.3. Conversiones de tipo . . . 292
A.8.1.4. Accesores . . . 292
A.8.1.5. Transformaciones . . . 292
A.8.1.6. Sistema de referencia espacial . . . 293
A.8.1.7. Comparaciones . . . 293
A.8.2. Poses temporales . . . 293
A.8.2.1. Entrada y salida . . . 293
A.8.2.2. Constructores . . . 293
A.8.2.3. Conversiones de tipo . . . 293
A.8.2.4. Accesores . . . 293
A.8.2.5. Transformaciones . . . 294

A.8.2.6. Modificaciones . . . 294
A.8.2.7. Restricciones . . . 294
A.8.2.8. Sistema de referencia espacial . . . 294
A.8.2.9. Operaciones de cuadro delimitador . . . 294
A.8.2.10. Operadores de distancia . . . 294
A.8.2.11. Comparaciones . . . 294
A.8.2.12. Agregaciones . . . 294
A.9. Operaciones para puntos de red temporales . . . 295
A.9.1. Tipos de red estáticos . . . 295
A.9.1.1. Constructores . . . 295
A.9.1.2. Conversiones de tipo . . . 295
A.9.1.3. Accesores . . . 295
A.9.1.4. Transformaciones . . . 295
A.9.1.5. Operaciones espaciales . . . 295
A.9.1.6. Comparaciones . . . 295
A.9.2. Puntos de red temporales . . . 295
A.9.2.1. Constructores . . . 295
A.9.2.2. Conversión de tipos . . . 295
A.9.2.3. Accesores . . . 296
A.9.2.4. Transformaciones . . . 296
A.9.2.5. Modifications . . . 296
A.9.2.6. Restricciones . . . 296
A.9.2.7. Operadores de distancia . . . 296
A.9.2.8. Operaciones espaciales . . . 296
A.9.2.9. Operaciones de cuadro delimitador . . . 296
A.9.2.10. Relaciones espaciales . . . 297
A.9.2.11. Comparaciones . . . 297
A.9.2.12. Agregaciones . . . 297
A.10. Búferes circulares temporales . . . 297
A.10.1. Búferes circulares estáticos . . . 297
A.10.1.1. Constructores . . . 297
A.10.1.2. Conversiones de tipo . . . 297
A.10.1.3. Accesores . . . 297
A.10.1.4. Transformaciones . . . 297
A.10.1.5. Operaciones espaciales . . . 297
A.10.1.6. Comparaciones . . . 298
A.10.2. Búferes circulares temporales . . . 298
A.10.2.1. Entrada y salida . . . 298
A.10.2.2. Constructores . . . 298

A.10.2.3. Conversiones de tipo . . . 298
A.10.2.4. Accesores . . . 298
A.10.2.5. Transformaciones . . . 298
A.10.2.6. Modificaciones . . . 299
A.10.2.7. Restricciones . . . 299
A.10.2.8. Operaciones de distancia . . . 299
A.10.2.9. Operaciones espaciales . . . 299
A.10.2.10 Operaciones de cuadro delimitador . . . 299
A.10.2.11 Relaciones espaciales . . . 299
A.10.2.12 Comparaciones . . . 299
A.10.2.13 Agregaciones . . . 299
A.11. Geometrías rígidas temporales . . . 300
A.11.1. Entrada y salida . . . 300
A.11.2. Constructores . . . 300
A.11.3. Conversiones de tipo . . . 300
A.11.4. Accesores . . . 300
A.11.5. Área recorrida . . . 300
A.11.6. Funciones espaciales . . . 300
A.11.7. Métricas de movimiento . . . 301
A.11.8. Modificaciones . . . 301
A.11.9. Restricciones . . . 301
A.11.10 Operadores de distancia . . . 301
A.11.11 Similitud . . . 301
A.11.12 Comparaciones . . . 301
A.11.13 Operaciones de cuadro delimitador . . . 301
A.11.14 Teselas . . . 302
A.11.15 Cajas delimitadoras . . . 302
A.11.16 Agregaciones . . . 302
A.12. JSON temporal . . . 302
A.12.1. Entrada y salida . . . 302
A.12.2. Constructores . . . 302
A.12.3. Conversiones de tipo . . . 302
A.12.4. Accesores . . . 302
A.12.5. Transformaciones . . . 303
A.12.6. Operaciones JSON temporales . . . 303
A.12.7. Restricciones . . . 303
A.12.8. Operaciones de cuadro delimitador . . . 303
A.12.9. Comparaciones . . . 303
A.12.10 Agregaciones . . . 304

A.13. Índices de celda H3 temporales . . . 304
A.13.1. Conversiones de tipo . . . 304
A.13.2. Cobertura geométrica estática . . . 304
A.13.3. Accesores . . . 304
A.13.4. Inspección del índice . . . 304
A.13.5. Jerarquía . . . 304
A.13.6. Conversiones de latitud/longitud . . . 305
A.13.7. Aristas dirigidas . . . 305
A.13.8. Vértices . . . 305
A.13.9. Recorrido de la malla . . . 305
A.13.10 Métricas . . . 305
A.13.11 Comparaciones ever y always . . . 305
A.13.12 Comparaciones temporales . . . 305
A.13.13 Operaciones de cuadro delimitador . . . 306
A.13.14 Funciones que devuelven conjuntos . . . 306
A.13.15 Relaciones espaciales . . . 306
A.13.16 Índices . . . 306
A.14. Índices de celda Quadbin temporales . . . 306
A.14.1. Conversiones de tipo . . . 306
A.14.2. Celdas estáticas . . . 307
A.14.3. Accesores . . . 307
A.14.4. Inspección del índice . . . 307
A.14.5. Jerarquía . . . 307
A.14.6. Conversiones geométricas . . . 307
A.14.7. Teselas y quadkeys . . . 307
A.14.8. Métricas . . . 307
A.14.9. Comparaciones . . . 307
A.14.10 Índices . . . 308
A.15. Nubes de Puntos Temporales . . . 308
A.15.1. Tipos estáticos `pcpoint` y `pcpatch` . . . 308
A.15.1.1. Constructores . . . 308
A.15.1.2. Accesores . . . 308
A.15.2. Tipos de conjunto `pcpointset` y `pcpatchset` . . . 308
A.15.2.1. Constructores . . . 308
A.15.3. Cuadro delimitador `tpcbox` . . . 308
A.15.3.1. Constructores . . . 308
A.15.3.2. Conversiones de tipo . . . 308
A.15.3.3. Accesores . . . 308
A.15.3.4. Transformaciones . . . 309

A.15.3.5. Operaciones de conjunto . . . 309
A.15.3.6. Operadores topológicos . . . 309
A.15.3.7. Operadores de posición relativa . . . 309
A.15.3.8. Comparaciones . . . 309
A.15.4. Tipo temporal `tpcpoint` . . . 309
A.15.4.1. Entrada y salida . . . 309
A.15.4.2. Constructores . . . 309
A.15.4.3. Accesores . . . 310
A.15.4.4. Conversiones de tipo . . . 310
A.15.4.5. Transformaciones . . . 310
A.15.4.6. Comparaciones . . . 310
A.15.4.7. Restricciones . . . 310
A.15.4.8. Modificaciones . . . 310
A.15.4.9. Fragmentación . . . 310
A.15.4.10. Operaciones de cuadro delimitador . . . 310
A.15.4.11. Relaciones espaciales . . . 311
A.15.5. Tipo temporal `tpcpatch` . . . 311
A.15.5.1. Entrada y salida . . . 311
A.15.5.2. Constructores . . . 311
A.15.5.3. Accesores . . . 311
A.15.5.4. Transformaciones . . . 311
A.15.5.5. Restricciones . . . 311
A.15.5.6. Modificaciones . . . 311
A.15.5.7. Fragmentación . . . 312
A.15.5.8. Relaciones espaciales . . . 312
A.15.5.9. Operaciones de cuadro delimitador . . . 312
A.15.6. Agregaciones . . . 312
A.15.7. Índices . . . 312
A.15.8. Representación binaria . . . 312
A.15.9. Salida MF-JSON . . . 312
A.15.10. Integración con PDAL . . . 312
A.15.11. Puente PCL . . . 313
A.16. Muestreo de Ráster . . . 313
A.16.1. Operador de Muestreo . . . 313
A.16.2. Muestreo Raquet / QUADBIN . . . 313
A.16.3. Accesores y Comparación de Raquet . . . 313
A.16.4. Operadores de Restricción y Predicado . . . 313

B. Generador de datos sintéticos **314**

B.1. Generador para tipos PostgreSQL 314

B.2. Generador para tipos PostGIS 315

B.3. Generador para tipos de rango, de tiempo y de cuadro delimitador MobilityDB 316

B.4. Generador para tipos temporales MobilityDB 316

B.5. Generación de tablas con valores aleatorios 317

B.6. Generador para tipos de red temporales 322

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. 38

5.1. Operación de inserción para valores temporales. 63

5.2. Operaciones de actualización y supresión para valores temporales. 64

5.3. Operaciones de modificación para tablas temporales en SQL. 64

7.1. Jerarquía de tipos espaciotemporales en MobilityDB. 82

7.2. Ilustración del uso de los tipos `tgeompoint` (arriba) y `tgeometry` (abajo) para modelizar la trayectoria y la franja de viento de las tormentas tropicales en 2024. 83

7.3. Visualización de la velocidad de un objeto móvil usando una rampa de color en QGIS. 92

9.1. Diferencia entre la distancia espacial y la distancia sincronizada. 105

9.2. Muestreo de flotantes temporales con interpolación discreta, escalonada y lineal. 106

9.3. Cambio de precisión de números flotantes temporales con interpolación discreta, escalonada y lineal. 108

9.4. Mosaicos multidimensionales para números flotantes temporales. 114

13.1. Ilustración del uso del tipo `tcbuffer` para modelar el búfer circular alrededor de la franja de viento de las tormentas tropicales en 2024. 165

Índice de cuadros

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

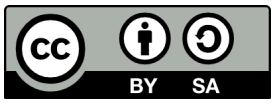
A.1. Instancias actuales de los tipos de plantilla en MobilityDB 280

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 topológicas 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`, `tgeometry`, `tgeography`, `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, donde `tgeometry` y `tgeography` aceptan geometrías/geografías arbitrarias, mientras que `tgeompoint` y `tgeogpoint` solo aceptan 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

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

- Mahmoud Sakr: cofundador del proyecto MobilityDB, MobilityDB workshop, copresidente del OGC Moving Feature Standard Working Group (MF-SWG)
- Vicky Vergara: integración con pgRouting, enlace con OSGeo
- Esteban Zimányi (chair): cofundador 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-1.3.tar.gz https://github.com/MobilityDB/MobilityDB/archive/v1.3.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 v1.3
```

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-16 postgresql-server-dev-16 postgresql-16-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 los tipos `cbuffer` (búfer circular, ver Capítulo 13) y `npoint` (*network point* o punto de red, ver Capítulo 12) proporcionados por MobilityDB. MobilityDB proporciona los siguientes tipos de conjunto y de rango:

- `set`: `intset`, `bigintset`, `floatset`, `textset`, `dateset`, `tstzset`, `geomset`, `geogset`, `cbufferset`, `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)"}';
-- Conjunto de búferes circulares: cada elemento es un cbuffer en WKT
SELECT cbufferset '{"Cbuffer(Point(1 1),0.5)", "Cbuffer(Point(2 2),1.0)"}';
```

Nótese que los elementos de los conjuntos `textset`, `geomset`, `geogset`, `cbufferset` 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.

- Devuelve 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]}
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB) o la representación hexadecimal binaria conocida (HexWKB)

  `asBinary({set,spans},endian text="") → bytea`

  `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 asBinary(dateset '{2001-01-01, 2001-01-03}');
-- \x01050001020000006e01000070010000
SELECT asBinary(intspan '[1, 3)');
-- \x0113000101000000003000000
SELECT asBinary(floatspanset '{[1, 2], [4, 5]}', 'XDR');
-- \x00000e00000002033ff0000000000004000000000000000003401000000000000004014000000000000
SELECT asHexWKB(dateset '{2001-01-01, 2001-01-03}');
-- 01050001020000006E01000070010000
SELECT asHexWKB(intspan '[1, 3)');
-- 0113000101000000003000000
SELECT asHexWKB(floatspanset '{[1, 2], [4, 5]}', 'XDR');
-- 00000E00000002033FF0000000000004000000000000000003401000000000000004014000000000000
```

- Entrar a partir de la representación binaria conocida (WKB) o a partir de la representación hexadecimal binaria conocida (HexWKB)

  `settypeFromBinary(bytea) → set`

  `spantypeFromBinary(bytea) → span`

  `spansettypeFromBinary(bytea) → spanset`

  `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 datesetFromBinary('\x01050001020000006e01000070010000');
-- {2001-01-01, 2001-01-03}
SELECT intspanFromBinary('\x0113000101000000003000000');
-- [1, 3)
SELECT floatspansetFromBinary(
  '\x00000e00000002033ff0000000000004000000000000000003401000000000000004014000000000000');
-- {[1, 2], [4, 5]}
SELECT datesetFromHexWKB('01050001020000006E01000070010000');
-- {2001-01-01, 2001-01-03}
SELECT intspanFromHexWKB('0113000101000000003000000');
-- [1, 3)
SELECT floatspansetFromHexWKB(
  '00000E00000002033FF0000000000004000000000000000003401000000000000004014000000000000');
-- {[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 rangos tiene un solo argumento que es una matriz de rangos 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

- Devuelve 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
```

- Devuelve 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
```

- Devuelve 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
```

- Devuelve el rango 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
```

- Devuelve el número de valores o los valores

  `numValues(set) → integer`

  `getValues(set) → base[]`

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

- Devuelve 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
```

- Devuelve el número de rangos o los rangos

  `numSpans(spanset) → integer`

  `spans(spanset) → span[]`

```
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
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)"}
```

- Devuelve 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]
```

- Devuelve el (number of) different dates

  `numDates(datespanset) → integer`

  `dates(datespanset) → dateset`

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

- Devuelve el start, end, or n-th date

  `startDate(datespanset) → date`

  `endDate(datespanset) → date`

  `dateN(datespanset,integer) → date`

  The functions do not take into account whether the bounds are inclusive or not.

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

- Devuelve el número o las marcas de tiempo diferentes

  `numTimestamps(tstzspanset) → integer`

  `timestamps(tstzspanset) → tstzset`

```
SELECT numTimestamps(tstzspanset '{[2001-01-01, 2001-01-03), (2001-01-03, 2001-01-05)}');
-- 3
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"}
```

- Devuelve la marca de tiempo inicial, final, o enésima

  `startTimestamp(tstzspanset) → timestamptz`

  `endTimestamp(tstzspanset) → timestamptz`

  `timestampN(tstzspanset,integer) → timestamptz`

  The functions do not take into account whether the bounds are inclusive or not.

```
SELECT startTimestamp(tstzspanset '{[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
```

2.5. Transformaciones

- Expandir o reducir los límites de un rango con un valor o un intervalo de tiempo

  `expand(numspan,base) → numspan`

  `expand(tstzspan,interval) → tstzspan`

  La función devuelve NULL si el valor o intervalo dado como segundo argumento es negativo y el rango resultante de desplazar los límites con el argumento está vacío.

```
SELECT expand(floatspan '[1, 3]', 1);
-- [0, 4]
SELECT expand(floatspan '[1, 3]', -1);
-- [2, 2]
SELECT expand(floatspan '[1, 3)', -1);
-- NULL
SELECT expand(tstzspan '[2001-01-01, 2001-01-03]', interval '1 day');
-- [2000-12-31, 2001-01-04]
SELECT expand(tstzspan '[2001-01-01, 2001-01-03]', interval '-1 day');
-- [2001-01-02, 2001-01-02]
SELECT expand(tstzspan '[2001-01-01, 2001-01-03]', interval '-2 day');
-- NULL
```

- Desplazar con un valor o rango 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 con un valor o un rango 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 rango 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 con los valores o rangos 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 al entero inferior o superior

  `floor({floatset,floatspans})` → `{floatset,floatspans}`

  `ceil({floatset,floatspans})` → `{floatset,floatspans}`

```
SELECT floor(floatset '{1.5,2.5}');
-- {1, 2}
SELECT ceil(floatspan '[1.5,2.5)');
-- [2, 3)
SELECT floor(floatspan '(1.5, 1.6)');
-- [1, 1]
SELECT ceil(floatspanset '{[1.5, 2.5],[3.5,4.5]}');
-- {[2, 3], [4, 5]}
```

- Redondear 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 a grados o radianes

  `degrees({floatset,floatspans}, normalize=false)` → `{floatset,floatspans}`

  `radians({floatset,floatspans})` → `{floatset,floatspans}`

  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(floatspanset '{[0, 45], [90, 135]}'), 3);
-- {[0, 0.785], [1.571, 2.356]}
```

- 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 rango 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

- Devuelve o especifica el identificador de referencia espacial

  `SRID(spatialset) → integer`

  `setSRID(spatialset) → spatialset`

```
SELECT SRID(geomset '{"Point(1 1)", "Point(2 2)"}');
-- 0
SELECT SRID(geogset '{"Linestring(1 1,2 2)","Polygon((1 1,1 2,2 2,2 1,1 1))"}');
-- 4326
SELECT SRID(geomset 'SRID=5676;{"Linestring(1 1,2 2)","Polygon((1 1,1 2,2 2,2 1,1 1))"}');
-- 5676
SELECT asEWKT(setSRID(geomset '{Point(1 1), Point(2 2)}', 5676));
-- SRID=5676;{"POINT(1 1)", "POINT(2 2)"}
SELECT asEWKT(setSRID(poseset '{"Pose(Point(1 1),1)", "Pose(Point(2 2),3)"}', 5676));
-- SRID=5676;{"Pose(POINT(2 2),3)", "Pose(POINT(1 1),1)"}
SELECT SRID(cbufferset 'SRID=4326;{"Cbuffer(Point(1 1),0.5)",
  "Cbuffer(Point(2 2),1.0)"}');
-- 4326
SELECT asEWKT(setSRID(cbufferset '{"Cbuffer(Point(1 1),0.5)",
  "Cbuffer(Point(2 2),1.0)"}', 3812));
-- SRID=3812;{"Cbuffer(POINT(1 1),0.5)", "Cbuffer(POINT(2 2),1)"}
```

- Transformar a una referencia espacial diferente

  `transform(spatialset,to_srid integer) → spatialset`

  `transformPipeline(spatialset,pipeline text,to_srid integer,is_forward bool=true) → spatialset`

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 en el siguiente formato:

`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 asEWKT(transform(geomset 'SRID=4326;{Point(2.340088 49.400250),
  Point(6.575317 51.553167)}', 3812), 6);
-- SRID=3812;{"POINT(502773.429981 511805.120402)", "POINT(803028.908265 751590.742629)"}
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, diferencia o intersección de conjuntos o de rangos

  `set {+, - , *} set → set`

  `spans {+, - , *} spans → 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)}
```

```
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]} */
```

```
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
```

2.8.3. Operaciones de división

Al crear índices para tipos de conjuntos o conjunto de rangos, lo que se almacena en el índice no es el valor real, sino un cuadro delimitador que *representa* el valor. En este caso, el índice proporcionará una lista de valores candidatos que *pueden* satisfacer el predicado de la consulta, y se necesita un segundo paso para filtrar los valores candidatos calculando el predicado de la consulta sobre los valores reales.

Sin embargo, cuando los cuadros delimitadores tienen un gran espacio vacío no cubierto por los valores reales, el índice generará muchos valores candidatos que no satisfacen el predicado de la consulta, lo que reduce la eficiencia del índice. En estas situaciones, puede ser mejor representar un valor no con un cuadro delimitador *único*, sino con *múltiples* cuadros delimitadores. Esto aumenta considerablemente la eficiencia del índice, siempre que el índice sea capaz de gestionar múltiples cuadros delimitadores por valor. Las siguientes funciones se utilizan para generar múltiples rangos a partir de un único valor de conjunto o conjunto de rangos.

- Devuelve una matriz de N rangos obtenida fusionando los elementos de un conjunto o los rangos de un conjunto de rangos

  `splitNSpans(set, integer) → span[]`

  `splitNSpans(spanset, integer) → span[]`

  El último argumento especifica el número de rangos de salida. Si el número de elementos o rangos de entrada es menor que el número especificado, la matriz resultante tendrá un rango por elemento o rango de entrada. De lo contrario, el número especificado de rangos de salida se obtendrá fusionando elementos o rangos de entrada consecutivos.

```
SELECT splitNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 1);
/* {"[1, 11)"} */
SELECT splitNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 3);
-- {"[1, 5)","[5, 8)","[8, 11)"}
SELECT splitNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 6);
-- {"[1, 3)","[3, 5)","[5, 7)","[7, 9)","[9, 10)","[10, 11)"}
SELECT splitNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 12);
/* {"[1, 2)","[2, 3)","[3, 4)","[4, 5)","[5, 6)","[6, 7)","[7, 8)","[8, 9)",
    "[9, 10)","[10, 11)"} */
```

```
SELECT splitNSpans(intspanset '{[1, 2), [3, 4), [5, 6), [7, 8), [9, 10)}');
-- {"[1, 2)","[3, 4)","[5, 6)","[7, 8)","[9, 10)"}
SELECT splitNSpans(floatspanset '{[1, 2), [3, 4), [5, 6), [7, 8), [9, 10)}', 3);
-- {"[1, 4)","[5, 8)","[9, 10)"}
SELECT splitNSpans(datespanset '{[2000-01-01, 2000-01-04), [2000-01-05, 2000-01-10)}', 3);
-- {"[2000-01-01, 2000-01-04)","[2000-01-05, 2000-01-10)"}
```

- Devuelve una matriz de rangos obtenida fusionando N elementos consecutivos de un conjunto o N rangos consecutivos de un conjunto de rangos

  `splitEachNSpans(set, integer) → span[]`

  `splitEachNSpans(spanset, integer) → span[]`

  El último argumento especifica el número de elementos de entrada que se fusionan para producir un rango de salida. Si la cantidad de elementos de entrada es menor que el número especificado, la matriz resultante tendrá un rango de salida por elemento. De lo contrario, la cantidad especificada de elementos de entrada consecutivos se fusionará en un único rango de salida en la respuesta. Observe que, a diferencia de la función splitNSpans, el número de rangos en el resultado depende del número de elementos o de rangos de entrada.

```
SELECT splitEachNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 1);
/* {"[1, 2)","[2, 3)","[3, 4)","[4, 5)","[5, 6)","[6, 7)","[7, 8)","[8, 9)",
    "[9, 10)","[10, 11)"} */
SELECT splitEachNSpans(intspanset '{[1, 2), [3, 4), [5, 6), [7, 8), [9, 10)}', 3);
-- {"[1, 6)","[7, 10)"}
SELECT splitEachNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 6);
-- {"[1, 7)","[7, 11)"}
SELECT splitEachNSpans(intset '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}', 12);
-- {"[1, 11)"}
```

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).

- Devuelve 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.

- Comparaciones tradicionales

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

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

```
SELECT intspan '[1,3]' = intspan '[1,4)';
-- true
SELECT floatspanset '{[1, 2),[2,3)}' = floatspanset '{[1,3)}';
```

```
-- true
SELECT tstzset '{2001-01-01, 2001-01-04}' <> tstzset '{2001-01-01, 2001-01-05}';
-- false
SELECT tstzspan '[2001-01-01, 2001-01-04)' <> tstzspan '[2001-01-03, 2001-01-05)';
-- true
SELECT floatspan '[3, 4]' < floatspan '(3, 4]';
-- true
SELECT intspanset '{[1,2],[3,4]}' < intspanset '{[3, 4]}';
-- true
SELECT floatspan '[3, 4]' > floatspan '[3, 4)';
-- true
SELECT tstzspan '[2001-01-03, 2001-01-04)' > tstzspan '[2001-01-02, 2001-01-05)';
-- true
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
SELECT tstzspan '[2001-01-03, 2001-01-05)' >= tstzspan '[2001-01-03, 2001-01-04)';
-- true
SELECT intspanset '{[1, 4)}' >= intspanset '{[1, 5), [6, 7)}';
-- false
```

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 `setUnion` y `spanUnion` 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 10). 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 rangos de tiempo de la granularidad dada (ver Sección 9.5)

- 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).

- Devuelve 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))
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB) o la representación hexadecimal binaria conocida (HexWKB)

  `asBinary(box,endian text=”) → bytea`

  `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 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
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 a partir de representación binaria conocida (WKB) o de la representación hexadecimal binaria conocida (HexWKB)

  `boxFromBinary(bytea) → box`

  `boxFromHexWKB(text) → box`

  En las funciones anteriores, `box` reemplaza cualquier tipo de cuadro delimitador, a saber, `tbox` o `stbox`.

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

- 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,tgeo}::stbox`

  `stbox({box2d,box3d,geo,times,tgeo}) → 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
```

- Devuelve 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
```

- Devuelve 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
```

- Área, volumen, perímetro

  `area(stbox, spheroid bool=true) → float`

  `volume(stbox) → float`

  `perimeter(stbox, spheroid bool=true) → float`

  Para cuadros geodésicos, el cálculo se realiza en el esferoide WGS 84 por defecto. Si el último argumento es falso, se utiliza un cálculo esférico más rápido. La función `volume` no acepta cuadros geodésicos.

```
SELECT area(stbox 'STBOX XT(((1,1),(3,3)),[2001-01-01,2001-01-03))');
-- 4
SELECT volume(stbox 'STBOX ZT(((1,1,1),(3,3,3)),[2001-01-01,2001-01-03))');
-- 8
SELECT perimeter(stbox 'STBOX XT(((1,1),(3,3)),[2001-01-01,2001-01-03))');
-- 8
SELECT area(stbox 'GEODSTBOX XT(((1,1),(3,3)),[2001-01-01,2001-01-03))');
-- 49209676328.36632
SELECT perimeter(stbox 'GEODSTBOX XT(((1,1),(3,3)),[2001-01-01,2001-01-03))', false);
-- 889221.9544681838
```

3.5. Transformaciones

- Desplazar y/o escalear el rango de valores de un cuadro delimitador con uno o dos valores

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

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

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

```
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 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 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])
```

- Desplazar y/o escalar el período de un cuadro delimitador con uno o dos intervalos. 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.

  `shiftTime(box,interval) → box`

  `scaleTime(box,interval) → box`

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

```
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])
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
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])
```

- Devuelve 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, espacial o temporal del cuadro delimitador con un valor o un intervalo

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

  `expandSpace(stbox,float) → stbox`

  `expandTime(box,interval) → box`

  La función devuelve NULL si el valor o intervalo dado como segundo argumento es negativo y el rango resultante de desplazar los límites con el argumento está vacío.

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

```
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
```

```
SELECT expandTime(tbox 'TBOXFLOAT XT((1,2),[2001-01-01,2001-01-03])', interval '1 day');
-- TBOXFLOAT XT((1,2),[2000-12-31,2001-01-04])
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)),[2001-01-02,2001-01-02])
SELECT expandTime(tbox 'TBOX XT((1,2),[2001-01-01,2001-01-03])', interval '-2 days');
-- NULL
```

- 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])
SELECT round(tstzspan '[2000-01-01, 2001-01-02]'::tbox);
-- The tbox must have X dimension
```

3.6. Sistema de referencia espacial

- Devuelve o especifica el identificador de referencia espacial

  `SRID(stbox) → integer`

  `setSRID(stbox) → stbox`

```
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
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`

  `transformPipeline(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:

  `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. Operaciones 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.

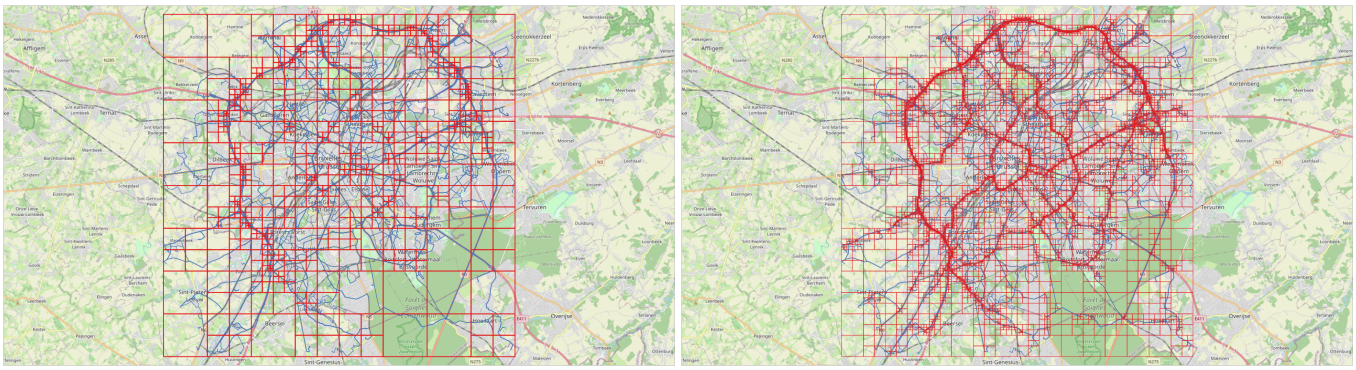


Figura 3.1: Grilla multiresolución sobre datos de Bruselas obtenidos mediante el generador BerlinMOD. Cada celda contiene como máximo 10.000 (izquierda) y 1.000 (derecha) instantes durante todo el período de simulación (cuatro días en este caso). A la izquierda, podemos ver la alta densidad de tráfico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

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 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 e intersección de dos 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
```

```
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?

  `tbox {<<, <<#} tbox → boolean`

  `stbox {<<, <<|, <</, <<#} stbox → 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?

  `tbox {>>, #>>} tbox → boolean`

  `stbox {>>, |>>, />>, #>>} stbox → 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?

  `tbox {&<, &<#} {tbox,stbox} → boolean`

  `stbox &<, &<|, &</, &<#} stbox → 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?

  `tbox {&>, #&>} tbox → boolean`

  `stbox {&>, |&>, /&>, #&>} stbox} → 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.

- Comparaciones tradicionales

  `box = box → boolean`

  `box <> box → boolean`

  `box < box → boolean`

  `box > box → boolean`

  `box <= box → boolean`

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

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

4.1. Introduction

MobilityDB proporciona varios tipos temporales basados en tipos PostgreSQL o PostGIS, a saber, `tbool`, `tint`, `tfloat`, `ttext`, `tgeometry`, `tgeography`, `tgeompoint` y `tgeogpoint`, que se basan, respectivamente, en los tipos de base `bool`, `integer`, `float`, `text`, `geometry` y `geography`, donde `tgeometry` y `tgeography` aceptan geometrías/geografías arbitrarias, mientras que `tgeompoint` y `tgeogpoint` solo aceptan puntos 2D o 3D con dimensión Z. Además, MobilityDB proporciona otros tipos espaciales específicos, a saber, `cbuffer` (buffer circular), `npoint` (punto de red) y `pose`, y sus tipos espaciotemporales correspondientes, a saber, `tcbuffer`, `tnpoint`, `tpose` y `trgeometry` (geometría rígida temporal). En este capítulo y en el siguiente, presentamos las funciones y operadores disponibles para todos los tipos temporales, mientras que las funciones específicas para los tipos espaciotemporales se detallarán en capítulos posteriores.

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. Nótese que los tipos `tgeometry` y `tgeography` solo admiten interpolación discreta o por escalonada, ya que no es posible interpolar linealmente dos geometrías/geografías arbitrarias.

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, atValue(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(atValue(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)}';
```

4.4. Temporalización de operaciones

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.

4.5. Notación

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,
- `ttypeInst`, `ttypSeq` y `ttypeSeqSet` representan cualquier tipo temporal con subtipo instante, secuencia y conjunto de secuencias
- `tdisc` representa cualquier tipo temporal con tipo de base discreto, es decir, `tbool`, `tint`, or `ttext`,
- `tcont` cualquier tipo temporal con tipo de base contínuo, es decir, `tfloat`, `tgeompoint`, or `tgeogpoint`,
- `ttypeDiscSeq` y `ttypeContSeq` representan cualquier tipo temporal con subtipo secuencia y, respectivamente, interpolación discreta y contínua,
- `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`
- `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`.

4.6. 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';
SELECT tgeometry 'Linestring(0 0,1 1)@2017-01-01 08:00:05';
SELECT tgeography 'Polygon((0 0,1 1,2 0,0 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}';
SELECT tgeometry '{Point(0 0)@2017-01-01 08:00:00,
  Linestring(0 0,0 1)@2017-01-02 08:05:00}';
SELECT tgeography '{Point(0 0)@2017-01-01 08:00:00,
  Polygon((0 0,1 1,2 0,0 0))@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)';
SELECT tgeometry '[Point(0 0)@2017-01-01 08:00:00,
  Linestring(0 0,0 1)@2017-01-02 08:05:00]';
SELECT tgeography '[Point(0 0)@2017-01-01 08:00:00,
  Polygon((0 0,1 1,2 0,0 0))@2017-01-02 08:05:00]';
```

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)}';
SELECT tgeometry
  '{[Point(0 0)@2017-01-01 08:00:00, Linestring(0 0,1 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 tgeography
  '{[Point(0 0)@2017-01-01 08:00:00, Polygon((0 0,1 1,2 0,0 0))@2017-01-01 08:05:00),
    [Point(0 1)@2017-01-01 08:10:00, Linestring(0 0,1 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 geometrías 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]'
SELECT tgeography 'SRID=7844;[Point(0 0)@2001-01-01,Linestring(1 0,0 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 tgeography 'SRID=7844;[SRID=4326;Point(0 0)@2001-01-01,
  SRID=4326;Linestring(1 0,0 1)@2001-01-02]';
-- ERROR: Geometry SRID (4326) does not match temporal type SRID (7844)
```

Si no se especifica, el SRID predeterminado para los geometrías temporales es 0 (desconocido) y para los geografías temporales es 4326 (WGS 84). Los puntos temporales con interpolación escalonada también pueden especificar el SRID, como se muestra a continuación.

```
SELECT tgeompoint 'SRID=5435,Interp=Step;[Point(0 0)@2001-01-01,
  Point(0 1)@2001-01-02]';
SELECT tgeogpoint 'Interp=Step;[SRID=4326;Point(0 0)@2001-01-01,
  SRID=4326;Point(0 1)@2001-01-02]';
```

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 7.3. 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.

- Devuelve 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]
  SELECT asText(tgeometry
    '[Point(1.55 1.55)@2001-01-01,Linestring(1.55 1.55,3.55 3.55)@2001-01-02]', 0);
  -- [Point(1 1)@2001-01-01, Linestring(1 1,3 3)@2001-01-02]
  ```

- Devuelve 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"} */
SELECT asMFJSON(tgeometry '{[Point(1 1)@2001-01-01 18:00:00+02,
  Linestring(1 1,2 2)@2001-01-01 18:10:00+02]}');
/* {"type":"MovingGeometry","sequences":[{"values":[{"type":"Point",
    "coordinates":[1,1]},{"type":"LineString","coordinates":[[1,1],[2,2]]}],
    "datetimes":["2001-01-01T17:00:00+01","2001-01-01T17:10:00+01"],
    "lower_inc":true,"upper_inc":true}],"interpolation":"Step"} */
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB) o la representación hexadecimal binaria conocida (HexWKB)

  `asBinary(ttype,endian text='') → bytea`

  `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 asBinary(tbool 'true@2001-01-01');
-- \x011a000101009c57d3c11c0000
SELECT asBinary(tfloat '1.5@2001-01-01');
-- \x0121000100000000000f83f009c57d3c11c0000
SELECT asHexWKB(tint '1@2001-01-01', 'XDR');
-- 00002301000000100001CC1D3579C00
SELECT asHexWKB(ttext 'AAA@2001-01-01');
-- 0129000104000000000000041414100009C57D3C11C0000
```

- Entrar a partir de la 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]}');
SELECT asText(tgeometryFromMFJSON(text
  '{"type":"MovingGeometry","sequences":[{"values":[{"type":"Point",
  "coordinates":[1,1]},{"type":"LineString","coordinates":[[1,1],[2,2]]}],
  "datetimes":["2001-01-01T17:00:00+01","2001-01-01T17:10:00+01"],
  "lower_inc":true,"upper_inc":true}],"interpolation":"Step"}'));
-- {[POINT(1 1)@2001-01-01 17:00:00, LINESTRING(1 1,2 2)@2001-01-01 17:10:00]}
```

- Entrar a partir de la representación binaria conocida (WKB) o de la representación hexadecimal binaria conocida (HexWKB)

  `ttypeFromBinary(bytea)` → `ttype`

  `ttypeFromHexWKB(text)` → `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('\x011a000101009c57d3c11c0000');
-- t@2001-01-01
SELECT tintFromBinary('\x00002301000000100001cc1d3579c00');
-- 1@2001-01-01
SELECT tfloatFromBinary('\x0121000100000000000f83f009c57d3c11c0000');
-- 1.5@2001-01-01
SELECT ttextFromBinary('\x0129000104000000000000000041414100009c57d3c11c0000');
-- "AAA"@2001-01-01
SELECT tboolFromHexWKB('011A000101009C57D3C11C0000');
-- t@2001-01-01
SELECT tintFromHexWKB('00002301000000100001CC1D3579C00');
-- 1@2001-01-01
SELECT tfloatFromHexWKB('0121000100000000000F83F009C57D3C11C0000');
-- 1.5@2001-01-01
SELECT ttextFromHexWKB('0129000104000000000000000041414100009C57D3C11C0000');
-- "AAA"@2001-01-01
```

4.7. 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 tgeography('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 tgeographySeq(ARRAY[tgeography '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 tgeographySeqSet(ARRAY[tgeography
  '[Point(0 0)@2001-01-01 08:00:00, Point(0 0)@2001-01-01 08:05:00)',
  '[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@2001-01-01, 3@2001-01-02, 4@2001-01-03, 5@2001-01-05]}
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(tgeometrySeqSetGaps(ARRAY[tgeometry 'Point(1 1)@2001-01-01',
  'Linestring(2 2,3 3)@2001-01-02', 'Point(4 2)@2001-01-03',
  'Polygon((5 0,6 1,7 0,5 0))@2001-01-05'], '1 day', 1));
/* {[POINT(1 1)@2001-01-01], [LINESTRING(2 2,3 3)@2001-01-02],
   [POINT(4 2)@2001-01-03], [POLYGON((5 0,6 1,7 0,5 0))@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]} */
```

4.8. 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 intervalo de marcas de tiempo

  `ttype::tstzspan`

```
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]
```

4.9. Accesores

- Devuelve 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
```

- Devuelve el subtipo temporal

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

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

- Devuelve el tipo base

  `tempBasetype(ttype) → text`

```
SELECT tempBasetype(tint '[1@2001-01-01, 2@2001-01-02, 3@2001-01-03]');
-- int4
SELECT tempBasetype(tgeompoint 'Point(1 1)@2001-01-01');
-- geometry
```

- Devuelve la interpolación

  `interp(ttype) → {'None','Discrete','Step','Linear'}`

```
SELECT interp(tbool 'true@2001-01-01');
-- None
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
SELECT interp(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02, Polygon((3 3,4 4,5 3,3 3))@2001-01-03]');
-- Step
```

- Devuelve 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
```

- Devuelve 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]}
```

- Devuelve el período delimitador en el que se define el valor temporal

  `timeSpan(ttype) → tstzspan`

```
SELECT timeSpan(tfloat '{1@2001-01-01, 2@2001-01-02, 1@2001-01-03}');
-- [2001-01-01, 2001-01-03]
SELECT timeSpan(tint '[1@2001-01-01, 1@2001-01-15)');
-- [2001-01-01, 2001-01-15)
```

- Devuelve el valor inicial, final, o enésimo

  `startValue(ttype) → base`

  `endValue(ttype) → base`

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

- Devuelve 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
```

- Devuelve 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
```

- Devuelve el número o los instantes diferentes

  `numInstants(ttype) → integer`

  `instants(ttype) → ttypeInst[]`

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

- Devuelve el instante inicial, final o enésimo

  `startInstant(ttype) → ttypeInst`

  `endInstant(ttype) → ttypeInst`

  `instantN(ttype,integer) → ttypeInst`

  Las funciones no tienen 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
```

- Devuelve el número o las marcas de tiempo diferentes

  `numTimestamps(ttype) → integer`

  `timestamps(ttype) → timestamptz[]`

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

- Devuelve la marca de tiempo inicial, final o enésima

  `startTimestamp(ttype) → timestamptz`

  `endTimestamp(ttype) → timestamptz`

  `timestampN(ttype,integer) → timestamptz`

  Las funciones no tienen 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
```

- Devuelve el número o las secuencias

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

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

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

- Devuelve 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)
```

- Devuelve 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]"} */
```

4.10. 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]} */
SELECT setInterp(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02]', 'linear');
-- ERROR:  The temporal type cannot have linear interpolation
```

- Desplazar y/o escalear el intervalo de tiempo de un valor temporal a uno o dos intervalos

  `shiftTime(ttype,interval) → ttype`

  `scaleTime(ttype,interval) → ttype`

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

  Cuando se escalea, 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. Igualmente, el ancho o intervalo dado debe ser estrictamente mayor que cero.

```
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]} */
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
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(tgeometry '{[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02, Point(1 1)@2001-01-03], [Point(2 2)@2001-01-04,
  Polygon((1 1,2 2,3 1,1 1))@2001-01-05]}', '1 day', '1 day'));
/* {[POINT(1 1)@2001-01-02 00:00:00, LINESTRING(1 1,2 2)@2001-01-02 06:00:00,
    POINT(1 1)@2001-01-02 12:00:00], [POINT(2 2)@2001-01-02 18:00:00,
    POLYGON((1 1,2 2,3 1,1 1))@2001-01-03 00:00:00]} */
```

- Devuelve un valor booleano temporal que indica si cada segmento de una secuencia (o conjunto de sequencias) temporal(es) continua tiene, respectivamente, al menos o como máximo una duración determinada.

  `segmentMinDuration(temp,interval,bool strict=true) → tbool`

  `segmentMaxDuration(temp,interval,bool strict=true) → tbool`

  Si el argumento `strict` no se especifica, se asume el valor `true` por defecto y, por lo tanto, la función verifica si la duración del segmento es estrictamente menor o mayor que el intervalo. Si el argumento es `false`, la función verifica si la duración del segmento es menor/mayor o *igual* que el intervalo.

```
SELECT segmentMinDuration(tfloat '[1@2001-01-01, 0@2001-01-03, -1@2001-01-04, -1@2001 ↩
    -01-06]', '1 day');
-- {[t@2001-01-01, f@2001-01-03, t@2001-01-04, t@2001-01-06)}
SELECT segmentMinDuration(tfloat '[1@2001-01-01, 0@2001-01-03, -1@2001-01-04, -1@2001 ↩
    -01-06]', '1 day', false);
-- {[t@2001-01-01, t@2001-01-06)}
SELECT segmentMaxDuration(tfloat '[1@2001-01-01, 0@2001-01-03, -1@2001-01-04, -1@2001 ↩
    -01-06]', '1 day');
-- {[f@2001-01-01 00:00:00+01, f@2001-01-06 00:00:00+01)}
SELECT segmentMaxDuration(tfloat '[1@2001-01-01, 0@2001-01-03, -1@2001-01-04, -1@2001 ↩
    -01-06]', '1 day', false);
-- {[f@2001-01-01, t@2001-01-03, f@2001-01-04, f@2001-01-06)}
```

- 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)}
SELECT ST_AsText((un).value), (un).time
FROM (SELECT unnest(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 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]}
-- LINESTRING(1 1,2 2) | {[2001-01-02, 2001-01-03)}
```

Capítulo 5

Tipos temporales (Parte 2)

5.1. 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). Please refer to this article for a detailed discussion about tuple versus attribute timestamping in temporal databases.

La operación `insert` agrega a un valor temporal los instantes de otro sin modificar los instantes existentes, como se ilustra en la Figura 5.1.

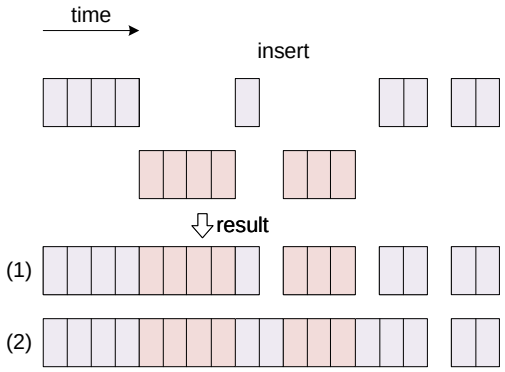


Figura 5.1: Operación de inserción para valores temporales.

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 5.2.

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

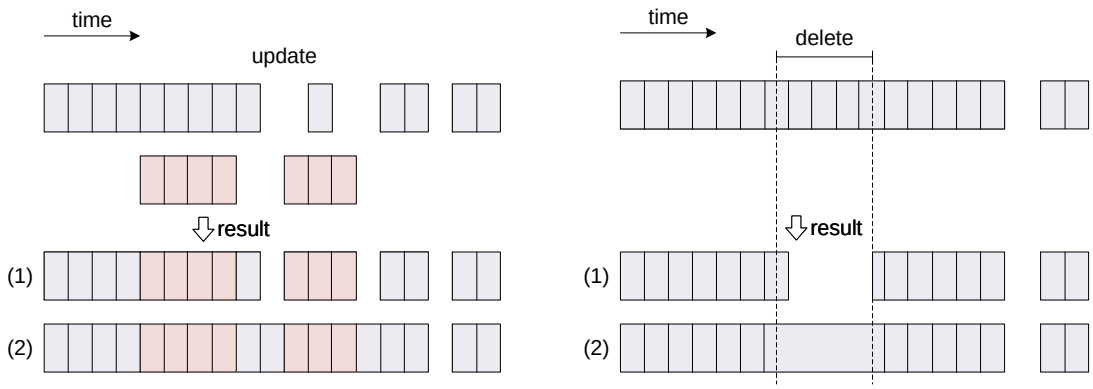


Figura 5.2: Operaciones de actualización y supresión para valores temporales.

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 5.2.

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 5.2), 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 5.3 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 5.1 y Figura 5.2). 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`.

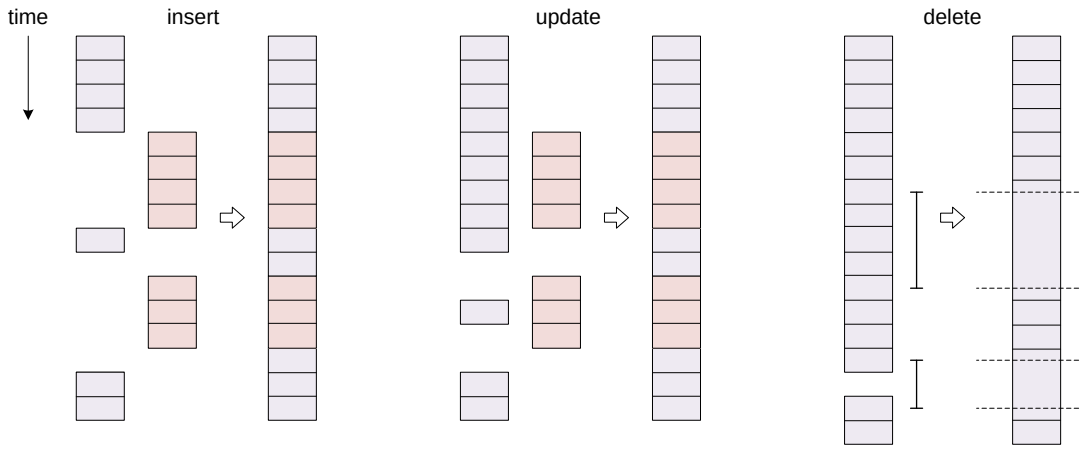


Figura 5.3: Operaciones de modificación para tablas temporales en SQL.

- 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]}',
  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]} */
SELECT asText(update(
  tgeometry '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03, Point(1 1)@2001-01-05]',
  tgeometry '[Linestring(2 2,3 3)@2001-01-02, Linestring(3 3,2 2)@2001-01-04]'));
/* [POINT(1 1)@2001-01-01, LINESTRING(2 2,3 3)@2001-01-02,
    LINESTRING(3 3,2 2)@2001-01-04], (POINT(3 3)@2001-01-04, POINT(1 1)@2001-01-05] */
```

- 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,interp=NULL) → ttype`

`appendInstant(ttypeInst,interp=NULL,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', 'discrete');
-- {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]} */
SELECT asText(appendInstant(tgeometry '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02],
  [Linestring(2 2,3 3)@2001-01-04, Point(3 3)@2001-01-05]}',
  tgeometry 'Linestring(1 1,2 2)@2001-01-06'));
/* {[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02], [LINESTRING(2 2,3 3)@2001-01-04,
     POINT(3 3)@2001-01-05, LINESTRING(1 1,2 2)@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 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, 'discrete' ORDER BY inst) FROM temp;
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03, 4@2001-01-04, 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 interpolación discreta o la tercera consulta con `tgeogpoint`, donde 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, NULL, 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, NULL, 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]} */
SELECT asText(appendSequence(tgeometry '{[Point(1 1)@2001-01-01, Point(2 2)@2001-01-02],
  [Linestring(3 3,2 2)@2001-01-04, Point(3 3)@2001-01-05]}',
  tgeometry '[Point(3 3)@2001-01-05, Linestring(3 3,1 1)@2001-01-06]'));
/* {[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02], [LINESTRING(3 3,2 2)@2001-01-04,
     POINT(3 3)@2001-01-05, LINESTRING(3 3,1 1)@2001-01-06]} */
```

- Fusionar los valores temporales

  `merge(ttype,ttype) → ttype`

  `merge(ttype[]) → ttype`

  `merge(ttype) → ttype`

  Los valores temporales solo pueden intersectar en su límite. En ese caso, para todos los tipos temporales excepto `tgeometry` y `tgeography`, sus valores de base en las marcas de tiempo comunes deben ser los mismos, de lo contrario se genera un error. Para los tipos `tgeometry` y `tgeography`, si el valor en sus marcas de tiempo comunes es diferente, el valor resultante después de la `merge` será la unión espacial de los valores. La razón de un comportamiento diferente para estos tipos es que son los únicos tipos temporales que no son tipos *escalares*, es decir, su valor en una marca de tiempo dada no es un único elemento del dominio del tipo base, sino más bien un subconjunto de su dominio.

  La tercera 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 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 asText(merge(tgeometry 'Linestring(1 1,5 1)@2001-01-01',
  tgeometry '{Linestring(5 1,10 1)@2001-01-01, Point(3 3)@2001-01-02}'));
-- {LINESTRING(1 1,5 1,10 1)@2001-01-01, POINT(3 3)@2001-01-02}
```

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

```
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 merge(inst) FROM temp;
-- {1@2001-01-01, 2@2001-01-02, 3@2001-01-03, 4@2001-01-04, 5@2001-01-05}
```

5.2. 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 o un conjunto de valores o, para un número temporal, un span o un conjunto de spans de valores

  `atValue(ttype,value) → ttype`

  `minusValue(ttype,value) → ttype`

  `atValues(ttype,valueset) → ttype`

  `minusValues(ttype,valueset) → ttype`

`atSpan(tnumber,span) → tnumber`

`minusSpan(tnumber,span) → tnumber`

`atSpanset(tnumber,spanset) → tnumber`

`minusSpanset(tnumber,spanset) → tnumber`

```
SELECT atValue(tint '[1@2001-01-01, 1@2001-01-15)', 1);
-- [1@2001-01-01, 1@2001-01-15)
SELECT atValues(tfloat '[1@2001-01-01, 4@2001-01-4)', floatset '{1, 3, 5}');
-- {[1@2001-01-01], [3@2001-01-03]}
SELECT atSpan(tfloat '[1@2001-01-01, 4@2001-01-4)', floatspan '[1,3]');
-- [1@2001-01-01, 3@2001-01-03]
SELECT atSpanset(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 asText(atValue(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 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 asText(atValue(tgeometry '[Point(0 0)@2001-01-01, Linestring(0 0,2 2)@2001-01-02,
  Linestring(0 0,2 2)@2001-01-03]', geometry 'Linestring(0 0,2 2)'));
-- {[LINESTRING(0 0,2 2)@2001-01-02, LINESTRING(0 0,2 2)@2001-01-03]}
```

```
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 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 minusSpan(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 minusSpanset(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)}
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 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 asText(minusValue(tgeometry '[Point(0 0)@2001-01-01, Linestring(0 0,2 2)@2001 ↩
   -01-02,
  Linestring(0 0,2 2)@2001-01-03]', geometry 'Linestring(0 0,2 2)'));
--  {[POINT(0 0)@2001-01-01, POINT(0 0)@2001-01-02)}
```

- 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)} */
```

5.3. Operadores de cuadro delimitador

Estos operadores determinan 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.

5.4. Comparaciones

5.4.1. 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
```

- Comparaciones tradicionales

  `ttype {=, <>, <, >, <=, >=} ttype → boolean`

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

5.4.2. 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 `%<=`.

- Comparaciones alguna vez y siempre

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

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

  Los operadores no tienen en cuenta si los límites son inclusivos o no.

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

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

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

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

```
SELECT tint '[1@2001-01-01, 1@2001-01-05]' ?<= 2;
-- true
SELECT tfloat '[1@2001-01-01, 1@2001-01-05)' %<= 4;
-- true
```

```
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
```

5.4.3. Comparaciones temporales

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`.

- Comparaciones temporales

  `{base,ttype} {#=, #<>, #<, #>, #<=, #>=} {base,ttype} → boolean`

```
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 '[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 tgeometry '[Point(0 0)@2001-01-01, Linestring(1 1,2 2)@2001-01-03]' #=
  geometry 'Linestring(2 2,1 1)';
-- {[f@2001-01-01, t@2001-01-03]}
```

```
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)}
SELECT tgeometry '[Point(0 0)@2001-01-01, Linestring(1 1,2 2)@2001-01-03]' #<>
  geometry 'Linestring(2 2,1 1)';
-- {[t@2001-01-01, f@2001-01-03]}
```

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

5.5. Funciones de utilidad

- Versión de la extensión MobilityDB

  `mobilitydb_version() → text`

```
SELECT mobilitydb_version();
-- MobilityDB 1.3.0
```

- Versión de la extensión MobilityDB y de sus dependencias

  `mobilitydb_full_version() → text`

```
/* MobilityDB 1.3.0, PostgreSQL 17.2, PostGIS 3.5.2, GEOS 3.12.0-CAPI-1.18.0, PROJ 9.2.0,
   JSON-C 0.13.1, GSL 2.5 */
```

Capítulo 6

Tipos alfanuméricos temporales

6.1. Notación

En Sección 4.5 presentamos la notación utilizada para definir la firma de las funciones y operadores para tipos temporales. A continuación, ampliamos estas notaciones para tipos alfanuméricos temporales.

- `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`,
- `number` represents any number base type, that is, `integer` or `float`,
- `numspan` represents any number span type, that is, either `intspan` or `floatspan`,

6.2. Conversión de tipos

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

- Convertir un número temporal en un rango o un cuadro delimitador temporal

  `tnumber::{span,tbox}`

  `valueSpan(tnumber)` → `numspan`

  `timeSpan(tnumber)` → `tstzspan`

  `tbox(tnumber)` → `tbox`

```
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 valueSpan(tfloat '(1@2001-01-01, 3@2001-01-03, 2@2001-01-05]');
-- (1, 3]
SELECT timeSpan(tfloat 'Interp=Step;(1@2001-01-01, 3@2001-01-03, 2@2001-01-05]');
-- (2001-01-01, 2001-01-05]
```

```
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])
```

- Convertir entre un booleano temporal y un entero temporal

  `tbool::tint`

  `tint(tbool) → tint`

```
SELECT tbool '[true@2001-01-01, false@2001-01-03]'::tint;
-- [1@2001-01-01, 0@2001-01-03]
```

- Convertir entre un entero integer y un número flotante temporal

  `tint::tfloat`

  `tfloat::tint`

  `tfloat(tint)`

  `tint(tfloat)`

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

6.3. Accessores

- Devuelve el intervalo de valores ignorando las brechas potenciales

  `valueSpan(tnumber) → numspan`

```
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])
```

- Devuelve el conjunto de valores de un número temporal

  `valueSet(tnumber) → numset`

```
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}
```

- Devuelve el valor mínimo, máximo, o promedio

  `minValue(torder) → base`

  `maxValue(torder) → base`

  `avgValue(tnumber) → base`

  Las funciones no tienen 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
SELECT avgValue(tfloat '{[1@2001-01-01, 2@2001-01-03), [3@2001-01-03, 5@2001-01-05)}');
-- 2.75
```

```
SELECT minValue(ttext '{[A@2001-01-01, B@2001-01-02], [C@2001-01-03, E@2001-01-05]}');
-- A
SELECT maxValue(ttext '{[A@2001-01-01, B@2001-01-02], [C@2001-01-03, E@2001-01-05]}');
-- E
```

- Devuelve el instante con el valor mínimo o máximo

  `minInstant(torder) → base`

  `maxInstant(torder) → base`

  Las funciones no tienen 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
```

6.4. Transformaciones

- Desplazar y/o escalear el rango de valores de un número temporal con uno o dos números

  `shiftValue(tnumber,base) → tnumber`

  `scaleValue(tnumber,width) → tnumber`

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

  Cuando se escalea, si el rango de valores del valor temporal es un valor único (por ejemplo, para un instante temporal), el resultado es el valor temporal. Además, el ancho data debe ser estrictamente superio que cero.

```
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 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 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]}
```

- Extraer de un flotante temporal con interpolación lineal las subsecuencias donde los valores permanecen en un rango de un ancho dado durante al menos una duración dada

  `stops(tfloat,maxDist=0.0,minDuration='0 minutes') → tfloatSeqSet`

  Si `maxDist` no se especifica, el valor 0.0 se asume por defecto y por lo tanto, la functión extrae los segmentos constantes del flotante temporal.

```
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
```

6.5. Restrictions

- Restringir al (complemento del) valor mínimo o máximo

  `atMin(torder)` → `torder`

  `atMax(torder)` → `torder`

  `minusMin(torder)` → `torder`

  `minusMax(torder)` → `torder`

  La función devuelve un valor nulo si el valor mínimo o máximo 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 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 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
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) un `tbox`

  `atTbox(tnumber,tbox)` → `tnumber`

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

  When the bounding box has both value and time dimensions, the functions restrict the temporal number with respect to the value *and* the time extents of the box.

```
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 minusSpan(minusTime(temp, box::tstzspan), box::floatspan) FROM temp;
-- {[1@2001-01-01], [2@2001-01-02, 3@2001-01-03)}
```

6.6. Operaciones booleanas

- Y temporal, 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)';
-- [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)}
```

```
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)
```

- Devuelve el tiempo cuando un 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]}
```

6.7. Operaciones matemáticas

- Adición, resta, multiplicación y división temporal

  `{number,tnumber} {+, -, *, /} {number,tnumber} → tnumber`

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

```
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)}
```

```
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)
```

```
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)}
```

```
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]
```

- Devuelve 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]
```

- Redondear al entero inferior o superior

  `floor(tfloat) → tfloat`

  `ceil(tfloat) → tfloat`

```
SELECT floor(tfloat '[0.5@2001-01-01, 1.5@2001-01-02]');
-- [0@2001-01-01, 1@2001-01-02]
SELECT ceil(tfloat '[0.5@2001-01-01, 0.6@2001-01-02, 0.7@2001-01-03]');
-- [1@2001-01-01, 1@2001-01-03]
```

- Redondear 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 '[0.7853981633974@2001-01-01, 2.35619449019923@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]
```

- Devuelve 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)} */
```

- Devuelve la tendencia de un flotante temporal con interpolación lineal, que indica si su valor es creciente, constante o decreciente, representado, respectivamente, por 1, 0 y -1.

  `trend(tfloat) → tint`

  Nótese que la tendencia es NULL para secuencias instantáneas.

```
SELECT trend(tfloat '[1@2001-01-01, 2@2001-01-02, 4@2001-01-03, 4@2001-01-04, 3@2001 ↩
    -01-05, 2@2001-01-06]');
-- [1@2001-01-01, 0@2001-01-03, -1@2001-01-04, -1@2001-01-06]
SELECT trend(tfloat '[1@2001-01-01]');
-- NULL
```

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

  `derivative(tfloat) → tfloat`

  El número flotante temporal debe tener interpolación linear. Nótese que esta función corresponde a la función speed para los puntos temporales.

```
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
```

- Devuelve 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
```

- Devuelve 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
```

- Devuelve el logaritmo natural y el logaritmo base 10 de un número flotante temporal

  `ln(tfloat) → tfloat`

  `log10(tfloat) → tfloat`

  El número flotante temporal no puede ser cero o negativo

```
SELECT ln(tfloat '{[1@2001-01-01, 10@2001-01-02, 5@2001-01-03],
  [1@2001-01-04, 1@2001-01-05]}');
/* {[0@2001-01-01, 2.302585092994046@2001-01-02, 1.6094379124341@2001-01-03],
   [0@2001-01-04, 0@2001-01-05]} */
SELECT log10(tfloat 'Interp=Step;[-10@2001-01-01, 10@2001-01-02]');
-- ERROR:  Cannot take logarithm of zero or a negative number
```

- Devuelve el exponencial (e elevado a la potencia dada) de un número flotante temporal

  `exp(tfloat) → tfloat`

```
SELECT exp(tfloat '{[1@2001-01-01, 10@2001-01-02],
  [1@2001-01-04, 1@2001-01-05]}');
/* {[2.718281828459045@2001-01-01, 22026.465794806718@2001-01-02],
   [2.718281828459045@2001-01-04, 2.718281828459045@2001-01-05]} */
SELECT exp(tfloat '{-10@2001-01-01, 0@2001-01-02, 10@2001-01-03}');
-- {0.000045399929762@2001-01-01, 1@2001-01-02, 22026.465794806718@2001-01-03}
```

- Devuelve el seno, coseno o tangente de un número flotante temporal (argumento en radianes)

  `sin(tfloat) → tfloat`

  `cos(tfloat) → tfloat`

  `tan(tfloat) → tfloat`

```
SELECT round(sin(tfloat '[0@2001-01-01, 6.283185307@2001-01-02]'), 6);
-- [0@2001-01-01, -0.000000@2001-01-02]
SELECT round(cos(tfloat '[0@2001-01-01, 3.141592654@2001-01-02]'), 6);
-- [1@2001-01-01, -1@2001-01-02]
SELECT round(tan(tfloat '{0@2001-01-01, 0.785398163@2001-01-02}'), 6);
-- {0@2001-01-01, 1@2001-01-02}
```

6.8. 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 7

Tipos temporales geométricos (Parte 1)

7.1. Tipos espaciotemporales

MobilityDB proporciona un conjunto de tipos espaciotemporales, a saber, `tgeometry` (geometría temporal), `tgeography` (geografía temporal), `tgeompoint` (punto geométrico temporal), `tgeogpoint` (punto geográfico temporal), `tcbuffer` (buffer circular temporal), `tnpoint` (punto de red temporal), `tpose` (pose temporal) y `trgeometry` (geometría rígida temporal). A nivel conceptual, estos tipos espaciotemporales se organizan en la jerarquía que se muestra en la Figura 7.1.

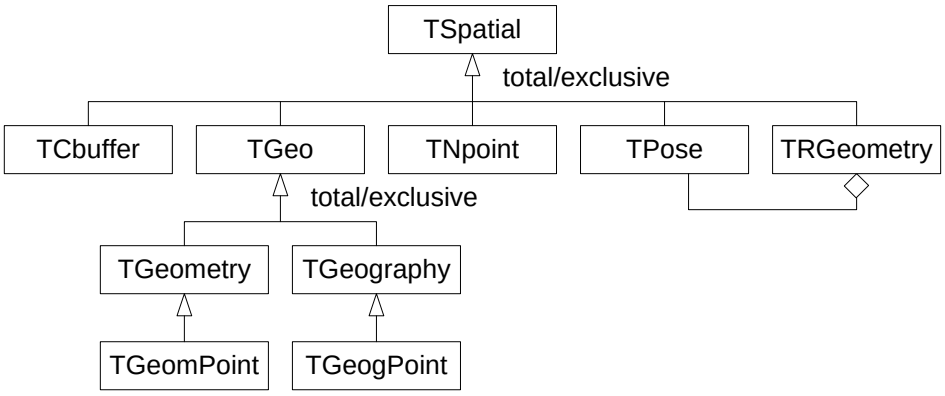


Figura 7.1: Jerarquía de tipos espaciotemporales en MobilityDB.

En este capítulo y el siguiente cubrimos el tipo `TGeo` y sus subtipos, es decir, los tipos espaciotemporales derivados de los tipos `geometry` y `geography` de PostGIS. En los capítulos siguientes, continuamos describiendo los tipos espaciotemporales restantes.

La Figura 7.2 ilustra el uso de los tipos `tgeompoint` (arriba) y `tgeometry` (abajo) para modelizar la trayectoria y la franja de viento de las tormentas tropicales en 2024. Los datos provienen de National Oceanic and Atmospherique Administration (NOAA). Como se ilustra en la figura, el tipo `tgeompoint` permite la interpolación *lineal*, mientras que el tipo `tgeometry` solo permite la interpolación *escalonada*.

7.2. Notación

Presentamos en la Sección 4.5 y en la Sección 6.1 la notación utilizada para definir la firma de las funciones y operadores para tipos temporales. A continuación, ampliamos estas notaciones para tipos espaciotemporales.

- `tspatial` representa un tipo espaciotemporal, por ejemplo, `tgeometry`, `tgeompoint`, `tpose`, o `tnpoint`,

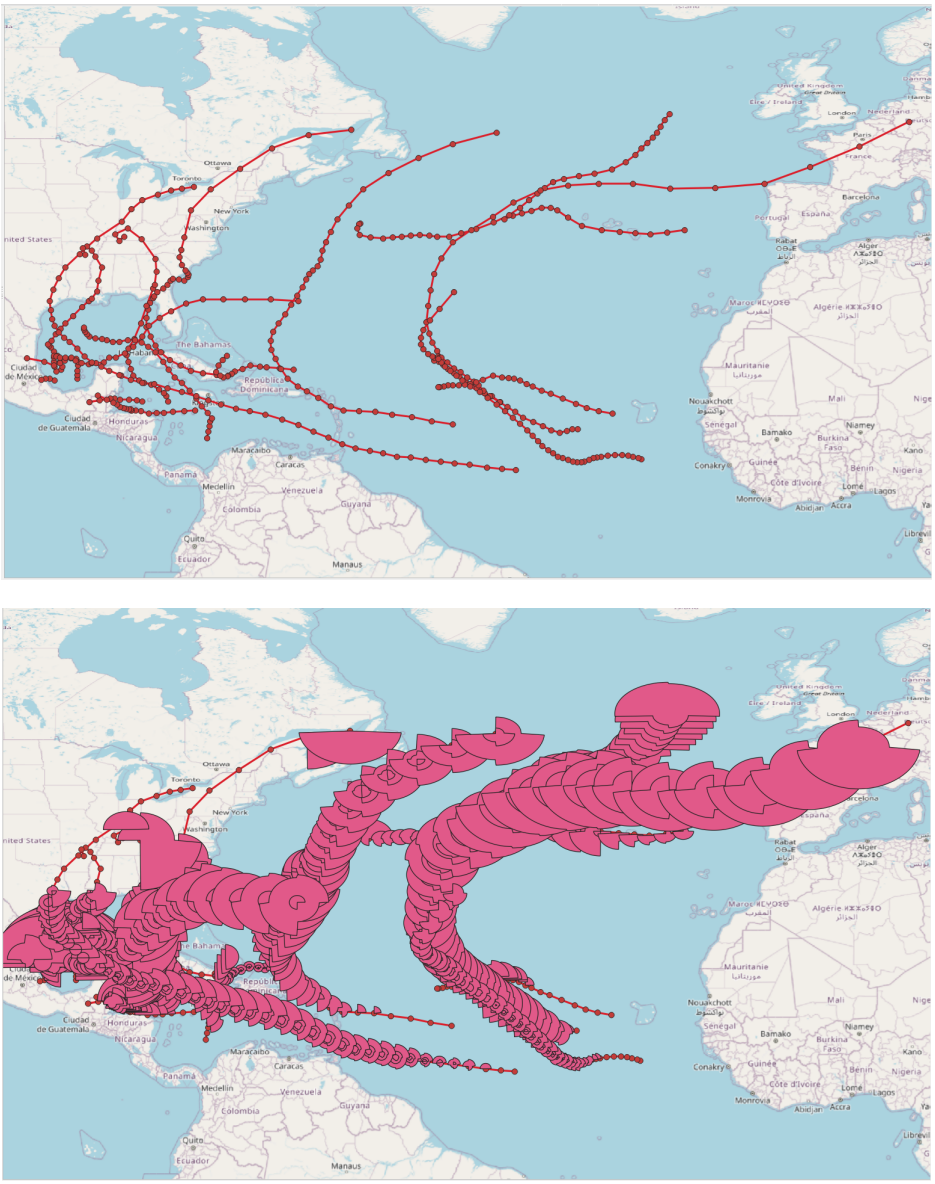


Figura 7.2: Ilustración del uso de los tipos `tgeompoint` (arriba) y `tgeometry` (abajo) para modelizar la trayectoria y la franja de viento de las tormentas tropicales en 2024.

- `tgeo` representa un tipo temporal geometría/geografía, es decir, `tgeometry`, `tgeography`, `tgeompoint`, o `tgeogpoint`,
- `tgeom` representa un tipo temporal geometría, es decir, `tgeometry` o `tgeompoint`,
- `tpoint` representa un tipo temporal punto, es decir, `tgeompoint` o `tgeogpoint`,
- `spatial` representa un tipo de base espacial, por ejemplo, `geometry`, `geography`, `pose`, o `npoint`,
- `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.
- `lines` representa los tipos `geometry` o `geography` restringidos a una (multi)línea.

A continuación, se especifica con el símbolo que la función admite geometrías en 3D y con el símbolo que la función admite geografías.

7.3. Entrada y salida

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

  `asText({tspatial,tspatial[],spatial[]}) → {text,text[]}`

  `asEWKT({tspatial,tspatial[],spatial[]}) → {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[tgeometry 'SRID=4326;[Point(0 0)@2001-01-01,
  Linestring(1 1,2 1)@2001-01-02]', 'Polygon((1 1,2 2,3 1,1 1))@2001-01-01']);
/* {"[POINT(0 0)@2001-01-01, LINESTRING(1 1,2 1)@2001-01-02]",
    "POLYGON((1 1,2 2,3 1,1 1))@2001-01-01"} */
SELECT asText(ARRAY[geometry 'Point(0 0)', 'Point(1 1)']);
-- {"POINT(0 0)","POINT(1 1)"}
```

```
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[tgeometry 'SRID=4326;[Point(0 0)@2001-01-01,
  Linestring(1 1,2 1)@2001-01-02]', 'Polygon((1 1,2 2,3 1,1 1))@2001-01-01']);
-- {"SRID=4326;[POINT(0 0)@2001-01-01, LINESTRING(1 1,2 1)@2001-01-02]",
    "POLYGON((1 1,2 2,3 1,1 1))@2001-01-01"}
SELECT asEWKT(ARRAY[geometry 'SRID=5676;Point(0 0)', 'SRID=5676;Point(1 1)']);
-- {"SRID=5676;POINT(0 0)","SRID=5676;POINT(1 1)"}
```

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

  `asMFJSON(tspatial,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)@2019-01-01 18:00:00.15+02');
/* {"type":"MovingPoint","coordinates":[[1,2]],"datetimes":["2019-01-01T17:00:00.15+01"],
    "interpolation":"None"} */
SELECT asMFJSON(tgeometry 'SRID=3812;Linestring(1 1,1 2)@2019-01-01 18:00:00.15+02');
/* {"type":"MovingGeometry",
    "crs":{"type":"Name","properties":{"name":"EPSG:3812"}},
    "values":[{"type":"LineString","coordinates":[[1,1],[1,2]]}],
    "datetimes":["2019-01-01T17:00:00.15+01"],"interpolation":"None"} */
SELECT asMFJSON(tgeompoint 'SRID=4326;
  Point(50.813810 4.384260)@2019-01-01 18:00:00.15+02', 3, 0, 2);
/* {"type":"MovingPoint","crs":{"type":"name","properties":{"name":"EPSG:4326"}},
    "stBoundedBy":{"bbox":[50.81,4.38,50.81,4.38],
    "period":{"begin":"2019-01-01 17:00:00.15+01","end":"2019-01-01 17:00:00.15+01"}},
    "coordinates":[[50.81,4.38]],"datetimes":["2019-01-01T17:00:00.15+01"],
    "interpolations":"None"} */
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB), la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o la representación hexadecimal extendida binaria conocida (Extended Well-Known Binary o EWKB)

  `asBinary(tgeo,endian text='') → bytea`

  `asEWKB(tgeo,endian text='') → bytea`

  `asHexEWKB(tgeo,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 asBinary(tgeompoint 'Point(1 2 3)@2001-01-01');
-- \x012e0011000000000000f03f0000000000000040000000000000000840009c57d3c11c0000
SELECT asEWKB(tgeogpoint 'SRID=7844;Point(1 2 3)@2001-01-01');
-- \x012f0071a41e0000000000000000f03f0000000000000040000000000000000840009c57d3c11c0000
SELECT asHexEWKB(tgeompoint 'SRID=3812;Point(1 2 3)@2001-01-01');
-- 012E0051E40E0000000000000000F03F0000000000000040000000000000000840009C57D3C11C0000
```

- Entrar a partir de la representación de texto conocido (Well-Known Text o WKT) o de la representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `tspatialFromText(text) → tspatial`

  `tspatialFromEWKT(text) → tspatial`

  En las funciones anteriores, `tspatial` reemplaza cualquier tipo espacial, como `tgeompoint`, `tgeometry` o `tpose`

```
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(tgeographyFromText(text
  '[Point(1 2)@2001-01-01, Linestring(1 2,3 4)@2001-01-02]'));
-- SRID=4326;[POINT(1 2)@2001-01-01, LINESTRING(1 2,3 4)@2001-01-02]
```

```
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, POINT(3 4)@2001-01-02]
SELECT asEWKT(tgeographyFromEWKT(text 'SRID=7844;[Point(1 2)@2001-01-01,
  Linestring(1 2,3 4)@2001-01-02]'));
-- SRID=7844;[POINT(1 2)@2001-01-01, LINESTRING(1 2,3 4)@2001-01-02]
```

- Entrar a partir de la representación JSON de características móviles (Moving Features JSON o MF-JSON)

  `tspatialFromMFJSON(text) → spatial`

  En la funcion anterior, `tspatial` reemplaza cualquier tipo espacial, como `tgeompoint`, `tgeometry` o `tpose`

```
SELECT asEWKT(tgeompointFromMFJSON(text '{"type":"MovingPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[[50.81,4.38]],
  "datetimes":["2019-01-01T17:00:00.15+01"],"interpolation":"None"}'));
-- SRID=4326;POINT(50.81 4.38)@2019-01-01 17:00:00.15+01
SELECT asEWKT(tgeogpointFromMFJSON(text '{"type":"MovingPoint","crs":{"type":"name",
  "properties":{"name":"EPSG:4326"}},"coordinates":[[50.81,4.38]],
  "datetimes":["2019-01-01T17:00:00.15+01"],"interpolation":"None"}'));
-- SRID=4326;POINT(50.81 4.38)@2019-01-01 17:00:00.15+01
SELECT asEWKT(tgeographyFromMFJSON(text '{"type":"MovingGeometry",
  "crs":{"type":"Name","properties":{"name":"EPSG:7844"}},
  "values":[{"type":"LineString","coordinates":[[1,1],[1,2]]}],
  "datetimes":["2019-01-01T17:00:00.15+01"],"interpolation":"None"}'));
-- SRID=7844;LINESTRING(1 1,1 2)@2019-01-01 17:00:00.15+01
```

- Entrar a partir de su representación binaria conocida (WKB), de la representación extendida binaria conocida (EWKB), o de la representación hexadecimal extendida binaria conocida (HexEWKB)

  `tspatialBinary(bytea) → tspatial`

  `tspatialFromEWKB(bytea) → tspatial`

  `tspatialFromHexEWKB(text) → tspatial`

  En las funciones anteriores, `tspatial` reemplaza cualquier tipo espacial, como `tgeompoint`, `tgeometry` o `tpose`

```
SELECT asEWKT(tgeompointFromBinary(
  '\x012e0011000000000000f03f0000000000000040000000000000000840009c57d3c11c0000'));
-- POINT Z (1 2 3)@2001-01-01
SELECT asEWKT(tgeogpointFromEWKB(
  '\x012f0071a41e0000000000000000f03f0000000000000040000000000000000840009c57d3c11c0000'));
-- SRID=7844;POINT Z (1 1 1)@2001-01-01
SELECT asEWKT(tgeompointFromHexEWKB(
  '012E0051E40E0000000000000000F03F0000000000000040000000000000000840009C57D3C11C0000'));
-- SRID=3812;POINT(1 2 3)@2001-01-01
```

7.4. Conversión de tipos

- Convertir un valor espaciotemporal a un rango temporal o a un cuadro delimitador espaciotemporal

  `tspatial::stbox`

  `stbox(tspatial)`

```
SELECT tgeompoint '[Point(1 1)@2001-01-01, Point(3 3)@2001-01-03]'::tstzspan;
-- [2001-01-01, 2001-01-03]
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 tgeography '[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 una geometría temporal y una geografía temporal

  `tgeometry::tgeography`

  `tgeography::tgeometry`

  `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((tgeography 'Linestring(0 0,1 1)@2001-01-01')::tgeometry);
-- LINESTRING(0 0,1 1)@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]} */
```

7.5. Accessores

- Obtenir la trayectoria o el área atravesada

  `trajectory(tpoint,unary_union=false) → geo`

  `traversedArea(tgeo,unary_union=false) → geo`

  Esta función es equivalente a getValues para los valores temporales alphanuméricos. El último argumento indica si se aplica la función PostGIS ST_UnaryUnion para eliminar geometrías redundantes en el resultado. Tenga en cuenta que establecer este argumento como verdadero implica un alto consumo computacional.

```
SELECT ST_AsText(trajectory(tgeompoint '[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02,
  Point(0 0)@2001-01-03]'));
-- LINESTRING(0 0,0 1,0 0)
SELECT ST_AsText(trajectory(tgeompoint '[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02,
```

```
  Point(0 0)@2001-01-03]', true));
-- LINESTRING(0 0,0 1)
SELECT ST_AsText(trajectory(tgeompoint '{[Point(0 0)@2001-01-01, Point(0 1)@2001-01-02),
  [Point(0 1)@2001-01-03, Point(0 0)@2001-01-04)}'));
-- LINESTRING(0 0,0 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),(1 1))
```

```
SELECT ST_AsText(traversedArea(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02, Point(1 1)@2001-01-03]'));
-- GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(1 1,2 2))
SELECT ST_AsText(traversedArea(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02, Point(1 1)@2001-01-03]', true));
-- LINESTRING(1 1,2 2)
```

- Devuelve el centroide como un punto temporal

  `centroid(tgeo) → tpoint`

```
SELECT asText(centroid(tgeometry '[Point(1 1)@2000-01-01, Linestring(1 1,3 3)@2000-01-02,
  Polygon((1 1,4 4,7 1,1 1))@2000-01-03]'));
-- Interp=Step;[POINT(1 1)@2000-01-01, POINT(2 2)@2000-01-02, POINT(4 2)@2000-01-03]
SELECT asText(centroid(tgeography '[MultiPoint(1 1,4 4,7 1)@2000-01-01,
  Polygon((1 1,4 4,7 1,1 1))@2000-01-02]'),6);
-- Interp=Step;[POINT(4 2.001727)@2000-01-01, POINT(4 2.001727)@2000-01-02]
```

- Devuelve 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]
```

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

  `isSimple(tpoint) → boolean`

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

```
SELECT isSimple(tgeompoint '[Point(0 0)@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
```

- Devuelve 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
```

- Devuelve 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]
```

- Devuelve 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
```

- Devuelve 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)
```

- Devuelve 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
```

- Devuelve 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)}
```

- Devuelve 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}
```

- Devuelve 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]
```

7.6. Transformaciones

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

  `round(tspatial,integer=0) → tgeo`

```
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(tgeography 'Linestring(1.12345 1.12345,2.12345 2.12345)@2001-01-01',  ↩
    2));
-- LINESTRING(1.12 1.12,2.12 2.12)@2001-01-01
```

- Devuelve 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 7.3 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) )
```

- Devuelve la transformación afín 3D de una geometría temporal para traducirla, rotarla y/o escalarla en un solo paso

  `affine(tgeo,float a,float b,float c,float d,float e,float f,float g,`
  `float h,float i,float xoff,float yoff,float zoff) → tgeo`

  `affine(tgeo,float a,float b,float d,float e,float xoff,float yoff) → tgeo`

Figura 7.3: Visualización de la velocidad de un objeto móvil usando una rampa de color en QGIS.

```
-- Rotate a 3D temporal point 180 degrees about the z axis
SELECT asEWKT(affine(temp, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), 0, 0, 0, 1,
  0, 0, 0))
FROM (SELECT tgeompoint '[POINT(1 2 3)@2001-01-01, POINT(1 4 3)@2001-01-02]' AS temp) t;
-- [POINT Z (-1 -2 3)@2001-01-01, POINT Z (-1 -4 3)@2001-01-02]
SELECT asEWKT(rotate(temp, pi()))
FROM (SELECT tgeompoint '[POINT(1 2 3)@2001-01-01, POINT(1 4 3)@2001-01-02]' AS temp) t;
-- [POINT Z (-1 -2 3)@2001-01-01, POINT Z (-1 -4 3)@2001-01-02]
-- Rotate a 3D temporal point 180 degrees in both the x and z axis
SELECT asEWKT(affine(temp, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin(pi()),
   0, sin(pi()), cos(pi()), 0, 0, 0))
FROM (SELECT tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,2 2)@2001-01-02]' AS temp) t;
-- [POINT(-1 -1)@2001-01-01, LINESTRING(-1 -1,-2 -2)@2001-01-02]
```

- Devuelve el punto temporal rotado en sentido antihorario sobre el punto de origen

  `rotate(tgeo,float radians) → tgeo`

  `rotate(tgeo,float radians,float x0,float y0) → tgeo`

  `rotate(tgeo,float radians,geometry pointOrigin) → tgeo`

```
-- Rotate a temporal point 180 degrees
SELECT asEWKT(rotate(tgeompoint '[Point(5 10)@2001-01-01, Point(5 5)@2001-01-02,
  Point(10 5)@2001-01-03]', pi()), 6);
-- [POINT(-5 -10)@2001-01-01, POINT(-5 -5)@2001-01-02, POINT(-10 -5)@2001-01-03]
-- Rotate 30 degrees counter-clockwise at x=5, y=10
SELECT asEWKT(rotate(tgeompoint '[Point(5 10)@2001-01-01, Point(5 5)@2001-01-02,
  Point(10 5)@2001-01-03]', pi()/6, 5, 10), 6);
-- [POINT(5 10)@2001-01-01, POINT(7.5 5.67)@2001-01-02, POINT(11.83 8.17)@2001-01-03]
-- Rotate 60 degrees clockwise from centroid
SELECT asEWKT(rotate(temp, -pi()/3, ST_Centroid(traversedArea(temp))), 2)
FROM (SELECT tgeometry '[Point(5 10)@2001-01-01, Point(5 5)@2001-01-02,
  Linestring(5 5,10 5)@2001-01-03]' AS temp) AS t;
/* [POINT(10.58 9.67)@2001-01-01, POINT(6.25 7.17)@2001-01-02,
    LINESTRING(6.25 7.17,8.75 2.83)@2001-01-03] */
```

- Devuelve un punto temporal escalado por factores dados

  `scale(tgeo,float Xfactor,float Yfactor,float Zfactor) → tgeo`

  `scale(tgeo,float Xfactor,float Yfactor) → tgeo`

  `scale(tgeo,geometry factor) → tgeo`

  `scale(tgeo,geometry factor,geometry origin) → tgeo`

```
SELECT asEWKT(scale(tgeompoint '[Point(1 2 3)@2001-01-01, Point(1 1 1)@2001-01-02]',
  0.5, 0.75, 0.8));
--  [POINT Z (0.5 1.5 2.4)@2001-01-01, POINT Z (0.5 0.75 0.8)@2001-01-02]
SELECT asEWKT(scale(tgeompoint '[Point(1 2 3)@2001-01-01, Point(1 1 1)@2001-01-02]',
  0.5, 0.75));
-- [POINT Z (0.5 1.5 3)@2001-01-01, POINT Z (0.5 0.75 1)@2001-01-02]
SELECT asEWKT(scale(tgeompoint '[Point(1 2 3)@2001-01-01, Point(1 1 1)@2001-01-02]',
```

```
  geometry 'Point(0.5 0.75 0.8)'));
-- [POINT Z (0.5 1.5 2.4)@2001-01-01, POINT Z (0.5 0.75 0.8)@2001-01-02]
SELECT asEWKT(scale(tgeometry '[Point(1 1)@2001-01-01, Linestring(1 1,2 2)@2001-01-02]',
  geometry 'Point(2 2)', geometry 'Point(1 1)'));
-- [POINT(1 1)@2001-01-01, LINESTRING(1 1,3 3)@2001-01-02]
```

- 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}
```

- Extraer de un punto temporal con interpolación lineal las subsecuencias donde el punto permanece dentro de un área con un tamaño máximo especificado durante al menos la duración dada

  `stops(tpoint,maxDist=0.0,minDuration='0 minutes') → tpoint`

  El tamaño del área se calcula como la diagonal del rectángulo mínimo rotado de los puntos de la subsecuencia. Si no se especifica `maxDist`, se asume 0.0 y, por lo tanto, la función extrae los segmentos constantes del punto temporal dado. La distancia se calcula en las unidades del sistema de coordenadas. Tenga en cuenta que, aunque la función acepta geometrías 3D, el cálculo siempre se realiza en 2D.

```
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]} */
```

Capítulo 8

Tipos temporales geométricos (Parte 2)

8.1. Restricciones

- Restringir a (al complemento de) una geometría

  `atGeometry(tgeom,geometry) → tgeom`

  `minusGeometry(tgeom,geometry]) → tgeom`

  La geometría debe ser 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))'));
-- {[POINT Z (2 1 1)@2001-01-02, POINT Z (3 1 1)@2001-01-03, POINT Z (3 1 3)@2001-01-05]}
SELECT asText(atGeometry(tgeometry 'Linestring(1 1,10 1)@2001-01-01',
  'Polygon((0 0,0 5,5 5,5 0,0 0))'));
-- LINESTRING(1 1,5 1)@2001-01-01
```

```
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))'));
-- {[POINT Z (1 1 1)@2001-01-01, POINT Z (2 1 1)@2001-01-02)}
SELECT asText(minusGeometry(tgeometry 'Linestring(1 1,10 1)@2001-01-01',
  'Polygon((0 0,0 5,5 5,5 0,0 0))'));
-- LINESTRING(5 1,10 1)@2001-01-01
```

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

  `atStbox(tgeom,stbox,borderInc bool=true) → tgeompoint`

  `minusStbox(tgeom,stbox,borderInc bool=true) → tgeompoint`

El tercer argumento opcional se utiliza para mosaicos multidimensionales (ver Sección 9.5) para excluir el borde superior de los mosaicos cuando un valor temporal se divide en varios mosaicos, de modo que todos los fragmentos de la geometría temporal sean exclusivos.

```
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(atStbox(tgeometry '[Point(1 1)@2001-01-01, Linestring(1 1,3 3)@2001-01-03,
  Point(2 2)@2001-01-04, Linestring(3 3,4 4)@2001-01-09]', stbox 'STBOX X((2,2),(3,3))'));
-- {[LINESTRING(2 2,3 3)@2001-01-03, POINT(2 2)@2001-01-04, POINT(3 3)@2001-01-09]}
```

```
SELECT asText(minusStbox(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])'));
-- {(POINT(2 2)@2001-01-02, 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]} */
SELECT asText(minusStbox(tgeometry '[Point(1 1)@2001-01-01,
  Linestring(1 1,3 3)@2001-01-03, Point(2 2)@2001-01-04,
  Linestring(1 1,4 4)@2001-01-09]', stbox 'STBOX X((2,2),(3,3))'));
/* {[POINT(1 1)@2001-01-01, LINESTRING(1 1,2 2)@2001-01-03,
   LINESTRING(1 1,2 2)@2001-01-04),[MULTILINESTRING((1 1,2 2),(3 3,4 4))@2001-01-09]} */
```

- Restringir a (al complemento de) un rango de altitud

  `atElevation(tgeom,zspan) → tgeom`

  `minusElevation(tgeom,zspan) → tgeom`

```
SELECT astext(atElevation(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(4 4 4)@2001-01-04, Point(1 1 1)@2001-01-07]', floatspan '[2,3]'));
/* {[POINT Z (2 2 2)@2001-01-02, POINT Z (3 3 3)@2001-01-03],
  [POINT Z (3 3 3)@2001-01-05, POINT Z (2 2 2)@2001-01-06]} */
SELECT astext(minusElevation(tgeompoint '[Point(1 1 1)@2001-01-01,
  Point(4 4 4)@2001-01-04, Point(1 1 1)@2001-01-07]', floatspan '[2,3]'));
/* {[POINT Z (2 2 2)@2001-01-02, POINT Z (3 3 3)@2001-01-03],
  [POINT Z (3 3 3)@2001-01-05, POINT Z (2 2 2)@2001-01-06]} */
```

8.2. Sistema de referencia espacial

- Devuelve o establece el identificador de referencia espacial

  `SRID(tgeo) → integer`

  `setSRID(tgeo) → tgeo`

```
SELECT SRID(tgeompoint 'Point(0 0)@2001-01-01');
-- 0
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(tgeo,to_srid integer) → tgeo`

  `transformPipeline(tgeo,pipeline text,to_srid integer,is_forward bool=true) → tgeo`

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:

`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(tgeo, 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(tgeo, 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

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

  `expandSpace({geo,tgeo},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. Operaciones de distancia

- Devuelve la distancia más pequeña que haya existido

  `{geo,tgeo} |=| {geo,tgeo} → 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 tgeometry '(Point(1 1)@2001-01-01, Linestring(3 1,1 1)@2001-01-03]' |=|
  geometry 'Linestring(1 3,2 2,3 3)';
-- 1
```

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
```

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

  `nearestApproachInstant({geo,tgeo},{geo,tgeo}) → tgeo`

  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(tgeometry
  '[Linestring(0 0 0,1 1 1)@2001-01-01, Point(0 0 0)@2001-01-03]', tgeometry
  '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04, Point(2 2 2)@2001-01-06]'));
-- LINESTRING Z (0 0 0,1 1 1)@2001-01-02
```

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
```

- Devuelve la distancia espacial mínima entre dos geometrías temporales sobre la unión de sus extensiones temporales

  `minDistance({tgeo,geo},{tgeo,geo}) → float`

  `minDistance(tgeo[],tgeo[]) → float`

  `minDistance(tgeo,tgeo) → float` (agregado)

  La forma escalar devuelve la distancia mínima alcanzada entre los dos argumentos sobre la intersección de sus extensiones temporales, equivalente al valor mínimo de la distancia temporal `<->`. La forma de arreglo generaliza esto a dos conjuntos de geometrías temporales y devuelve el mínimo sobre todos los pares. La forma de agregado, utilizada con `GROUP BY`, devuelve la distancia mínima alcanzada sobre todos los pares agregados en el grupo.

```
SELECT minDistance(tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-03)',
  geometry 'Point(0 1)');
-- 0.707106781186548
SELECT minDistance(
  ARRAY[tgeompoint '[Point(0 0)@2001-01-01, Point(1 1)@2001-01-03)',
        tgeompoint '[Point(5 5)@2001-01-01, Point(6 6)@2001-01-03)'],
  ARRAY[tgeompoint '[Point(0 1)@2001-01-01, Point(1 0)@2001-01-03)']);
```

```
-- 0
SELECT g, minDistance(t1.Trip, t2.Trip)
FROM Trips t1, Trips t2, Groups g
WHERE t1.GroupId = g AND t2.GroupId = g AND t1.TripId < t2.TripId
GROUP BY g;
```

La forma de agregado se adapta bien a las consultas tipo BerlinMOD que calculan la distancia mínima de aproximación entre pares de trayectorias agrupados por licencia o tipo de vehículo. En un grupo de una sola fila, el agregado se reduce a la distancia mínima escalar entre el par.

- Devuelve los pares de índices que cumplen la relación entre dos arreglos de geometrías temporales, generalizando las relaciones espaciales escalares a una unión espacial conjunto-conjunto

  `eDwithinPairs(tgeo[],tgeo[],float) → setof(i integer, j integer)`

  `aDwithinPairs(tgeo[],tgeo[],float) → setof(i integer, j integer)`

  `eIntersectsPairs(tgeo[],tgeo[]) → setof(i integer, j integer)`

  `aIntersectsPairs(tgeo[],tgeo[]) → setof(i integer, j integer)`

  `eDisjointPairs(tgeo[],tgeo[]) → setof(i integer, j integer)`

  `aDisjointPairs(tgeo[],tgeo[]) → setof(i integer, j integer)`

  `eTouchesPairs(tgeometry[],tgeometry[]) → setof(i integer, j integer)`

  `aTouchesPairs(tgeometry[],tgeometry[]) → setof(i integer, j integer)`

  `tDwithinPairs(tgeo[],tgeo[],float) → setof(i integer, j integer, periods tstzspanset)`

  `tIntersectsPairs(tgeo[],tgeo[]) → setof(i integer, j integer, periods tstzspanset)`

  `tDisjointPairs(tgeo[],tgeo[]) → setof(i integer, j integer, periods tstzspanset)`

  `tTouchesPairs(tgeometry[],tgeometry[]) → setof(i integer, j integer, periods tstzspanset`

  Cada función resuelve el producto cruzado podado de los dos arreglos de entrada en una sola llamada y devuelve los índices basados en uno `i` y `j` de los pares que cumplen la relación. Las funciones con prefijo `e` devuelven los pares para los que la relación se cumple alguna vez, las funciones con prefijo `a` los pares para los que se cumple siempre, y las funciones con prefijo `t` devuelven además los períodos durante los cuales se cumple. La caja espaciotemporal de cada entrada se usa como prefiltro de caja envolvente, de modo que la relación exacta se calcula solo sobre los pares candidatos que sobreviven, y los pares cuyas extensiones temporales no se solapan se excluyen, igual que en las relaciones escalares. Las funciones están definidas para puntos temporales y geometrías temporales tanto en el caso planar como en el geodésico, excepto la relación de toca, que está definida solo para geometrías planares. Los índices se relacionan con las identidades originales mediante un `array_agg(identidad ORDER BY ...)` paralelo.

```
SELECT i, j FROM eDwithinPairs(
  ARRAY[tgeompoint '[Point(0 0)@2001-01-01, Point(0 10)@2001-01-02]',
        tgeompoint '[Point(50 50)@2001-01-01, Point(50 60)@2001-01-02]'],
  ARRAY[tgeompoint '[Point(1 0)@2001-01-01, Point(1 10)@2001-01-02]'], 2.0);
-- 1 | 1
SELECT i, j, periods FROM tDwithinPairs(
  ARRAY[tgeompoint '[Point(0 0)@2001-01-01, Point(0 10)@2001-01-02]'],
  ARRAY[tgeompoint '[Point(1 0)@2001-01-01, Point(1 10)@2001-01-02]'], 2.0);
-- un par, con el período durante el cual está dentro de la distancia
SELECT i, j FROM aDisjointPairs(
  ARRAY[tgeompoint '[Point(0 0)@2001-01-01, Point(0 10)@2001-01-02]'],
  ARRAY[tgeompoint '[Point(5 0)@2001-01-01, Point(5 10)@2001-01-02]']);
-- 1 | 1
```

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

  `shortestLine({geo,tgeo},{geo,tgeo}) → 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(tgeometry
  '[Linestring(0 0 0,1 1 1)@2001-01-01, Point(0 0 0)@2001-01-03]', tgeometry
  '[Point(2 0 0)@2001-01-02, Point(1 1 1)@2001-01-04, Point(2 2 2)@2001-01-06]'));
-- LINESTRING Z (0 0 0,2 0 0)
```

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.

- Devuelve la distancia temporal

  `{geo,tgeo} <-> {geo,tgeo} → 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 tgeometry '[Point(0 0)@2001-01-01, Linestring(0 0,1 1)@2001-01-02]' <->
  tgeometry '[Point(0 1)@2001-01-01, Point(1 0)@2001-01-02)';
-- Interp=Step;[1@2001-01-01, 1@2001-01-02]
```

8.5. 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:

- Las relaciones *alguna vez* determinan determinan si la relación topológica o de distancia se satisface alguna vez (ver Sección 5.4.2) 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 5.4.2) 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 se han generalizado para geometrías temporales, solo las derivadas de las siguientes funciones: `ST_Contains`, `ST_Covers`, `ST_Disjoint`, `ST_Intersects`, `ST_Touches` y `ST_DWithin`. Estas funciones solo admiten geometrías 2D, y solo las funciones `ST_Covers`, `ST_Intersects` y `ST_DWithin` admiten geografías. Por lo tanto, esto mismo aplica a las funciones de MobilityDB derivadas de ellas, excepto que admiten 3D para puntos temporales, es decir, `tgeompoint` y `tgeogpoint`. Como se mencionó anteriormente, cada una de las funciones PostGIS mencionadas, como `ST_Contains`, tiene tres versiones generalizadas en MobilityDB: `eContains`, `aContains` y `tContains`. Además, no todas las combinaciones de parámetros son relevantes para las funciones generalizadas. Por ejemplo, `tContains(tpoint, geometría)` solo es relevante cuando la geometría es un solo punto, y `tContains(tpoint, tpoint)` es equivalente a `tintersects(tpoint, geometría)`.

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

8.5.1. Relaciones alguna vez o siempre

Presentamos a continuación las relaciones 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(geometry,tgeom)` → `boolean`

  `aContains(geometry,tgeom)` → `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))',
  tgeometry '[Linestring(1 1,4 4)@2001-01-01, Point(3 3)@2001-01-04]');
-- true
```

- Ever or always covers

  `eCovers({geometry,tgeom},{geometry,tgeom})` → `boolean`

  `aCovers({geometry,tgeom},{geometry,tgeom})` → `boolean`

  Please refer to the documentation of the ST_Contains and the ST_Covers function in PostGIS for detailed explanations about the difference between the two functions.

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

- Está disjunto alguna vez

  `eDisjoint({geometry,tgeom},{geometry,tgeom}) → boolean`

  `aDisjoint({geo,tgeo},{geo,tgeo}) → 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))',
  tgeometry '[Linestring(1 1 1,2 2 2)@2001-01-01, Point(2 2 2)@2001-01-03]');
-- true
```

- Está alguna vez a distancia de

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

  `aDwithin({geometry,tgeom},{geometry,tgeom},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,tgeo},{geo,tgeo}) → boolean`

  `aIntersects({geometry,tgeom},{geometry,tgeom}) → 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({geometry,tgeom},{geometry,tgeom}) → boolean`

  `aTouches({geometry,tgeom},{geometry,tgeom}) → 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
```

8.5.2. Relaciones espaciotemporales

- Contiene temporal

  `tContains(geometry,tgeom) → 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,tgeo},{geo,tgeo}) → 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({geo,tgeo},{geo,tgeo},float) → 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,tgeo},{geo,tgeo}) → 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({geometry,tgeom},{geometry,tgeom}) → 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 Euclidean 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.

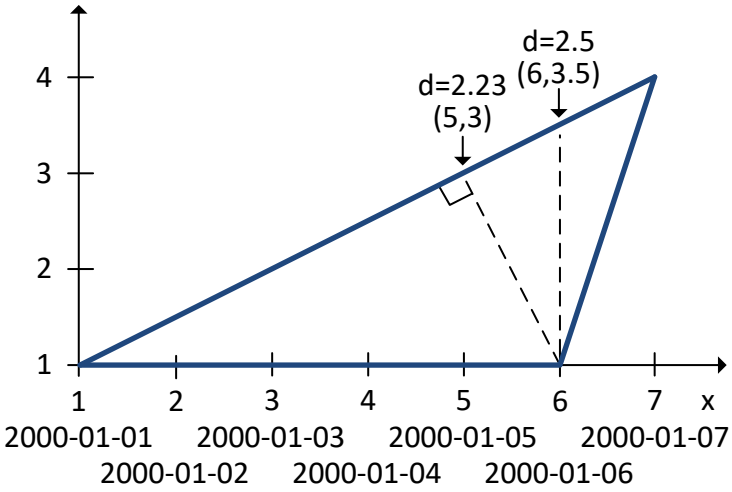


Figura 9.1: Diferencia entre la distancia espacial y la distancia sincronizada.

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(temporal,duration interval,torigin timestamptz='2000-01-03',`

```
interp='discrete') → temporal
```

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 tsample(tbool '[true@2001-01-01,false@2001-01-02]', '12 hours', '2001-01-01', 'step ↩
    ');
-- [t@2001-01-01, f@2001-01-02]
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 tsample(tfloat '[1@2001-01-01,5@2001-01-05]', '3 days', '2001-01-01', 'linear');
-- [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}
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', 'step'));
-- Interp=Step;[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.

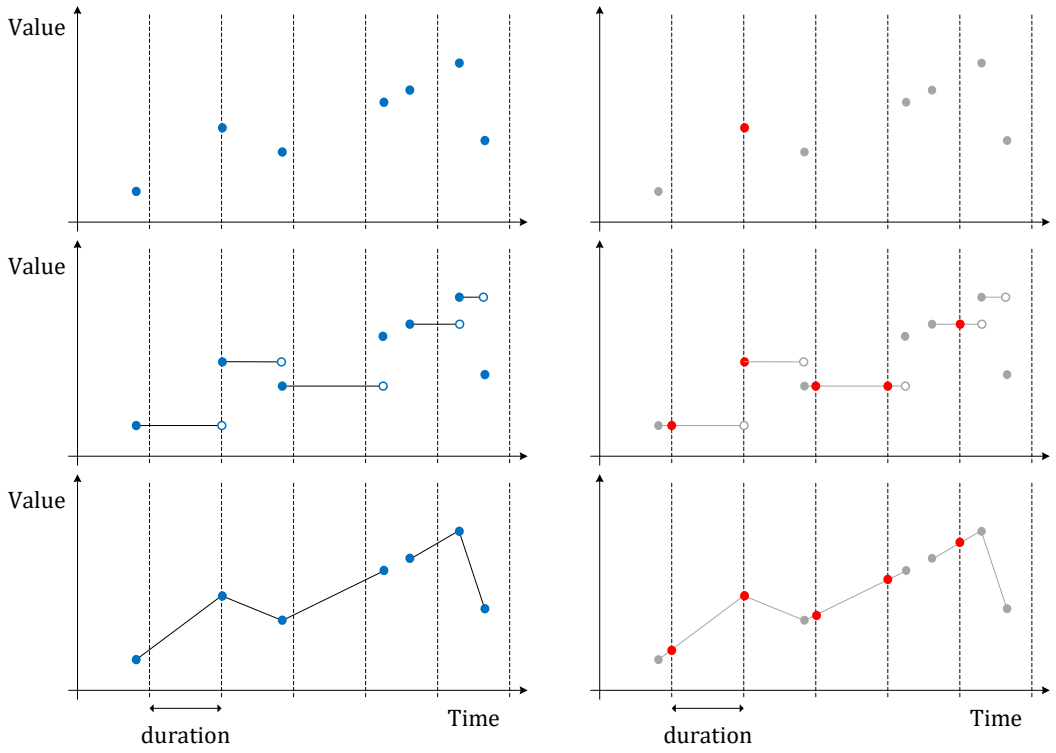


Figura 9.2: Muestreo de flotantes temporales con interpolación discreta, escalonada y lineal.

- 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,tcbuffer,tpose,tgeo,trgeometry},duration interval,torigin
timestamptz='2000-01-03')
 → {tnumber,tgeompoint,tcbuffer,tpose,tgeo,trgeometry}
```

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.

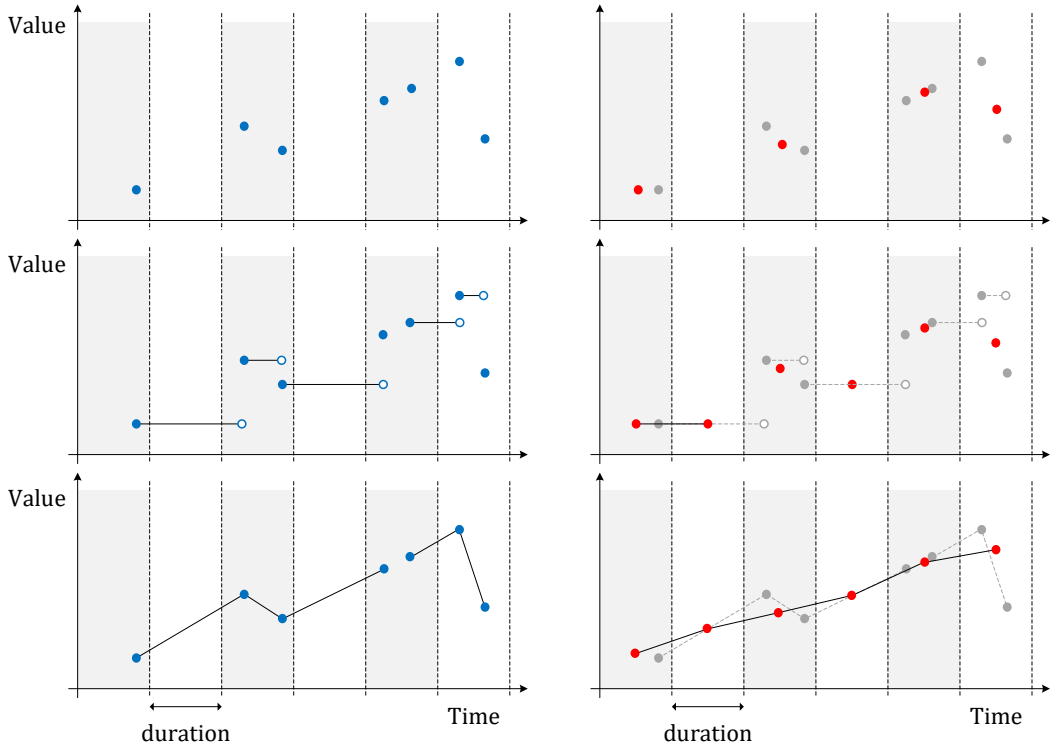


Figura 9.3: Cambio de precisión de números flotantes temporales con interpolación discreta, escalonada y lineal.

9.3. Similaridad

- Devuelve la distancia de Hausdorff discreta, la distancia de Fréchet discreta, la distancia de distorsión de tiempo dinámica (Dynamic Time Warp o DTW), la distancia de Hausdorff promedio, o la distancia de subsecuencia común más larga (Longest Common SubSequence o LCSS) entre dos valores temporales

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

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

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

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

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

  La función `lcssDistance` requiere un parámetro adicional que especifica el umbral de distancia por debajo del cual dos valores se consideran coincidentes.

  Estas funciones tienen 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, las funciones requerien 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
```

```
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
```

```
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
```

```
SELECT round(averageHausdorffDistance(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]') ↩
     ::numeric, 6);
-- 0.283333
SELECT round(averageHausdorffDistance(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);
-- 0.385218
```

```
SELECT lcssDistance(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]',  ↩
     1.0);
-- 3
SELECT round(lcssDistance(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]', 1.0)::numeric, 6);
-- 2.000000
```

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

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

  `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 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)
```

```
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. Operaciones de división

Al crear índices para tipos temporales, lo que se almacena en el índice no es el valor real sino un cuadro delimitador que *representa* el valor. En este caso, el índice proporcionará una lista de valores candidatos que *pueden* satisfacer el predicado de la consulta, y se necesita un segundo paso para filtrar los valores candidatos calculando el predicado de la consulta sobre los valores reales.

Sin embargo, cuando los cuadros delimitadores tienen un gran espacio vacío no cubierto por los valores reales, el índice generará muchos valores candidatos que no satisfacen el predicado de la consulta, lo que reduce la eficiencia del índice. En estas situaciones, puede ser mejor representar un valor no con un cuadro delimitador *único*, sino con *múltiples* cuadros delimitadores. Esto aumenta considerablemente la eficiencia del índice, siempre que el índice sea capaz de gestionar múltiples cuadros delimitadores por valor. Las siguientes funciones se utilizan para generar múltiples cuadros delimitadores para un único valor temporal.

- Devuelve una matriz de N rangos de tiempo a partir de los instantes o segmentos de un valor temporal

  `splitNSpans(temp, integer) → tstzspan[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta o continua. El último argumento especifica el número de rangos de salida. Si el número de instantes o segmentos es inferior o igual al número especificado, la matriz resultante tendrá un rango por instante o segmento del valor temporal. En caso contrario, el número de rangos especificado se obtendrá fusionando instantes o segmentos consecutivos.

```
SELECT splitNSpans(ttext '{A@2000-01-01, B@2000-01-02, A@2000-01-03, B@2000-01-04,
  A@2000-01-05}', 1);
-- {"[2000-01-01, 2000-01-05]"}
SELECT splitNSpans(tfloat '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 2@2000-01-04,
  1@2000-01-05}', 2);
-- {"[2000-01-01, 2000-01-03]","[2000-01-04, 2000-01-05]"}
SELECT splitNSpans(tgeompoint '[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(1 1)@2000-01-03, Point(2 2)@2000-01-04, Point(1 1)@2000-01-05]', 2);
-- {"[2000-01-01, 2000-01-03]","[2000-01-03, 2000-01-05]"}
SELECT splitNSpans(tgeogpoint '{[Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02],
  [Point(1 1 1)@2000-01-03, Point(2 2 2)@2000-01-04], [Point(1 1 1)@2000-01-05]}', 2);
-- {"[2000-01-01, 2000-01-04])","[2000-01-05, 2000-01-05])"}
SELECT splitNSpans(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 2@2000-01-04,
  2@2000-01-05}', 6);
```

```
/* {"[2000-01-01, 2000-01-01]","[2000-01-02, 2000-01-02]","[2000-01-03, 2000-01-03]",
    "[2000-01-04, 2000-01-04]","[2000-01-05, 2000-01-05]"} */
SELECT splitNSpans(ttext '[A@2000-01-01, B@2000-01-02, A@2000-01-03, B@2000-01-04,
  A@2000-01-05]', 6);
/* {"[2000-01-01, 2000-01-02]","[2000-01-02, 2000-01-03]",
    "[2000-01-03, 2000-01-04]","[2000-01-04, 2000-01-05]"} */
```

- Devuelve una matriz de rangos de tiempo obtenida fusionando N instantes o segmentos consecutivos de un valor temporal

  `splitEachNSpans(temp, integer) → tstzspan[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta o continua. El último argumento especifica el número de instantes o segmentos de entrada que se fusionan para producir un rango de salida. Si el número de instantes o segmentos es inferior o igual al número especificado, la matriz resultante tendrá un único rango por secuencia. En caso contrario, el número especificado de instantes o segmentos consecutivos será fusionado en cada rango de salida. Observe que, a diferencia de la función splitNSpans, el número de cuadros en el resultado depende del número de instantes o de segmentos de entrada.

```
SELECT splitEachNSpans(ttext '{A@2000-01-01, B@2000-01-02, A@2000-01-03, B@2000-01-04,
  A@2000-01-05}', 1);
/* {"[2000-01-01, 2000-01-01]","[2000-01-02, 2000-01-02]","[2000-01-03, 2000-01-03]",
     "[2000-01-04, 2000-01-04]","[2000-01-05, 2000-01-05]} */
SELECT splitEachNSpans(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 2@2000-01-04,
  1@2000-01-05]', 2);
--  {"[2000-01-01, 2000-01-03]","[2000-01-03, 2000-01-05]"}
SELECT splitEachNSpans(tgeompoint '{[Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02],
  [Point(1 1 1)@2000-01-03, Point(2 2 2)@2000-01-04], [Point(1 1 1)@2000-01-05]}', 2);
--  {"[2000-01-01, 2000-01-02]","[2000-01-03, 2000-01-04]","[2000-01-05, 2000-01-05]"}
SELECT splitEachNSpans(tgeogpoint '[Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02,
  Point(1 1 1)@2000-01-03, Point(2 2 2)@2000-01-04, Point(1 1 1)@2000-01-05]', 6);
-- {"[2000-01-01, 2000-01-05])"}
SELECT splitEachNSpans(tgeogpoint '{[Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02],
  [Point(1 1 1)@2000-01-03, Point(2 2 2)@2000-01-04], [Point(1 1 1)@2000-01-05]}', 6);
-- {"[2000-01-01, 2000-01-02]","[2000-01-03, 2000-01-04]","[2000-01-05, 2000-01-05]"}
```

- Devuelve una matriz de N cuadros temporales obtenida fusionando los instantes o segmentos de un número temporal

  `splitNTboxes(tnumber, integer) → tbox[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta or continua. El último argumento especifica el número de cuadros de salida. Si el número de instantes o segmentos es inferior o igual al numéro especificado, la matriz resultante tendrá un cuadro por instante o segmento del número temporal. En caso contrario, En caso contrario, el número de cuadros especificado se obtendrá fusionando instantes o segmentos consecutivos.

```
SELECT splitNTboxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05}', 1);
-- {"TBOXINT XT([1, 5),[2000-01-01, 2000-01-05])"}
SELECT splitNTboxes(tfloat '{[1@2000-01-01, 2@2000-01-02], [1@2000-01-03, 4@2000-01-04],
  [1@2000-01-05]}', 2);
/* {"TBOXFLOAT XT([1, 4],[2000-01-01, 2000-01-04])",
    "TBOXFLOAT XT([1, 1],[2000-01-05, 2000-01-05])"} */
SELECT splitNTboxes(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 3);
/* {"TBOXFLOAT XT([1, 2],[2000-01-01, 2000-01-03])",
    "TBOXFLOAT XT([1, 4],[2000-01-03, 2000-01-04])",
    "TBOXFLOAT XT([1, 4],[2000-01-04, 2000-01-05])"} */
SELECT splitNTboxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05}', 6);
/* {"TBOXINT XT([1, 2),[2000-01-01, 2000-01-01])",
    "TBOXINT XT([2, 3),[2000-01-02, 2000-01-02])",
    "TBOXINT XT([1, 2),[2000-01-03, 2000-01-03])",
    "TBOXINT XT([4, 5),[2000-01-04, 2000-01-04])",
```

```
    "TBOXINT XT([1, 2),[2000-01-05, 2000-01-05])"} */
SELECT splitNTboxes(tint '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 6);
/* {"TBOXINT XT([1, 3),[2000-01-01, 2000-01-02])",
    "TBOXINT XT([1, 3),[2000-01-02, 2000-01-03])",
    "TBOXINT XT([1, 5),[2000-01-03, 2000-01-04])",
    "TBOXINT XT([1, 5),[2000-01-04, 2000-01-05])"} */
```

- Devuelve una matriz de cuadros temporales obtenida fusionando N instantes o segmentos consecutivos de un número temporal

  `splitEachNTboxes(tnumber, integer) → tbox[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta or continua. El último argumento especifica el número de instantes o segmentos de entrada que se fusionan para producir un cuadro de salida. Si el número de instantes o segmentos es inferior o igual al número especificado, la matriz resultante tendrá un único cuadro por secuencia. En caso contrario, el número especificado de instantes o segmentos consecutivos será fusionado en cada cuadro de salida. Observe que, a diferencia de la función splitNTboxes, el número de cuadros en el resultado depende del número de instantes o de segmentos de entrada.

```
SELECT splitEachNTboxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05}', 1);
/* {"TBOXINT XT([1, 2),[2000-01-01, 2000-01-01])",
    "TBOXINT XT([2, 3),[2000-01-02, 2000-01-02])",
    "TBOXINT XT([1, 2),[2000-01-03, 2000-01-03])",
    "TBOXINT XT([4, 5),[2000-01-04, 2000-01-04])"} */
SELECT splitEachNTboxes(tint '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 1);
/* {"TBOXINT XT([1, 3),[2000-01-01, 2000-01-02])",
    "TBOXINT XT([1, 3),[2000-01-02, 2000-01-03])",
    "TBOXINT XT([1, 5),[2000-01-03, 2000-01-04])",
    "TBOXINT XT([1, 5),[2000-01-04, 2000-01-05])"} */
SELECT splitEachNTboxes(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 3);
/* {"TBOXFLOAT XT([1, 4],[2000-01-01, 2000-01-04])",
    "TBOXFLOAT XT([1, 4],[2000-01-04, 2000-01-05])"} */
SELECT splitEachNTboxes(tfloat '{[1@2000-01-01, 2@2000-01-02], [1@2000-01-03, 4@2000 ↩
    -01-04],
  [1@2000-01-05]}', 6);
/* {"TBOXFLOAT XT([1, 2],[2000-01-01, 2000-01-02])",
    "TBOXFLOAT XT([1, 4],[2000-01-03, 2000-01-04])",
    "TBOXFLOAT XT([1, 1],[2000-01-05, 2000-01-05])"} */
```

- Devuelve ya sea una matriz de N cuadros espaciales obtenida fusionando los segmentos de una (multi)línea o una matriz de N cuadros espaciotemporales obtenida fusionando los instantes o segmentos de un punto temporal

  `splitNStboxes(lines, integer) → stbox[]`

  `splitNStboxes(tpoint, integer) → stbox[]`

  Para puntos temporales, la elección entre instantes o segmentos depende de si la interpolación es discreta or continua. El último argumento especifica el número de cuadros de salida. Si el número de instantes o segmentos es menor que el número dado, la matriz resultante tendrá un cuadro por segmento de la (multi)línea o un cuadro por instante o segmento del punto temporal. En caso contrario, el número de cuadros especificado se obtendrá fusionando instantes o segmentos consecutivos.

```
SELECT splitNStboxes(geometry 'Linestring(1 1,2 2,3 1,4 2,5 1)', 1);
-- {"STBOX X((1,1),(5,2))"}
SELECT splitNStboxes(geometry 'Linestring(1 1,2 2,3 1,4 2,5 1)', 2);
-- {"STBOX X((1,1),(3,2))","STBOX X((3,1),(5,2))"}
SELECT splitNStboxes(geography 'Linestring(1 1 1,2 2 1,3 1 1,4 2 1,5 1 1)', 6);
/* {"SRID=4326;GEODSTBOX Z((1,1,1),(2,2,1))",
    "SRID=4326;GEODSTBOX Z((2,1,1),(3,2,1))",
    "SRID=4326;GEODSTBOX Z((3,1,1),(4,2,1))",
    "SRID=4326;GEODSTBOX Z((4,1,1),(5,2,1))"} */
```

```
SELECT splitNStboxes(geometry 'MultiLinestring((1 1,2 2),(3 1,4 2),(5 1,6 2))', 2);
-- {"STBOX X((1,1),(4,2))","STBOX X((5,1),(6,2))"}
```

```
SELECT splitNStboxes(tgeompoint '{Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(3 1)@2000-01-03, Point(4 2)@2000-01-04, Point(5 1)@2000-01-05}', 1);
-- {"STBOX XT(((1,1),(5,2)),[2000-01-01, 2000-01-05])"}
SELECT splitNStboxes(tgeompoint '[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(3 1)@2000-01-03, Point(4 2)@2000-01-04, Point(5 1)@2000-01-05]');
/* {"STBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-02])",
    "STBOX XT(((2,1),(3,2)),[2000-01-02, 2000-01-03])",
    "STBOX XT(((3,1),(4,2)),[2000-01-03, 2000-01-04])",
    "STBOX XT(((4,1),(5,2)),[2000-01-04, 2000-01-05])"} */
SELECT splitNStboxes(tgeompoint '{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02],
  [Point(3 1)@2000-01-03, Point(4 2)@2000-01-04], [Point(5 1)@2000-01-05]}', 2);
/* {"STBOX XT(((1,1),(4,2)),[2000-01-01, 2000-01-04])",
    "STBOX XT(((5,1),(5,1)),[2000-01-05, 2000-01-05])"} */
SELECT splitNStboxes(tgeogpoint '{Point(1 1 1)@2000-01-01, Point(2 2 1)@2000-01-02,
  Point(3 1 1)@2000-01-03, Point(4 2 1)@2000-01-04, Point(5 1 1)@2000-01-05}', 6);
/* {"SRID=4326;GEODSTBOX ZT(((1,1,1),(1,1,1)),[2000-01-01, 2000-01-01])",
    "SRID=4326;GEODSTBOX ZT(((2,2,1),(2,2,1)),[2000-01-02, 2000-01-02])",
    "SRID=4326;GEODSTBOX ZT(((3,1,1),(3,1,1)),[2000-01-03, 2000-01-03])",
    "SRID=4326;GEODSTBOX ZT(((4,2,1),(4,2,1)),[2000-01-04, 2000-01-04])",
    "SRID=4326;GEODSTBOX ZT(((5,1,1),(5,1,1)),[2000-01-05, 2000-01-05])"} */
SELECT splitNStboxes(tgeompoint '[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(3 1)@2000-01-03, Point(4 2)@2000-01-04, Point(5 1)@2000-01-05]', 6);
/* {"STBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-02])",
    "STBOX XT(((2,1),(3,2)),[2000-01-02, 2000-01-03])",
    "STBOX XT(((3,1),(4,2)),[2000-01-03, 2000-01-04])",
    "STBOX XT(((4,1),(5,2)),[2000-01-04, 2000-01-05])"} */
```

- Devuelve ya sea una matriz de cuadros espaciales obtenida fusionando N segmentos consecutivos de una (multi)línea o una matriz de cuadros espaciotemporales obtenida fusionando N instantes o segmentos consecutivos de un punto temporal

  `splitEachNStboxes(lines, integer) → stbox[]`

  `splitEachNStboxes(tpoint, integer) → stbox[]`

  Para puntos temporales, la elección entre instantes o segmentos depende de si la interpolación es discreta or continua. El último argumento especifica el número de instantes o segmentos de entrada que se fusionan para producir un cuadro de salida. Si el número de instantes o segmentos es menor que el número dado, la matriz resultante tendrá un único cuadro por secuencia. En caso contrario, el número especificado de instantes o segmentos consecutivos será fusionado en cada cuadro de salida. Observe que, a diferencia de la función splitNStboxes, el número de cuadros en el resultado depende del número de instantes o de segmentos de entrada.

```
SELECT splitEachNStboxes(geometry 'Linestring(1 1,2 2,3 1,4 2,5 1)', 1);
-- {"STBOX X((1,1),(5,2))"}
SELECT splitEachNStboxes(geometry 'Linestring(1 1,2 2,3 1,4 2,5 1)', 2);
-- {"STBOX X((1,1),(3,2))","STBOX X((3,1),(5,2))"}
SELECT splitEachNStboxes(geography 'Linestring(1 1 1,2 2 1,3 1 1,4 2 1,5 1 1)', 6);
/* {"SRID=4326;GEODSTBOX Z((1,1,1),(2,2,1))",
    "SRID=4326;GEODSTBOX Z((2,1,1),(3,2,1))",
    "SRID=4326;GEODSTBOX Z((3,1,1),(4,2,1))",
    "SRID=4326;GEODSTBOX Z((4,1,1),(5,2,1))"} */
SELECT splitEachNStboxes(geometry 'MultiLinestring((1 1,2 2),(3 1,4 2),(5 1,6 2))', 2);
-- {"STBOX X((1,1),(4,2))","STBOX X((5,1),(6,2))"}
```

```
SELECT splitEachNStboxes(tgeompoint '{Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(3 1)@2000-01-03, Point(4 2)@2000-01-04, Point(5 1)@2000-01-05}', 1);
/* {"STBOX XT(((1,1),(1,1)),[2000-01-01, 2000-01-01])",
    "STBOX XT(((2,2),(2,2)),[2000-01-02, 2000-01-02])",
    "STBOX XT(((3,1),(3,1)),[2000-01-03, 2000-01-03])",
    "STBOX XT(((4,2),(4,2)),[2000-01-04, 2000-01-04])",
```

```
    "STBOX XT(((5,1),(5,1)),[2000-01-05, 2000-01-05])"} */
SELECT splitEachNStboxes(tgeompoint '[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(3 1)@2000-01-03, Point(4 2)@2000-01-04, Point(5 1)@2000-01-05]', 1);
/* {"STBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-02])",
    "STBOX XT(((2,1),(3,2)),[2000-01-02, 2000-01-03])",
    "STBOX XT(((3,1),(4,2)),[2000-01-03, 2000-01-04])",
    "STBOX XT(((4,1),(5,2)),[2000-01-04, 2000-01-05])"} */
SELECT splitEachNStboxes(tgeogpoint '{[Point(1 1)@2000-01-01, Point(2 2)@2000-01-02],
  [Point(3 1)@2000-01-03, Point(4 2)@2000-01-04], [Point(5 1)@2000-01-05]}', 2);
/* {"SRID=4326;GEODSTBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-02])",
    "SRID=4326;GEODSTBOX XT(((3,1),(4,2)),[2000-01-03, 2000-01-04])",
    "SRID=4326;GEODSTBOX XT(((5,1),(5,1)),[2000-01-05, 2000-01-05])"} */
```

9.5. Mosaicos multidimensionales

Los mosaicos multidimensionales son un 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. Se pueden utilizar para calcular histogramas multidimensionales, donde los valores temporales se agregan de acuerdo con la partición subyacente del dominio. Por otro lado, los mosaicos multidimensionales también se pueden utilizar para fines de indexación, donde el cuadro delimitador de un valor temporal se puede fragmentar en múltiples cuadros para mejorar la eficiencia del índice. Finalmente, los mosaicos multidimensionales se pueden utilizar para fragmentar valores temporales de acuerdo con una malla multidimensional definida sobre el dominio subyacente. Esto permite la distribución de un conjunto de datos en un clúster de servidores, donde cada servidor contiene una partición del conjunto de datos. La ventaja de este mecanismo de partición es que preserva la proximidad en valor/espacio y tiempo, a diferencia de los mecanismos de partición tradicionales basados en hash.

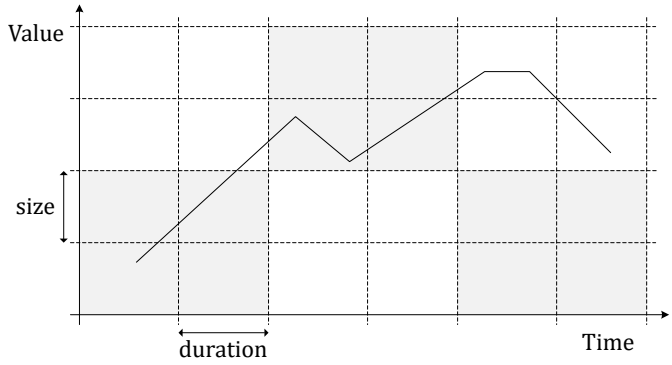


Figura 9.4: Mosaicos multidimensionales para números flotantes temporales.

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 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 {} .

9.5.1. Operaciones de intervalos

- Devuelve una matriz de intervalos que cubre el rango de valores o de tiempo con intervalos de la misma amplitud o duración

  `bins(numspans,width number,origin number=0) → span[]`

  `bins(datespans,duration interval,origin date='2000-01-03') → span[]`

  `bins(tstzspans,duration interval,origin timestamptz='2000-01-03') → span[]`

  Si el origen no se especifica, su valor se establece por defecto en 0 para rangos de valores y en lunes 3 de enero de 2000 para rangos de tiempo.

```
SELECT bins(floatspan '[-10, -1]', 2.5, -7);
-- {"[-12, -9.5)","[-9.5, -7)","[-7, -4.5)","[-4.5, -2)","[-2, 0.5)"}
SELECT bins(intspan '[15, 25]', 2);
-- {"[14, 16)","[16, 18)","[18, 20)","[20, 22)","[22, 24)","[24, 26)","[26, 28)"}
SELECT index, span
FROM unnest(bins(datespan '[2001-01-15, 2001-01-25]', '2 days'))
  WITH ORDINALITY t(span, index);
-- 1 | [2001-01-15, 2001-01-17)
-- 2 | [2001-01-17, 2001-01-19)
-- 3 | [2001-01-19, 2001-01-21)
-- ...
SELECT index, span
FROM unnest(bins(tstzspanset '{[2001-01-15, 2001-01-17],[2001-01-22, 2001-01-25]}',
  '2 days', '2001-01-02')) WITH ORDINALITY t(span, index);
-- 1 | [2001-01-15, 2001-01-16)
-- 2 | [2001-01-16, 2001-01-17]
-- 3 | [2001-01-22, 2001-01-24)
-- 4 | [2001-01-24, 2001-01-25]
```

- Devuelve el rango que contiene un número o una marca de tiempo

  `getBin(number,size number,origin number=0) → span`

  `getBin(date,duration interval,origin date='2000-01-03') → datespan`

  `getBin(timestamptz,duration interval,origin timestamptz='2000-01-03') → tstzspan`

  Si el origen no se especifica, su valor se establece por defecto en 0 para rangos de valores y en lunes 3 de enero de 2000 para rangos de tiempo.

```
SELECT getBin(2, 2);
-- [2, 4)
SELECT getBin(2, 2.5, 1.5);
-- [1.5, 4)
SELECT getBin('2001-01-04', interval '1 week');
-- [2001-01-01, 2001-01-08)
SELECT getBin('2001-01-04', interval '1 week', '2001-01-07');
-- [2000-12-31, 2001-01-07)
```

9.5.2. Operaciones de mosaicos

- Devuelve el conjunto de mosaicos que cubre un cuadro delimitador temporal con mosaicos del mismo tamaño y/o duración {}

  `valueTiles(tbox,size float,vorigin float=0) → {(index,tile)}`

  `timeTiles(tbox,duration interval,torigin timestamptz='2000-01-03') → {(index,tile)}`

```
valueTimeTiles(tbox,size float,duration interval,vorigin float=0,
 torigin timestamptz='2000-01-03') → {(index,tile)}
```

Si el origen de la dimensión de valores y/o de tiempo no se especifica, su valor se establece por defecto en 0 y en el lunes 3 de enero de 2000, respectivamente.

```
SELECT (gr).index, (gr).tile
FROM (SELECT valueTiles(tfloat '[15@2001-01-15, 25@2001-01-25]'::tbox, 2.0) AS gr) t;
-- 1 | TBOXFLOAT X([14, 16))
-- 2 | TBOXFLOAT X([16, 18))
-- 3 | TBOXFLOAT X([18, 20))
-- ...
SELECT (gr).index, (gr).tile
FROM (SELECT timeTiles(tfloat '[15@2001-01-15, 25@2001-01-25]'::tbox, '2 days') AS gr) t;
-- 1 | TBOX T([2001-01-15, 2001-01-17))
-- 2 | TBOX T([2001-01-17, 2001-01-19))
-- 3 | TBOX T([2001-01-19, 2001-01-21))
-- ...
SELECT (gr).index, (gr).tile
FROM (SELECT valueTimeTiles(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 valueTimeTiles(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))")
-- ...
```

- Devuelve el conjunto de mosaicos que cubre un cuadro delimitador espaciotemporal con mosaicos del mismo tamaño y/o duración {}

```
spaceTiles(stbox,xsize float,[ysize float,zsize float,]
 sorigin geompoint='Point(0 0 0)',borderInc bool=true) → {(index,tile)}
timeTiles(stbox,duration interval,torigin timestamptz='2000-01-03',
 borderInc bool=true) → {(index,tile)}
spaceTimeTiles(stbox,xsize float,[ysize float,zsize float,]duration interval,
 sorigin geompoint='Point(0 0 0)',torigin timestamptz='2000-01-03',
 borderInc bool=true) → {(index,tile)}
```

Si el origen de las dimensiones espacial y/o de tiempo no se especifican, su valor se establece por defecto en `'Point (0 0 0)'` y en el lunes 3 de enero de 2000, respectivamente. El argumento opcional `borderInc` indica si se incluye el borde superior de la extensión y, por lo tanto, se generan mosaicos adicionales que contienen el borde.

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 spaceTiles(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 timeTiles(tgeompoint '[Point(3 3)@2001-01-15,
  Point(15 15)@2001-01-25]'::stbox, '2 days');
-- (1,"STBOX T([2001-01-15, 2001-01-17))")
```

```
-- (2,"STBOX T([2001-01-17, 2001-01-19))")
-- (3,"STBOX T([2001-01-19, 2001-01-21))")
-- ...
SELECT spaceTiles(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 spaceTiles(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 spaceTimeTiles(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 spaceTimeTiles(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))")
-- ...
```

- Devuelve el mosaico temporal que cubre un valor y/o una marca de tiempo

  `getValueTile(value float,vsize float,vorigin float=0.0) → tbox`

  `getTboxTimeTile(time timestamptz,duration interval,torigin timestamptz='2000-01-03') torigin timestamptz='2000-01-03') → tbox`

  `getValueTimeTile(value float,time timestamptz,vsize float,duration interval, vorigin float=0.0,torigin timestamptz='2000-01-03') → tbox`

  Si el origen de las dimensiones de valores y/o de tiempo no se especifica, su valor se establece por defecto en 0 y en el lunes 3 de enero de 2000, respectivamente.

```
SELECT getValueTile(15, 2);
-- TBOX ([14,16))
SELECT getTboxTimeTile('2001-01-15', interval '2 days');
-- TBOX ([2001-01-15,2001-01-17))
SELECT getValueTimeTile(15, '2001-01-15', 2, interval '2 days');
-- TBOX ([14,16),[2001-01-15,2001-01-17))
SELECT getValueTimeTile(15, '2001-01-15', 2, interval '2 days', 1, '2001-01-02');
-- TBOX XT([15,17),[2001-01-14,2001-01-16))
```

- Devuelve el mosaico espaciotemporal que cubre un punto y/o una marca de tiempo

  `getSpaceTile(point geometry,xsize float,[ysize float,zsize float], sorigin geompoint='Point(0 0 0)') → stbox`

  `getStboxTimeTile(time timestamptz,duration interval, torigin timestamptz='2000-01-03') → stbox`

  `getSpaceTimeTile(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 la dimensión espacial y/o de 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.

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 getSpaceTile(geometry 'Point(1 1 1)', 2.0);
-- STBOX Z((0,0,0),(2,2,2))
SELECT getStboxTimeTile(timestamptz '2001-01-01', interval '2 days');
-- STBOX T([2001-01-01,2001-01-03))
SELECT getSpaceTimeTile(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 getSspaceTimeTile(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))
```

9.5.3. Operaciones de cuadro delimitador

Estas operaciones fragmentan el cuadro delimitador de un valor temporal con respecto a un mosaico multidimensional. Ofrecen una alternativa a las operaciones de la Sección 9.4 para dividir los cuadros delimitadores especificando el tamaño máximo de los cuadros en las distintas dimensiones.

- Devuelve una matriz de rangos de valores o de tiempo obtenidos a partir de los instantes o segmentos de un número temporal con respecto a un mosaico de valores o de tiempo

  `valueBins(tnumber,vsize number,vorigin number=0) → numspan[]`

  `timeBins(temp,duration interval,torigin timestamptz='2000-01-03') → tstzspan[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta o continua. Si no se especifica el origen de los valores, se establece por defecto en 0 para los rangos de valores y en el lunes 3 de enero de 2000 para los rangos de tiempo.

```
SELECT valueBins(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05}', 3);
-- {"[1, 3)","[4, 5)"}
SELECT valueBins(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 3, 1);
-- {"[1, 4)","[4, 4]"}
```

```
SELECT timeBins(ttext '{AAA@2000-01-01, BBB@2000-01-02, AAA@2000-01-03, CCC@2000-01-04,
  AAA@2000-01-05}', '3 days');
-- {"[2000-01-01, 2000-01-02]","[2000-01-03, 2000-01-05]"}
SELECT timeBins(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', '3 days', '2000-01-01');
-- {"[2000-01-01, 2000-01-04)","[2000-01-04, 2000-01-05)"}
```

- Devuelve una matriz de cuadros temporales obtenidos a partir de los instantes o segmentos de un número temporal con respecto a un mosaico de valores y/o tiempo

  `valueBoxes(tnumber,size number,vorigin number=0) → tbox[]`

  `timeBoxes(tnumber,duration interval,torigin timestamptz='2000-01-03') → tbox[]`

  `valueTimeBoxes(tnumber,size number,duration interval,vorigin number=0,`
  `torigin timestamptz='2000-01-03') → tbox[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta o continua. Si el origen de la dimensión de valores y/o de tiempo no se especifica, se establece por defecto en 0 y en el lunes 3 de enero de 2000, respectivamente.

```
SELECT valueBoxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
1@2000-01-05}', 3);
/* {"TBOXINT XT([1, 3),[2000-01-01, 2000-01-05])",
    "TBOXINT XT([4, 5),[2000-01-04, 2000-01-04])"} */
SELECT timeBoxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
1@2000-01-05}', '3 days');
/* {"TBOXINT XT([1, 3),[2000-01-01, 2000-01-02])",
    "TBOXINT XT([1, 5),[2000-01-03, 2000-01-05])"} */
SELECT valueTimeBoxes(tint '{1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05}', 3, '3 days');
/* {"TBOXINT XT([1, 3),[2000-01-01, 2000-01-02])",
    "TBOXINT XT([1, 2),[2000-01-03, 2000-01-05])",
    "TBOXINT XT([4, 5),[2000-01-04, 2000-01-04])"} */
SELECT valueTimeBoxes(tfloat '[1@2000-01-01, 2@2000-01-02, 1@2000-01-03, 4@2000-01-04,
  1@2000-01-05]', 3, '3 days', 1, '2000-01-01');
/* {"TBOXFLOAT XT([1, 4),[2000-01-01, 2000-01-04))",
    "TBOXFLOAT XT([1, 4),(2000-01-04, 2000-01-05])",
    "TBOXFLOAT XT([4, 4],[2000-01-04, 2000-01-04])"} */
```

- Devuelve una matriz de cuadros espaciotemporales obtenidos a partir de los instantes o segmentos de un punto temporal con respecto a un mosaico espacial y/o temporal

  `spaceBoxes(tgeompoint,xsize float,[ysize float,zsize float,]`
  `sorigin geompoint='Point(0 0 0)',borderInc bool=true) → stbox[]`
  `timeBoxes(tgeompoint,duration interval,torigin timestamptz='2000-01-03',`
  `borderInc bool=true) → stbox[]`
  `spaceTimeBoxes(tgeompoint,xsize float,[ysize float,zsize float,]duration interval,`
  `sorigin geompoint='Point(0 0 0)',torigin timestamptz='2000-01-03',`
  `borderInc bool=true) → stbox[]`

  La elección entre instantes o segmentos depende de si la interpolación es discreta o continua. Los argumentos `ysize` y `zsize` son opcionales, se supone que el tamaño de las dimensiones faltantes es igual a `xsize`. El SRID de las coordenadas del mosaico se determina por el punto temporal y los tamaños se dan en las unidades del SRID. Si se proporciona el origen de las coordenadas espaciales, que debe ser un punto, su dimensionalidad y SRID deben ser iguales a los del punto temporal, de lo contrario se genera un error. Si el origen de la dimensión espacial 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. El argumento opcional `borderInc` indica si se incluye el borde superior de la extensión y, por lo tanto, se generan mosaicos adicionales que contienen el borde.

```
SELECT spaceBoxes(tgeompoint '{Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(1 1)@2000-01-03, Point(4 4)@2000-01-04, Point(1 1)@2000-01-05}', 3);
/* {"STBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-05])",
   "STBOX XT(((4,4),(4,4)),[2000-01-04, 2000-01-04])"} */
SELECT timeBoxes(tgeompoint '{Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02,
  Point(1 1 1)@2000-01-03, Point(4 4 4)@2000-01-04, Point(1 1 1)@2000-01-05}',
  interval '2 days', '2000-01-01');
/* {"STBOX ZT(((1,1,1),(2,2,2)),[2000-01-01, 2000-01-02])",
    "STBOX ZT(((1,1,1),(4,4,4)),[2000-01-03, 2000-01-04])",
    "STBOX ZT(((1,1,1),(1,1,1)),[2000-01-05, 2000-01-05])"} */
SELECT spaceTimeBoxes(tgeompoint '{Point(1 1)@2000-01-01, Point(2 2)@2000-01-02,
  Point(1 1)@2000-01-03, Point(4 4)@2000-01-04, Point(1 1)@2000-01-05}', 3, '3 days');
/* {"STBOX XT(((1,1),(2,2)),[2000-01-01, 2000-01-02])",
    "STBOX XT(((1,1),(1,1)),[2000-01-03, 2000-01-05])",
    "STBOX XT(((4,4),(4,4)),[2000-01-04, 2000-01-04])"} */
SELECT spaceTimeBoxes(tgeompoint '[Point(1 1 1)@2000-01-01, Point(2 2 2)@2000-01-02,
  Point(1 1 1)@2000-01-03, Point(4 4 4)@2000-01-04, Point(1 1 1)@2000-01-05]',
  3, interval '3 days', 'Point(1 1 1)', '2000-01-01');
/* {"STBOX ZT(((1,1,1),(4,4,4)),[2000-01-01, 2000-01-04))",
    "STBOX ZT(((1,1,1),(4,4,4)),(2000-01-04, 2000-01-05])",
    "STBOX ZT(((4,4,4),(4,4,4)),[2000-01-04, 2000-01-04])"} */
```

9.5.4. Operaciones de división

Estas funciones fragmentan un valor temporal con respecto a una secuencia de intervalos (ver la Sección 9.5.1) o un mosaico multidimensional (ver la Sección 9.5.2).

- Fragmentar un número temporal con respecto a intervalos de valores y/o de tiempo {}

  `valueSplit(tnumber,width number,origin number=0) → {(number,tnumber)}`

  `timeSplit(ttype,duration interval,origin timestamptz='2000-01-03') → {(time,temp)}`

  `valueTimeSplit(tnumber,width number,duration interval,vorigin number=0, torigin timestamptz='2000-01-03') → {(number,time,tnumber)}`

  El resultado es un conjunto de pares o un conjunto de triples. Si no se especifica el origen de la dimensión de valores o temporal, se establecen por defecto en 0 y en el lunes 3 de enero de 2000, respectivamente.

```
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)}")
```

```
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)} */
/* ... */
```

```
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 un punto temporal con respecto a una malla espacial o espacio-temporal {}

  `spaceSplit(tgeompoint,xsize float,[ysize float,zsize float,]`
  `origin geompoint='Point(0 0 0)',bitmatrix boolean=true,borderInc bool=true) →`
  `{(point,tpoint)}`

  `timeSplit(tgeompoint,duration interval,torigin timestamptz='2000-01-03',`
  `bitmatrix boolean=true,borderInc boolean=true) → {(point,time,tpoint)}`

  `spaceTimeSplit(tgeompoint,xsize float,[ysize float,zsize float,]`
  `duration interval,sorigin geompoint='Point(0 0 0)',`
  `torigin timestamptz='2000-01-03',bitmatrix boolean=true,borderInc boolean=true) →`
  `{(point,time,tpoint)}`

  El resultado es un conjunto de pares o un conjunto de triples. Si el origen de la dimensión espacial 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. El argumento opcional `borderInc` indica si se incluye el borde superior de la extensión espacial y, por lo tanto, se generan mosaicos adicionales que contienen el borde.

```
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)}
-- ...
```

```
SELECT (sp).time, astext((sp).temp) AS tpoint
FROM (SELECT timeSplit(tgeompoint '[Point(1 1)@2001-02-01, Point(10 10)@2001-02-10]',
  interval '2 days') AS sp) t;
-- 2001-01-31 | [POINT(1 1)@2001-02-01, POINT(2 2)@2001-02-02)
-- 2001-02-02 | [POINT(2 2)@2001-02-02, POINT(4 4)@2001-02-04)
-- 2001-02-04 | [POINT(4 4)@2001-02-04, POINT(6 6)@2001-02-06)
-- ...
```

```
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.

---

nota
Los nombres `tMin`, `tMax`, `merge`, `appendInstant` y `appendSequence` designan tanto una función escalar como una agregación. El nombre canónico de la forma de agregación lleva el sufijo `Agg` (`tMinAgg`, `tMaxAgg`, `mergeAgg`, `appendInstantAgg` y `appendSequenceAgg`), de modo que los nombres escalar y de agregación sean distintos. Los nombres de agregación sin sufijo siguen disponibles como alias de compatibilidad y están obsoletos; se eliminarán en una versión mayor futura. Las formas escalares (los accesores de caja `tMin`/`tMax` y el constructor binario `merge`) conservan sus nombres.

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, o temporal

  `tAnd(tbool) → tbool`

  `tOr(tbool) → tbool`

```
SELECT tAnd(NoEmps #> 6) FROM Department;
-- {[t@2001-01-01, f@2001-04-01, f@2001-10-01)}
SELECT tOr(NoEmps #> 6) FROM Department;
-- {[t@2001-01-01, f@2001-08-01, f@2001-10-01)}
```

- Mínimo, máximo, suma y promedio temporal

  `tMin(ttype) → ttype`

  `tMax(ttype) → ttype`

  `tSum(tnumber) → {tnumberSeq,tnumberSeqSet}`

  `tAvg(tnumber) → {tfloatSeq,tfloatSeqSet}`

```
SELECT tMin(NoEmps) FROM Department;
-- {[10@2001-01-01, 4@2001-02-01, 6@2001-06-01, 6@2001-10-01)}
SELECT tMax(NoEmps) FROM Department;
-- {[10@2001-01-01, 12@2001-04-01, 6@2001-08-01, 6@2001-10-01)}
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)} */
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) */
```

- 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)} */
```

- Mínimo, máximo, suma y promedio de ventana

  `wMin(tnumber,interval) → {tnumberSeq,tnumberSeqSet}`

  `wMax(tnumber,interval) → {tnumberSeq,tnumberSeqSet}`

  `wSum(tint,interval) → {tintSeq,tintSeqSet}`

  `wAvg(tint,interval) → {tfloatDiscSeq,tfloatSeqSet}`

```
SELECT wMin(NoEmps, interval '2 days') FROM Department;
-- {[10@2001-01-01, 4@2001-04-01, 6@2001-06-03, 6@2001-10-03)}
SELECT wMax(NoEmps, interval '2 days') FROM Department;
-- {[10@2001-01-01, 12@2001-04-01, 6@2001-08-03, 6@2001-10-03)}
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)} */
SELECT round(wAvg(NoEmps, interval '2 days'), 3) FROM Department;
/*  Interp=Step;{[10@2001-01-01, 7@2001-02-01, 8.667@2001-04-01, 8@2001-04-03,
   7.333@2001-06-01, 9@2001-06-03, 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 5.3 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 geometrías temporales, todas las relaciones espaciales con la semántica alguna vez o siempre (ver la Sección 8.5) 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 eIntersects(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

Poses temporales

El tipo `pose` se utiliza para representar la *ubicación* y la *orientación* de objetos geométricos dentro de sistemas de coordenadas anclados a la superficie de la Tierra o dentro de otros sistemas de coordenadas astronómicas. La ubicación se representa mediante un punto 2D o 3D. Para las poses 2D, la orientación se define mediante un ángulo de rotación en (-π, π] expresado en radianes. Para las poses 3D, la orientación se define mediante cuatro valores flotantes `W`, `X`, `Y`, `Z`, que representan un cuaternión unitario $Q = \langle W, X, Y, Z \rangle$ donde $\| Q^2 \| = \sqrt{W^2 + X^2 + Y^2 + Z^2} = 1$.

El grupo de trabajo de estándares GeoPose (SWG), que trabaja bajo los auspicios del Open Geospatial Consortium, ha definido un estándar para intercambiar información de poses entre distintos usuarios, dispositivos y plataformas. Se puede encontrar más información sobre el estándar en el repositorio de GitHub del SWG GeoPose.

El tipo `pose` sirve como tipo base para definir el tipo de pose temporal `tpose`. El tipo `tpose` tiene una funcionalidad similar al tipo de punto temporal `tgeompoint`. Por lo tanto, la mayoría de las funciones y operadores descritos anteriormente para el tipo `tgeompoint` también son aplicables para el tipo `tpose`. Además, hay funciones específicas definidas para el tipo `tpose`. En este capítulo cubrimos estas funciones.

El tipo `tpose` se utiliza para definir el tipo `trgeometry` (es decir, geometría rígida temporal) definido en el siguiente capítulo. La implementación de estos tipos en MobilityDB se ha estudiado en la siguiente tesis doctoral.

- Maxime Schoemans, *Managing Rigid Temporal Geometries in Moving Object Databases*, tesis doctoral, Université libre de Bruxelles, 2025.

11.1. Poses estáticas

Una `pose` 2D es un par de la forma `(point2D,radius)` donde `point2D` es un punto geométrico 2D y `radius` es un valor `float` que representa un ángulo de rotación en (-π, π] expresado en radianes. Una `pose` 3D es una tupla de la forma `(point3D,W,Y,X,Z)` donde `point3D` es un punto geométrico 3D, y `W`, `X`, `Y` y `Z` son cuatro `floats` que representan un cuaternión *unitario*. Ejemplos de entrada de valores de pose son los siguientes:

```
SELECT pose 'Pose(Point(1 1), 0.5)';
SELECT pose 'Pose(Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)';
```

Se puede especificar un SRID para una pose ya sea al comienzo del literal de pose o antes del literal de punto, como se muestra a continuación.

```
SELECT pose 'SRID=3812;Pose(Point(1 1), 0.5)';
SELECT pose 'Pose(SRID=5676;Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)';
```

Los valores del tipo pose deben satisfacer varias restricciones para que estén bien definidos. Ejemplos de valores incorrectos del tipo pose son los siguientes.

```
-- Punto vacío
select pose 'Pose(Point empty, 0.5)';
-- Valor de punto incorrecto
SELECT pose 'Pose(Linestring(1 1,2 2), 1.0)';
-- Valor de radio incorrecto
SELECT pose 'Pose(Point(1 1), -10.0)';
-- Punto 3D incorrecto
SELECT pose 'Pose(Point Z(1 1), 1.0)';
-- Orientación 3D incompleta
SELECT pose 'Pose(Point Z(1 1 1), 1.0)';
```

A continuación damos las funciones y operadores para el tipo pose.

11.1.1. Entrada y salida

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

  `asText({pose,pose[]}) → {text,text[]}`

  `asEWKT({pose,pose[]}) → {text,text[]}`

```
SELECT asText(pose 'SRID=4326;Pose(Point(0 0),1)');
-- Pose(POINT(0 0),1)
SELECT asText(ARRAY[pose 'Pose(Point(0 0),1)', 'Pose(Point(1 1),2)']);
-- {"Pose(POINT(0 0),1)","Pose(POINT(1 1),2)"}
SELECT asEWKT(pose 'SRID=4326;Pose(Point(0 0),1)');
-- SRID=4326;Pose(Point(0 0),1)
SELECT asEWKT(ARRAY[pose 'Pose(SRID=5676;Point(0 0),1)', 'Pose(SRID=5676;Point(1 1),2)']);
-- {"Pose(SRID=5676;POINT(0 0),1)","Pose(SRID=5676;POINT(1 1),2))"}
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB), la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o HexEWKB)

  `asBinary(pose,endian text='') → bytea`

  `asEWKB(pose,endian text='') → bytea`

  `asHexEWKB(pose,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 asBinary(pose 'Pose(Point(1 2),1)');
-- \x0101000000000000f03f0000000000000040000000000000f03f
SELECT asEWKB(pose 'SRID=7844;Pose(Point(1 2),1)');
-- \x0141a41e000000000000000f03f0000000000000040000000000000f03f
SELECT asHexEWKB(pose 'SRID=3812;Pose(Point(1 2),1)');
-- 0141E40E000000000000000F03F0000000000000040000000000000F03F
```

- Entrada desde la representación de texto conocido (Well-Known Text o WKT) o desde la representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `poseFromText(text) → pose`

  `poseFromEWKT(text) → pose`

```
SELECT asEWKT(poseFromText(text 'Pose(Point(1 2),1)'));
-- Pose(POINT(1 2),1)
SELECT asEWKT(poseFromEWKT(text 'SRID=3812;Pose(Point(1 2),1)'));
-- SRID=3812;Pose(Point(1 2),1)
```

- Entrada desde la representación binaria conocida (Well-Known Binary o WKB), desde la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o desde la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o HexEWKB)

  `poseFromBinary(bytea) → pose`

  `poseFromEWKB(bytea) → pose`

  `poseFromHexEWKB(text) → pose`

```
SELECT asEWKT(poseFromBinary(
  '\x0101000000000000f03f0000000000000040000000000000f03f'));
-- Pose(POINT(1 2),1)
SELECT asEWKT(poseFromEWKB(
  '\x0141a41e000000000000000f03f0000000000000040000000000000f03f'));
-- SRID=7844;Pose(Point(1 2),1)
SELECT asEWKT(poseFromHexEWKB(
  '0141E40E000000000000000F03F0000000000000040000000000000F03F'));
-- SRID=3812;Pose(POINT(1 2),1)
```

11.1.2. Constructores

- Constructor para poses

  `pose(geompoint2D,float) → pose`

  `pose(geompoint3D,float,float,float,float) → pose`

```
SELECT asText(pose(ST_Point(1,1), radians(45)), 6);
-- Pose(POINT(1 1),0.785398)
SELECT asEWKT(pose(ST_Point(1,1,3812), radians(45)), 6);
-- SRID=3812;Pose(POINT(1 1),0.785398)
SELECT asText(pose(ST_PointZ(1,1,1), 1, 0, 0, 0));
-- Pose(POINT Z (1 1 1),1,0,0,0)
```

11.1.3. Conversión de tipos

Los valores del tipo `pose` se pueden convertir al tipo de punto `geometry` utilizando un `CAST` explícito o utilizando la notación `::` como se muestra a continuación.

- Convertir una pose y, opcionalmente, una marca de tiempo o un período, en una caja espaciotemporal

  `pose::stbox`

  `stbox(pose) → stbox`

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

```
SELECT stbox(pose 'SRID=5676;Pose(Point(1 1),0.3)');
-- SRID=5676;STBOX X((1,1),(1,1))
SELECT stbox(pose 'Pose(Point(1 1),0.3)', timestamptz '2001-01-01');
-- STBOX XT(((1,1),(1.3,1.3)),[2001-01-01, 2001-01-01])
SELECT stbox(pose 'Pose(Point(1 1),0.3)', tstzspan '[2001-01-01,2001-01-02]');
-- STBOX XT(((1,1),(1.3,1.3)),[2001-01-01, 2001-01-02])
```

- Convertir una pose en un punto geométrico

  `pose::geompoint`

```
SELECT ST_AsText(pose(ST_Point(1, 1), 1)::geometry);
-- Point(1 1)
SELECT ST_AsEWKT(pose(ST_PointZ(1, 1, 1, 5676), 1, 0, 0, 0)::geometry);
-- SRID=5676;POINT(1 1 1)
```

11.1.4. Accesores

- Devuelve el punto

  `point(pose) → geompoint`

```
SELECT ST_AsText(point(pose 'Pose(Point(1 1), 0.3)'));
-- Point(1 1)
```

- Devuelve la rotación

  `rotation(pose2D) → float`

```
SELECT rotation(pose 'Pose(Point(1 1), 0.3)');
-- 0.3
```

- Devuelve la orientación

  `orientation(pose3D) → (X,W,Z,T)`

```
SELECT orientation(pose 'Pose(Point Z(1 1 1), 0, 0, 0, 1)');
-- (0, 0, 0, 1)
```

11.1.5. Transformaciones

- Redondear el punto y la orientación de la pose al número de posiciones decimales

  `round(pose,integer=0) → pose`

```
SELECT asText(round(pose(ST_Point(1.123456789,1.123456789), 0.123456789), 6));
-- Pose(POINT(1.123457 1.123457),0.123457)
```

11.1.6. Sistema de referencia espacial

- Devuelve o establece el identificador de referencia espacial

  `srid(pose) → integer`

  `setSRID(pose) → pose`

```
SELECT SRID(pose 'Pose(SRID=5676;Point(1 1), 0.3)');
-- 5676
SELECT asEWKT(setSRID(pose 'Pose(Point(0 0),1)', 4326));
-- SRID=4326;Pose(POINT(0 0),1)
```

- Transformar a un identificador de referencia espacial

  `transform(pose,integer) → pose`

  `transformPipeline(pose,pipeline text,to_srid integer,is_forward bool=true) → pose`

  La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando la pose 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 de la pose de entrada se ignora y el SRID de la pose 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(pose 'SRID=4326;Pose(Point(4.35 50.85),1)', 3812), 6);
-- SRID=4326;Pose(POINT(648679.018035 671067.055638),1)
```

```
WITH test(pose, pipeline) AS (
  SELECT pose 'Pose(SRID=4326;Point(4.3525 50.846667),1)',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(pose, pipeline, 4326), pipeline,
  4326, false), 6)
FROM test;
-- SRID=4326;Pose(POINT(4.3525 50.846667),1)
```

11.1.7. Comparaciones

Los operadores de comparación (=, < y así sucesivamente) están disponibles para poses. 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 poses.

- Comparaciones tradicionales

  `pose {=, <>, <, >, <=, >=} pose`

```
SELECT pose 'Pose(Point(3 3), 0.5)' = pose 'Pose(Point(3 3), 0.5)';
-- true
SELECT pose 'Pose(Point(3 3), 0.5)' <> pose 'Pose(Point(3 3), 0.6)';
-- true
SELECT pose 'Pose(Point(3 3), 0.5)' < pose 'Pose(Point(3 3), 0.6)';
-- true
SELECT pose 'Pose(Point(3 3), 0.6)' > pose 'Pose(Point(2 2), 0.6)';
-- true
SELECT pose 'Pose(Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)' <= pose 'Pose(Point Z(2 2 2), 0.5, ↩
    0.5, 0.5, 0.5)';
-- true
SELECT pose 'Pose(Point(1 1), 0.6)' >= pose 'Pose(Point(1 1), 0.5)';
-- true
```

- ¿Son las poses aproximadamente iguales con respecto a un valor épsilon?

  `pose ~= pose → boolean`

```
SELECT pose 'Pose(SRID=5676;Point(1 1), 0.3)' ~=
  pose 'Pose(SRID=5676;Point(1 1.0000001), 0.30000001)';
-- true
```

11.1.8. Soporte de OGC GeoPose v1.0

El tipo `pose` de MobilityDB implementa el modelo de datos que subyace al estándar OGC GeoPose v1.0 (21-056r10): una `position` más una `orientation` en un marco de referencia conocido, donde la orientación es un cuaternión unitario en la convención de Hamilton o, de forma equivalente, una terna yaw / pitch / roll bajo la convención Tait-Bryan intrínseca ZYX. Esta sección agrupa la superficie SQL que expone esa interoperabilidad: E/S JSON para las clases de conformidad Basic del estándar, una función auxiliar de renormalización explícita para quienes ejecutan composiciones largas, y los accesores de ángulos de Euler que esperan los consumidores Basic-YPR.

11.1.8.1. Entrada y salida JSON

- Convertir hacia o desde la codificación JSON de OGC GeoPose v1.0 (clase de conformidad Basic-Quaternion o Basic-YPR)

`asGeoPose(pose,conformance int4=0,maxdecimaldigits int4=-1) → text`

`poseFromGeoPose(text) → pose`

El argumento `conformance` selecciona la clase de salida: `0` = Basic-Quaternion (predeterminado, sin pérdida), `1` = Basic-YPR (yaw, pitch, roll en grados, Tait-Bryan intrínseco ZYX). El argumento `maxdecimaldigits` es el número de dígitos significativos que se conservan en los números JSON; `-1` utiliza el valor predeterminado sin pérdida de json-c. La función de entrada detecta automáticamente la clase de conformidad a partir de las claves JSON (el miembro `quaternion` → Basic-Quaternion, el miembro `angles` → Basic-YPR). Las clases de conformidad Basic exigen un marco exterior geográfico; la implementación acepta el SRID 4326 (o 0, tratado como geográfico) y rechaza los SRID proyectados en el límite de la conversión.

```
SELECT asGeoPose(pose 'Pose(Point(8 47 1500), 0.7071067811865476, 0, 0,  ↩
    0.7071067811865475)', 0, 6);
-- {"position":{"lat":47,"lon":8,"h":1500},"quaternion":{"x":0,"y":0,"z":0.707107,"w ↩
    ":0.707107}}
SELECT asGeoPose(pose 'Pose(Point(8 47 1500), 0.7071067811865476, 0, 0,  ↩
    0.7071067811865475)', 1, 6);
-- {"position":{"lat":47,"lon":8,"h":1500},"angles":{"yaw":90,"pitch":0,"roll":0}}
SELECT asEWKT(poseFromGeoPose(
  '{"position":{"lat":47,"lon":8,"h":1500},"angles":{"yaw":90,"pitch":0,"roll":0}}'));
-- Pose(POINT Z (8 47 1500),0.7071067811865476,0,0,0.7071067811865475)
```

11.1.8.2. Renormalización de cuaterniones

- Devuelve una pose cuyo cuaternión de orientación se ha llevado de nuevo a la norma unitaria. Una pose 2D se devuelve sin cambios, ya que su orientación es un único ángulo que no deriva. A una pose 3D se le divide el cuaternión por su norma euclidiana, de modo que composiciones largas de SLERP (o cualquier otra vía que acumule deriva de punto flotante en `|q|`) pueden restaurarse al invariante `|q| = 1` que requiere el código de SLERP y de descomposición de Euler.

  `poseNormalize(pose) → pose`

```
SELECT asText(poseNormalize(pose 'Pose(Point(1 1 1), 0.5, 0.5, 0.5, 0.5)'));
-- Pose(POINT Z (1 1 1),0.5,0.5,0.5,0.5)
```

11.1.8.3. Accesores de ángulos de Euler

- Devuelve el ángulo yaw, pitch o roll de una pose, en radianes, bajo la convención Tait-Bryan intrínseca ZYX que exige la clase de conformidad Basic-YPR de OGC GeoPose

  `yaw(pose) → float`

  `pitch(pose) → float`

  `roll(pose) → float`

  Para una pose 2D, `yaw` devuelve la rotación almacenada `theta` —por convención, el yaw del marco del cuerpo— y `pitch` y `roll` devuelven `0`. Para una pose 3D, los tres valores provienen de la descomposición Tait-Bryan intrínseca ZYX del cuaternión de orientación. El término de pitch `asin` se acota a `[-1, 1]` para absorber la pequeña deriva numérica que pueden introducir las composiciones largas de cuaterniones.

```
SELECT yaw(pose 'Pose(Point(1 1), 0.5)');
-- 0.5
SELECT pitch(pose 'Pose(Point(1 1), 0.5)');
-- 0
SELECT roll(pose 'Pose(Point(0 0 0), 0.7071067811865476, 0, 0, 0.7071067811865475)');
-- 0
SELECT yaw(pose 'Pose(Point(0 0 0), 0.7071067811865476, 0, 0, 0.7071067811865475)');
-- 1.5707963267948966
```

- Eleva los accesores de ángulos de Euler a través de una pose temporal, devolviendo un flotante temporal (radianes) por cada instante bajo la convención Tait-Bryan intrínseca ZYX

  `yaw(tpose) → tfloat`

  `pitch(tpose) → tfloat`

  `roll(tpose) → tfloat`

```
SELECT asText(yaw(tpose '[Pose(Point(0 0), 0.0)@2000-01-01, Pose(Point(1 1), 0.5)@2000 ↩
    -01-02]'));
-- [0@2000-01-01, 0.5@2000-01-02]
```

11.1.8.4. Registro de metadatos de marcos

El estándar OGC GeoPose v1.0 distingue el *marco exterior* (la referencia global, por ejemplo, WGS-84 geográfico o ECEF) del *marco interior* (el marco del cuerpo cuya orientación es el cuaternión de la pose). Las clases de conformidad Basic exigen WGS-84 geográfico como marco exterior y un marco interior implícito de ejes de cuerpo dextrógiros; la clase Advanced admite pilas de marcos con nombre.

En MobilityDB v1, el tipo `pose` codifica el marco exterior de forma implícita mediante su SRID y utiliza el marco interior convencional de ejes de cuerpo dextrógiros. La tabla `geopose_frames` documenta esta correspondencia y siembra un registro paralelo a la tabla `pointcloud_formats` de pgPointCloud, de modo que el soporte futuro de la clase Advanced solo necesite ampliar el catálogo en lugar de rediseñarlo.

```
SELECT frame_id, authority, code, name, is_geographic
FROM geopose_frames ORDER BY frame_id;
-- 1 | EPSG | 4326 | WGS-84 geographic (lat/lon/h)                | t
-- 2 | EPSG | 4978 | WGS-84 ECEF (Earth-Centred Earth-Fixed)      | f
-- 3 | OGC  | LTP  | Local Tangent Plane (East-North-Up)         | f
-- 4 | OGC  | BODY | Right-handed body axes (default inner frame) | f
```

Tres funciones auxiliares SQL proporcionan una interfaz de consulta estable para el código que no quiere codificar de forma rígida la disposición de la tabla:

- `geopose_frame_srid(int) → int` — el SRID de PostGIS de un marco, o `NULL` si el marco es paramétrico (LTP, BODY).
- `geopose_frame_name(int) → text` — el nombre legible del marco.
- `geopose_frame_is_geographic(int) → boolean` — `true` para los marcos lat/lon/h, `false` para los cartesianos o proyectados.

Los usuarios pueden registrar marcos personalizados insertándolos en `geopose_frames`; el catálogo está marcado como una tabla de configuración, por lo que `pg_dump` preserva las filas del usuario.

11.1.8.5. Transformación rígida cuerpo↔mundo

La función `applyPose` aplica la transformación de cuerpo rígido codificada por una pose (la correspondencia *cuerpo→mundo* de OGC GeoPose) a una geometría en el marco del cuerpo, produciendo la geometría correspondiente en el marco del mundo. La transformación es

```
world = R(q) · body + p
```

donde `(p, q)` son la posición y la orientación de la pose. La forma estática toma una única pose y una geometría estática; la forma temporal eleva la transformación rígida por instante de una `tpose` a lo largo de sus instantes, produciendo una trayectoria `tgeompoint` en el marco del mundo de la geometría del cuerpo. La versión v1 admite geometrías de cuerpo de tipo punto y multipunto; las líneas y los polígonos quedan aplazados. La interpolación lineal de la trayectoria resultante es la cuerda de cada segmento, que aproxima la verdadera trayectoria de cuerpo rígido (un arco circular bajo SLERP), la misma concesión que MobilityDB ya adopta para las trayectorias espaciales.

`applyPose(geometry, pose) → geometry`

`applyPose(geometry, tpose) → tgeompoint`

```
-- A body sensor offset 1 unit along the body X axis, traced through a
-- tpose that ends 90 degrees yawed and translated to (10, 20).
SELECT asText(applyPose(ST_Point(1,0),
  tpose '[Pose(Point(0 0), 0)@2026-01-01,
          Pose(Point(10 20), 1.5707963267948966)@2026-01-02]'));
-- [POINT(1 0)@2026-01-01, POINT(10 21)@2026-01-02]
```

11.1.8.6. Transformaciones de marco entre SRID

La función `transform(pose, srid)` lleva los valores de pose de un SRID a otro. El flujo de trabajo n.º 6 del plan de GeoPose temporal amplía el comportamiento preexistente de *solo posición* con la corrección de orientación necesaria cuando el marco de origen y el de destino tienen bases distintas en el punto de la pose. La versión v1 implementa el caso canónico de OGC GeoPose —WGS-84 geográfico (EPSG:4326) ↔ WGS-84 ECEF (EPSG:4978)— utilizando la base estándar Este-Norte-Arriba en el punto geográfico como pivote de rotación. Para los pares de SRID cuya corrección de orientación aún no está definida, `transform` emite un `NOTICE` y deja pasar la orientación sin cambios.

Un viaje de ida y vuelta a través de ECEF y de regreso aterriza exactamente en la pose de entrada:

```
SELECT asEWKT(round(
  transform(transform(pose 'SRID=4326;Pose(Point(8 47 0), 1, 0, 0, 0)', 4978),
            4326), 6));
-- SRID=4326;Pose(POINT Z (8 47 0),1,0,0,0)

-- At the equator-meridian (lat=lon=0), the body identity quaternion in
-- the ECEF basis becomes the canonical ENU->ECEF rotation:
SELECT asEWKT(round(
  transform(pose 'SRID=4326;Pose(Point(0 0 0), 1, 0, 0, 0)', 4978), 6));
-- SRID=4978;Pose(POINT Z (6378137 0 0),0.5,-0.5,-0.5,-0.5)
```

La implementación también corrige un error preexistente de solo posición en el que `transform` no actualizaba el campo SRID del resultado, de modo que la pose devuelta ahora lleva el SRID de destino como se espera.

11.1.8.7. E/S JSON temporal

- Convierte una pose temporal hacia o desde una envoltura JSON TemporalGeoPose. Cada carga útil por instante es un documento GeoPose de clase Basic estrictamente válido según OGC más un miembro `validTime`; la envoltura registra la clase de conformidad, la interpolación y (para secuencias y conjuntos de secuencias) la inclusión de los límites del período. Un consumidor GeoPose ajeno a MEOS puede iterar sobre `instants[]` (o cada `sequences[].instants[]`) y consumir cada elemento como un documento GeoPose estático.

  `asGeoPose(tpose,conformance int4=0,maxdecimaldigits int4=-1) → text`

  `tposeFromGeoPose(text) → tpose`

  La forma de la envoltura es

```
{
  "type":          "TemporalGeoPose",
  "version":       "1.0",
  "conformance":   "Basic-Quaternion" | "Basic-YPR",
  "interpolation": "None" | "Discrete" | "Step" | "Linear",
  "instants":      [...]                  // for TInstant + TSequence
  "lower_inc":     true|false,            // for TSequence only
  "upper_inc":     true|false,            // for TSequence only
  "sequences":     [{...}, ...]           // for TSequenceSet only
}
```

```
SELECT asGeoPose(tpose '[Pose(Point(8 47), 0)@2026-01-01,
                        Pose(Point(9 48), 0.5)@2026-01-02]',
                 1, 4);
-- {"type":"TemporalGeoPose","version":"1.0","conformance":"Basic-YPR",
--  "interpolation":"Linear","lower_inc":true,"upper_inc":true,
--  "instants":[
--    {"position":{"lat":47,"lon":8,"h":0},
--     "angles":{"yaw":0,"pitch":0,"roll":0},
--     "validTime":"2026-01-01 00:00:00+01"},
--    {"position":{"lat":48,"lon":9,"h":0},
--     "angles":{"yaw":28.65,"pitch":0,"roll":0},
--     "validTime":"2026-01-02 00:00:00+01"}]}
```

11.1.8.8. Velocidad y velocidad angular

- Devuelve la velocidad de una pose temporal como un flotante temporal: la magnitud de la velocidad de la componente de posición (distancia recorrida por unidad de tiempo). La orientación no forma parte de este escalar; para la velocidad angular véase `angularSpeed`.

  `speed(tpose) → tfloat`

```
SELECT asText(speed(tpose '[Pose(Point(0 0), 0)@2000-01-01, Pose(Point(1 1), 0.5)@2000 ↩
    -01-02]'));
-- Interp=Step;[1.6368e-5@2000-01-01, 1.6368e-5@2000-01-02]
-- (sqrt(2) units / 86400 seconds -- units depend on the SRID's CRS)
```

- Devuelve la velocidad angular de una pose temporal como un flotante temporal con interpolación por pasos (radianes por unidad de tiempo). Para las poses 2D es el delta theta del arco más corto por segmento dividido por la duración del segmento; para las poses 3D es el ángulo de arco de SLERP `2 · acos(|q1 · q2|)` dividido por la duración del segmento. SLERP tiene por construcción velocidad angular constante a lo largo de un segmento, por lo que el resultado es constante a trozos.

  `angularSpeed(tpose) → tfloat`

```
-- 90-degree yaw rotation over one day -> pi/2 rad / 86400 s
SELECT asText(angularSpeed(tpose
  '[Pose(Point(0 0 0), 1, 0, 0, 0)@2000-01-01,
    Pose(Point(0 0 0), 0.7071067811865476, 0, 0, 0.7071067811865475)@2000-01-02]'));
-- Interp=Step;[1.8181e-5@2000-01-01, 1.8181e-5@2000-01-02]
```

11.2. Poses temporales

El tipo de pose temporal `tpose` permite representar la evolución en el tiempo de la posición de los objetos y su orientación. Como todos los tipos temporales, se presenta en tres subtipos, a saber, instante, secuencia y conjunto de secuencias. A continuación se dan ejemplos de valores de `tpose` en estos subtipos.

```
SELECT tpose 'Pose(Point(1 1), 0.5)@2001-01-01';
SELECT tpose '{Pose(Point(1 1), 0.3)@2001-01-01, Pose(Point(1 1), 0.5)@2001-01-02,
  Pose(Point(1 1), 0.5)@2001-01-03}';
SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-01, Pose(Point(1 1), 0.4)@2001-01-02,
  Pose(Point(1 1), 0.5)@2001-01-03]';
SELECT tpose '{[Pose(Point(1 1), 1)@2001-01-01, Pose(Point(2 2), 1)@2001-01-02],
  [Pose(Point(2 2), 2)@2001-01-04, Pose(Point(2 2), 3)@2001-01-05]}';
```

Al igual que para los tipos de punto temporal, el tipo de pose temporal acepta modificadores de tipo (o `typmod` en la terminología de PostgreSQL) para especificar el subtipo, la dimensionalidad del punto y/o el identificador de referencia espacial (SRID). Los valores posibles para el subtipo son `Instant`, `Sequence` y `SequenceSet` y los valores posibles para el tipo de geometría son `Point` o `PointZ`. 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.

```
SELECT asEWKT(tpose(Sequence,5676) 'SRID=5676;[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]');
-- SRID=5676;[Pose(POINT(1 1),0.2)@2001-01-01, Pose(POINT(1 1),0.5)@2001-01-03]
SELECT tpose(Sequence) 'Pose(Point(1 1), 0.2)@2001-01-01';
-- ERROR: Temporal type (Instant) does not match column type (Sequence)
SELECT tpose(PointZ) 'Pose(Point(1 1), 0.2)@2001-01-01';
-- Column has Z dimension but the tpose value does not
SELECT tpose(5676) 'Pose(Point(1 1), 0.2)@2001-01-01';
-- ERROR:  Temporal pose SRID (0) does not match column SRID (5676)
```

Los valores de pose 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 asText(tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(2 2), 0.4)@2001-01-02, Pose(Point(3 3), 0.6)@2001-01-03)');
-- [Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(3 3),0.6)@2001-01-03)
SELECT asText(tpose '{[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(2 2), 0.3)@2001-01-02, Pose(Point(2 2), 0.5)@2001-01-03),
  [Pose(Point(2 2), 0.5)@2001-01-03, Pose(Point(2 2), 0.7)@2001-01-04)}');
/* {[Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(2 2),0.3)@2001-01-02,
   Pose(Point(2 2),0.7)@2001-01-04)} */
```

11.3. Validez de las poses temporales

Los valores de pose 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 tpose 'NULL@2001-01-01 08:05:00';
SELECT tpose 'Point(0 0)@NULL';
-- El tipo base no es una pose
SELECT tpose 'Point(0 0)@2001-01-01 08:05:00';
```

A continuación damos las funciones y operadores para las poses temporales. La mayoría de las funciones y operadores para tipos temporales y tipos espaciotemporales descritos en los capítulos precedentes se pueden aplicar para los tipos de pose temporal. Por lo tanto, en las firmas de las funciones, el tipo `pose` se puede utilizar siempre que se especifiquen los tipos `base` y `spatial`, y el tipo `tpose` se puede utilizar siempre que se especifiquen los tipos `ttype` y `tpatial`. Para evitar la redundancia, a continuación solo presentamos algunos ejemplos de estas funciones y operadores para poses temporales y nos remitimos a los capítulos precedentes para explicaciones detalladas sobre ellos.

11.4. Entrada y salida

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

  `asText({tpose,tpose[]})` → `{text,text[]}`

  `asEWKT({tpose,tpose[]})` → `{text,text[]}`

```
SELECT asText(tpose 'SRID=4326;[Pose(Point(0 0),1)@2001-01-01,
  Pose(Point(1 1),2)@2001-01-02)');
-- [Pose(Point(0 0),1)@2001-01-01, Pose(Point(1 1),2)@2001-01-02)
SELECT asText(ARRAY[pose 'Pose(Point(0 0),1)', 'Pose(Point(1 1),2)']);
-- {"Pose(POINT(0 0),1)","Pose(POINT(1 1),2)"}
SELECT asEWKT(tpose 'SRID=4326;[Pose(Point(0 0),1)@2001-01-01,
```

```
  Pose(Point(1 1),2)@2001-01-02)');
-- SRID=4326;[Pose(Point(0 0),1)@2001-01-01, Pose(Point(1 1),2)@2001-01-02)
SELECT asEWKT(ARRAY[pose 'Pose(SRID=5676;Point(0 0),1)',
  'Pose(SRID=5676;Point(1 1),2)']);
-- {"Pose(SRID=5676;POINT(0 0),1)","Pose(SRID=5676;POINT(1 1),2))"}
```

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

  `asMFJSON(tpose) → text`

```
SELECT asMFJSON(tpose 'Pose(Point(1 2),0.5)@2001-01-01', 3);
/* {"type":"MovingPose","bbox":[[1,2],[1,2]],"period":{"begin":"2001-01-01T00:00:00+01",
  "end":"2001-01-01T00:00:00+01","lower_inc":true,"upper_inc":true},
  "values":[{"position":{"lat":2,"lon":1},"rotation":0.5}],
  "datetimes":["2001-01-01T00:00:00+01"],"interpolation":"None"} */
SELECT asMFJSON(tpose 'Pose(Point Z(1 2 3),1,0,0,0)@2001-01-01', 3);
/* {"type":"MovingPose","bbox":[[1,2,3],[1,2,3]],
  "period":{"begin":"2001-01-01T00:00:00+01","end":"2001-01-01T00:00:00+01",
  "lower_inc":true,"upper_inc":true},
  "values":[{"position":{"lat":2,"lon":1,"h":3},"quaternion":{"x":0,"y":0,"z":0,"w":1}}],
  "datetimes":["2001-01-01T00:00:00+01"],"interpolation":"None"} */
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB), la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o HexEWKB)

  `asBinary(tpose,endian text='') → bytea`

  `asEWKB(tpose,endian text='') → bytea`

  `asHexEWKB(tpose,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 asBinary(tpose 'Pose(Point(1 2),1)@2001-01-01');
-- \x013a0001000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000
SELECT asEWKB(tpose 'SRID=7844;Pose(Point(1 2),1)@2001-01-01');
-- \x013a0001000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000
SELECT asHexEWKB(tpose 'SRID=3812;Pose(Point(1 2),1)@2001-01-01');
-- 013A0001000000000000F03F0000000000000040000000000000F03F009C57D3C11C0000
```

- Entrada desde la representación de texto conocido (Well-Known Text o WKT) o desde la representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `tposeFromText(text) → tpose`

  `tposeFromEWKT(text) → tpose`

```
SELECT asEWKT(tposeFromText(text '[Pose(Point(1 2),1)@2001-01-01,
  Pose(Point(3 4),2)@2001-01-02]'));
-- [Pose(POINT(1 2),1)@2001-01-01, Pose(POINT(3 4),2)@2001-01-02]
SELECT asEWKT(tposeFromEWKT(text 'SRID=3812;[Pose(Point(1 2),1)@2001-01-01,
  Pose(Point(3 4),2)@2001-01-02]'));
-- SRID=3812;[Pose(Point(1 2),1)@2001-01-01, Pose(Point(3 4),2)@2001-01-02]
```

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

  `tposeFromMFJSON(bytea) → tpose`

```
SELECT asEWKT(tposeFromMFJSON(asMFJSON(tpose
  'SRID=3812;Pose(Point(1 2),0.5)@2001-01-01', 3)));
-- SRID=3812;Pose(POINT(1 2),0.5)@2001-01-01
SELECT asEWKT(tposeFromMFJSON(asMFJSON(tpose
  'SRID=5676;Pose(Point Z(1 2 3),1,0,0,0)@2001-01-01', 3)));
-- SRID=5676;Pose(POINT Z(1 2 3),1,0,0,0)@2001-01-01
```

- Entrada desde la representación binaria conocida (Well-Known Binary o WKB), desde la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o desde la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o HexEWKB)

  `tposeFromBinary(bytea) → tpose`

  `tposeFromEWKB(bytea) → tpose`

  `tposeFromHexEWKB(text) → tpose`

```
SELECT asEWKT(tposeFromBinary(
  '\x013a0001000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000'));
-- Pose(POINT(1 2),1)@2001-01-01
SELECT asEWKT(tposeFromEWKB(
  '\x013a0001000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000'));
-- SRID=7844;Pose(Point(1 1),2)@2001-01-01
SELECT asEWKT(tposeFromHexEWKB(
  '013A0041E40E0000E40E000000000000000F03F...40000000000000F03F009C57D3C11C0000'));
-- SRID=3812;Pose(POINT(1 2),1)@2001-01-01
```

11.5. Constructores

- Constructor para poses temporales con valor constante

  `tpose(pose,timestamptz) → tposeInst`

  `tpose(pose,tstzset) → tposeDiscSeq`

  `tpose(pose,tstzspan,interp='linear') → tposeContSeq`

  `tpose(pose,tstzspanset,interp='linear') → tposeSeqSet`

```
SELECT asText(tpose('Pose(Point(1 1), 0.5)', timestamptz '2001-01-01'));
-- Pose(Point(1 1),0.5)@2001-01-01
SELECT asText(tpose('Pose(Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)',
  tstzset '{2001-01-01, 2001-01-05}'));
/* {Pose(POINT Z (1 1 1),0.5,0.5,0.5,0.5)@2001-01-01,
  Pose(POINT Z (1 1 1),0.5,0.5,0.5,0.5)@2001-01-05} */
SELECT asText(tpose('Pose(Point(1 1), 0.5)',
  tstzspan '[2001-01-01, 2001-01-02]'));
-- [Pose(Point(1 1),0.5)@2001-01-01, Pose(Point(1 1),0.5)@2001-01-02]
SELECT asText(tpose('Pose(Point(1 1), 0.2)',
  tstzspanset '{[2001-01-01, 2001-01-03]}', 'step'));
-- Interp=Step;{[Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(1 1),0.2)@2001-01-03]}
```

- Constructor para poses temporales de subtipo secuencia

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

```
SELECT asText(tposeSeq(ARRAY[tpose 'Pose(Point(1 1), 0.3)@2001-01-01',
  'Pose(Point(2 2), 0.5)@2001-01-02', 'Pose(Point(1 1), 0.5)@2001-01-03']));
/* {Pose(Point(1 1),0.3)@2001-01-01, Pose(Point(2 2),0.5)@2001-01-02,
   Pose(Point(1 1),0.5)@2001-01-03} */
SELECT asText(tposeSeq(ARRAY[tpose 'Pose(Point(1 1), 0.2)@2001-01-01',
  'Pose(Point(1 1), 0.4)@2001-01-02', 'Pose(Point(1 1), 0.5)@2001-01-03']));
/* [Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(1 1),0.4)@2001-01-02,
   Pose(Point(1 1),0.5)@2001-01-03] */
```

- Constructor para poses temporales de subtipo conjunto de secuencias

  `tposeSeqSet(tpose[]) → tposeSeqSet`

  `tposeSeqSetGaps(tposeInst[],maxt=NULL,maxdist=NULL,interp='linear') → tposeSeqSet`

```
SELECT asText(tposeSeqSet(ARRAY[tpose
  '[Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(2 2),0.4)@2001-01-02]',
  '[Pose(Point(2 2),0.6)@2001-01-03, Pose(Point(2 2),0.8)@2001-01-04]']));
/* {[Pose(Point(1 1),0.2)@2001-01-01, Pose(Point(2 2),0.4)@2001-01-02],
   [Pose(Point(2 2),0.6)@2001-01-03, Pose(Point(2 2),0.6)@2001-01-04]} */
SELECT asText(tposeSeqSetGaps(ARRAY[tpose 'Pose(Point(1 1),0.1)@2001-01-01',
  'Pose(Point(1 1),0.3)@2001-01-03', 'Pose(Point(1 1),0.5)@2001-01-05'], '1 day'));
/* {[Pose(Point(1 1),0.1)@2001-01-01], [Pose(Point(1 1),0.3)@2001-01-03],
   [Pose(Point(1 1),0.5)@2001-01-05]} */
```

- Construir una pose temporal 2D a partir de un punto geométrico temporal y un flotante temporal

  `tpose(tgeompoint, tfloat) → tpose`

  El marco temporal del resultado es la intersección de los marcos temporales del punto temporal y el flotante temporal. Si no se intersecan, se devuelve un valor nulo

```
SELECT asText(tpose(tgeompoint '[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02)',
  tfloat '[1@2001-01-01, 2@2001-01-02)'));
-- [Pose(Point(1 1),1)@2001-01-01, Pose(Point(1 1),2)@2001-01-02)
SELECT asText(tpose(tgeompoint '[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02)',
  tfloat '[2@2001-01-03, 3@2001-01-04)'));
-- NULL
```

11.6. Conversión de tipos

Un valor de pose temporal se puede convertir hacia y desde un punto geométrico temporal. Esto se puede hacer utilizando un `CAST` explícito o utilizando la notación `::`. Se devuelve un valor nulo si alguno de los valores de punto geométrico que lo componen no se puede convertir en un valor `pose`.

- Convertir una pose temporal en un punto geométrico temporal

  `tpose::tgeompoint`

```
SELECT asText((tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.3)@2001-01-02)')::tgeompoint);
-- [POINT(1 1)@2001-01-01, POINT(1 1)@2001-01-02)
SELECT asEWKT((tpose '[Pose(Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)@2001-01-01,
  Pose(Point Z(2 2 2), 1, 0, 0, 0)@2001-01-02)')::tgeompoint);
-- [POINT Z (1 1 1)@2001-01-01, POINT Z (2 2 2)@2001-01-02)
```

- Convertir una pose temporal geodésica en un punto geográfico temporal

  `tpose::tgeogpoint`

```
SELECT asText((tpose '[GeodPose(Point(1 1), 0.2)@2001-01-01,
  GeodPose(Point(1 1), 0.3)@2001-01-02)')::tgeogpoint);
-- [POINT(1 1)@2001-01-01, POINT(1 1)@2001-01-02)
```

11.7. Accesores

- Devuelve los valores

  `getValues(tpose) → poseset`

```
SELECT asText(getValues(tpose '{[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02)}'));
-- {"Pose(Point(1 1),0.3)","Pose(Point(1 1),0.5)"}
SELECT asEWKT(getValues(tpose 'SRID=5676;{[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.3)@2001-01-02)}'));
-- SRID=5676;{"Pose(Point(1 1),0.3)"}
```

- Devuelve los puntos

  `points(tpose) → geomset`

```
SELECT asEWKT(points(tpose 'SRID=5676;{Pose(Point(3 3), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02}'));
-- SRID=5676;{Point(1 1), Point(3 3)}
```

- Devuelve la rotación

  `rotation(tpose2d) → tfloat`

```
SELECT rotation(tpose 'SRID=5676;{[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.3)@2001-01-02)}');
-- {[0.3@2001-01-01, 0.3@2001-01-02)}
```

- Devuelve el valor en una marca de tiempo

  `valueAtTimestamp(tpose,timestamptz) → pose`

```
SELECT asText(valueAtTimestamp(tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(3 3), 0.5)@2001-01-03)', '2001-01-02'));
-- Pose(Point(2 2),0.4)
SELECT asText(valueAtTimestamp(tpose
  '[Pose(Point Z(1 1 1), 0.5, 0.5, 0.5, 0.5)@2001-01-01,
    Pose(Point Z(3 3 3), 1, 0, 0, 0)@2001-01-03)', timestamptz '2001-01-02'), 6);
-- Pose(POINT Z (2 2 2),0.866025,0.288675,0.288675,0.288675)
```

11.8. Transformaciones

- Transformar a otro subtipo

  `tposeInst(tpose) → tposeInst`

  `tposeSeq(tpose) → tposeSeq`

  `tposeSeqSet(tpose) → tposeSeqSet`

```
SELECT asText(tposeSeq(tpose 'Pose(Point(1 1), 0.5)@2001-01-01', 'discrete'));
-- {Pose(Point(1 1),0.5)@2001-01-01}
SELECT asText(tposeSeq(tpose 'Pose(Point(1 1), 0.5)@2001-01-01'));
-- [Pose(Point(1 1),0.5)@2001-01-01]
SELECT asText(tposeSeqSet(tpose 'Pose(PointZ(1 1 1), 0.5, 0.5, 0.5, 0.5)@2001-01-01'));
-- {[Pose(POINT Z (1 1 1),0.5,0.5,0.5,0.5)@2001-01-01]}
```

- Transformar a otra interpolación

  `setInterp(tpose, interp) → tpose`

```
SELECT asText(setInterp(tpose 'Pose(Point(1 1),0.2)@2001-01-01','linear'));
-- [Pose(Point(1 1),0.2)@2001-01-01]
SELECT asText(setInterp(tpose '{[Pose(Point Z(1 1 1),1,0,0,0)@2001-01-01],
  [Pose(Point Z(2 2 2),0,0,0,1)@2001-01-02]}', 'discrete'));
-- {Pose(POINT Z (1 1 1),1,0,0,0)@2001-01-01, Pose(POINT Z (2 2 2),0,0,0,1)@2001-01-02}
```

- Redondear los puntos y las orientaciones de la pose temporal al número de posiciones decimales

  `round(tpose,integer=0) → tpose`

```
SELECT asText(round(tpose '{[Pose(Point(1 1.123456789),0.123456789)@2001-01-01,
  Pose(Point(1 1),0.5)@2001-01-02)}', 3));
-- {[Pose(POINT(1 1.123),0.123)@2001-01-01, Pose(POINT(1 1),0.5)@2001-01-02)}
```

11.9. Modificaciones

- Agregar un instante temporal o una secuencia temporal

  `appendInstant(tpose,tposeInst) → tpose`

  `appendSequence(tpose,tposeSeq) → tpose`

```
SELECT asText(appendInstant(tpose 'Pose(Point(1 1), 0.5)@2001-01-01', 'Pose(Point(2 2), 1) ↩
    @2001-01-02'));
-- [Pose(POINT(1 1),0.5)@2001-01-01, Pose(POINT(2 2),1)@2001-01-02]
SELECT asText(appendSequence(tpose '[Pose(Point(1 1), 0.5)@2001-01-01]', '[Pose(Point(2 2) ↩
    , 1)@2001-01-02]'));
-- {[Pose(POINT(1 1),0.5)@2001-01-01], [Pose(POINT(2 2),1)@2001-01-02]}
```

- Fusionar las poses temporales

  `merge(tpose,tpose) → tpose`

  `merge(tpose[]) → tpose`

  `merge(tpose) → tpose`

  `mergeAgg(tpose) → tpose`

```
SELECT asText(merge(tpose
  '[Pose(Point(1 1 1),1,0,0,0)@2001-01-01, Pose(Point(2 2 2),0,1,0,0)@2001-01-02]',
  '[Pose(Point(2 2 2),0,1,0,0)@2001-01-02, Pose(Point(3 3 3),0,0,0,1)@2001-01-03]'));
/* [Pose(POINT Z (1 1 1),1,0,0,0)@2001-01-01, Pose(POINT Z (2 2 2),0,1,0,0)@2001-01-02,
    Pose(POINT Z (3 3 3),0,0,0,1)@2001-01-03] */
SELECT asText(merge(ARRAY[tpose
  '{[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(2 2),0.2)@2001-01-02],
    [Pose(Point(3 3),0.3)@2001-01-03, Pose(Point(4 4),0.4)@2001-01-04]}',
  '[Pose(Point(4 4),0.4)@2001-01-04, Pose(Point(5 5),0.5)@2001-01-05]']));
/* {[Pose(POINT(1 1),0.1)@2001-01-01, Pose(POINT(2 2),0.2)@2001-01-02],
    [Pose(POINT(3 3),0.3)@2001-01-03, Pose(POINT(5 5),0.5)@2001-01-05]} */
WITH temp(inst) AS (
  SELECT tpose 'Pose(Point(1 1),0.1)@2001-01-01' UNION
  SELECT tpose 'Pose(Point(2 2),0.2)@2001-01-02' UNION
  SELECT tpose 'Pose(Point(3 3),0.3)@2001-01-03' )
SELECT asText(mergeAgg(inst)) FROM temp;
/* {Pose(POINT(1 1),0.1)@2001-01-01, Pose(POINT(2 2),0.2)@2001-01-02,
    Pose(POINT(3 3),0.3)@2001-01-03} */
```

11.10. Restricciones

- Restringir al (complemento de) un valor o un conjunto de valores

  `atValue(tpose,value) → tpose`

  `minusValue(tpose,value) → tpose`

  `atValues(tpose,valueset) → tpose`

  `minusValues(tpose,valueset) → tpose`

```
SELECT atValue(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-03]', 'Pose(Point(2 2), 0.5)');
-- {[Pose(Point(2 2),0.5)@2001-01-02]}
SELECT minusValue(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-03]', 'Pose(Point(2 2), 0.5)');
/* {[Pose(Point(2 2),0.3)@2001-01-01, Pose(Point(2 2),0.5)@2001-01-02),
   (Pose(Point(2 2),0.5)@2001-01-02, Pose(Point(2 2),0.7)@2001-01-03]} */
```

- TODO Restringir al (complemento de) una geometría

  `atGeometry(tpose,geometry) → tpose`

  `minusGeometry(tpose,geometry) → tpose`

```
SELECT atGeometry(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
SELECT minusGeometry(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-03]',
  'Polygon((40 40,40 50,50 50,50 40,40 40))');
/* {(Pose(Point(2 2),0.342593)@2001-01-01 05:06:40.364673+01,
   Pose(Point(2 2),0.7)@2001-01-03]} */
```

11.11. Sistema de referencia espacial

- Devuelve o establece el identificador de referencia espacial

  `SRID(tpose) → integer`

  `setSRID(tpose) → tpose`

```
SELECT SRID(tpose 'SRID=5676;[Pose(Point(0 0),1)@2001-01-01,
  Pose(Point(1 1),2)@2001-01-02)'));
-- 5676
SELECT asEWKT(setSRID(tpose '[Pose(Point(0 0),1)@2001-01-01,
  Pose(Point(1 1),2)@2001-01-02)', 5676));
-- SRID=5676;[Pose(POINT(0 0),1)@2001-01-01, Pose(POINT(1 1),2)@2001-01-02)
```

- Transformar a un identificador de referencia espacial

  `transform(tpose,integer) → tpose`

  `transformPipeline(tpose,pipeline text,to_srid integer,is_forward bool=true) → tpose`

```
SELECT asEWKT(transform(tpose 'SRID=4326;Pose(Point(4.35 50.85),1)@2001-01-01',
  3812));
-- SRID=3812;Pose(POINT(648679.0180353033 671067.0556381135),1)@2001-01-01
```

```
WITH test(tpose, pipeline) AS (
  SELECT tpose 'SRID=4326;{Pose(Point(4.3525 50.846667),1)@2001-01-01,
    Pose(Point(-0.1275 51.507222),2)@2001-01-02}',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(tpose, pipeline, 4326),  pipeline,
  4326, false), 6)
FROM test;
/* SRID=4326;{Pose(POINT(4.3525 50.846667),1)@2001-01-01,
   Pose(POINT(-0.1275 51.507222),2)@2001-01-02} */
```

11.12. Operaciones de cuadro delimitador

- Operadores topológicos

  `{tstzspan,stbox,tpose} {&&, <@, @>, ~=, -|-} {tstzspan,stbox,tpose} → boolean`

```
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' && tstzspan '[2001-01-02,2001-01-04]';
-- true
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02]' @> stbox(pose 'Pose(Point(1 1), 0.5)');
-- true
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' ~= tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.35)@2001-01-02, Pose(Point(1 1), 0.5)@2001-01-03]';
-- true
```

- Operadores de posición

  `{stbox,tpose} {<<, &<, >>, &>} {stbox,tpose} → boolean`

  `{stbox,tpose} {<<|, &<|, |>>, |&>} {stbox,tpose} → boolean`

  `{stbox,tpose} {<</, &</, />>, /&>} {stbox,tpose} → boolean`

  `{tstzspan,stbox,tpose} {<<#, &<#, #>>, #&>} {tstzspan,stbox,tpose} → boolean`

```
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02]' << stbox(pose 'Pose(Point(2 2), 0.5)');
-- true
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02]' <<| stbox(pose 'Pose(Point(2 2), 0.5)');
-- true
SELECT tpose '[Pose(Point(1 1 1), 1,0,0,0)@2001-01-01,
  Pose(Point(1 1 1), 1,0,0,0)@2001-01-02]' <<| stbox(pose 'Pose(Point(2 2 2), 1,0,0,0)');
-- true
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-02]' <<# tstzspan '[2001-01-03, 2001-01-04]';
-- true
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-03,
  Pose(Point(1 1), 0.5)@2001-01-05]' #>>
  tpose '[Pose(Point(1 1), 0.3)@2001-01-01, Pose(Point(1 1), 0.5)@2001-01-02]';
-- true
```

11.13. Operaciones de distancia

A continuación presentamos las funciones que calculan operaciones de distancia para poses temporales. Tenga en cuenta que para estas operaciones solo se considera el componente de punto, no la orientación.

- Devuelve la distancia más pequeña alguna vez

  `{geo,pose,tpose} |=| {geo,pose,tpose} → float`

```
SELECT tpose '[Pose(Point(2 2), 0.3)@2001-01-01, Pose(Point(2 2), 0.7)@2001-01-02]' |=|
  geometry 'Linestring(50 50,55 55)';
-- 67.88225099390856
SELECT tpose '[Pose(Point(2 2), 0.3)@2001-01-01, Pose(Point(2 2), 0.7)@2001-01-02]' |=|
  pose 'Pose(Point(1 1), 0.5)';
-- 1.4142135623730951
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' |=| pose 'Pose(Point(2 2), 0.2)';
```

```
-- 1.4142135623730951
SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' |=| geometry 'Linestring(2 2,2 1,3 1)';
-- 1
```

El operador `|=|` se puede utilizar para realizar búsquedas del vecino más cercano utilizando un índice GiST o SP-GiST (ver Sección 10.2).

- Devuelve el instante de la primera pose temporal en el que los dos argumentos están a la distancia más cercana

  `nearestApproachInstant({geo,pose,tpose},{geo,pose,tpose}) → tpose`

```
SELECT asText(nearestApproachInstant(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)'));
-- Pose(POINT(2 2),0.3)@2001-01-01
SELECT asText(nearestApproachInstant(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-02]', pose 'Pose(Point(1 1), 0.5)'));
-- Pose(POINT(2 2),0.3)@2001-01-01
```

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

  `shortestLine({geo,pose,tpose},{geo,pose,tpose}) → geometry`

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

```
SELECT ST_AsText(shortestLine(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)'));
-- LINESTRING(2 2,50 50)
SELECT ST_AsText(shortestLine(tpose '[Pose(Point(2 2), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.7)@2001-01-02]', pose 'Pose(Point(1 1), 0.5)'));
-- LINESTRING(2 2,1 1)
```

- Devuelve la distancia temporal

  `tpose <-> tpose → tfloat`

```
SELECT round(tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' <-> pose 'Pose(Point(2 2), 0.2)', 6);
-- [1.414214@2001-01-01, 1.414214@2001-01-03]
SELECT round(tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' <-> geometry 'Point(50 50)', 6);
-- [69.296465@2001-01-01, 69.296465@2001-01-03]
SELECT round(tpose '[Pose(Point(1 1), 0.3)@2001-01-01,
  Pose(Point(2 2), 0.5)@2001-01-03]' <->
  tpose '[Pose(Point(1 2), 0.3)@2001-01-02, Pose(Point(2 1), 0.5)@2001-01-04]', 6);
-- [0.707107@2001-01-02, 0.5@2001-01-02 12:00:00+01, 0.707107@2001-01-03]
```

11.14. Relaciones espaciales

Las relaciones topológicas y de distancia descritas en la Sección 8.5 tales como `eIntersects`, `aDwithin` o `tContains` solo consideran la ubicación de las geometrías, no su orientación. Para poder aplicar estas relaciones espaciales a las poses temporales, se deben transformar a puntos geométricos temporales. Esto se puede realizar fácilmente utilizando las funciones `geometry` y `tgeompoint` o utilizando un casting explícito `::` como se muestra en los ejemplos a continuación.

- Relaciones alguna vez y siempre

```
SELECT eContains(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tgeompoint(tpose '[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(1 1),0.3)@2001-01-03)'));
-- true
SELECT eDisjoint(geometry(pose 'Pose(Point(2 2), 0.0)'),
  tgeompoint(tpose '[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(1 1),0.3)@2001-01-03)'));
```

```
-- true
SELECT eIntersects(
  tpose '[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(1 1),0.3)@2001-01-03)'::tgeompoint,
  tpose '[Pose(Point(2 2),0.0)@2001-01-01, Pose(Point(2 2),1)@2001-01-03)'::tgeompoint);
-- false
```

- Relaciones espaciotemporales

```
SELECT tContains(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tgeompoint(tpose '[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(1 1),0.3)@2001-01-03)'));
-- {[t@2001-01-01 00:00:00+01, t@2001-01-03 00:00:00+01)}
SELECT tDisjoint(geometry(pose 'Pose(Point(2 2), 0.0)'),
  tgeompoint(tpose '[Pose(Point(1 1),0.1)@2001-01-01, Pose(Point(1 1),0.3)@2001-01-03)'));
-- [t@2001-01-01, t@2001-01-03)
SELECT tDwithin(
 tpose '[Pose(Point(1 1),0.3)@2001-01-01,Pose(Point(1 1),0.5)@2001-01-03)'::tgeompoint,
 tpose '[Pose(Point(1 1),0.5)@2001-01-01,Pose(Point(3 3),0.3)@2001-01-03)'::tgeompoint,1);
/* {[t@2001-01-01 00:00:00+01, t@2001-01-01 16:58:14.025894+01],
  (f@2001-01-01 16:58:14.025894+01, f@2001-01-03 00:00:00+01)} */
```

11.15. Comparaciones

- Comparaciones tradicionales

`tpose {=, <>, <, >, <=, >=} tpose → boolean`

```
SELECT tpose '{[Pose(Point(1 1), 0.1)@2001-01-01,
  Pose(Point(1 1), 0.3)@2001-01-02),
  [Pose(Point(1 1), 0.3)@2001-01-02, Pose(Point(1 1), 0.5)@2001-01-03]}' =
  tpose '[Pose(Point(1 1), 0.1)@2001-01-01, Pose(Point(1 1), 0.5)@2001-01-03]';
-- true
SELECT tpose '{[Pose(Point(1 1), 0.1)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]}' <>
  tpose '[Pose(Point(1 1), 0.1)@2001-01-01, Pose(Point(1 1), 0.5)@2001-01-03]';
-- false
SELECT tpose '[Pose(Point(1 1), 0.1)@2001-01-01,
  Pose(Point(1 1), 0.5)@2001-01-03]' <
  tpose '[Pose(Point(1 1), 0.1)@2001-01-01, Pose(Point(1 1), 0.6)@2001-01-03]';
-- true
```

- Comparaciones alguna vez y siempre

`tpose {?=, %=} tpose → boolean`

```
SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.4)@2001-01-04)' ?= Pose(Point(1 1), 0.3);
-- true
SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.2)@2001-01-04)' &= Pose(Point(1 1), 0.2);
-- true
```

- Comparaciones temporales

`tpose {#=, #<>} tpose → tbool`

```
SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.4)@2001-01-03)' #= pose 'Pose(Point(1 1), 0.3)';
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-01,
  Pose(Point(1 1), 0.8)@2001-01-03)' #<>
  tpose '[Pose(Point(1 1), 0.3)@2001-01-01, Pose(Point(1 1), 0.7)@2001-01-03)';
-- {[t@2001-01-01, f@2001-01-02], (t@2001-01-02, t@2001-01-03)}
```

11.16. Agregaciones

Las tres funciones de agregación para poses temporales se ilustran a continuación.

- Conteo temporal

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

```
WITH Temp(temp) AS (
  SELECT tpose '[Pose(Point(1 1), 0.1)@2001-01-01,
    Pose(Point(1 1), 0.3)@2001-01-03)' UNION
  SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-02,
    Pose(Point(1 1), 0.4)@2001-01-04)' UNION
  SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-03,
    Pose(Point(1 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(tpose) → {tintSeq,tintSeqSet}`

```
WITH Temp(temp) AS (
  SELECT tpose '[Pose(Point(1 1), 0.1)@2001-01-01,
    Pose(Point(1 1), 0.3)@2001-01-03)' UNION
  SELECT tpose '[Pose(Point(1 1), 0.2)@2001-01-02,
    Pose(Point(1 1), 0.4)@2001-01-04)' UNION
  SELECT tpose '[Pose(Point(1 1), 0.3)@2001-01-03,
    Pose(Point(1 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)} */
```

11.17. Indexación

Se pueden crear índices GiST y SP-GiST para columnas de tablas de poses temporales. Un ejemplo de creación de índice es el siguiente:

```
CREATE INDEX Trips_Trip_SPGist_Idx ON Trips USING SPGist(Trip);
```

Los índices GiST y SP-GiST almacenan el cuadro delimitador para las poses temporales, que es un `stbox`, como todos los tipos espaciotemporales.

Un índice GiST o SP-GiST puede acelerar las consultas que involucran los siguientes operadores:

- `<<`, `&<`, `&>`, `>>`, `<<|`, `&<|`, `|&>`, `|>>`, que solo consideran la dimensión espacial en las poses temporales,
- `<<#`, `&<#`, `#&>`, `#>>`, que solo consideran la dimensión temporal en las poses temporales,
- `&&`, `@>`, `<@`, `~=`, `-|-`, y `|=|` , que consideran tantas dimensiones como las que comparten la columna indexada y el argumento de la consulta.

Estos operadores trabajan con cuadros delimitadores, no con los valores completos.

Capítulo 12

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`.

La guía operativa sobre cómo elegir entre `tnpoint` y `tgeompoint`; el contrato de herencia del SRID (el SRID se lee de la ruta registrada en `ways`, no se lleva explícitamente en el valor); los peligros numéricos que las cargas de trabajo de producción deben anticipar (la limitación del parámetro de posición a `[0, 1]`, el manejo de SRID mixtos, los errores de ruta no registrada, las operaciones entre redes); y las convenciones de durabilidad y almacenamiento se recogen en Sección 12.19.

12.1. 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.

12.1.1. Constructores

- Constructores de puntos o segmentos de red

  `npoint(bigint,double precision) → npoint`

  `nsegment(bigint,double precision,double precision) → nsegment`

```
SELECT npoint(76, 0.3);
SELECT nsegment(76, 0.3, 0.5);
```

12.1.2. Conversión de tipos

Los valores de los tipos `npoint` y `nsegment` se pueden convertir al tipo `geometry` utilizando un `CAST` explícito o utilizando la notación `::` como se muestra a continuación. De manera similar, los valores `geometry` del subtipo `point` o `linestring` (restringidos a dos puntos) se pueden convertir, respectivamente, a valores `npoint` y `nsegment`. Para ello, se debe encontrar la ruta que interseca los puntos dados, donde se supone una tolerancia de 0,00001 unidades (según el sistema de coordenadas), por lo que se considera que un punto y una ruta que están cerca se intersecan. Si no se encuentra dicha ruta, se devuelve un valor nulo.

- Convertir entre un punto o un segmento de red y una geometría

  `{npoint,nsegment}::geometry`

  `geometry::{npoint,nsegment}`

```
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)
```

```
SELECT geometry 'SRID=5676;Point(279.269156511873 811.497076880187)'::npoint;
-- NPoint(3,0.781413)
SELECT geometry 'SRID=5676;LINESTRING(406.729536784738 702.58583437902,
  383.570801314823 845.137059419277)'::nsegment;
-- NSegment(3,0.6,0.9)
SELECT geometry 'SRID=5676;Point(279.3 811.5)'::npoint;
-- NULL
SELECT geometry 'SRID=5676;LINESTRING(406.7 702.6,383.6 845.1)'::nsegment;
-- NULL
```

12.1.3. Accesores

- Devuelve 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
```

- Devuelve la posición

  `getPosition(npoint) → float`

```
SELECT getPosition(npoint 'Npoint(63, 0.3)');
-- 0.3
```

- Devuelve 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
```

12.1.4. 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)
```

12.1.5. Operaciones espaciales

- Devuelve 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
```

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
```

12.1.6. 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.

- Comparaciones tradicionales

  `npoint {=, <>, <, >, <=, >=} npoint`

  `nsegment {=, <>, <, >, <=, >=} nsegment`

```
SELECT npoint 'Npoint(3, 0.5)' = npoint 'Npoint(3, 0.5)';
-- true
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, 0.6)';
-- true
SELECT nsegment 'Nsegment(3, 0.5, 0.5)' > nsegment 'Nsegment(2, 0.5, 0.5)';
-- true
SELECT npoint 'Npoint(1, 0.5)' <= npoint 'Npoint(2, 0.5)';
-- true
SELECT npoint 'Npoint(1, 0.6)' >= npoint 'Npoint(1, 0.5)';
-- true
```

12.2. 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)}
```

12.3. 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)';
```

Damos a continuación las funciones y operadores 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.

12.4. 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]}
```

12.5. 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 entre un punto de red temporal y un punto de geometría temporal

  `tnpoint::tgeompoint`

  `tgeompoint::tnpoint`

```
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) */
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
```

12.6. Entrada y Salida MF-JSON

Un punto de red temporal se escribe y se lee en Moving-Features JSON. Cada instante representa su punto de red como `{"route":<route>,"position":<position>}` — el id de ruta como un número JSON y la posición como un float en `[0, 1]` — coincidiendo con la superficie de accesores SQL (`route`, `getPosition`).

- Genera un punto de red temporal como MF-JSON y lo lee de vuelta; el ciclo de ida y vuelta es sin pérdida. La salida toma los argumentos de opciones, banderas y dígitos decimales compartidos con los demás tipos temporales.

  `asMFJSON(tnpoint,integer,integer,integer)` → `text`

  `tnpointFromMFJSON(text)` → `tnpoint`

```
SELECT tnpointFromMFJSON(asMFJSON(tnpoint 'NPoint(1, 0.5)@2001-01-01'))
  = tnpoint 'NPoint(1, 0.5)@2001-01-01';
-- true
```

12.7. Accessores

- Devuelve 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)"}
```

- Devuelve 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}
```

- Devuelve 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)
```

- Devuelve 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
```

- Devuelve 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]} */
```

- Devuelve 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] */
```

12.8. Transformaciones

- 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)}
```

12.9. Modificaciones

- Agregar un instante temporal o una secuencia temporal

  `appendInstant(tnpoint,tnpointInst) → tnpoint`

  `appendSequence(tnpoint,tnpointSeq) → tnpoint`

```
SELECT asText(appendInstant(tnpoint 'Npoint(1, 0.1)@2001-01-01', 'Npoint(1, 0.2)@2001 ↩
    -01-02'));
-- [NPoint(1,0.1)@2001-01-01 00:00:00+01, NPoint(1,0.2)@2001-01-02 00:00:00+01]
SELECT asText(appendSequence(tnpoint '[Npoint(1, 0.1)@2001-01-01]', '[Npoint(1, 0.2)@2001 ↩
    -01-02]'));
-- {[NPoint(1,0.1)@2001-01-01 00:00:00+01], [NPoint(1,0.2)@2001-01-02 00:00:00+01]}
```

- Fusionar los puntos de red temporales

  `merge(tnpoint,tnpoint) → tnpoint`

  `merge(tnpoint[]) → tnpoint`

  `mergeAgg(tnpoint) → tnpoint`

```
SELECT asText(merge(tnpoint '[Npoint(1, 0.1)@2001-01-01, Npoint(1, 0.3)@2001-01-03]',
  '[Npoint(1, 0.3)@2001-01-03, Npoint(1, 0.5)@2001-01-05]'));
-- [NPoint(1,0.1)@2001-01-01 00:00:00+01, NPoint(1,0.5)@2001-01-05 00:00:00+01]
SELECT asText(merge(ARRAY[tnpoint '[Npoint(1, 0.1)@2001-01-01, Npoint(1, 0.2)@2001-01-02]' ↩
    ,
  '[Npoint(1, 0.2)@2001-01-02, Npoint(1, 0.3)@2001-01-03]']));
-- [NPoint(1,0.1)@2001-01-01 00:00:00+01, NPoint(1,0.3)@2001-01-03 00:00:00+01]
WITH temp(inst) AS (
  SELECT tnpoint 'Npoint(1, 0.1)@2001-01-01' UNION
  SELECT tnpoint 'Npoint(1, 0.2)@2001-01-02' UNION
  SELECT tnpoint 'Npoint(1, 0.3)@2001-01-03' )
SELECT asText(mergeAgg(inst)) FROM temp;
-- {NPoint(1,0.1)@2001-01-01 00:00:00+01, NPoint(1,0.2)@2001-01-02 00:00:00+01, NPoint ↩
    (1,0.3)@2001-01-03 00:00:00+01}
```

12.10. Restricciones

- Restringir al (complemento de) un valor o un conjunto de valores

  `atValue(tnpoint,value) → tnpoint`

`minusValue(tnpoint,value) → tnpoint`

`atValues(tnpoint,valueset) → tnpoint`

`minusValues(tnpoint,valueset) → 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 al (complemento de) 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]} */
```

12.11. Operaciones de distancia

- Devuelve la distancia más cercana

  `{geo,npoint,tnpoint} |=| {geo,npoint,tnpoint} → float`

  El operador `|=|` se puede utilizar para realizar búsquedas más cercanas utilizando un índice GiST o SP-GIST (ver Sección 10.2).

```
SELECT tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-02]' |=|
   geometry 'SRID=5676;Linestring(50 50,55 55)';
-- 31.69220882252415
SELECT tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-02]' |=|
  npoint 'NPoint(1, 0.5)';
-- 19.49691305292373
SELECT tnpoint '[NPoint(2, 0.3)@2001-01-01, NPoint(2, 0.7)@2001-01-02]' |=|
  tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.7)@2001-01-02]';
-- 5.231180723735304
```

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

  `nearestApproachInstant({geo,npoint,tnpoint},{geo,npoint,tnpoint}) → tnpoint`

```
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
```

- Devuelve la línea que conecta el punto de aproximación más cercano entre los dos argumentos

  `shortestLine({geo,npoint,tpoint},{geo,npoint,tpoint}) → geometry`

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

```
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)
```

- Devuelve 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)';
-- [2.349883008750 63@2001-01-02 00:00:00+01, 2.349883008750 63@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]'
-- [2.349883008750 63@2001-01-02 00:00:00+01, 2.349883008750 63@2001-01-03 00:00:00+01]
```

12.12. Operaciones espaciales

- Devuelve 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)
```

12.13. Operaciones de cuadro delimitador

- Operadores topológicos

  `{tstzspan,stbox,tnpoint} {&&, <@, @>, ~=, -|-} {tstzspan,stbox,tnpoint} → boolean`

```
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' &&
  tstzspan '[2001-01-02,2001-01-03]';
-- 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 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
```

- Operadores de posición

  `{stbox,tnpoint} {<<, &<, >>, &>} {stbox,tnpoint} → boolean`

  `{stbox,tnpoint} {<<|, &<|, |>>, |&>} {stbox,tnpoint} → boolean`

  `{tstzspan,stbox,tnpoint} {<<#, &<#, #>>, #&>} {tstzspan,stbox,tnpoint} → boolean`

```
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.5)@2001-01-02]' <<
  stbox(npoint 'NPoint(1, 0.2)');
-- true
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)@2001-01-03, NPoint(1, 0.5)@2001-01-05]' #&>
  tstzspan '[2001-01-01,2001-01-03]';
-- true
SELECT tnpoint '[NPoint(1, 0.3)@2001-01-03, NPoint(1, 0.3)@2001-01-05]' #>>
  tnpoint '[NPoint(1, 0.3)@2001-01-01, NPoint(1, 0.3)@2001-01-02]';
-- true
```

12.14. Relaciones espaciales

Las relaciones topológicas y de distancia descritas en Sección 8.5, como `eIntersects`, `aDwithin` o `tContains`, se definen en el espacio geográfico, mientras que los puntos de la red temporal se definen en el espacio de la red. Para aplicar estas relaciones espaciales a los puntos de la red temporal, estos deben transformarse en puntos geométricos temporales. Esto se puede realizar fácilmente mediante las funciones `geometry` y `tgeompoint` o mediante una conversión explícita de valores con `::`, como se muestra en los ejemplos a continuación.

- Relaciones alguna vez y siempre

```
SELECT eContains(geometry 'SRID=5676;Polygon((0 0,0 50,50 50,50 0,0 0))',
  tgeompoint(tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)'));
-- false
SELECT eDisjoint(geometry(npoint 'NPoint(2, 0.0)'),
  tgeompoint(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)'::tgeompoint,
  tnpoint '[NPoint(2, 0.0)@2001-01-01, NPoint(2, 1)@2001-01-03)'::tgeompoint);
-- false
```

- Relaciones espaciotemporales

```
SELECT tContains(geometry 'SRID=5676;Polygon((0 0,0 50,50 50,50 0,0 0))',
  tgeompoint(tnpoint '[NPoint(1, 0.1)@2001-01-01, NPoint(1, 0.3)@2001-01-03)'));
-- [f@2001-01-01 00:00:00+01, f@2001-01-03 00:00:00+01)
SELECT tDisjoint(geometry(npoint 'NPoint(2, 0.0)'),
  tgeompoint(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)'::tgeompoint,
  tnpoint '[NPoint(1, 0.5)@2001-01-01, NPoint(1, 0.3)@2001-01-03)'::tgeompoint, 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)} */
```

12.15. Comparaciones

- Comparaciones tradicionales

```
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
```

- Comparaciones alguna vez y siempre

  `{npoint,tnpoint} {?=,%=} {npoint,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
```

- Comparaciones temporales

  `{npoint,tnpoint} {#=, #<>} {npoint,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)}
```

12.16. 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)} */
```

12.17. 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.

12.18. Agregación

- Devuelve el cuadro delimitador espaciotemporal que cubre cada posición materializada en un conjunto de puntos de red. El agregado es polimórfico con los agregados `extent` de los demás tipos temporales espaciales, por lo que una sola consulta puede calcular un único cuadro a través de un conjunto heterogéneo de columnas `tgeompoint`, `tgeometry` y `tnpoint`. La extensión espacial de cada instante se resuelve a través del registro de ways, lo que domina el coste sobre una entrada grande.

  `extent(tnpoint) → stbox`

```
SELECT extent(temp) FROM tbl_tnpoint;
```

12.19. Guía de producción

Esta sección recoge el conocimiento operativo que las cargas de trabajo de producción necesitan además de la referencia de funciones: cuándo `npoint` / `tnpoint` son el tipo adecuado, cómo fluyen los SRID desde el registro de la red, los peligros numéricos que afectan a las cargas de trabajo de seguimiento restringidas a una red, y la historia de durabilidad de los valores serializados. La mayoría de estos puntos aparecen por separado a lo largo del capítulo; el objetivo aquí es un único lugar que un nuevo contribuyente pueda leer de principio a fin antes de instrumentar una aplicación.

12.19.1. Cuándo usar tnpoint frente a tgeompoint

Un `tgeompoint` rastrea una posición en el espacio libre; un `tnpoint` rastrea una posición restringida a una red conocida de rutas. Cada valor de punto de red es un par `(rid, pos)` donde `rid` es el identificador de ruta y `pos` es la posición relativa a lo largo de esa ruta, normalizada a `[0, 1]` (del inicio de la ruta al final de la ruta).

Use `tgeompoint` cuando el objeto en movimiento no esté restringido a una red, cuando la topología de la red sea desconocida, o cuando solo importe la geometría de la trayectoria — trazas GPS de vehículos a nivel de datos crudos, movimiento de animales, flujos AIS.

Use `tnpoint` cuando el movimiento esté sobre una red conocida y se desee semántica relativa a la red: distancias medidas a lo largo de la ruta, no en el espacio libre; agregaciones agrupadas por segmento de ruta; uniones espaciales restringidas a la "misma red". Las cargas de trabajo canónicas son el seguimiento de redes ferroviarias y de carreteras, donde los flujos emparejados con el mapa se convierten en valores `tnpoint`.

La componente de posición de un `tnpoint` se puede recuperar como un `tgeompoint` mediante la conversión estándar (o como una `geometry` mediante `geometry(npoint)`); una `geometry` se puede proyectar de vuelta sobre la red cargada con `npoint(geometry)`. En consecuencia, una sola columna `tnpoint` subsume una columna `tgeompoint` emparejada con el mapa cuando se necesita semántica relativa a la red.

12.19.2. Herencia del SRID desde el registro de la red

Un valor `npoint` no lleva su propio SRID; hereda el SRID de la ruta referenciada por su campo `rid`. El conjunto de rutas conocidas — el registro *ways* — se carga en una caché por proceso (véase `meos/src/npoint/ways_meos.c`) la primera vez que se ejecuta una función de red, y está indexado por el gid de ruta. Todo accesor de `npoint` que necesite la geometría de la ruta (`npoint_to_geompoint`, `npoint_to_stbox`, todos los predicados espaciales y funciones de distancia) consulta esta caché.

En la extensión de PostgreSQL la caché se rellena automáticamente desde la tabla `public.ways`. En MEOS independiente la aplicación debe llamar a `meos_register_ways(...)` con la tabla de rutas antes de cualquier operación `npoint`; `meos_finalize_ways()` es idempotente y se puede llamar varias veces de forma segura. Las operaciones entre tipos entre un `npoint` y una geometría validan que el SRID de la ruta coincida con el SRID de la geometría; las discrepancias generan un error explícito en lugar de reproyectar silenciosamente.

12.19.3. Peligros numéricos

Cuatro peligros afectan de forma fiable a las cargas de trabajo reales de seguimiento restringidas a una red. La implementación mitiga cada uno como sigue:

- *Limitación del parámetro de posición a [0, 1]*. Una posición fuera de `[0, 1]` no tiene significado en una ruta que no se ha extendido.

  Mitigación: `npoint_make` rechaza `pos < 0` o `pos > 1` en la construcción con el mensaje "The relative position must be a real number between 0 and 1", y la misma comprobación se aplica en las entradas de los analizadores WKT, WKB y MFJSON. Los extremos `0.0` (inicio de ruta) y `1.0` (fin de ruta) son válidos y admitidos — el conjunto de pruebas ejercita ambos. Los datos de flujo corruptos o sin recortar fallan de forma explícita en lugar de contaminar una consulta posterior.

- *Manejo de SRID mixtos*. Un `tnpoint` hereda su SRID de su ruta, mientras que un argumento `geometry` lleva su SRID explícitamente.

  Mitigación: cada operador entre tipos (`tnpoint <-> geometry`, `tnpoint && geometry`, `distance(tnpoint, geometry)`) valida que los dos SRID coincidan mediante `ensure_same_srid`; las discrepancias generan un error explícito en lugar de reproyectar silenciosamente un lado sobre el otro.

- *Ruta no registrada*. Un `npoint` que referencia un gid de ruta no presente en la caché de ways no puede resolverse a una geometría.

  Mitigación: `npoint_make` llama a `ensure_route_exists` en la construcción; los constructores SQL WKT, WKB, MFJSON y `npoint(...)` fluyen todos por esta comprobación y generan "There is no route with gid value <n> in table ways" si no se encuentra. Los usuarios de MEOS independiente que omitan el arranque `meos_register_ways(...)` verán este error en la primera operación de red, no más tarde como una consulta silenciosa de resultado cero.

- *Operaciones entre redes*. Dos valores `tnpoint` cuyas rutas pertenecen a redes diferentes (o a componentes conexas diferentes de la misma red) no pueden relacionarse mediante distancia de red.

  Mitigación: los operadores de distancia y de relación espacial entre dos valores `tnpoint` bajan primero ambos lados a `tgeompoint` y calculan el resultado como distancia / topología en el espacio libre. Este es un comportamiento documentado: quienes necesiten distancia de red verdadera deben restringir su consulta a una sola componente conexa aguas arriba.

12.19.4. Durabilidad y almacenamiento

La representación en disco de un `npoint` es una tupla de 16 bytes: un id de ruta de 8 bytes seguido de una posición de 8 bytes. La disposición de bytes es estable: `asBinary(tnpoint)` / `tnpointFromBinary(bytea)` realizan un ciclo de ida y vuelta que preserva el valor bit a bit, y las serializaciones WKB / HexEWKB siguen el patrón estándar de los tipos temporales de bandera de endianidad seguida de la carga útil. El par `asMFJSON(tnpoint)` / `tnpointFromMFJSON(text)` es bidireccional, por lo que MovingFeatures-JSON es un formato de intercambio viable en igualdad de condiciones con WKB. Cada instante se representa como `{"route":<route>,"position":<position>}`, con los nombres de campo coincidiendo con la superficie de accesores SQL (`route`, `getPosition`) según la convención de descriptores en lenguaje natural.

El id de ruta se emite como un número JSON, que realiza un ciclo de ida y vuelta sin pérdida entre los codificadores y decodificadores de MEOS (ambos conservan la precisión completa de `int64` mediante `INT64_FORMAT` / `strtoll`). Sin embargo, no es sin pérdida en consumidores JavaScript / ECMAScript: el tipo `Number` tiene una mantisa de 53 bits, por lo que los ids de ruta más allá de `2^53` (~9 × 10^15) pierden precisión cuando los analiza un `JSON.parse` genérico del lado del navegador. Para redes derivadas de OSM esto importa en la práctica — los ids de vía de OSM cruzaron `2^53` en 2025 — y la mitigación recomendada del lado de JS es analizar con una biblioteca que materialice los números como `BigInt` (por ejemplo `json-bigint`) en lugar de confiar en el comportamiento por defecto. Esta es una decisión de diseño deliberada, no un descuido: la alternativa (emitir la ruta como una cadena JSON, como hace `th3index` para su carga útil de celda int64) divergiría de la forma JSON que devuelven todos los accesores numéricos de este tipo y rompería el ciclo de ida y vuelta con el análisis ingenuo de enteros en cada consumidor que *sí* maneja correctamente los enteros de 64 bits.

La salida de `pg_dump` de una tabla que contiene columnas `npoint` o `tnpoint` usa la representación WKT por defecto; `pg_dump --binary-upgrade` usa la representación WKB. Ambas son estables en el ciclo de ida y vuelta, pero un volcado solo se puede restaurar en una base de datos cuya tabla `public.ways` contenga todos los ids de ruta referenciados: el registro de rutas es lógicamente una clave foránea en cada columna `npoint`, aunque no se imponga como tal.

Capítulo 13

Búferes circulares temporales

Un tipo de búfer circular estático `cbuffer` representa un punto 2D y un radio mayor o igual que 0 alrededor del punto. El radio representa el «impacto» del punto sobre su entorno. Este impacto puede interpretarse como ruido, contaminación o aspectos similares según los requisitos de la aplicación.

El tipo `cbuffer` sirve como tipo base para definir el tipo de búfer circular temporal `tcbuffer`. El tipo `tcbuffer` tiene una funcionalidad similar al tipo de punto temporal `tgeompoint` con la excepción de que solo considera dos dimensiones. Por lo tanto, la mayoría de las funciones y operadores descritos anteriormente para el tipo `tgeompoint` también son aplicables para el tipo `tcbuffer`. Además, hay funciones específicas definidas para el tipo `tcbuffer`.

La Figura 13.1 ilustra el uso del tipo `tcbuffer` para modelar el búfer circular alrededor de la franja de viento de las tormentas tropicales en 2024. Los datos se obtienen de la National Oceanic and Atmospherique Administration (NOAA). Como se ilustra en la figura, los tipos `tgeompoint` y `tcbuffer` permiten la interpolación *lineal*, mientras que, como hemos visto en la Figura 7.2, el tipo `tgeometry` solo permite la interpolación *escalonada*.

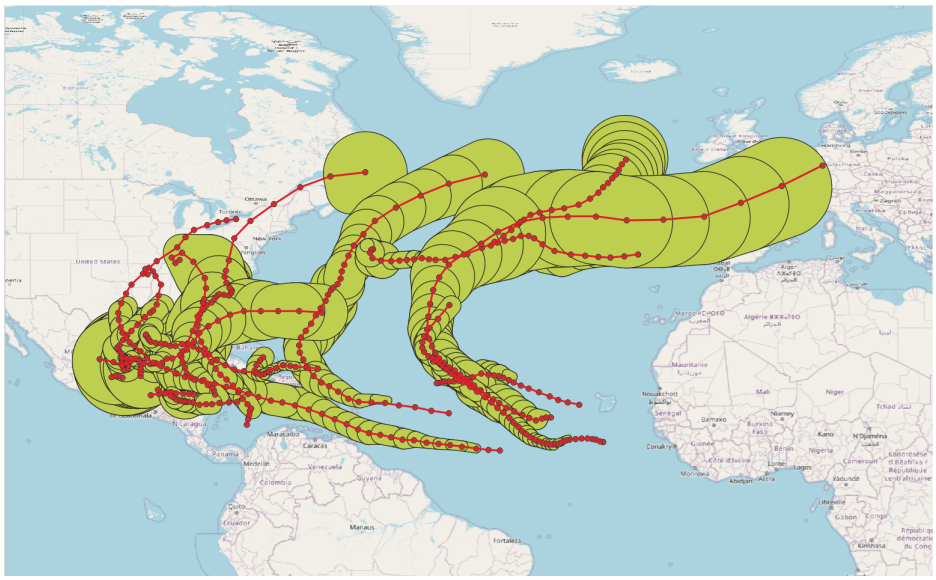


Figura 13.1: Ilustración del uso del tipo `tcbuffer` para modelar el búfer circular alrededor de la franja de viento de las tormentas tropicales en 2024.

13.1. Búferes circulares estáticos

Un valor `cbuffer` es un par de la forma `(point,radius)` donde `point` es un punto geométrico y `radius` es un valor `float` mayor o igual que cero que representa un radio alrededor del punto expresado utilizando las unidades del SRID del punto. Ejemplos de entrada de valores de búfer circular son los siguientes:

```
SELECT cbuffer 'Cbuffer(Point(1 1), 0.3)';
SELECT cbuffer 'Cbuffer(Point(1 1), 10.0)';
```

El tipo `cbuffer` corresponde al resultado de la función de PostGIS ST_MinimumBoundingRadius.

Los valores del tipo de búfer circular deben satisfacer varias restricciones para que estén bien definidos. Ejemplos de valores incorrectos del tipo de búfer circular son los siguientes.

```
-- valor de punto incorrecto
SELECT cbuffer 'Cbuffer(Linestring(1 1,2 2), 1.0)';
-- punto 3D incorrecto
SELECT cbuffer 'Cbuffer(Point Z(1 1 1), 1.0)';
-- valor de radio incorrecto
SELECT cbuffer 'Cbuffer(Point(1 1), -2.0)';
```

A continuación damos las funciones y operadores para el tipo de búfer circular.

13.1.1. Constructores

- Constructor para búferes circulares

  `cbuffer(geompoint,float) → cbuffer`

```
SELECT asText(cbuffer(ST_Point(1,1), 0.3));
-- Cbuffer(POINT(1 1),0.3)
```

13.1.2. Conversión de tipos

Los valores del tipo `cbuffer` se pueden convertir al tipo `geometry` utilizando un `CAST` explícito o utilizando la notación `::` como se muestra a continuación. De manera similar, una `geometry` se puede convertir a un valor `cbuffer` utilizando la función `ST_MinimumBoundingRadius` de PostGIS.

- Convertir entre un búfer circular y una geometría

  `geometry::cbuffer`

  `cbuffer::geometry`

```
SELECT ST_AsText(cbuffer(ST_Point(1,1), 1)::geometry);
-- CURVEPOLYGON(CIRCULARSTRING(0 1,2 1,0 1))
SELECT asText(geometry 'SRID=5676;CIRCULARSTRING(0 1,2 1,0 1)'::cbuffer);
-- Cbuffer(POINT(1 1),1)
SELECT asText(geometry 'SRID=5676;LINESTRING(0 0,2 2)'::cbuffer);
-- Cbuffer(POINT(1 1),1.414213562373095)
```

- Convertir un búfer circular y, opcionalmente, una marca de tiempo o un período, a un cuadro espaciotemporal

  `stbox(cbuffer) → stbox`

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

```
SELECT stbox(cbuffer 'SRID=5676;Cbuffer(Point(1 1),0.3)');
-- SRID=5676;STBOX X((0.7,0.7),(1.3,1.3))
SELECT stbox(cbuffer 'Cbuffer(Point(1 1),0.3)', timestamptz '2001-01-01');
-- STBOX XT(((0.7,0.7),(1.3,1.3)),[2001-01-01, 2001-01-01])
SELECT stbox(cbuffer 'Cbuffer(Point(1 1),0.3)', tstzspan '[2001-01-01,2001-01-02]');
-- STBOX XT(((0.7,0.7),(1.3,1.3)),[2001-01-01, 2001-01-02])
```

13.1.3. Accesores

- Devuelve el punto

  `point(cbuffer) → geompoint`

```
SELECT ST_AsText(point(cbuffer 'Cbuffer(Point(1 1), 0.3)'));
-- Point(1 1)
```

- Devuelve el radio

  `radius(cbuffer) → float`

```
SELECT radius(cbuffer 'Cbuffer(Point(1 1), 0.3)');
-- 0.3
```

13.1.4. Transformaciones

- Redondear el punto y el radio del búfer circular en el número de lugares decimales

  `round(cbuffer,integer=0) → cbuffer`

```
SELECT asText(round(cbuffer(ST_Point(1.123456789,1.123456789), 0.123456789), 6));
-- Cbuffer(POINT(1.123457 1.123457),0.123457)
```

13.1.5. Operaciones espaciales

- Devuelve o establece el identificador de referencia espacial

  `SRID(cbuffer) → integer`

  `setSRID(cbuffer) → cbuffer`

```
SELECT SRID(cbuffer 'Cbuffer(SRID=5676;Point(1 1), 0.3)');
-- 5676
SELECT asEWKT(setSRID(cbuffer 'Cbuffer(Point(0 0),1)', 4326));
-- SRID=4326;Cbuffer(POINT(0 0),1)
```

- Transformar a una referencia espacial diferente

  `transform(cbuffer,integer) → cbuffer`

  `transformPipeline(cbuffer,pipeline text,to_srid integer,is_forward bool=true) → cbuffer`

  La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando el búfer circular 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 búfer circular de entrada se ignora y el SRID del búfer circular 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(cbuffer 'SRID=4326;Cbuffer(Point(4.35 50.85),1)', 3812));
-- SRID=3812;Cbuffer(POINT(648679.0180353033 671067.0556381135),1)
```

```
WITH test(cbuffer, pipeline) AS (
  SELECT cbuffer 'Cbuffer(SRID=4326;Point(4.3525 50.846667),1)',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(cbuffer, pipeline, 4326), pipeline,
  4326, false), 6)
FROM test;
-- SRID=4326;Cbuffer(POINT(4.3525 50.846667),1)
```

13.1.6. Comparaciones

Los operadores de comparación (=, < y así sucesivamente) están disponibles para búferes circulares. Comparan primero los puntos y luego el radio de los argumentos. 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 búferes circulares.

- Comparaciones tradicionales

  `cbuffer {=, <>, <, >, <=, >=} cbuffer`

```
SELECT cbuffer 'Cbuffer(Point(3 3), 0.5)' = cbuffer 'Cbuffer(Point(3 3), 0.5)';
-- true
SELECT cbuffer 'Cbuffer(Point(3 3), 0.5)' <> cbuffer 'Cbuffer(Point(3 3), 0.6)';
-- true
SELECT cbuffer 'Cbuffer(Point(3 3), 0.5)' < cbuffer 'Cbuffer(Point(3 3), 0.6)';
-- true
SELECT cbuffer 'Cbuffer(Point(3 3), 0.6)' > cbuffer 'Cbuffer(Point(2 2), 0.6)';
-- true
SELECT cbuffer 'Cbuffer(Point(1 1), 0.5)' <= cbuffer 'Cbuffer(Point(2 2), 0.5)';
-- true
SELECT cbuffer 'Cbuffer(Point(1 1), 0.6)' >= cbuffer 'Cbuffer(Point(1 1), 0.5)';
-- true
```

13.2. Búferes circulares temporales

El tipo de búfer circular temporal `tcbuffer` permite representar el movimiento de objetos junto con un radio circular alrededor de ellos. Como todos los tipos temporales, se presenta en tres subtipos, a saber, instante, secuencia y conjunto de secuencias. A continuación se dan ejemplos de valores de `tcbuffer` en estos subtipos.

```
SELECT tcbuffer 'Cbuffer(Point(1 1), 0.5)@2001-01-01';
SELECT tcbuffer '{Cbuffer(Point(1 1), 0.3)@2001-01-01, Cbuffer(Point(1 1), 0.5)@2001-01-02,
  Cbuffer(Point(1 1), 0.5)@2001-01-03}';
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01, Cbuffer(Point(1 1), 0.4)@2001-01-02,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]';
SELECT tcbuffer '{[Cbuffer(Point(1 1), 1)@2001-01-01, Cbuffer(Point(2 2), 1)@2001-01-02],
  [Cbuffer(Point(2 2), 2)@2001-01-04, Cbuffer(Point(2 2), 3)@2001-01-05]}';
```

El tipo de búfer circular temporal acepta modificadores de tipo (o `typmod` en la terminología de PostgreSQL) para especificar el subtipo y/o el identificador de referencia espacial (SRID). Los valores posibles para el subtipo son `Instant`, `Sequence` y `SequenceSet`. Los dos argumentos son opcionales y si alguno de ellos no se especifica para una columna, se permiten valores de cualquier subtipo y/o SRID.

```
SELECT asEWKT(tcbuffer(Sequence,5676) 'SRID=5676;[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]');
-- SRID=5676;[Cbuffer(POINT(1 1),0.2)@2001-01-01, Cbuffer(POINT(1 1),0.5)@2001-01-03]
SELECT tcbuffer(Sequence) 'Cbuffer(Point(1 1), 0.2)@2001-01-01';
-- ERROR: Temporal type (Instant) does not match column type (Sequence)
SELECT tcbuffer(5676) 'Cbuffer(Point(1 1), 0.2)@2001-01-01';
-- ERROR:  Temporal circular buffer SRID (0) does not match column SRID (5676)
```

Los valores de búfer circular 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 asText(tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(2 2), 0.4)@2001-01-02, Cbuffer(Point(3 3), 0.6)@2001-01-03)');
-- [Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(3 3),0.6)@2001-01-03)
SELECT asText(tcbuffer '{[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(2 2), 0.3)@2001-01-02, Cbuffer(Point(2 2), 0.5)@2001-01-03),
  [Cbuffer(Point(2 2), 0.5)@2001-01-03, Cbuffer(Point(2 2), 0.7)@2001-01-04)}');
/* {[Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(2 2),0.3)@2001-01-02,
   Cbuffer(Point(2 2),0.7)@2001-01-04)} */
```

13.3. Validez de los búferes circulares temporales

Los valores de búfer circular 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 tcbuffer 'NULL@2001-01-01 08:05:00';
SELECT tcbuffer 'Point(0 0)@NULL';
-- El tipo base no es un búfer circular
SELECT tcbuffer 'Point(0 0)@2001-01-01 08:05:00';
```

A continuación damos las funciones y operadores para el búfer circular temporal. La mayoría de las funciones y operadores para tipos temporales descritos en los capítulos precedentes se pueden aplicar para los tipos de búfer circular temporal. Por lo tanto, en las firmas de las funciones, la notación `base` representa un `cbuffer` y las notaciones `ttype`, `tpoint` y `tgeompoint` también representan un `tcbuffer`. Además, las funciones que tienen un argumento de tipo `geometry` aceptan además un argumento de tipo `cbuffer`. Para evitar la redundancia, a continuación solo presentamos algunos ejemplos de estas funciones y operadores para búferes circulares temporales.

13.4. Entrada y salida

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

  `asText({tcbuffer,tcbuffer[],cbuffer[]}) → {text,text[]}`

  `asEWKT({tcbuffer,tcbuffer[],cbuffer[]}) → {text,text[]}`

```
SELECT asText(tcbuffer 'SRID=4326;[Cbuffer(Point(0 0),1)@2001-01-01,
  Cbuffer(Point(1 1),2)@2001-01-02)');
-- [Cbuffer(Point(0 0),1)@2001-01-01, Cbuffer(Point(1 1),2)@2001-01-02)
SELECT asText(ARRAY[cbuffer 'Cbuffer(Point(0 0),1)', 'Cbuffer(Point(1 1),2)']);
-- {"Cbuffer(POINT(0 0),1)","Cbuffer(POINT(1 1),2)"}
SELECT asEWKT(tcbuffer 'SRID=4326;[Cbuffer(Point(0 0),1)@2001-01-01,
  Cbuffer(Point(1 1),2)@2001-01-02)');
-- SRID=4326;[Cbuffer(Point(0 0),1)@2001-01-01, Cbuffer(Point(1 1),2)@2001-01-02)
SELECT asEWKT(ARRAY[cbuffer 'Cbuffer(SRID=5676;Point(0 0),1)',
  'Cbuffer(SRID=5676;Point(1 1),2)']);
-- {"Cbuffer(SRID=5676;POINT(0 0),1)","Cbuffer(SRID=5676;POINT(1 1),2))"}
```

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

  `asMFJSON(tcbuffer) → text`

  Un búfer circular siempre es planar 2D. Cada valor es un objeto con un miembro `point`, un arreglo de coordenadas `[x, y]` y un miembro numérico `radius`, que reflejan los accesores `point` y `radius`.

```
SELECT asMFJSON(tcbuffer 'Cbuffer(Point(1 2),0.5)@2001-01-01', 3);
/* {"type":"MovingCircularBuffer","bbox":[[0.5,1.5],[1.5,2.5]],
  "period":{"begin":"2001-01-01T00:00:00+01","end":"2001-01-01T00:00:00+01",
  "lower_inc":true,"upper_inc":true},
  "values":[{"point":[1,2],"radius":0.5}],
  "datetimes":["2001-01-01T00:00:00+01"],"interpolation":"None"} */
SELECT asMFJSON(tcbuffer 'SRID=4326;[Cbuffer(Point(1 1),0.2)@2001-01-01,
  Cbuffer(Point(2 2),0.4)@2001-01-02)', 3);
/* {"type":"MovingCircularBuffer","crs":{"type":"Name",
  "properties":{"name":"EPSG:4326"}},"bbox":[[0.8,0.8],[2.4,2.4]],
  "period":{"begin":"2001-01-01T00:00:00+01","end":"2001-01-02T00:00:00+01",
  "lower_inc":true,"upper_inc":false},
  "values":[{"point":[1,1],"radius":0.2},{"point":[2,2],"radius":0.4}],
  "datetimes":["2001-01-01T00:00:00+01","2001-01-02T00:00:00+01"],
  "lower_inc":true,"upper_inc":false,"interpolation":"Linear"} */
```

- Devuelve la representación binaria conocida (Well-Known Binary o WKB), la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o la representación hexadecimal extendida binaria conocida (Extended Well-Known Binary o EWKB)

  `asBinary(tcbuffer,endian text='') → bytea`

  `asEWKB(tcbuffer,endian text='') → bytea`

  `asHexEWKB(tcbuffer,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 asBinary(tcbuffer 'Cbuffer(Point(1 2),1)@2001-01-01');
-- \x013b0001000000000000f03f000000000000004000000000000f03f009c57d3c11c0000
SELECT asEWKB(tcbuffer 'SRID=7844;Cbuffer(Point(1 2),1)@2001-01-01');
-- \x013b0041a41e0000000000000000f03f000000000000004000000000000f03f009c57d3c11c0000
SELECT asHexEWKB(tcbuffer 'SRID=3812;Cbuffer(Point(1 2),1)@2001-01-01');
-- 013B0041E40E0000000000000000F03F000000000000004000000000000F03F009C57D3C11C0000
```

- Entrar a partir de la representación de texto conocido (Well-Known Text o WKT) o de la representación extendida de texto conocido (Extended Well-Known Text o EWKT)

  `tcbufferFromText(text) → tcbuffer`

  `tcbufferFromEWKT(text) → tcbuffer`

```
SELECT asEWKT(tcbufferFromText(text '[Cbuffer(Point(1 2),1)@2001-01-01,
  Cbuffer(Point(3 4),2)@2001-01-02]'));
-- [Cbuffer(POINT(1 2),1)@2001-01-01, Cbuffer(POINT(3 4),2)@2001-01-02]
SELECT asEWKT(tcbufferFromEWKT(text 'SRID=3812;[Cbuffer(Point(1 2),1)@2001-01-01,
  Cbuffer(Point(3 4),2)@2001-01-02]'));
-- SRID=3812;[Cbuffer(Point(1 2),1)@2001-01-01, Cbuffer(Point(3 4),2)@2001-01-02]
```

- Entrar a partir de la representación JSON de características móviles (Moving Features JSON o MF-JSON)

  `tcbufferFromMFJSON(text) → tcbuffer`

```
SELECT asEWKT(tcbufferFromMFJSON(asMFJSON(tcbuffer
  'SRID=3812;Cbuffer(Point(1 2),0.5)@2001-01-01', 3)));
-- SRID=3812;Cbuffer(POINT(1 2),0.5)@2001-01-01
SELECT asEWKT(tcbufferFromMFJSON(asMFJSON(tcbuffer
  'SRID=5676;[Cbuffer(Point(1 2),1)@2001-01-01, Cbuffer(Point(3 4),2)@2001-01-02]', 3)));
-- SRID=5676;[Cbuffer(POINT(1 2),1)@2001-01-01, Cbuffer(POINT(3 4),2)@2001-01-02]
```

- Entrar a partir de la representación binaria conocida (WKB), de la representación extendida binaria conocida (EWKB), o de la representación hexadecimal extendida binaria conocida (HexEWKB)

`tcbufferFromBinary(bytea) → tcbuffer`

`tcbufferFromEWKB(bytea) → tcbuffer`

`tcbufferFromHexEWKB(text) → tcbuffer`

```
SELECT asEWKT(tcbufferFromBinary(
  '\x013b0001000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000'));
-- Cbuffer(POINT(1 2),1)@2001-01-01
SELECT asEWKT(tcbufferFromEWKB(
  '\x013b0041a41e0000000000000000f03f0000000000000040000000000000f03f009c57d3c11c0000'));
-- SRID=7844;Cbuffer(Point(1 1),2)@2001-01-01
SELECT asEWKT(tcbufferFromHexEWKB(
  '013B0041E40E000000000000000F03F0000000000000040000000000000F03F009C57D3C11C0000'));
-- SRID=3812;Cbuffer(POINT(1 2),1)@2001-01-01
```

13.5. Constructores

- Constructor para búferes circulares temporales con valor constante

`tcbuffer(cbuffer,timestamptz) → tcbufferInst`

`tcbuffer(cbuffer,tstzset) → tcbufferDiscSeq`

`tcbuffer(cbuffer,tstzspan,interp='linear') → tcbufferContSeq`

`tcbuffer(cbuffer,tstzspanset,interp='linear') → tcbufferSeqSet`

```
SELECT asText(tcbuffer('Cbuffer(Point(1 1), 0.5)', timestamptz '2001-01-01'));
-- Cbuffer(Point(1 1),0.5)@2001-01-01
SELECT asText(tcbuffer('Cbuffer(Point(1 1), 0.3)',
  tstzset '{2001-01-01, 2001-01-03, 2001-01-05}'));
/* {Cbuffer(Point(1 1),0.3)@2001-01-01, Cbuffer(Point(1 1),0.3)@2001-01-03,
   Cbuffer(Point(1 1),0.3)@2001-01-05} */
SELECT asText(tcbuffer('Cbuffer(Point(1 1), 0.5)',
  tstzspan '[2001-01-01, 2001-01-02]'));
-- [Cbuffer(Point(1 1),0.5)@2001-01-01, Cbuffer(Point(1 1),0.5)@2001-01-02]
SELECT asText(tcbuffer('Cbuffer(Point(1 1), 0.2)',
  tstzspanset '{[2001-01-01, 2001-01-03]}', 'step'));
-- Interp=Step;{[Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(1 1),0.2)@2001-01-03]}
```

- Constructor para búferes circulares temporales de subtipo secuencia

`tcbufferSeq(tcbufferInst[],interp='linear',leftInc bool=true,`
`rightInc bool=true) →tcbufferSeq`

```
SELECT asText(tcbufferSeq(ARRAY[tcbuffer 'Cbuffer(Point(1 1), 0.3)@2001-01-01',
  'Cbuffer(Point(2 2), 0.5)@2001-01-02', 'Cbuffer(Point(1 1), 0.5)@2001-01-03']));
/* [Cbuffer(Point(1 1),0.3)@2001-01-01, Cbuffer(Point(2 2),0.5)@2001-01-02,
   Cbuffer(Point(1 1),0.5)@2001-01-03] */
SELECT asText(tcbufferSeq(ARRAY[tcbuffer 'Cbuffer(Point(1 1), 0.3)@2001-01-01',
  'Cbuffer(Point(2 2), 0.5)@2001-01-02', 'Cbuffer(Point(1 1), 0.5)@2001-01-03'],
  'discrete'));
/* {Cbuffer(POINT(1 1),0.3)@2001-01-01, Cbuffer(POINT(2 2),0.5)@2001-01-02,
   Cbuffer(POINT(1 1),0.5)@2001-01-03} */
SELECT asText(tcbufferSeq(ARRAY[tcbuffer 'Cbuffer(Point(1 1), 0.2)@2001-01-01',
  'Cbuffer(Point(1 1), 0.4)@2001-01-02', 'Cbuffer(Point(1 1), 0.5)@2001-01-03'], 'step'));
/* Interp=Step;[Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(1 1),0.4)@2001-01-02,
   Cbuffer(Point(1 1),0.5)@2001-01-03] */
```

- Constructor para búferes circulares temporales de subtipo conjunto de secuencias

  `tcbufferSeqSet(tcbuffer[]) → tcbufferSeqSet`

  `tcbufferSeqSetGaps(tcbufferInst[],maxt=NULL,maxdist=NULL,interp='linear') → tcbufferSeqSet`

```
SELECT asText(tcbufferSeqSet(ARRAY[tcbuffer
  '[Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(2 2),0.4)@2001-01-02]',
  '[Cbuffer(Point(2 2),0.6)@2001-01-03, Cbuffer(Point(2 2),0.8)@2001-01-04]']));
/* {[Cbuffer(Point(1 1),0.2)@2001-01-01, Cbuffer(Point(2 2),0.4)@2001-01-02],
   [Cbuffer(Point(2 2),0.6)@2001-01-03, Cbuffer(Point(2 2),0.6)@2001-01-04]} */
SELECT asText(tcbufferSeqSetGaps(ARRAY[tcbuffer 'Cbuffer(Point(1 1),0.1)@2001-01-01',
  'Cbuffer(Point(1 1),0.3)@2001-01-03', 'Cbuffer(Point(1 1),0.5)@2001-01-05'], '1 day'));
/* {[Cbuffer(Point(1 1),0.1)@2001-01-01], [Cbuffer(Point(1 1),0.3)@2001-01-03],
   [Cbuffer(Point(1 1),0.5)@2001-01-05]} */
```

- Construir un búfer circular temporal a partir de un punto de geometría temporal y un flotante temporal

  `tcbuffer(tgeompoint, tfloat) → tcbuffer`

  El marco de tiempo del resultado es la intersección de los marcos de tiempo del punto temporal y el flotante temporal. Si no se intersecan, se devuelve un valor nulo

```
SELECT asText(tcbuffer(tgeompoint '[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02)',
  tfloat '[1@2001-01-01, 2@2001-01-02)'));
-- [Cbuffer(Point(1 1),1)@2001-01-01, Cbuffer(Point(1 1),2)@2001-01-02)
SELECT asText(tcbuffer(tgeompoint '[POINT(1 1)@2001-01-01, POINT(2 2)@2001-01-02)',
  tfloat '[2@2001-01-03, 3@2001-01-04)'));
-- NULL
```

13.6. Conversión de tipos

Un valor de búfer circular 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 `cbuffer`.

- Convertir un búfer circular temporal a un punto de geometría temporal

  `tcbuffer::tgeompoint`

```
SELECT asText((tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-02)')::tgeompoint);
-- [POINT(1 1)@2001-01-01, POINT(1 1)@2001-01-02)
```

- Convertir un búfer circular temporal a un flotante temporal

  `tcbuffer::tfloat`

```
SELECT (tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-02)')::tfloat;
-- [0.2@2001-01-01, 0.3@2001-01-02)
```

13.7. Accesores

- Devuelve los valores

  `getValues(tcbuffer) → cbufferset`

```
SELECT asText(getValues(tcbuffer '{[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02)}'));
-- {"Cbuffer(Point(1 1),0.3)","Cbuffer(Point(1 1),0.5)"}
SELECT asEWKT(getValues(tcbuffer 'SRID=5676;{[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-02)}'));
-- SRID=5676;{"Cbuffer(Point(1 1),0.3)"}
```

- Devuelve los puntos o los radios

  `points(tcbuffer) → geomset`

  `radius(tcbuffer) → floatset`

```
SELECT asEWKT(points(tcbuffer 'SRID=5676;{Cbuffer(Point(3 3), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02}'));
-- SRID=5676;{Point(1 1), Point(3 3)}
SELECT radius(tcbuffer 'SRID=5676;{Cbuffer(Point(3 3), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02}');
-- {0.3, 0.5}
```

- Devuelve el valor en una marca de tiempo

  `valueAtTimestamp(tcbuffer,timestamptz) → cbuffer`

```
SELECT asText(valueAtTimestamp(tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(3 3), 0.5)@2001-01-03)', '2001-01-02'));
-- Cbuffer(Point(2 2),0.4)
```

13.8. Transformaciones

- Transformar un búfer circular temporal en otro subtipo

  `tcbufferInst(tcbuffer) → tcbufferInst`

  `tcbufferSeq(tcbuffer) → tcbufferSeq`

  `tcbufferSeqSet(tcbuffer) → tcbufferSeqSet`

```
SELECT asText(tcbufferSeq(tcbuffer 'Cbuffer(Point(1 1), 0.5)@2001-01-01', 'discrete'));
-- {Cbuffer(Point(1 1),0.5)@2001-01-01}
SELECT asText(tcbufferSeq(tcbuffer 'Cbuffer(Point(1 1), 0.5)@2001-01-01'));
-- [Cbuffer(Point(1 1),0.5)@2001-01-01]
SELECT asText(tcbufferSeqSet(tcbuffer 'Cbuffer(Point(1 1), 0.5)@2001-01-01'));
-- {[Cbuffer(Point(1 1),0.5)@2001-01-01]}
```

- Transformar un búfer circular temporal en otra interpolación

  `setInterp(tcbuffer, interp) → tcbuffer`

```
SELECT asText(setInterp(tcbuffer 'Cbuffer(Point(1 1),0.2)@2001-01-01','linear'));
-- [Cbuffer(Point(1 1),0.2)@2001-01-01]
SELECT asText(setInterp(tcbuffer '{[Cbuffer(Point(1 1),0.1)@2001-01-01],
  [Cbuffer(Point(1 1),0.2)@2001-01-02]}', 'discrete'));
-- {Cbuffer(Point(1 1),0.1)@2001-01-01, Cbuffer(Point(1 1),0.2)@2001-01-02}
```

- Redondear los puntos y los radios del búfer circular temporal en el número de lugares decimales

  `round(tcbuffer,integer) → tcbuffer`

```
SELECT asText(round(tcbuffer '{[Cbuffer(Point(1 1.123456789),0.123456789)@2001-01-01,
  Cbuffer(Point(1 1),0.5)@2001-01-02)}', 3));
-- {[Cbuffer(Point(1 1.123),0.123)@2001-01-01, Cbuffer(Point(1 1),0.5)@2001-01-02)}
```

13.9. Modificaciones

- Agregar un instante temporal o una secuencia temporal

  `appendInstant(tcbuffer,tcbufferInst) → tcbuffer`

  `appendSequence(tcbuffer,tcbufferSeq) → tcbuffer`

```
SELECT asText(appendInstant(tcbuffer 'Cbuffer(Point(1 1), 0.1)@2001-01-01', 'Cbuffer(Point ↩
    (2 2), 0.2)@2001-01-02'));
-- [Cbuffer(POINT(1 1),0.1)@2001-01-01 00:00:00+01, Cbuffer(POINT(2 2),0.2)@2001-01-02 ↩
    00:00:00+01]
SELECT asText(appendSequence(tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01]', '[Cbuffer( ↩
    Point(2 2), 0.2)@2001-01-02]'));
-- {[Cbuffer(POINT(1 1),0.1)@2001-01-01 00:00:00+01], [Cbuffer(POINT(2 2),0.2)@2001-01-02 ↩
    00:00:00+01]}
```

- Fusionar los búferes circulares temporales

  `merge(tcbuffer,tcbuffer) → tcbuffer`

  `merge(tcbuffer[]) → tcbuffer`

  `mergeAgg(tcbuffer) → tcbuffer`

```
SELECT asText(merge(tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), ↩
    0.3)@2001-01-03]',
  '[Cbuffer(Point(1 1), 0.3)@2001-01-03, Cbuffer(Point(1 1), 0.5)@2001-01-05]'));
-- [Cbuffer(POINT(1 1),0.1)@2001-01-01 00:00:00+01, Cbuffer(POINT(1 1),0.5)@2001-01-05 ↩
    00:00:00+01]
SELECT asText(merge(ARRAY[tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 ↩
    1), 0.2)@2001-01-02]',
  '[Cbuffer(Point(1 1), 0.2)@2001-01-02, Cbuffer(Point(1 1), 0.3)@2001-01-03]']));
-- [Cbuffer(POINT(1 1),0.1)@2001-01-01 00:00:00+01, Cbuffer(POINT(1 1),0.3)@2001-01-03 ↩
    00:00:00+01]
WITH temp(inst) AS (
  SELECT tcbuffer 'Cbuffer(Point(1 1), 0.1)@2001-01-01' UNION
  SELECT tcbuffer 'Cbuffer(Point(2 2), 0.2)@2001-01-02' )
SELECT asText(mergeAgg(inst)) FROM temp;
-- {Cbuffer(POINT(1 1),0.1)@2001-01-01 00:00:00+01, Cbuffer(POINT(2 2),0.2)@2001-01-02 ↩
    00:00:00+01}
```

13.10. Operaciones espaciales

- Devuelve o establece el identificador de referencia espacial

  `SRID(tcbuffer) → integer`

  `setSRID(tcbuffer) → tcbuffer`

```
SELECT SRID(tcbuffer 'SRID=5676;[Cbuffer(Point(0 0),1)@2001-01-01,
  Cbuffer(Point(1 1),2)@2001-01-02)');
-- 5676
SELECT asEWKT(setSRID(tcbuffer '[Cbuffer(Point(0 0),1)@2001-01-01,
  Cbuffer(Point(1 1),2)@2001-01-02)', 5676));
-- SRID=5676;[Cbuffer(POINT(0 0),1)@2001-01-01, Cbuffer(POINT(1 1),2)@2001-01-02)
```

- Transformar a una referencia espacial diferente

  `transform(tcbuffer,integer) → tcbuffer`

  `transformPipeline(tcbuffer,pipeline text,to_srid integer,is_forward bool=true) → tcbuffer`

La función `transform` especifica la transformación con un SRID de destino. Se genera un error cuando el búfer circular temporal 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 búfer circular temporal de entrada se ignora y el SRID del búfer circular 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(tcbuffer 'SRID=4326;Cbuffer(Point(4.35 50.85),1)@2001-01-01',
  3812));
-- SRID=3812;Cbuffer(POINT(648679.0180353033 671067.0556381135),1)@2001-01-01
```

```
WITH test(tcbuffer, pipeline) AS (
  SELECT tcbuffer 'SRID=4326;{Cbuffer(Point(4.3525 50.846667),1)@2001-01-01,
    Cbuffer(Point(-0.1275 51.507222),2)@2001-01-02}',
    text 'urn:ogc:def:coordinateOperation:EPSG::16031' )
SELECT asEWKT(transformPipeline(transformPipeline(tcbuffer, pipeline, 4326),  pipeline,
  4326, false), 6)
FROM test;
/* SRID=4326;{Cbuffer(POINT(4.3525 50.846667),1)@2001-01-01,
   Cbuffer(POINT(-0.1275 51.507222),2)@2001-01-02} */
```

- Devuelve el área atravesada por el búfer circular temporal

  `traversedArea(tcbuffer) → geometry`

```
SELECT ST_AsText(traversedArea(tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]'));
-- CURVEPOLYGON(CIRCULARSTRING(0.7 1,1.3 1,0.7 1))
```

- Devuelve el punto geométrico temporal dado por los centros de un búfer circular temporal

  `centroid(tcbuffer) → tgeompoint`

```
SELECT asText(centroid(tcbuffer '[Cbuffer(Point(1 1), 0.5)@2001-01-01,
  Cbuffer(Point(3 3), 0.5)@2001-01-03]'));
-- [POINT(1 1)@2001-01-01 00:00:00+01, POINT(3 3)@2001-01-03 00:00:00+01]
```

- Devuelve la envolvente convexa del área atravesada por el búfer circular temporal

  `convexHull(tcbuffer) → geometry`

  La envolvente convexa se calcula sobre el área atravesada linealizada: los arcos circulares que delimitan la región barrida se convierten primero en segmentos rectos y la envolvente se calcula luego con GEOS. Por lo tanto, el resultado es un `POLYGON` lineal que aproxima el contorno circular, no un `CURVEDPOLYGON` curvo. Utilice `traversedArea` para obtener la región barrida curva exacta.

```
SELECT ST_GeometryType(convexHull(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(4 0), 1)@2001-01-02]'));
-- ST_Polygon
SELECT ST_NPoints(convexHull(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(4 0), 1)@2001-01-02]'));
-- 131
SELECT round(ST_Area(convexHull(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(4 0), 1)@2001-01-02]'))::numeric, 3);
-- 11.140
-- The exact swept region has area pi + 8 = 11.1416; the stroked hull is
-- slightly smaller because the arcs are replaced by chords.
```

13.11. Operaciones de distancia

- Devuelve la distancia más cercana

  `{geo,cbuffer,tcbuffer} |=| {geo,cbuffer,tcbuffer} → float`

  El operador `|=|` se puede utilizar para realizar búsquedas del vecino más cercano utilizando un índice GiST o SP-GiST (ver Sección 10.2).

```
SELECT tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]' |=| geometry 'Linestring(50 50,55 55)';
-- 67.58225099390856
SELECT tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]' |=| cbuffer 'Cbuffer(Point(1 1), 0.5)';
-- 0.6142135623730951
```

- TODO Devuelve el instante del primer búfer circular temporal en el que los dos argumentos están a la distancia más cercana

  `nearestApproachInstant({geo,cbuffer,tcbuffer},{geo,cbuffer,tcbuffer}) → tcbuffer`

```
SELECT nearestApproachInstant(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)');
-- Cbuffer(Point(2 2),0.349928)@2001-01-01 02:59:44.402905+01
SELECT nearestApproachInstant(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]', cbuffer 'Cbuffer(Point(1 1), 0.5)');
-- Cbuffer(Point(2 2),0.592181)@2001-01-01 17:31:51.080405+01
```

- TODO Devuelve la línea que conecta el punto de aproximación más cercano entre los dos argumentos

  `shortestLine({geo,cbuffer,tcbuffer},{geo,cbuffer,tcbuffer}) → geometry`

  La función devuelve la primera línea que encuentre si hay más de una

```
SELECT ST_AsText(shortestLine(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]', geometry 'Linestring(50 50,55 55)'));
-- LINESTRING(50 50,2.212132034355964 2.212132034355964)
SELECT ST_AsText(shortestLine(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-02]', cbuffer 'Cbuffer(Point(1 1), 0.5)'));
-- LINESTRING(1.353553390593274 1.353553390593274,1.787867965644036 1.787867965644036)
```

- TODO Devuelve la distancia temporal

  `{geo,cbuffer,tcbuffer} <-> {geo,cbuffer,tcbuffer} → tfloat`

```
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]' <-> cbuffer 'Cbuffer(Point(1 1), 0.2)';
-- [2.349883008750630@2001-01-02 00:00:00+01, 2.349883008750630@2001-01-03 00:00:00+01]
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]' <-> geometry 'Point(50 50)';
-- [25.04966669450440@2001-01-01 00:00:00+01, 26.40856884262320@2001-01-03 00:00:00+01]
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]' <->
  tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-02, Cbuffer(Point(1 1), 0.5)@2001-01-04]'
-- [2.349883008750630@2001-01-02 00:00:00+01, 2.349883008750630@2001-01-03 00:00:00+01]
```

- Devuelve la distancia espacial mínima, ignorando el tiempo

  `minDistance(tcbuffer,{geo,cbuffer}) → float`

  `minDistance(tcbuffer,tcbuffer) → float` (agregado)

  El resultado es la distancia espacial entre las áreas barridas por los dos argumentos durante toda su vida útil, igual a `ST_Distance(t traversedArea(...))`. La forma de dos argumentos de búfer circular temporal es un agregado, de modo que una combinación cruzada agrupada por una clave produce el mínimo sobre cada par del grupo.

```
SELECT minDistance(tcbuffer '[Cbuffer(Point(0 0), 0.5)@2001-01-01,
  Cbuffer(Point(10 0), 0.5)@2001-01-02]', geometry 'Point(5 5)');
-- 4.5
SELECT minDistance(tcbuffer '[Cbuffer(Point(0 0), 0.5)@2001-01-01,
  Cbuffer(Point(10 0), 0.5)@2001-01-02]', cbuffer 'Cbuffer(Point(5 5), 1)');
-- 3.5
SELECT g, minDistance(t1.Noise, t2.Noise)
FROM Vessel v1, VesselTrip t1, Vessel v2, VesselTrip t2
WHERE v1.MMSI = t1.MMSI AND v2.MMSI = t2.MMSI
GROUP BY t1.ShipTypeLabel, t2.ShipTypeLabel;
```

13.12. Restricciones

- Restringir al (complemento de) un valor o un conjunto de valores

  `atValue(tcbuffer,value) → tcbuffer`

  `minusValue(tcbuffer,value) → tcbuffer`

  `atValues(tcbuffer,valueset) → tcbuffer`

  `minusValues(tcbuffer,valueset) → tcbuffer`

```
SELECT asText(atValue(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-03]', cbuffer 'Cbuffer(Point(2 2), 0.5)'));
-- {[Cbuffer(Point(2 2),0.5)@2001-01-02]}
SELECT asText(minusValue(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-03]', cbuffer 'Cbuffer(Point(2 2), 0.5)'));
/* {[Cbuffer(Point(2 2),0.3)@2001-01-01, Cbuffer(Point(2 2),0.5)@2001-01-02),
   (Cbuffer(Point(2 2),0.5)@2001-01-02, Cbuffer(Point(2 2),0.7)@2001-01-03]} */
```

- TODO Restringir al (complemento de) una geometría

  `atGeometry(tcbuffer,geometry) → tcbuffer`

  `minusGeometry(tcbuffer,geometry) → tcbuffer`

```
SELECT asText(atGeometry(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-03]', 'Polygon((0 0,0 2,2 2,2 0,0 0))'));
-- {[Cbuffer(POINT(2 2),0.3)@2001-01-01, Cbuffer(POINT(2 2),0.7)@2001-01-03]}
SELECT asText(minusGeometry(tcbuffer '[Cbuffer(Point(2 2), 0.3)@2001-01-01,
  Cbuffer(Point(2 2), 0.7)@2001-01-03]', 'Polygon((0 0,0 2,2 2,2 0,0 0))'));
/* {(Cbuffer(Point(2 2),0.342593)@2001-01-01 05:06:40.364673+01,
   Cbuffer(Point(2 2),0.7)@2001-01-03 00:00:00+01]} */
```

- Restringir un búfer circular temporal a (el complemento de) una caja espaciotemporal

  `atStbox(tcbuffer,stbox,borderInc bool=true) → tcbuffer`

  `minusStbox(tcbuffer,stbox,borderInc bool=true) → tcbuffer`

```
SELECT asText(atStbox(tcbuffer '[Cbuffer(Point(1 1), 0.5)@2000-01-01,
  Cbuffer(Point(3 3), 0.5)@2000-01-03]', stbox 'STBOX X((0,0),(2,2))'));
-- {[Cbuffer(POINT(1 1),0.5)@2000-01-01 00:00:00+01, Cbuffer(POINT(2 2),0.5)@2000-01-02 ↩
    00:00:00+01)}
```

13.13. Comparaciones

- Comparaciones tradicionales

  `tcbuffer {=, <>, <, >, <=, >=} tcbuffer → boolean`

```
SELECT tcbuffer '{[Cbuffer(Point(1 1), 0.1)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-02),
  [Cbuffer(Point(1 1), 0.3)@2001-01-02, Cbuffer(Point(1 1), 0.5)@2001-01-03]}' =
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.5)@2001-01-03]';
-- true
SELECT tcbuffer '{[Cbuffer(Point(1 1), 0.1)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]}' <>
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.5)@2001-01-03]';
-- false
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]' <
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.6)@2001-01-03]';
-- false
```

- Comparaciones alguna vez y siempre

  `{tcbuffer,cbuffer} {?=,%=} {tcbuffer,cbuffer} → boolean`

```
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.4)@2001-01-03)' ?= cbuffer 'Cbuffer(Point(1 1), 0.3)';
-- true
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.4)@2001-01-03)' ?=
  tcbuffer '[Cbuffer(Point(1 1), 0.4)@2001-01-01, Cbuffer(Point(1 1), 0.2)@2001-01-03)';
-- true
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.2)@2001-01-04)' %= cbuffer 'Cbuffer(Point(1 1), 0.2)';
-- true
```

- Comparaciones temporales

  `tcbuffer {#=, #<>} tcbuffer → tbool`

```
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.4)@2001-01-03)' #= cbuffer 'Cbuffer(Point(1 1), 0.3)';
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.8)@2001-01-03)' #<>
  tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01, Cbuffer(Point(1 1), 0.7)@2001-01-03)';
-- {[t@2001-01-01, f@2001-01-02], (t@2001-01-02, t@2001-01-03)}
```

13.14. Operaciones de cuadro delimitador

- Operadores topológicos

  `{tstzspan,stbox,tcbuffer} {&&, <@, @>, ~=, -|-} {tstzspan,stbox,tcbuffer} → boolean`

```
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' && cbuffer 'Cbuffer(Point(1 1), 0.5)';
-- true
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' @> stbox(cbuffer 'Cbuffer(Point(1 1), 0.5)');
-- true
SELECT cbuffer 'Cbuffer(Point(1 1), 0.5)'::geometry <@
  tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01, Cbuffer(Point(1 1), 0.5)@2001-01-02]';
-- true
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03]' ~= tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.35)@2001-01-02, Cbuffer(Point(1 1), 0.5)@2001-01-03]';
-- true
```

- Operadores de posición

  `{stbox,tcbuffer} {<<, &<, >>, &>} {stbox,tcbuffer} → boolean`

  `{stbox,tcbuffer} {<<|, &<|, |>>, |&>} {stbox,tcbuffer} → boolean`

  `{tstzspan,stbox,tcbuffer} {<<#, &<#, #>>, #&>} {tstzspan,stbox,tcbuffer} → boolean`

```
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' << cbuffer 'Cbuffer(Point(1 1), 0.2)'
-- false
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' <<| stbox(cbuffer 'Cbuffer(Point(1 1), 0.5)')
-- false
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' &> cbuffer 'Cbuffer(Point(1 1), 0.3)'::geometry
-- true
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-02]' >>#
  tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-03, Cbuffer(Point(1 1), 0.5)@2001-01-05]'
-- true
```

13.15. Relaciones espaciales

- TODO Relaciones alguna vez y siempre

  `eContains(geometry,tcbuffer) → boolean`

  `aContains(geometry,tcbuffer) → boolean`

  `eDisjoint({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `aDisjoint({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `eDwithin({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer},float) → boolean`

  `aDwithin({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer},float) → boolean`

  `eIntersects({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `aIntersects({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `eTouches({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `aTouches({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

```
SELECT eContains(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.3)@2001-01-03)');
-- false
SELECT eDisjoint(cbuffer 'Cbuffer(Point(2 2), 0.0)',
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.3)@2001-01-03)');
-- true
SELECT eIntersects(tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-03)',
  tcbuffer '[Cbuffer(Point(2 2), 0.0)@2001-01-01, Cbuffer(Point(2 2), 1)@2001-01-03)');
-- false
```

- TODO Relaciones espaciotemporales

  `tContains(geometry,tcbuffer) → boolean`

  `tDisjoint({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `tDwithin({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer},float) → boolean`

  `tIntersects({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

  `tTouches({geometry,cbuffer,tcbuffer},{geometry,cbuffer,tcbuffer}) → boolean`

```
SELECT tDisjoint(geometry 'Polygon((0 0,0 50,50 50,50 0,0 0))',
  tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01, Cbuffer(Point(1 1), 0.3)@2001-01-03)');
-- {[t@2001-01-01 00:00:00+01, t@2001-01-03 00:00:00+01)}
SELECT tDwithin(tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 1), 0.5)@2001-01-03)', tcbuffer '[Cbuffer(Point(1 1), 0.5)@2001-01-01,
  Cbuffer(Point(1 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)} */
```

13.16. Agregaciones

Las tres funciones agregadas para búferes circulares temporales se ilustran a continuación.

- Conteo temporal

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

```
WITH Temp(temp) AS (
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01,
    Cbuffer(Point(1 1), 0.3)@2001-01-03)' UNION
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-02,
    Cbuffer(Point(1 1), 0.4)@2001-01-04)' UNION
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-03,
    Cbuffer(Point(1 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(tcbuffer) → {tintSeq,tintSeqSet}`

```
WITH Temp(temp) AS (
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01,
    Cbuffer(Point(1 1), 0.3)@2001-01-03)' UNION
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-02,
    Cbuffer(Point(1 1), 0.4)@2001-01-04)' UNION
  SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-03,
    Cbuffer(Point(1 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)} */
```

- TODO Centroide temporal

  `tCentroid(tcbuffer) → tgeompoint`

```
WITH Temp(temp) AS (
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.1)@2001-01-01,
  Cbuffer(Point(1 1), 0.3)@2001-01-03)' UNION
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.2)@2001-01-01,
  Cbuffer(Point(1 1), 0.4)@2001-01-03)' UNION
SELECT tcbuffer '[Cbuffer(Point(1 1), 0.3)@2001-01-01,
  Cbuffer(Point(1 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)} */
```

13.17. Simplificación

- Devuelve un búfer circular temporal simplificado asegurándose de que los centros consecutivos estén al menos separados por una cierta distancia o intervalo de tiempo

  `minDistSimplify(tcbuffer,mindist float) → tcbuffer`

  `minTimeDeltaSimplify(tcbuffer,mint interval) → tcbuffer`

  La simplificación opera sobre la trayectoria de los centros del búfer circular temporal y conserva el radio en cada instante que sobrevive. 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 valor dado.

```
SELECT asText(minDistSimplify(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(1 0), 1)@2001-01-02,
    Cbuffer(Point(4 0), 1)@2001-01-03]', 2));
-- [Cbuffer(POINT(0 0),1)@2001-01-01, Cbuffer(POINT(4 0),1)@2001-01-03]
-- The result keeps the interpolation of the value it simplifies.
SELECT interp(minDistSimplify(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(1 0), 1)@2001-01-02,
    Cbuffer(Point(4 0), 1)@2001-01-03]', 2));
-- Linear
SELECT asText(minTimeDeltaSimplify(tcbuffer
  '[Cbuffer(Point(0 0), 1)@2001-01-01, Cbuffer(Point(1 0), 1)@2001-01-02,
    Cbuffer(Point(4 0), 1)@2001-01-04]', interval '2 days'));
-- [Cbuffer(POINT(0 0),1)@2001-01-01, Cbuffer(POINT(4 0),1)@2001-01-04]
```

- Devuelve un búfer circular temporal simplificado usando el algoritmo de Douglas-Peucker

  `maxDistSimplify(tcbuffer,maxdist float,syncdist=true) → tcbuffer`

  `douglasPeuckerSimplify(tcbuffer,maxdist float,syncdist=true) → tcbuffer`

  La diferencia entre las dos funciones es que `maxDistSimplify` utiliza una versión de un solo paso del algoritmo mientras que `douglasPeuckerSimplify` utiliza el algoritmo recursivo estándar. Al igual que las funciones anteriores, la simplificación opera sobre la trayectoria de los centros y conserva el radio en cada instante que sobrevive.

```
SELECT asText(douglasPeuckerSimplify(tcbuffer '[Cbuffer(Point(1 1),0.5)@2000-01-01,
  Cbuffer(Point(2 2),0.5)@2000-01-02, Cbuffer(Point(3 1),0.5)@2000-01-03,
  Cbuffer(Point(4 4),0.5)@2000-01-04]', 2.0));
-- {Cbuffer(POINT(1 1),0.5)@2000-01-01 00:00:00+01, Cbuffer(POINT(4 4),0.5)@2000-01-04 ↩
    00:00:00+01}
```

13.18. Indexación

Se pueden crear índices GiST y SP-GiST para columnas de tabla de búferes circulares 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 búferes circulares temporales, que es un `stbox`, como todos los tipos espaciotemporales.

Un índice GiST o SP-GiST puede acelerar las consultas que involucran a los siguientes operadores:

- `<<`, `&<`, `&>`, `>>`, `<<|`, `&<|`, `|&>`, `|>>`, que solo consideran la dimensión espacial en búferes circulares temporales,
- `<<#`, `&<#`, `#&>`, `#>>`, que solo consideran la dimensión temporal en búferes circulares temporales,
- `&&`, `@>`, `<@`, `~=`, `-|-` 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.

Capítulo 14

Geometrías rígidas temporales

Una geometría rígida temporal (`trgeometry`) es una geometría de referencia 2D estática —normalmente un polígono que describe la forma de un cuerpo en movimiento— emparejada con una pose temporal (`tpose`) que describe cómo esa geometría se traslada y rota a lo largo del tiempo. La geometría materializada en el instante `t` es la geometría de referencia rotada y trasladada de acuerdo con la pose interpolada en `t`.

Esta es la representación natural para objetos en movimiento cuya *forma* importa, no solo su posición. El ejemplo motivador es el tráfico marítimo AIS, donde el casco de cada barco (un pentágono derivado de los desplazamientos de antena `A`, `B`, `C`, `D`) sigue una trayectoria de pose definida por informes de latitud/longitud y rumbo. La geometría materializada en cualquier instante es el contorno real del barco en ese momento.

El tipo `trgeometry` se construye sobre Capítulo 11. La mayoría de los operadores de acceso, comparación, topológicos, de posición y de indexación tienen la misma forma que las funciones `tpose` correspondientes; este capítulo cubre la superficie específica de trgeometry: la materialización de pose a forma, el área recorrida, la distancia materializada y las restricciones espaciales sobre la trayectoria del centroide.

La guía operativa para elegir entre `trgeometry`, `tpose` y `tgeometry`; las convenciones de marco que la implementación impone (forma de referencia solo 2D, una única geometría de referencia compartida por fila, requisito de mismo SRID, semántica de restricción espacial basada en el centroide); los riesgos numéricos que las cargas de trabajo de producción deben anticipar (advertencia de muestreo para `traversedArea` en segmentos de rotación, tolerancia de submuestreo adaptativo en la distancia materializada, rechazo de SRID mixtos); y las convenciones de durabilidad y almacenamiento se recopilan en Sección 14.18.

14.1. Entrada y salida

Un literal `trgeometry` empareja una geometría de referencia con una pose temporal. La geometría de referencia va primero, separada de la pose por un punto y coma; la pose sigue la sintaxis estándar de `tpose` (instante, secuencia o conjunto de secuencias).

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(0 0), 0.5)@2001-01-01';
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));{Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(5 0), 0.5)@2001-01-02, Pose(Point(0 0), 0.0)@2001-01-03}';
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(10 0), 1.5)@2001-01-02]';
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));{[Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(5 0), 0.0)@2001-01-02], [Pose(Point(0 0), 0.0)@2001-01-04, Pose(Point(2 0), 0.0) ↩
    @2001-01-05]}';
```

La geometría de referencia puede ser cualquier polígono 2D o superficie poliédrica. La interpolación de la pose es lineal en la traslación y de camino angular más corto en la rotación. Los SRID se heredan de la geometría de referencia y de la pose; deben coincidir.

Se puede especificar un SRID para una trgeometry ya sea al comienzo del literal o antes de la geometría de referencia, como se muestra a continuación.

```
SELECT trgeometry 'SRID=25832;Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(0 0), 0.5)@2001 ↩
    -01-01';
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));SRID=25832;Pose(Point(0 0), 0.5)@2001 ↩
    -01-01';
```

La interpolación escalonada también se puede especificar de la misma manera que para otros tipos temporales:

```
SELECT trgeometry 'Interp=Step;Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001 ↩
    -01-01, Pose(Point(10 0), 0.0)@2001-01-02]';
```

Se admiten los formatos de transmisión estándar.

- Devuelve la representación de texto conocido (Well-Known Text o WKT)

  `asText({trgeometry,trgeometry[]} [, maxdecimaldigits int=15]) → text`

```
SELECT asText(trgeometry 'Polygon((1 1,2 2,3 1,1 1));Pose(Point(1.123456789 1.123456789),  ↩
    0.5)@2000-01-01', 6);
-- POLYGON((1 1,2 2,3 1,1 1));Pose(POINT(1.123457 1.123457),0.5)@2000-01-01
```

- Devuelve la representación extendida de texto conocido (Extended Well-Known Text o EWKT), prefijada con el SRID

  `asEWKT({trgeometry,trgeometry[]} [, maxdecimaldigits int=15]) → text`

```
SELECT asEWKT(trgeometry 'SRID=25832;Polygon((1 1,2 2,3 1,1 1));Pose(Point(1 1),0.5)@2000 ↩
    -01-01');
-- SRID=25832;POLYGON((1 1,2 2,3 1,1 1));Pose(POINT(1 1),0.5)@2000-01-01
```

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

  `asMFJSON(trgeometry [, options int=0 [, flags int=0 [, maxdecimaldigits int=15]]]) → text`

```
SELECT asMFJSON(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(0 0),0)@2001-01-01') ↩
    ;
```

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

  `asEWKB(trgeometry [, endian text]) → bytea`

  La función complementaria `asHexEWKB` devuelve los mismos bytes como un texto codificado en hexadecimal.

```
SELECT length(asEWKB(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(1 1), 0.5)@2001 ↩
    -01-01'));
-- 130
-- The bytes round-trip through the matching input function.
SELECT asText(trgeometryFromHexEWKB(asHexEWKB(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0)); ↩
    Pose(Point(1 1), 0.5)@2001-01-01')));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));Pose(POINT(1 1),0.5)@2001-01-01
```

- Cada formato de salida tiene un analizador complementario:

  `trgeometryFromText(text) → trgeometry`

  `trgeometryFromEWKT(text) → trgeometry`

  `trgeometryFromMFJSON(text) → trgeometry`

  `trgeometryFromEWKB(bytea) → trgeometry`

  `trgeometryFromHexEWKB(text) → trgeometry`

```
SELECT asText(trgeometryFromEWKT('SRID=4326;Polygon((0 0,1 0,1 1,0 1,0 0));
  Pose(Point(1 1), 0.5)@2001-01-01'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));Pose(POINT(1 1),0.5)@2001-01-01
SELECT asText(trgeometryFromHexEWKB(asHexEWKB(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0)); ↩
    Pose(Point(1 1), 0.5)@2001-01-01')));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));Pose(POINT(1 1),0.5)@2001-01-01
```

14.2. Constructores

Una `trgeometry` se puede construir directamente a partir de sus tipos componentes: una geometría de referencia más una pose temporal:

- Construir una geometría rígida temporal a partir de una geometría de referencia y una pose temporal

  `trgeometry(geometry, tpose) → trgeometry`

```
SELECT asText(trgeometry(
  geometry 'Polygon((0 0,1 0,1 1,0 1,0 0))',
  tpose 'Pose(Point(0 0), 0.0)@2001-01-01'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));Pose(POINT(0 0),0)@2001-01-01
```

  Ambos argumentos deben compartir el mismo SRID. La geometría de referencia puede ser un polígono o una superficie poliédrica; la pose puede ser de cualquier subtipo temporal.

- Agregar un único instante a una trgeometry existente, ampliando la trayectoria de pose subyacente

  `appendInstant(trgeometry, trgeometry [, maxdist float [, maxt interval]]) → trgeometry`

```
SELECT appendInstant(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(5 0), 0.0)@2001-01-02');
```

- Agregar una secuencia completa a una trgeometry existente

  `appendSequence(trgeometry, trgeometry) → trgeometry`

```
SELECT asText(appendSequence(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(20 0), 0.0)@2001-01-03, Pose(Point(30 0), 0.0)@2001-01-04]'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));
-- {[Pose(POINT(0 0),0)@2001-01-01, Pose(POINT(10 0),0)@2001-01-02],
--   [Pose(POINT(20 0),0)@2001-01-03, Pose(POINT(30 0),0)@2001-01-04]}
```

14.3. Conversión de tipos

La `trgeometry` se puede proyectar a sus tipos componentes o a una trayectoria de puntos puros.

- Devuelve la pose temporal subyacente

  `tpose(trgeometry) → tpose`

```
SELECT asText(tpose(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001 ↩
    -01-01, Pose(Point(10 0), 0.5)@2001-01-02]'));
-- [Pose(POINT(0 0),0)@..., Pose(POINT(10 0),0.5)@...]
```

- Devuelve la trayectoria del centroide (antena) como un punto temporal

  `tgeompoint(trgeometry) → tgeompoint`

```
SELECT asText(tgeompoint(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0) ↩
    @2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]'));
-- [POINT(0 0)@..., POINT(10 0)@...]
```

14.4. Accesores

Se aplican los accesores estándar de los tipos temporales: cada uno devuelve la geometría materializada en la marca de tiempo solicitada (la forma de referencia rotada y trasladada de acuerdo con la pose interpolada), o un valor de metadatos derivado de la trayectoria de pose.

- Devuelve la geometría materializada al inicio, al final o en una marca de tiempo elegida

  `startValue(trgeometry) → geometry`

  `endValue(trgeometry) → geometry`

  `valueAtTimestamp(trgeometry, timestamptz) → geometry`

```
SELECT asText(startValue(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0) ↩
    @2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0))

SELECT asText(valueAtTimestamp(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (10 0), 0.0)@2001-01-02]',
  timestamptz '2001-01-01 12:00:00'));
-- POLYGON((5 0,6 0,6 1,5 1,5 0))
```

  Tenga en cuenta que la rotación interpola linealmente a lo largo del camino angular más corto; una rotación de 90° entre dos instantes separados por un día devuelve un polígono rotado 45° en el punto medio.

- Devuelve el número de instantes, secuencias o marcas de tiempo distintos

  `numInstants(trgeometry) → integer`

  `numSequences(trgeometry) → integer`

  `numTimestamps(trgeometry) → integer`

```
SELECT numInstants(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- 2
SELECT numSequences(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- 1
SELECT numTimestamps(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- 2
```

- Devuelve el arreglo de instantes, secuencias o segmentos entre instantes constituyentes

  `instants(trgeometry) → trgeometry[]`

  `sequences(trgeometry) → trgeometry[]`

  `segments(trgeometry) → trgeometry[]`

```
SELECT array_length(
  instants(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));{Pose(Point(0 0), 0.0)@2001-01-01,  ↩
      Pose(Point(5 0), 0.5)@2001-01-02, Pose(Point(0 0), 0.0)@2001-01-03}'),
  1);
-- 3
```

- Devuelve el conjunto de puntos del centroide (antena) distintos vistos a lo largo de la trgeometry

  `points(trgeometry) → geomset`

```
SELECT asText(points(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));{Pose(Point(0 0), 0.0) ↩
    @2001-01-01, Pose(Point(5 5), 0.0)@2001-01-02, Pose(Point(0 0), 0.0)@2001-01-03}'));
-- {"POINT(0 0)", "POINT(5 5)"}
```

- Devuelve el ángulo de rotación como un flotante temporal

  `rotation(trgeometry) → tfloat`

```
SELECT asText(rotation(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0) ↩
    @2001-01-01, Pose(Point(0 0), 1.5)@2001-01-02]'));
-- [0@..., 1.5@...]
```

- Devuelve la geometría de referencia estática

  `geom(trgeometry) → geometry`

```
SELECT ST_AsText(geom(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(5 0), 0.5) ↩
    @2001-01-01'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0))
```

14.5. Área recorrida

La función `traversedArea` devuelve la unión de cada geometría materializada sobre el dominio temporal de la trgeometry: la huella espacial del cuerpo en movimiento.

- Devuelve la unión de los polígonos materializados a lo largo del dominio temporal de la trgeometry

  `traversedArea(trgeometry [, unary_union boolean = true]) → geometry`

```
SELECT ST_AsText(traversedArea(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (10 0), 0.0)@2001-01-02]'));
-- POLYGON((0 0, 11 0, 11 1, 0 1, 0 0))
```

  Con `unary_union = false` la función devuelve una GeometryCollection de los polígonos materializados por instante sin disolverlos, lo que resulta útil cuando quien llama desea hacer su propio posprocesamiento.

  Advertencia de muestreo: la implementación muestrea en los instantes de entrada, de modo que los segmentos de traslación pura son exactos, pero la cinta barrida entre dos instantes donde ocurre una rotación se aproxima mediante la envolvente convexa de los dos polígonos extremos. Agregar submuestras intermedias es un refinamiento futuro documentado.

14.6. Funciones espaciales

Estas funciones derivan un valor espacial del cuerpo en movimiento. `centroid` y `bodyPointTrajectory` devuelven puntos temporales que siguen una posición a lo largo del tiempo; `convexHull` devuelve una geometría estática.

- Devuelve el centroide del cuerpo en movimiento como un punto temporal

  `centroid(trgeometry) → tgeompoint`

```
SELECT asText(centroid(
  trgeometry 'Polygon((0 0, 1 0, 1 1, 0 1, 0 0));[Pose(Point(0 0),0)@2001-01-01, Pose( ↩
      Point(2 0),0)@2001-01-02]'));
-- Interp=Step;[POINT(0.5 0.5)@2001-01-01, POINT(2.5 0.5)@2001-01-02]
```

- Devuelve la envolvente convexa del área recorrida por el cuerpo en movimiento

  `convexHull(trgeometry) → geometry`

```
SELECT ST_GeometryType(convexHull(
  trgeometry 'Polygon((0 0, 1 0, 1 1, 0 1, 0 0));[Pose(Point(0 0),0)@2001-01-01, Pose( ↩
      Point(2 0),0)@2001-01-02]'));
-- ST_Polygon
```

- Devuelve la trayectoria en el marco del mundo de un punto del marco del cuerpo transportado por la geometría rígida en movimiento. El punto se expresa en el marco local de la forma de referencia y la pose variable en el tiempo se le aplica en cada instante.

  `bodyPointTrajectory(trgeometry, geometry) → tgeompoint`

```
SELECT asText(bodyPointTrajectory(
  trgeometry 'Polygon((0 0, 1 0, 1 1, 0 1, 0 0));[Pose(Point(0 0),0)@2001-01-01, Pose( ↩
      Point(2 0),0)@2001-01-02]',
  geometry 'Point(0 0)'));
-- Interp=Step;[POINT(0 0)@2001-01-01, POINT(2 0)@2001-01-02]
```

14.7. Métricas de movimiento

Estas funciones informan sobre el movimiento de una geometría rígida temporal a lo largo de su trayectoria del centroide.

- Devuelve la longitud total recorrida por el centroide, o su longitud acumulada a lo largo del tiempo

  `length(trgeometry) → float`

  `cumulativeLength(trgeometry) → tfloat`

```
SELECT length(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2001-01-01,  ↩
    Pose(Point(3 4),0)@2001-01-02]');
-- 5
```

- Devuelve la velocidad del centroide como un flotante temporal

  `speed(trgeometry) → tfloat`

```
SELECT speed(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- Interp=Step;[0.000115740740741@2001-01-01, 0.000115740740741@2001-01-02]
```

- Devuelve el centroide ponderado en el tiempo de la trayectoria del centroide como una geometría

  `twCentroid(trgeometry) → geometry`

```
SELECT ST_AsText(twCentroid(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0) ↩
    @2001-01-01, Pose(Point(3 4),0)@2001-01-02]'));
-- POINT(1.5 2)
```

14.8. Modificaciones

Se aplican las funciones estándar de modificación de secuencias: `round`, `setInterp`, `shiftTime`, `scaleTime`, `shiftScaleTim` `deleteTimestamp`, etc.

- Redondear todos los componentes numéricos a un número dado de posiciones decimales

  `round(trgeometry, integer) → trgeometry`

```
SELECT asText(round(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(1.123456789 1.123456789),  ↩
      0.123456)@2001-01-01',
  3));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));Pose(POINT(1.123 1.123),0.123)@...
```

- Convertir entre interpolaciones lineal y escalonada

  `setInterp(trgeometry, text) → trgeometry`

```
SELECT asText(setInterp(trgeometry 'Interp=Step;Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', 'linear'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));
-- {[Pose(POINT(0 0),0)@2001-01-01, Pose(POINT(0 0),0)@2001-01-02),
--  [Pose(POINT(10 0),0)@2001-01-02]}
```

- Desplazar, escalar, o desplazar y escalar el dominio temporal de una trgeometry

  `shiftTime(trgeometry, interval) → trgeometry`

  `scaleTime(trgeometry, interval) → trgeometry`

  `shiftScaleTime(trgeometry, interval, interval) → trgeometry`

```
SELECT asText(shiftTime(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', interval '1 day' ↩
    ));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));
-- [Pose(POINT(0 0),0)@2001-01-02, Pose(POINT(10 0),0)@2001-01-03]
SELECT asText(scaleTime(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', interval '2 days ↩
    '));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));
-- [Pose(POINT(0 0),0)@2001-01-01, Pose(POINT(10 0),0)@2001-01-03]
SELECT asText(shiftScaleTime(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', interval '1 day' ↩
    , interval '2 days'));
-- POLYGON((0 0,1 0,1 1,0 1,0 0));
-- [Pose(POINT(0 0),0)@2001-01-02, Pose(POINT(10 0),0)@2001-01-04]
```

14.9. Restricciones espaciales

Las funciones `atGeometry`, `minusGeometry`, `atStbox` y `minusStbox` restringen una trgeometry a (el complemento de) una geometría o una caja espaciotemporal. La semántica refleja la de `tpose`: una geometría rígida temporal está «en» una región en el instante `t` si y solo si su centroide (antena) en `t` se encuentra en esa región.

- Restringir una trgeometry a (el complemento de) una geometría

  `atGeometry(trgeometry, geometry) → trgeometry`

  `minusGeometry(trgeometry, geometry) → trgeometry`

```
SELECT temporalTime(atGeometry(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
    (4 0), 0.0)@2001-01-05]',
  geometry 'Polygon((1 -1,3 -1,3 1,1 1,1 -1))'));
-- {[2001-01-02, 2001-01-04]}

SELECT temporalTime(minusGeometry(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
    (4 0), 0.0)@2001-01-05]',
  geometry 'Polygon((1 -1,3 -1,3 1,1 1,1 -1))'));
-- {[2001-01-01, 2001-01-02), (2001-01-04, 2001-01-05]}
```

  La restricción contra una geometría vacía devuelve NULL para `atGeometry` y la entrada original para `minusGeometry`.

- Restringir una trgeometry a (el complemento de) una caja espaciotemporal

  `atStbox(trgeometry, stbox [, border_inc boolean = true]) → trgeometry`

  `minusStbox(trgeometry, stbox [, border_inc boolean = true]) → trgeometry`

```
SELECT temporalTime(atStbox(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (4 0), 0.0)@2001-01-05]',
  stbox 'STBOX X((1, -1), (3, 1))'));
-- {[2001-01-02, 2001-01-04]}

SELECT temporalTime(atStbox(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (4 0), 0.0)@2001-01-05]',
  stbox 'STBOX T([2001-01-02, 2001-01-04])'));
-- {[2001-01-02, 2001-01-04]}
```

Si la STBox solo tiene un componente temporal, la llamada se reduce a `atTime` estándar; si solo tiene un componente espacial, se conserva el dominio temporal.

14.10. Operaciones de distancia

La distancia entre una trgeometry y una geometría / pose / caja espaciotemporal / otra trgeometry se calcula sobre la geometría *materializada*: la forma de referencia rotada y trasladada de acuerdo con la pose. La distancia es exacta en cada marca de tiempo compartida; para entradas de secuencia, una bisección recursiva adaptativa emite marcas intermedias donde la trayectoria de distancia se desvía de una línea recta.

- Devuelve la distancia temporal entre dos trgeometries

  `{geometry,trgeometry} <-> {geometry,trgeometry} → tfloat`

  `tDistance({geometry,trgeometry}, {geometry,trgeometry}) → tfloat`

```
SELECT round(maxValue(tDistance(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(0 0), 0.0)@2001-01-01',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(5 0), 0.0)@2001-01-01')), 6);
-- 4

SELECT asText(tDistance(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (0 0), 0.0)@2001-01-02]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(2 0), 0.0)@2001-01-01, Pose(Point ↩
      (10 0), 0.0)@2001-01-02]'));
-- [1@2001-01-01 ..., 9@2001-01-02 ...]
```

  El despachador cubre cada combinación de subtipos: TINSTANT × TINSTANT, TINSTANT × TSEQUENCE asimétrica, TSEQUENCE × TSEQUENCE, TSEQUENCE × TSEQUENCESET, TSEQUENCESET × TSEQUENCESET. El núcleo de submuestreo adaptativo subyace a cada combinación continua; las combinaciones de instantes se materializan una vez. Los segmentos de traslación pura convergen en profundidad 0 (dos marcas extremas); los segmentos con mucha rotación emiten hasta 32 marcas por intervalo entre instantes. La cota sobre la aproximación interpolada LINEAL entre marcas es `1e-3` por construcción.

- Devuelve la distancia más pequeña entre dos trgeometries sobre su dominio temporal compartido

  `{geometry,trgeometry,stbox} |=| {geometry,trgeometry,stbox} → float`

  `nearestApproachDistance({geometry,trgeometry,stbox}, {geometry,trgeometry,stbox}) → float`

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(10 0), 0.0)@2001-01-02]'
       |=|
       trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(50 0), 0.0)@2001-01-01, Pose ↩
           (Point(50 0), 0.0)@2001-01-02]';
-- 39
```

El operador `|=|` se puede utilizar para búsquedas de vecino más cercano contra índices GiST o SP-GiST (véase Sección 14.17).

- Devuelve el instante en el que los dos argumentos están a su menor distancia mutua

  `nearestApproachInstant({geometry,trgeometry}, {geometry,trgeometry}) → trgeometry`

```
SELECT asText(nearestApproachInstant(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (10 0), 0.0)@2001-01-02]',
  geometry 'Point(5 5)'));
-- POLYGON((5 0,6 0,6 1,5 1,5 0));Pose(POINT(5 0),0)@2001-01-01 12:00:00+00
```

- Devuelve la línea que conecta los dos argumentos en su aproximación más cercana

  `shortestLine({geometry,trgeometry}, {geometry,trgeometry}) → geometry`

```
SELECT ST_AsText(shortestLine(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(0 0), 0.0)@2001-01-01',
  geometry 'Point(10 0)'));
-- LINESTRING(1 0,10 0)
```

14.11. Similitud

Las funciones de similitud comparan dos geometrías rígidas temporales a través de sus trayectorias del centroide.

- Devuelve la distancia de Hausdorff discreta, la distancia de Fréchet discreta, o la distancia de deformación dinámica del tiempo (Dynamic Time Warp) entre dos geometrías rígidas temporales

  `hausdorffDistance(trgeometry, trgeometry) → float`

  `frechetDistance(trgeometry, trgeometry) → float`

  `dynTimeWarpDistance(trgeometry, trgeometry) → float`

```
SELECT round(frechetDistance(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2000-01-01, Pose(Point(2 ↩
      0),0)@2000-01-03]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2000-01-01, Pose(Point(0 ↩
      2),0)@2000-01-03]'), 6);
-- 2.828427
SELECT round(hausdorffDistance(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2000-01-01, Pose(Point(2 ↩
      0),0)@2000-01-03]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2000-01-01, Pose(Point(0 ↩
      2),0)@2000-01-03]'), 6);
-- 2
```

- Devuelve los pares de correspondencia entre dos geometrías rígidas temporales con respecto a la distancia de Fréchet discreta o a la distancia de deformación dinámica del tiempo (Dynamic Time Warp), como un conjunto de pares de índices de instante `(i,j)`

  `frechetDistancePath(trgeometry, trgeometry) → {(i,j)}`

  `dynTimeWarpPath(trgeometry, trgeometry) → {(i,j)}`

```
SELECT frechetDistancePath(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
    [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
    [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(20 0), 0.0)@2001-01-02]');
-- (0,0)
-- (1,1)
```

```
SELECT dynTimeWarpPath(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
    [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
    [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(20 0), 0.0)@2001-01-02]');
-- (0,0)
-- (1,1)
```

14.12. Comparaciones

Los operadores de igualdad y de siempre / alguna vez comparan dos trgeometries sobre la geometría materializada: tanto la forma de referencia como la trayectoria de pose deben coincidir en cada instante compartido.

- Probar la (des)igualdad exacta de dos trgeometries

  `trgeometry = trgeometry → boolean`

  `trgeometry <> trgeometry → boolean`

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]' = trgeometry ' ↩
      Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]';
-- t
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]' <> trgeometry ' ↩
      Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]';
-- f
```

- Probar si la geometría materializada es alguna vez (no) igual al operando en algún momento

  `{geometry,trgeometry} ?= {geometry,trgeometry} → boolean`

  `{geometry,trgeometry} ?<> {geometry,trgeometry} → boolean`

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(5 0), 0.0)@2001-01-02]'
       ?= geometry 'Polygon((5 0,6 0,6 1,5 1,5 0))';
-- t
```

- Probar si la geometría materializada es siempre (no) igual al operando

  `{geometry,trgeometry}%= {geometry,trgeometry} → boolean`

  `{geometry,trgeometry}%<> {geometry,trgeometry} → boolean`

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]' %= trgeometry ' ↩
      Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]';
-- t
```

14.13. Operadores de caja delimitadora

Los operadores topológicos y de posición comparan las cajas delimitadoras espaciotemporales de los operandos; ambos pueden ser una `trgeometry`, un `stbox`, una `geometry` o un `tstzspan`.

- Operadores topológicos estándar sobre cajas delimitadoras

```
SELECT trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose( ↩
    Point(10 0), 0.0)@2001-01-02]'
       &&
       stbox 'STBOX X((5, -1), (15, 1))';
-- t
```

- Operadores de posición espacial y temporal

  El conjunto completo <<, >>, &<, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, #&> acepta `trgeometry` en cualquiera de los lados.

14.14. Mosaicos

Estas funciones devuelven las cajas espaciotemporales de una cuadrícula multidimensional que la geometría rígida temporal interseca, calculadas sobre su trayectoria del centroide.

- Devuelve las cajas espaciotemporales de una geometría rígida temporal dividida con respecto a una cuadrícula espacial, temporal o espaciotemporal

  `spaceBoxes(trgeometry, xsize float [, ysize float [, zsize float]], sorigin geometry = 'Point(0 0 0)', bitmatrix bool = true, borderInc bool = true) → stbox[]`

  `timeBoxes(trgeometry, interval, torigin timestamptz = '2000-01-03', bitmatrix bool = true, borderInc bool = true) → stbox[]`

  `spaceTimeBoxes(trgeometry, xsize float [, ysize float [, zsize float]], interval, sorigin geometry = 'Point(0 0 0)', torigin timestamptz = '2000-01-03', bitmatrix bool = true, borderInc bool = true) → stbox[]`

```
SELECT array_length(spaceBoxes(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0),0)@2001-01-01, Pose(Point(10 ↩
      0),0)@2001-01-02]',
  2.0), 1);
-- 6
```

14.15. Cajas delimitadoras

Estas funciones descomponen una geometría rígida temporal en las cajas delimitadoras o los lapsos de tiempo de sus segmentos o de un número solicitado de contenedores.

- Devuelve el arreglo de cajas espaciotemporales o lapsos de tiempo de los segmentos de una geometría rígida temporal

  `stboxes(trgeometry) → stbox[]`

  `spans(trgeometry) → tstzspan[]`

```
SELECT stboxes(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- {"STBOX XT(((0,0),(11,1)),[2001-01-01, 2001-01-02])"}
SELECT spans(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- {"[2001-01-01, 2001-01-02]"}
```

- Devuelve las cajas espaciotemporales o los lapsos de tiempo de una geometría rígida temporal dividida en un número dado de contenedores, o con un número dado de segmentos por contenedor

  `splitNStboxes(trgeometry, integer) → stbox[]`

  `splitEachNStboxes(trgeometry, integer) → stbox[]`

`splitNSpans(trgeometry, integer) → tstzspan[]`

`splitEachNSpans(trgeometry, integer) → tstzspan[]`

```
SELECT splitNStboxes(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', 2);
-- {"STBOX XT(((0,0),(11,1)),[2001-01-01, 2001-01-02])"}
SELECT splitNSpans(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]', 2);
-- {"[2001-01-01, 2001-01-02]"}
```

14.16. Agregaciones

- Número de trgeometries solapadas en cada instante, equivalente a `tcount` sobre la tpose subyacente

  `tcount(trgeometry) → tint`

```
SELECT tcount(trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));
  [Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point(10 0), 0.0)@2001-01-02]');
-- {[1@2001-01-01, 1@2001-01-02]}
```

- Caja delimitadora agregada de una columna de trgeometries

  `extent(trgeometry) → stbox`

```
SELECT extent(temp) FROM tbl_trgeometry;
```

- Combinar dos trgeometries con la misma geometría de referencia en un único conjunto de secuencias

  `merge(trgeometry, trgeometry) → trgeometry`

  `mergeAgg(trgeometry) → trgeometry`

```
SELECT asText(merge(
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-01, Pose(Point ↩
      (5 0), 0.0)@2001-01-02]',
  trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));[Pose(Point(0 0), 0.0)@2001-01-04, Pose(Point ↩
      (5 0), 0.0)@2001-01-05]'));
```

14.17. Indexación

Hay tres tipos de índices disponibles sobre columnas `trgeometry`. Cada uno utiliza la caja delimitadora espaciotemporal de la geometría materializada y admite consultas topológicas, de posición y de distancia KNN.

- `GIST(temp)` — GiST de árbol R.
- `SPGIST(temp trgeometry_quadtree_ops)` — SP-GiST de árbol cuádruple.
- `SPGIST(temp trgeometry_kdtree_ops)` — SP-GiST de árbol kd.

```
CREATE INDEX tbl_trgeometry_gist_idx
  ON tbl_trgeometry
  USING GIST(temp);

-- topological lookup
SELECT count(*) FROM tbl_trgeometry
WHERE temp && stbox 'STBOX XT(((0, 0), (10, 10)),[2001-01-01, 2001-01-02])';

-- KNN (ordered scan)
```

```
SELECT mmsi
FROM tbl_trgeometry
ORDER BY temp <-> trgeometry 'Polygon((0 0,1 0,1 1,0 1,0 0));Pose(Point(100 100), 0.0)@2001 ↩
    -01-01'
LIMIT 10;
```

La variante de árbol kd suele ganar en cargas de trabajo dominadas por consultas KNN con una cota de ordenación ajustada; la variante de árbol cuádruple es más uniforme. El GiST de árbol R es el valor predeterminado seguro para cargas de trabajo mixtas de lectura/escritura.

14.18. Guía de producción

Esta sección recoge el conocimiento operativo que las cargas de trabajo de producción necesitan además de la referencia de funciones: cuándo `trgeometry` es el tipo adecuado en relación con sus vecinos (`tpose` y `tgeometry`), las convenciones de marco que la implementación impone y los riesgos numéricos que las cargas de trabajo reales de cuerpos rígidos encuentran. La mayoría de estos se exponen por separado a lo largo del capítulo; el objetivo aquí es disponer de un único lugar donde un nuevo contribuyente pueda leer de principio a fin antes de instrumentar una aplicación.

14.18.1. Cuándo usar trgeometry frente a tpose frente a tgeometry

Tres tipos temporales abarcan el espacio de diseño de cuerpo rígido / forma arbitraria. Elija el tipo más estrecho que se ajuste a la carga de trabajo: los tipos más estrechos pagan menos almacenamiento y producen cajas delimitadoras más nítidas para el índice espacial.

- `tpose` — solo posición y orientación temporal; el cuerpo se trata como un punto con un rumbo. Úselo cuando la forma sea irrelevante para la consulta (seguimiento cinemático, sustitutos de colisión basados solo en el rumbo).
- `trgeometry` — una única forma de referencia estática que se traslada y rota a lo largo del tiempo según una pose temporal. Úselo cuando la forma *importa* pero es rígida: barcos, vehículos, drones con geometría fija, zonas de exclusión que se ajustan al cuerpo. La geometría de referencia se almacena una vez por fila, no por instante, de modo que el almacenamiento está dominado por la `tpose` subyacente.
- `tgeometry` — geometría arbitraria temporal cuya forma misma varía a lo largo del tiempo. Úselo cuando la forma no sea rígida: franjas de viento de tormentas, plumas en expansión, polígonos en evolución.

Una `trgeometry` se puede proyectar a su `tpose` subyacente (trayectoria del centroide) o a una `tgeometry` totalmente materializada mediante las conversiones de Sección 14.3. La dirección inversa (elevar una `tgeometry` de vuelta a una `trgeometry`) no está en general bien definida, ya que los cambios de forma arbitrarios no se factorizan limpiamente en rotación + traslación de una referencia fija.

14.18.2. Convenciones de marco

- *Geometría de referencia solo 2D*. La forma de referencia es un polígono 2D o una superficie poliédrica; el componente de rotación de la pose es el ángulo de guiñada (yaw) 2D de Capítulo 11. No hay representación de cuerpo rígido 3D en este tipo; las cargas de trabajo que necesiten orientación 3D completa (cuaternión) sobre una forma 3D deben componer `tpose` con el paso de materialización de forma explícita.
- *Geometría de referencia única compartida*. Cada instante de una `trgeometry` comparte la misma forma de referencia; solo varía la pose. La forma se almacena una vez por fila, no por instante.
- *Requisito de mismo SRID*. La geometría de referencia y la pose temporal deben compartir un SRID; el constructor rechaza las discrepancias con el error estándar de MEOS `ensure_same_srid`. La geometría materializada hereda este SRID.
- *Semántica de restricción espacial basada en el centroide*. `atGeometry`, `minusGeometry`, `atStbox` y `minusStbox` prueban el centroide de la forma materializada contra la región, no la forma materializada en sí. Una trgeometry cuyo centroide se encuentra fuera de una región pero cuyo cuerpo se extiende dentro de la región se *excluye*. Esto refleja la semántica subyacente de `tpose` (una pose es un punto) y es una decisión de diseño deliberada; las cargas de trabajo que necesiten contención de la forma completa deben materializar primero a `tgeometry`.

14.18.3. Riesgos numéricos

Cuatro riesgos afectan de forma fiable a las cargas de trabajo reales de cuerpos rígidos. La implementación mitiga cada uno de la siguiente manera:

- *Entradas con SRID mixtos*. Construir una `trgeometry` a partir de una geometría de referencia y una `tpose` en SRID diferentes produciría silenciosamente posiciones materializadas erróneas si los SRID no se comprobaran.

  Mitigación: el constructor y cada operador entre tipos pasan por `ensure_same_srid`, que genera un error que nombra ambos SRID. La ruta de error es ruidosa y explícita; no hay transformación implícita.

- *Advertencia de muestreo de traversedArea*. La función `traversedArea` muestrea en los instantes de entrada y los conecta con la envolvente convexa de los dos polígonos extremos. Los segmentos de traslación pura son exactos, pero las rotaciones entre instantes se aproximan: para una rotación de 90° entre dos instantes, la cinta barrida es más ancha de lo que captura la envolvente convexa.

  Mitigación: documentado en Sección 14.5. Las cargas de trabajo que necesiten cotas ajustadas del área barrida deben sobremuestrear la `tpose` de entrada antes de construir la `trgeometry`; el patrón de `setInterp` + inserción de subinstantes del capítulo de `tpose` es el enfoque estándar. El submuestreo adaptativo dentro de `traversedArea` es un refinamiento futuro documentado.

- *Tolerancia de submuestreo adaptativo en la distancia materializada*. Los núcleos de distancia continua (`tDistance`, `nearestApp`
  materializan la forma rígida por cada intervalo entre instantes y emiten marcas intermedias donde la trayectoria de distancia se desvía de una línea recta. El tope de profundidad es de 32 marcas por intervalo y la aproximación interpolada LINEAL entre marcas está acotada por `1e-3` (en unidades del CRS).

  Mitigación: la cota de `1e-3` es una decisión de diseño documentada. Los segmentos de traslación pura convergen en profundidad 0; los segmentos con mucha rotación usan el presupuesto completo de 32 marcas. Las cargas de trabajo que necesiten una precisión más nítida en un operador específico deben preprocesar las poses de entrada a instantes más densos en lugar de confiar en una recursión más ajustada.

- *Sorpresa de la restricción espacial basada en el centroide*. `atGeometry(trgeometry, geometry)` prueba el centroide de la pose contra la región, no la forma materializada. Un barco largo cuya popa está en un puerto pero cuya proa ha cruzado el límite del puerto se informará como dentro del puerto mientras su centroide esté dentro.

  Mitigación: documentado como una decisión de diseño de nivel D en Sección 14.9. Las cargas de trabajo que necesiten semántica de forma completa deben convertir mediante `trgeometry::tgeometry` antes de aplicar la restricción espacial. La conversión materializa un polígono por instante y es, en consecuencia, más costosa.

14.18.4. Durabilidad y almacenamiento

La representación en disco de una `trgeometry` es el `GSERIALIZED` de referencia seguido de la carga útil de la pose temporal. La disposición de bytes es estable: `asBinary(trgeometry)` / `trgeometryFromBinary(bytea)` realiza un ciclo de ida y vuelta que preserva el valor bit a bit (incluido el SRID de la forma de referencia, las banderas Z y M, y el subtipo de la `tpose` subyacente), y las serializaciones WKB / EWKB / HexEWKB siguen el patrón estándar de PostGIS de bandera de endianidad seguida de carga útil.

El par `asMFJSON(trgeometry)` / `trgeometryFromMFJSON(text)` es bidireccional. Cada instante se representa como `{"geometry":<reference-geojson>,"values":[<pose-payload>]}`: la geometría de referencia usa la forma GeoJSON estándar de PostGIS y la carga útil de la pose sigue la disposición de la clase de conformidad OGC GeoPose Basic-YPR (`{"position":...,"quaternion":...}` para 3D / `{"position":...,"rotation":...}` para 2D). La geometría de referencia se emite una vez por instante en el esquema actual; un trabajo futuro de esquema podría elevarla a un encabezado a nivel de capítulo para evitar la repetición por instante en secuencias largas.

La salida de `pg_dump` de una tabla que contiene columnas `trgeometry` usa la representación WKT de forma predeterminada; `pg_dump --binary-upgrade` usa la representación WKB. Ambas son estables en el ciclo de ida y vuelta entre actualizaciones de versión mayor de PostgreSQL.

Capítulo 15

Tipos JSON en MobilityDB

PostgreSQL proporciona dos tipos para almacenar datos JSON: `json` y `jsonb`. El tipo `json` almacena una copia exacta del texto de entrada, que debe volver a analizarse cada vez que se procesa. Por otro lado, el tipo `jsonb` almacena los datos JSON en un formato binario descompuesto que hace que su entrada sea ligeramente más lenta debido a la sobrecarga de conversión añadida, pero significativamente más rápida de procesar, ya que no es necesario volver a analizarla. El tipo `jsonb` también admite indexación.

En MobilityDB, los tipos `textset` y `ttext` se utilizan para representar, respectivamente, conjuntos de valores JSON y valores JSON temporales. Además, el tipo `jsonb` sirve como tipo base para definir los tipos `jsonbset` y `tjsonb`. La mayoría de las funciones y operadores descritos en los capítulos anteriores para los tipos conjunto y temporal también son aplicables a los tipos JSON correspondientes. Además, hay funciones específicas definidas para estos tipos, que se derivan de las funciones correspondientes de los tipos `json` y `jsonb`.

En este capítulo, describimos los tipos JSON de MobilityDB y sus operaciones asociadas. Remitimos a la documentación de PostgreSQL para una explicación detallada de los tipos `json` y `jsonb` y su funcionalidad. Hemos buscado habilitar la misma sintaxis de las funciones originales de PostgreSQL para los tipos correspondientes de MobilityDB, como se ilustra en este documento.

Considere por ejemplo el operador `->`, que extrae un campo de objeto JSON con una clave dada.

```
SELECT jsonb '{"unit": "km", "speed": 10}' -> text 'speed';
-- 10
```

Aplicar el mismo operador a un conjunto JSONB y a un valor JSONB temporal produce los siguientes resultados

```
SELECT jsonbset '{"{\"unit\": \"km\", \"speed\": 10}",
  "{\"unit\": \"km\", \"speed\": 20}"}' -> text 'speed';
-- {"10", "20"}
SELECT tjsonb '[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]' -> text 'speed';
-- {[10@2001-01-01, 20@2001-01-02]}
```

que se obtienen aplicando el operador de PostgreSQL `->` a cada elemento del conjunto JSONB y a cada instante del valor JSONB temporal.

15.1. Operaciones sobre conjuntos JSONB

Al manipular colecciones JSON de estructura variable, puede ocurrir que un elemento esté definido en algunos de los documentos pero no en todos. PostgreSQL proporciona el *modo lax* para este propósito, donde el argumento `null_value_treatment` determina el comportamiento en el caso en que una función devuelve un valor NULL. El argumento puede tomar uno de los siguientes valores: `'raise_exception'`, `'use_json_null'`, `'delete_key'`, `'return_target'`, donde `'use_json_null'` es el valor por defecto. En PostgreSQL el modo lax se admite solo para operaciones de *actualización* (no de *consulta*) con la función `jsonb_set_lax`. En MobilityDB, hemos mantenido la misma semántica para la función correspondiente `jsonbset_set_lax`

pero habilitamos un comportamiento similar para *todas* las operaciones JSON que pueden devolver un valor nulo, salvo que reemplazamos el valor `'return_target'` por `'return_null'`, ya que el primero no tiene sentido para operaciones sobre conjuntos. Para operadores como `->`, se utiliza el valor por defecto `'use_json_null'` y no se puede cambiar, mientras que para las funciones correspondientes `jsonbset_object_field`, el último argumento especifica el comportamiento en el caso en que la función devuelve NULL. Ilustramos este comportamiento para la función `jsonbset_object_field` a continuación.

- Extraer un campo de objeto JSON especificado por una clave

  `{textset,jsonbset} -> text → {textset,jsonbset}`

  `jsonbset ->> text → textset`

  `jsonbset_object_field(jsonbset,text,null_handle text='use_json_null') → jsonbset`

  `jsonbset_object_field_text(jsonbset,text,null_handle text='use_json_null') → textset`

```
SELECT textset '{[{"unit": "km", "speed": 10},
  {"unit": "km", "speed": 20}]}' -> text 'speed';
-- {["10", "20"]}
SELECT jsonbset '[{"position":"Point(1 1)", "speed":10},
  {"position":"Point(2 2)", "speed":20}]' ->> text 'position';
-- ["Point(1 1)", "Point(2 2)"]
SELECT textset '{[{"unit": "km", "speed": 10},
{"unit": "km"}, {"unit": "km", "speed": 10}]}' -> text 'speed';
-- {[10, null, 10]}
```

  Mostramos a continuación el comportamiento de la función `jsonbset_object_field` para los posibles valores del argumento `null_handle`. Todas las funciones JSON que pueden devolver un valor NULL se comportan de forma similar en MobilityDB.

```
SELECT jsonbset_object_field(textset '{[{"unit": "km", "speed": 10},
  {"unit": "km"}, {"unit": "km", "speed": 10}]}', 'speed',
  'raise_exception');
-- ERROR:  The lifted operation returned NULL
SELECT jsonbset_object_field(textset '{[{"unit": "km", "speed": 10},
  {"unit": "km"}, {"unit": "km", "speed": 10}]}', 'speed',
  'use_json_null');
-- {[10, null, 10]}
SELECT jsonbset_object_field(textset '{[{"unit": "km", "speed": 10},
  {"unit": "km"}, {"unit": "km", "speed": 10}]}', 'speed',
  'delete_key');
-- {[10, 10), [10]}
SELECT jsonbset_object_field(textset '{[{"unit": "km", "speed": 10},
  {"unit": "km"}, {"unit": "km", "speed": 10}]}', 'speed',
  'return_null');
-- NULL
```

- Extraer un campo de objeto JSON especificado por una ruta

  `{textset,jsonbset} #> text[] → {textset,jsonbset}`

  `jsonbset_extract_path(jsonbset,text[],null_handle text='use_json_null') → jsonbset`

  `jsonbset_extract_path_text(jsonbset,text[],null_handle text='use_json_null') → textset`

```
SELECT textset '[{"speed": {"unit": "km", "value": 10}},
  {"speed": {"unit": "km", "value": 20}}]' #> ARRAY[text 'speed', 'value'];
-- {["10", "20"]}
SELECT jsonbset '[{"position":"Point(1 1)", "PoIs":["Grand Place", "La Bourse"]},
  {"position":"Point(2 2)", "PoIs":["Palais Royal", "Manneken Pis"]}]' #>>
  ARRAY[text 'PoIs', '1'];
-- ["La Bourse", "Manneken Pis"]
```

- Extraer un elemento de un arreglo JSON

  `jsonbset_array_element(jsonbset,integer,null_handle text='use_json_null') → jsonbset`

  `jsonbset_array_element_text(jsonbset,integer,null_handle text='use_json_null') → jsonbse`

```
SELECT jsonbset_array_element(textset '[["Grand Place", "La Bourse"],
  ["Palais Royal", "Manneken Pis"]]', 0);
-- ["Grand Place", "Palais Royal"]
SELECT jsonbset_array_element(jsonbset '[["Grand Place", "La Bourse"],
  ["Palais Royal", "Manneken Pis"]]', 1);
-- ["La Bourse", "Manneken Pis"]
```

- Extraer un valor alfanumérico de un conjunto JSONB dado por una clave

  `intset(jsonbset,text,null_handle text='raise_exception') → tint`

  `floatset(jsonbset,text,null_handle text='raise_exception') → tfloat`

  `textset(jsonbset,text,null_handle text='raise_exception') → textset`

  Tenga en cuenta que el valor `use_json_null` no puede utilizarse para las funciones anteriores y por lo tanto se utiliza el valor por defecto `'raise_exception'`.

```
SELECT intset(jsonbset '{"{\"speed\": 10, \"units\": \"km/h\"}",
  "{\"speed\": 20, \"units\": \"km/h\"}"}', 'speed');
-- {10, 20}
SELECT floatset(jsonbset '{"{\"speed\": 10, \"units\": \"km/h\"}",
  "{\"speed\": 20, \"units\": \"km/h\"}", "{\"speed\": 25}"}', 'speed');
-- {10, 20, 25}
SELECT textset(jsonbset '{"{\"road\": \"Bvd Gén. Jacques\", \"category\": \"primary\"}",
  "{\"road\": \"Bvd de la Cambre\", \"category\": \"residential\"}"}', 'category');
-- {primary, residential}
```

- Concatenación JSONB temporal

  `jsonbset || {jsonb,jsonbset} → jsonbset`

```
SELECT jsonbset '{[{"speed":10}, {"speed":20}]}' ||
  '{"unit":"km"}'::jsonb;
-- {[{"unit": "km", "speed": 10}, {"unit": "km", "speed": 20}]}
SELECT jsonbset '{[{"speed":10}, {"speed":20}]}' ||
  jsonbset '{[{"Position":"Point(1 1)"}, {"Position":"Point(2 2)"}]}';
/* {[{"speed": 10, "Position": "Point(1 1)"},
     {"speed": 20, "Position": "Point(2 2)"}]} */
```

- Eliminación JSONB temporal

  `jsonbset - {int,text,text[]} → jsonbset`

  `jsonbset #- text[] → jsonbset`

```
SELECT jsonbset '{[{"unit": "km", "speed": 10},
  {"unit": "km", "speed": 20}]}' - 'unit';
-- {[{"speed": 10}, {"speed": 20}]}
SELECT jsonbset '[["Grand Place", "La Bourse"],
  ["Palais Royal", "Manneken Pis"]]' - 1;
-- [["Grand Place"], ["Palais Royal"]]
SELECT jsonbset '[{"location":"Point(1 1)", "PoIs": ["Grand Place", "La Bourse"],
  "location":"Point(2 2)", "PoIs": ["Palais Royal", "Manneken Pis"]]' #-
  ARRAY[text "\"PoIs\"", "1"];
-- ["La Bourse", "Manneken Pis"]
```

  Como se muestra a continuación, el resultado de una operación de eliminación puede ser un registro vacío o un arreglo vacío. Para eliminar estos valores, puede utilizarse la función `minusValue`.

```
SELECT jsonbset '{[{"unit": "km", "speed": 10, "light": true},
  {"unit": "km", "speed": 20}]}' - ARRAY[text 'unit', 'speed'];
-- {[{"light": true}, {}]}
SELECT jsonbset '[["Grand Place", "La Bourse"],
  ["Palais Royal"]]' - 0;
-- [["La Bourse"], []]
```

```
SELECT minusValues(jsonbset '{[{"unit": "km", "speed": 10, "light": true},
  {"unit": "km", "speed": 20}]}' - ARRAY[text 'unit', 'speed'],
  jsonbset '{"[]","{}"}');
--  {[{"light": true}, {"light": true})}
SELECT minusValues(jsonbset '[["Grand Place", "La Bourse"],
  ["Palais Royal"]]' - 0,  jsonbset '{"[]","{}"}');
-- {[["La Bourse"], ["La Bourse"])}
```

- Existencia JSONB temporal

  `jsonbset ? jsonb → tbool`

  `jsonbset ?| jsonb[] → tbool`

  `jsonbset ?& jsonb[] → tbool`

  Como en PostgreSQL, los operadores `?|` y `?&` comprueban, respectivamente, si *alguna* o *todas* las cadenas del arreglo de texto existen como claves de nivel superior o elementos de arreglo.

```
SELECT jsonbset '"{\"geom\": \"Point(1 1)\"}"' ? text 'geom';
-- t
SELECT jsonbset '{{"geom": "Point(1 1)"}, {"geom": "Point(2 2)"},
  {"geom": "Point(1 1)"}}' ?| ARRAY[text 'geom'];
-- {t, t, t}
SELECT jsonbset '[{"speed": 10, "units": "km/h"},
  {"speed": 20, "units": "km/h"}]' ?& ARRAY['speed', 'units'];
-- {[t, t]}
```

- Contiene/contenido JSONB temporal

  `{jsonbset,jsonb} @> {jsonbset,jsonb} → tbool`

  `{jsonbset,jsonb} <@ {jsonbset,jsonb} → tbool`

```
SELECT jsonbset '"{\"geom\": \"Point(1 1)\"}"' @> jsonb '{"geom": "Point(1 1)"}';
-- t
SELECT jsonbset '{{"geom": "Point(1 1)"}, {"geom": "Point(2 2)"},
  {"geom": "Point(1 1)"}}' @> jsonb '{"geom": "Point(1 1)"}';
-- {t, f, t}
SELECT jsonb '{"geom": "Point(1 1)"}' <@ jsonbset '{[{"geom": "Point(1 1)"},
  {"geom": "Point(1 1)"}, {"geom": "Point(1 1)"}],
  [{"geom": "Point(2 2)"}, {"geom": "Point(2 2)"}]}';
-- {[t, t], [f, f]}
```

- Asignación JSONB temporal

  `jsonbset_set(jsonbset,path text[],jsonb,create boolean=true) → jsonbset`

  `jsonbset_set_lax(jsonbset,path text[],jsonb,create boolean=true,handle_null text) → jsonbset`

  Devuelve el valor `jsonbset` con el elemento especificado por la ruta reemplazado por `jsonb`, o añadido si `create` es verdadero y el elemento no existe. Todos los pasos anteriores de la ruta deben existir, o se devuelve el objetivo sin cambios. Como ocurre con los operadores orientados a rutas, los enteros negativos que aparecen en la ruta cuentan desde el final de los arreglos JSON. Si el último paso de la ruta es un índice de arreglo que está fuera de rango, y create_if_missing es verdadero, el nuevo valor se añade al principio del arreglo si el índice es negativo, o al final del arreglo si es positivo.

La función `jsonbset_set_lax` se comporta de forma idéntica a `jsonbset_set` si el valor dado no es NULL; de lo contrario, se comporta según el valor de `handle_null`, que debe ser uno de `'raise_exception'`, `'use_json_null'`, `'delete_key'` o `'return_source'`. Como se indicó anteriormente, mantuvimos el comportamiento de PostgreSQL para esta función habilitando el valor `'return_source'`, mientras que para todas las demás operaciones de *consulta*, este valor se ha reemplazado por `'return_null'`.

```
SELECT jsonbset_set(jsonbset '[{"speed":10}, {"speed":20}]',
  ARRAY['units'], '"km/h"'::jsonb);
-- [{"speed": 10, "units": "km/h"}, {"speed": 20, "units": "km/h"}]
SELECT jsonbset_set(jsonbset '[{"speed":10}, {"speed":20}]',
  ARRAY['units'], '"km/h"'::jsonb, false);
-- [{"speed": 10}, {"speed": 20}]
SELECT jsonbset_set(jsonbset '[{"speed": 10, "units": "km/h"},
  {"speed": 20, "units": "km/h"}]', ARRAY['units'], '"mi/h"'::jsonb);
-- [{"speed": 10, "units": "mi/h"}, {"speed": 20, "units": "mi/h"}]
SELECT jsonbset_set_lax(jsonbset '[{"speed": 10, "units": "km/h"},
  {"speed": 20}]', ARRAY['units'], 'null'::jsonb, true, 'delete_key');
-- [{"speed": 10, "units": "km/h"}, {"speed": 20}]
```

- Inserción JSONB temporal

  `jsonbset_insert(jsonbset,path text[],jsonb,after boolean=false)` → `jsonbset`

  Devuelve el valor `jsonbset` con `jsonb` insertado. Si el elemento especificado por la ruta es un elemento de arreglo, el nuevo valor se insertará antes de ese elemento si `after` es falso, o después en caso contrario. Si el elemento especificado por la ruta es un campo de objeto, el nuevo valor se insertará solo si el objeto no contiene ya esa clave. Todos los pasos anteriores de la ruta deben existir, o se devuelve el objetivo sin cambios. Como ocurre con los operadores orientados a rutas, los enteros negativos que aparecen en la ruta cuentan desde el final de los arreglos JSON. Si el último paso de la ruta es un índice de arreglo que está fuera de rango, el nuevo valor se añade al principio del arreglo si el índice es negativo, o al final del arreglo si es positivo.

```
SELECT jsonbset_insert(jsonbset '[{"speed":10}, {"speed":20}]',
  ARRAY['units'], '"km/h"'::jsonb);
-- [{"speed": 10, "units": "km/h"}, {"speed": 20, "units": "km/h"}]
SELECT jsonbset_insert(jsonbset '[{"speed":10}, {"speed":20}]',
  ARRAY['units'], '"km/h"'::jsonb, false);
-- [{"speed": 10}, {"speed": 20}]
SELECT jsonbset_insert(jsonbset '[{"speed": 10, "units": "km/h"},
  {"speed": 20, "units": "km/h"}]', ARRAY['units'], '"mi/h"'::jsonb);
-- [{"speed": 10, "units": "mi/h"}, {"speed": 20, "units": "mi/h"}]
```

- Devolver un conjunto JSONB sin nulos

  `jsonbset_strip_nulls(jsonbset,strip_in_arrays bool=false)` → `jsonbset`

  El último argumento indica si los elementos de arreglo nulos también se eliminan. Los valores nulos simples nunca se eliminan.

```
SELECT jsonbset_strip_nulls(textset '[{"speed": 10, "lights": null},
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse", null]}]');
/* [{"speed": 10},
   {"PoIs": ["Grand Place", "La Bourse", null], "speed": 20}] */
SELECT jsonbset_strip_nulls(jsonbset '[{"speed": 10, "lights": null},
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse", null]}]', true);
/* [{"speed": 10},
   {"PoIs": ["Grand Place", "La Bourse"], "speed": 20}] */
SELECT jsonbset_strip_nulls(jsonbset
  '{{"road": "Bvd Gén. Jacques", "category": "primary"},
  {"road": "Bvd de la Cambre", "category": "primary"},
  {"road": "rue de l''Abbaye", "category": null}}');
/* [{"road": "Bvd Gén. Jacques", "category": "primary"},
   {"road": "Bvd de la Cambre", "category": "primary"},
   {"road": "rue de l'Abbaye"}] */
```

Como se muestra a continuación, la función no elimina los valores nulos simples. Para este propósito puede utilizarse la función `minusValue`.

```
SELECT jsonbset_strip_nulls(textset '[{"speed": 10}, null,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}]');
/* [{"speed":10}, null,
   {"speed":20,"PoIs":["Grand Place","La Bourse"]}] */
SELECT minusValues(textset '[{"speed": 10}, null,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}]', 'null');
/* {[{"speed": 10}, {"speed": 10}),
   [{"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}]}
```

- ¿Devuelve la ruta JSON algún elemento para el conjunto JSONB especificado?

  `jsonbset @? jsonpath → tbool`

  `jsonbset_path_exists(jsonbset,vars jsonb='{}',silent boolean=false) → tbool`

  `jsonbset_path_exists_tz(jsonbset,vars jsonb='{}',silent boolean=false) → tbool`

  El operador `@?` suprime los siguientes errores: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Las funciones anteriores también pueden suprimir estos tipos de errores estableciendo el último argumento en verdadero. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT jsonbset '[{"speed":10}, {"speed": 20, "units": "km/h"}]' @?
  '$.units ? (@ == "km/h")';
-- [f, t]
SELECT jsonbset_path_exists(jsonbset
  '{{"road": "Bvd Gén. Jacques", "category": "primary"},
  {"road": "Bvd Gén. Jacques", "category": "primary"},
  {"road": "Bvd de la Cambre", "category": "residential"}}',
  '$.category ? (@ == "residential")');
-- {f, f, t}
-- TODO, it works without ".datetime()"
SELECT jsonbset_path_exists_tz(jsonbset
  '[{"speed":10, "cameraId": "25", "lastInspection": "2000-08-01 12:00:00"},
    {"speed":20, "cameraId": "35", "lastInspection": "2000-03-01 12:00:00"}]',
  '$.lastInspection ? (@.datetime() > "2000-06-01".datetime() )');
-- [t, f]
```

- Devolver el resultado de una comprobación de predicado de ruta JSON para un conjunto JSONB

  `jsonbset @@ jsonpath → tbool`

  `jsonbset_path_match(jsonbset,vars jsonb='{}',silent boolean=false) → tbool`

  `jsonbset_path_match_tz(jsonbset,vars jsonb='{}',silent boolean=false) → tbool`

  El operador `@@` suprime los siguientes errores: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Las funciones anteriores también pueden suprimir estos tipos de errores estableciendo el último argumento en verdadero. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT jsonbset '[{"speed":10}, {"speed": 20, "units": "km/h"}]' @@
  '$.units == "km/h"';
-- [f, t]
SELECT jsonbset_path_match(jsonbset
  '{{"road": "Bvd Gén. Jacques", "category": "primary"},
  {"road": "Bvd Gén. Jacques", "category": "primary"},
  {"road": "Bvd de la Cambre", "category": "residential"}}',
  '$.category == "residential"');
-- {f, f, t}
-- TODO, it works without ".datetime()"
SELECT jsonbset_path_match_tz(jsonbset
  '[{"speed":10, "cameraId": "25", "lastInspection": "2000-08-01 12:00:00"},
```

```
    {"speed":20, "cameraId": "35", "lastInspection": "2000-03-01 12:00:00"}]',
  '$.lastInspection.datetime() > "2000-06-01".datetime()');
-- [t, f]
```

- Devolver todos los elementos devueltos por una ruta JSON de un conjunto JSONB, como un arreglo JSON

  `jsonbset_path_query_array(jsonbset,vars jsonb='{}',silent boolean=false) → jsonbset`

  `jsonbset_path_query_array_tz(jsonbset,vars jsonb='{}',silent boolean=false) → jsonbset`

  La función anterior puede suprimir los siguientes errores estableciendo el último argumento en verdadero: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT jsonbset_path_query_array(jsonbset
  '[{"speed":10}, {"speed": 20, "units": "km/h"}]',
  '$ ? (@.speed >= $min && @.speed <= $max)', '{"min":10, "max":30}');
-- [[{"speed": 10}], [{"speed": 20, "units": "km/h"}]]
-- TODO, it works without "_tz"
SELECT jsonbset_path_query_array_tz(jsonbset
  '[{"cameraId":25, "inspections":["2000-01-03", "2000-01-06", "2000-01-09"]},
   {"cameraId":35, "inspections":["2000-01-04", "2000-01-07", "2000-01-10"]}]',
  '$.inspections ? (@.timestamp_tz() >= $min.timestamp_tz() &&
   @.timestamp_tz() <= $max.timestamp_tz())', '{"min":"2000-01-05", "max":"2000-01-09"}');
-- [["2000-01-06", "2000-01-09"], ["2000-01-07"]]
```

- Devolver el primer elemento devuelto por una ruta JSON de un conjunto JSONB

  `jsonbset_path_query_first(jsonbset,vars jsonb='{}',silent boolean=false) → jsonbset`

  `jsonbset_path_query_first_tz(jsonbset,vars jsonb='{}',silent boolean=false) → jsonbset`

  Las funciones anteriores pueden suprimir los siguientes errores estableciendo el último argumento en verdadero: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT jsonbset_path_query_first(jsonbset
  '[{"speed":10}, {"speed": 20, "units": "km/h"}]',
  '$.speed ? (@ >= $min && @ <= $max'), '{"min":10, "max":30}');
-- [[{"speed": 10}], [{"speed": 20, "units": "km/h"}]]
-- TODO, it works without "_tz"
SELECT jsonbset_path_query_array_tz(jsonbset
  '[{"cameraId":25, "inspections":["2000-01-03", "2000-01-06", "2000-01-09"]},
   {"cameraId":35, "inspections":["2000-01-04", "2000-01-07", "2000-01-10"]}]',
  '$.inspections ? (@.datetime() >= $min.datetime() && @.datetime() <= $max.datetime())',
  '{"min":"2000-01-05", "max":"2000-01-09"}');
-- [["2000-01-06", "2000-01-09"], ["2000-01-07"]]
```

15.2. Valores JSONB temporales

El tipo temporal `tjsonb` permite representar la evolución en el tiempo de valores `jsonb`. Como todos los tipos temporales, viene en tres subtipos, a saber, instante, secuencia y conjunto de secuencias. A continuación se dan ejemplos de valores `tjsonb` en estos subtipos.

```
-- Instant
SELECT tjsonb '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01';
-- Sequence with discrete interpolation
SELECT tjsonb '{{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02}';
-- Sequence with step interpolation
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]';
```

```
-- Sequence set
SELECT tjsonb '{[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02],
  [{"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03,
  {"vehicleId": 1, "location": "Point(3 3)"}@2001-01-04]}';
```

Como puede verse arriba, para el subtipo `Instant` necesitamos encerrar el valor JSONB entre comillas y escapar las comillas internas; de lo contrario, la llave de apertura del valor JSONB se interpretaría como el comienzo de un subtipo `Sequence` con interpolación discreta.

El tipo `tjsonb` acepta modificadores de tipo (o `typmod` en la terminología de PostgreSQL) para especificar el subtipo. Los valores posibles para el subtipo son `Instant`, `Sequence` y `SequenceSet`. El argumento es opcional y, si no se especifica para una columna, se permiten valores de cualquier subtipo.

```
SELECT tjsonb(Instant) '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01';
--  {"location": "Point(1 1)", "vehicleId": 1}@2001-01-01
SELECT tjsonb(Sequence) '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01';
-- ERROR: Temporal type (Instant) does not match column type (Sequence)
```

Los valores JSONB temporales de subtipo secuencia o conjunto de secuencias se convierten a una forma normal de modo que valores equivalentes tengan representaciones idénticas. Para ello, los valores de instantes consecutivos se fusionan cuando es posible. Tres valores de instantes consecutivos pueden fusionarse en dos si sus valores JSONB son iguales. Asimismo, dos secuencias consecutivas que son adyacentes y tienen el mismo valor JSONB de final e inicio pueden fusionarse en una sola. A continuación se dan ejemplos de transformación a una forma normal.

```
SELECT asText(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]');
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
   {"location": "Point(1 1)", "vehicleId": 1}@2001-01-03] */
SELECT asText(tjsonb '{[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03],
  ({"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-04]}');
/* {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
   {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02,
   {"location": "Point(2 2)", "vehicleId": 1}@2001-01-04]} */
```

15.3. Validez de los valores JSONB temporales

Los valores JSONB temporales deben satisfacer las restricciones especificadas en Sección 4.3 para que estén bien definidos. Se genera un error siempre que no se satisfaga alguna de estas restricciones. A continuación se dan ejemplos de valores incorrectos.

```
-- Null values are not allowed
SELECT tjsonb 'NULL@2001-01-01 08:05:00';
SELECT tjsonb 'Point(0 0)@NULL';
-- Base type is not a JSONB
SELECT tjsonb 'Point(0 0)@2001-01-01 08:05:00';
```

A continuación damos las funciones y operadores para JSONB temporal. La mayoría de las funciones y operadores para tipos temporales descritos en los capítulos anteriores pueden aplicarse a los tipos JSONB temporales. Por lo tanto, en las firmas de las funciones dadas anteriormente, la notación `base` representa un valor `jsonb` y la notación `ttype` representa un valor `tjsonb`. Para evitar redundancia, solo presentamos a continuación algunos ejemplos de estas funciones y operadores para valores JSONB temporales.

15.4. Entrada y salida

- Devolver la representación en texto bien conocido (WKT)

  `asText({tjsonb,tjsonb[]}) → {text,text[]}`

```
SELECT asText(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]');
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
   {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02] */
SELECT asText(ARRAY[tjsonb '"{\"vehicleId\":1, \"location\": \"Point(1 1)\"}"@2001-01-01',
  '"{\"vehicleId\": 1, \"location\": \"Point(2 2)\"}"@2001-01-02']);
/* {"{\"location\": \"Point(1 1)\", \"vehicleId\": 1}@2001-01-01",
   "{\"location\": \"Point(2 2)\", \"vehicleId\": 1}@2001-01-02"} */
```

  Como puede verse arriba, para los arreglos de valores JSONB, los elementos deben encerrarse entre comillas y, por lo tanto, las comillas internas deben escaparse.

- Devolver la representación en binario bien conocido (WKB) o en binario bien conocido extendido hexadecimal (HexWKB)

  `asBinary(tjsonb,endian text=”) → bytea`

  `asHexWKB(tjsonb,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 asBinary(tjsonb '"{\"vehicleId\": 1, \"location\": \"Point(1 2)\"}"@2001-01-01');
-- \x01410001380000000000000002000020080000800900000000a000000090000106c6f636174696f6e7...
SELECT asHexWKB(tjsonb '"{\"vehicleId\": 1, \"location\": \"Point(1 2)\"}"@2001-01-01');
-- 01410001380000000000000002000020080000800900000000A000000090000106C6F636174696F6E7...
```

- Devolver la representación JSON de objetos en movimiento (MF-JSON)

  `asMFJSON(text) → tjsonb`

```
SELECT asMFJSON(tjsonb '[{"Position": "Point(1 1)"}@2001-01-01,
  {"Position": "Point(2 2)"}@2001-01-02]');
/* {"type":"MovingJsonb","values":[{"Position": "Point(1 1)"},{"Position": "Point(2 2)"}],
   "datetimes":["2001-01-01T00:00:00+01","2001-01-02T00:00:00+01"],
   "lower_inc":true,"upper_inc":true,"interpolation":"Step"} */
```

- Entrada a partir de la representación en texto bien conocido (WKT)

  `tjsonbFromText(text) → tjsonb`

```
SELECT tjsonbFromText(text
  '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]');
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
   {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02] */
```

- Entrada a partir de la representación en binario bien conocido (WKB) o en binario bien conocido hexadecimal (HexWKB)

  `tjsonbFromBinary(bytea) → tjsonb`

  `tjsonbFromHexWKB(text) → tjsonb`

```
SELECT tjsonbFromBinary(asBinary(
  tjsonb '[{"vehicleId": 1, "location": "Point(1 2)"}@2001-01-01]'));
-- [{"location": "Point(1 2)", "vehicleId": 1}@2001-01-01]
SELECT tjsonbFromHexWKB(asHexWKB(
  tjsonb '[{"vehicleId": 1, "location": "Point(1 2)"}@2001-01-01]'));
-- [{"location": "Point(1 2)", "vehicleId": 1}@2001-01-01]
```

- Entrada a partir de la representación JSON de objetos en movimiento (MF-JSON)

  `tjsonbFromMFJSON(text) → tjsonb`

```
SELECT tjsonbFromMFJSON('{"type": "MovingJsonb", "interpolation": "Step",
  "values": [{"Position": "Point(1 1)"}, {"Position": "Point(2 2)"}],
  "datetimes": ["2001-01-01T10:00:00Z","2001-01-01T12:00:00Z"]}');
/* [{"Position": "Point(1 1)"}@2001-01-01 11:00:00+01,
   {"Position": "Point(2 2)"}@2001-01-01 13:00:00+01] */
```

15.5. Constructores

- Constructor de valores JSONB temporales que tienen un valor constante

  `tjsonb(jsonb,timestamptz) → tjsonbInst`

  `tjsonb(jsonb,tstzset) → tjsonbDiscSeq`

  `tjsonb(jsonb,tstzspan) → tjsonbContSeq`

  `tjsonb(jsonb,tstzspanset) → tjsonbSeqSet`

```
SELECT tjsonb('{"vehicleId": 1, "location": "Point(1 1)"}', timestamptz '2001-01-01');
-- {"location": "Point(1 1)", "vehicleId": 1}@2001-01-01
SELECT tjsonb('{"vehicleId": 1, "location": "Point(1 1)"}',
  tstzset '{2001-01-01, 2001-01-03, 2001-01-05}');
/* {{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-03,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-05} */
SELECT tjsonb('{"vehicleId": 1, "location": "Point(1 1)"}',
  tstzspan '[2001-01-01, 2001-01-02]');
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-02] */
SELECT tjsonb('{"vehicleId": 1, "location": "Point(1 1)"}',
  tstzspanset '{[2001-01-01, 2001-01-02],[2001-01-03, 2001-01-04]}');
/* {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
     {"location": "Point(1 1)", "vehicleId": 1}@2001-01-02],
    [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-03,
     {"location": "Point(1 1)", "vehicleId": 1}@2001-01-04]} */
```

- Constructor de valores JSONB temporales del subtipo secuencia

  `tjsonbSeq(tjsonbInst[]},leftInc bool=true,rightInc bool=true) →tjsonbSeq`

```
SELECT tjsonbSeq(ARRAY[tjsonb
  '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01',
  '"{\"vehicleId\": 1, \"location\": \"Point(2 2)\"}"@2001-01-02',
  '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-03']);
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-03] */
```

- Constructor de valores JSONB temporales del subtipo conjunto de secuencias

  `tjsonbSeqset(tjsonb[]) → tjsonbSeqSet`

  `tjsonbSeqSetGaps(tjsonbInst[],maxt=NULL,maxdist=NULL) → tjsonbSeqSet`

```
SELECT tjsonbSeqSet(ARRAY[tjsonb
  '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]',
  '[{"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03,
    {"vehicleId": 1, "location": "Point(3 3)"}@2001-01-04]']);
/* {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
```

```
    {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02],
   [{"location": "Point(2 2)", "vehicleId": 1}@2001-01-03,
    {"location": "Point(3 3)", "vehicleId": 1}@2001-01-04} */
SELECT tjsonbSeqSetGaps(ARRAY[tjsonb
  '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01',
  '"{\"vehicleId\": 1, \"location\": \"Point(2 2)\"}"@2001-01-03',
  '"{\"vehicleId\": 1, \"location\": \"Point(3 3)\"}"@2001-01-05'], interval '1 day');
/* {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01],
    [{"location": "Point(2 2)", "vehicleId": 1}@2001-01-03],
    [{"location": "Point(3 3)", "vehicleId": 1}@2001-01-05]} */
```

15.6. Conversiones

- Convertir un JSONB temporal a un `tstzspan`

  `tjsonb::tstzpan`

```
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]'::tstzspan;
-- [2001-01-01, 2001-01-02]
```

- Convertir entre un JSONB temporal y un texto

  `tjsonb::text`

  `text::tjsonb`

```
SELECT (text '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]')::jsonb;
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02] */
SELECT pg_typeof(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]'::text);
-- text
```

- Convertir entre un JSONB temporal y un texto temporal

  `tjsonb::ttext`

  `ttext::tjsonb`

```
SELECT (ttext '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]')::tjsonb;
/* [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(2 2)", "vehicleId": 1}@2001-01-02] */
SELECT pg_typeof(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]'::ttext);
-- ttext
```

15.7. Accesores

- Devolver los valores

  `getValues(tjsonb) → jsonbset`

```
SELECT getValues(tjsonb
  '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02]');
-- {{"location": "Point(1 1)", "vehicleId": 1}}
```

- Devolver el valor en una marca de tiempo

  `valueAtTimestamp(tjsonb,timestamptz) → jsonb`

```
SELECT valueAtTimestamp(tjsonb
  '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]', '2001-01-02');
-- {"location": "Point(2 2)", "vehicleId": 1}
```

15.8. Transformaciones

- Transformar un JSONB temporal a otro subtipo

  `tjsonbInst(tjsonb) → tjsonbInst`

  `tjsonbSeq(tjsonb,interp='step') → tjsonbSeq`

  `tjsonbSeqSet(tjsonb) → tjsonbSeqSet`

```
SELECT tjsonbSeq(tjsonb '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01');
-- [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01]
SELECT tjsonbSeq(tjsonb '"{\"vehicleId\": 1, \"location\": \"Point(1 1)\"}"@2001-01-01',
  'discrete');
-- {{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01}
SELECT tjsonbSeqSet(tjsonb '"{\"vehicleId\": 1,\"location\":\"Point(1 1)\"}"@2001-01-01');
-- {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01]}
```

- Transformar un valor JSONB temporal a otra interpolación

  `setInterp(tjsonb,interp) → tjsonb`

```
SELECT setInterp(tjsonb '"{\"vehicleId\": 1,\"location\": \"Point(1 1)\"}"@2001-01-01',' ↩
    step');
-- [{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01]
SELECT setInterp(tjsonb '{[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01],
  [{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02]}', 'discrete');
/* {{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-02} */
```

15.9. Operaciones JSON temporales

En esta sección, damos las versiones temporales de las funciones para los tipos `json` y `jsonb`. Remitimos a la documentación de PostgreSQL para explicaciones detalladas de estas funciones. Hemos buscado mantener en la medida de lo posible la misma sintaxis para las versiones no temporales y temporales de estas funciones, como se ilustra en este documento.

Como se indicó en Sección 4.4, una forma común de generalizar las operaciones tradicionales a los tipos temporales es aplicar la operación *en cada instante*, lo que produce un valor temporal como resultado. Considere por ejemplo el operador `->` que extrae un campo de objeto JSON con la clave dada.

```
SELECT jsonb '{"unit": "km", "speed": 10}' -> text 'speed';
-- 10
```

Aplicar el mismo operador a un valor JSON o JSONB temporal produce el siguiente resultado

```
SELECT ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' -> text 'speed';
-- {["10"@2001-01-01, "20"@2001-01-02]}
SELECT tjsonb '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' -> text 'speed';
-- {["10"@2001-01-01, "20"@2001-01-02]}
```

que se obtiene aplicando el operador de PostgreSQL `->` en cada instante del valor JSON o JSONB temporal.

Al manipular colecciones JSON de estructura variable, puede ocurrir que un elemento esté definido en algunos de los documentos pero no en todos. PostgreSQL proporciona el modo lax para este propósito, donde el argumento `null_value_treatment` determina el comportamiento en el caso en que una función devuelve un valor NULL. El argumento puede tomar uno de los siguientes valores: `'raise_exception'`, `'use_json_null'`, `'delete_key'`, `'return_target'`, donde `'use_json_null'` es el valor por defecto. En PostgreSQL el modo lax se admite solo para operaciones de *actualización* (no de *consulta*) con la función `jsonb_set_lax`. En MobilityDB, hemos mantenido la misma semántica para la función correspondiente `tjsonb_set_lax`, pero habilitamos un comportamiento similar para *todas* las operaciones JSON temporales que pueden devolver un valor nulo, salvo que reemplazamos el valor `'return_target'` por `'return_null'`, ya que el primero no tiene sentido para operaciones temporales. Para operadores como `->`, se utiliza el valor por defecto `'use_json_null'` y no se puede cambiar, mientras que para las funciones correspondientes `tjson_object_field` y `tjsonb_object_field`, el último argumento especifica el comportamiento en el caso en que la función devuelve NULL. Ilustramos este comportamiento para la función `tjson_object_field` a continuación.

- Extraer un campo de objeto JSON temporal especificado por una clave

  `{ttext,tjsonb} -> text → {ttext,tjsonb}`

  `tjsonb ->> text → ttext`

  `tjson_object_field(ttext,text,null_handle text='use_json_null') → ttext`

  `tjsonb_object_field(tjsonb,text,null_handle text='use_json_null') → tjsonb`

  `tjsonb_object_field_text(tjsonb,text,null_handle text='use_json_null') → ttext`

```
SELECT ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' -> text 'speed';
-- {["10"@2001-01-01, "20"@2001-01-02]}
SELECT tjsonb '[{"position":"Point(1 1)", "speed":10}@2001-01-01,
  {"position":"Point(2 2)", "speed":20}@2001-01-03]' ->> text 'position';
-- ["Point(1 1)"@2001-01-01, "Point(2 2)"@2001-01-03]
SELECT ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
{"unit": "km"}@2001-01-02, {"unit": "km", "speed": 10}@2001-01-03]}' -> text 'speed';
-- {[10@2001-01-01, null@2001-01-02, 10@2001-01-03]}
```

  Mostramos a continuación el comportamiento de la función `tjson_object_field` para los posibles valores del argumento `null_handle`. Todas las funciones JSON que pueden devolver un valor NULL se comportan de forma similar en MobilityDB.

```
SELECT tjson_object_field(ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km"}@2001-01-02, {"unit": "km", "speed": 10}@2001-01-03]}', 'speed',
  'raise_exception');
-- ERROR:  The lifted operation returned NULL
SELECT tjson_object_field(ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km"}@2001-01-02, {"unit": "km", "speed": 10}@2001-01-03]}', 'speed',
  'use_json_null');
-- {[10@2001-01-01, null@2001-01-02, 10@2001-01-03]}
SELECT tjson_object_field(ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km"}@2001-01-02, {"unit": "km", "speed": 10}@2001-01-03]}', 'speed',
  'delete_key');
-- {[10@2001-01-01, 10@2001-01-02), [10@2001-01-03]}
SELECT tjson_object_field(ttext '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km"}@2001-01-02, {"unit": "km", "speed": 10}@2001-01-03]}', 'speed',
  'return_null');
-- NULL
```

- Extraer un campo de objeto JSON temporal especificado por una ruta

  `{ttext,tjsonb} #> text[] → {ttext,tjsonb}`

  `tjson_extract_path(ttext,text[],null_handle text='use_json_null') → ttext`

  `tjsonb_extract_path(tjsonb,text[],null_handle text='use_json_null') → tjsonb`

  `tjsonb_extract_path_text(tjsonb,text[],null_handle text='use_json_null') → ttext`

```
SELECT ttext '[{"speed": {"unit": "km", "value": 10}}@2001-01-01,
  {"speed": {"unit": "km", "value": 20}}@2001-01-02]' #> ARRAY[text 'speed', 'value'];
-- {["10"@2001-01-01, "20"@2001-01-02]}
SELECT tjsonb '[{"position":"Point(1 1)", "PoIs":["Grand Place", "La Bourse"]}@2001-01-01,
  {"position":"Point(2 2)", "PoIs":["Palais Royal", "Manneken Pis"]}@2001-01-03]' #>>
  ARRAY[text 'PoIs', '1'];
-- ["La Bourse"@2001-01-01, "Manneken Pis"@2001-01-03]
```

- Extraer un elemento de un arreglo JSON temporal

  `tjson_array_element(tjsonb,integer,null_handle text='use_json_null')` → `tjsonb`

  `tjsonb_array_element(tjsonb,integer,null_handle text='use_json_null')` → `tjsonb`

  `tjsonb_array_element_text(tjsonb,integer,null_handle text='use_json_null')` → `tjsonb`

```
SELECT tjson_array_element(ttext '[["Grand Place", "La Bourse"]@2001-01-01,
  ["Palais Royal", "Manneken Pis"]@2001-01-02]', 0);
-- ["Grand Place"@2001-01-01, "Palais Royal"@2001-01-02]
SELECT tjsonb_array_element(tjsonb '[["Grand Place", "La Bourse"]@2001-01-01,
  ["Palais Royal", "Manneken Pis"]@2001-01-02]', 1);
-- ["La Bourse"@2001-01-01, "Manneken Pis"@2001-01-02]
```

- Extraer un valor alfanumérico temporal de un valor JSONB temporal dado por una clave

  `tbool(tjsonb,text,null_handle text='raise_exception')` → `tbool`

  `tint(tjsonb,text,null_handle text='raise_exception')` → `tint`

  `tfloat(tjsonb,text,interp='linear',null_handle text='raise_exception')` → `tfloat`

  `ttext(tjsonb,text, null_handle text='raise_exception')` → `ttext`

  Tenga en cuenta que el valor `use_json_null` no puede utilizarse para las funciones anteriores y por lo tanto se utiliza el valor por defecto `'raise_exception'`.

```
SELECT tbool(tjsonb '[{"speed": 10, "lights": "on"}@2001-01-01,
  {"speed": 20, "lights": "on"}@2001-01-02]', 'lights');
-- [true@2001-01-01, true@2001-01-02]
SELECT tint(tjsonb '[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20, "units": "km/h"}@2001-01-02]', 'speed');
-- [10@2001-01-01, 20@2001-01-02]
SELECT tfloat(tjsonb '{[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20, "units": "km/h"}@2001-01-02],[{"speed": 20}@2001-01-03,
  {"speed": 20}@2001-01-04]}', 'speed', 'step');
-- Interp=Step;{[10@2001-01-01, 20@2001-01-02], [20@2001-01-03, 20@2001-01-04]}
SELECT ttext(tjsonb '[{"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-01,
  {"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-02,
  {"road": "Bvd de la Cambre", "category": "residential"}@2001-01-03]', 'category');
-- [primary@2001-01-01, residential@2001-01-03]
```

- Concatenación JSONB temporal

  `tjsonb || {jsonb,tjsonb}` → `tjsonb`

```
SELECT tjsonb '{[{"speed":10}@2001-01-01, {"speed":20}@2001-01-02]}' ||
  '{"unit":"km"}'::jsonb;
-- {[{"unit": "km", "speed": 10}@2001-01-01, {"unit": "km", "speed": 20}@2001-01-02]}
SELECT tjsonb '{[{"speed":10}@2001-01-01, {"speed":20}@2001-01-03]}' ||
  tjsonb '{[{"Position":"Point(1 1)"}@2001-01-02, {"Position":"Point(2 2)"}@2001-01-04]}';
/* {[{"speed": 10, "Position": "Point(1 1)"}@2001-01-02,
     {"speed": 20, "Position": "Point(2 2)"}@2001-01-03]} */
```

- Eliminación JSONB temporal

  `tjsonb - {int,text,text[]} → tjsonb`

  `tjsonb #- text[] → tjsonb`

```
SELECT tjsonb '{[{"unit": "km", "speed": 10}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' - 'unit';
-- {[{"speed": 10}@2001-01-01, {"speed": 20}@2001-01-02]}
SELECT tjsonb '[["Grand Place", "La Bourse"]@2001-01-01,
  ["Palais Royal", "Manneken Pis"]@2001-01-02]' - 1;
-- [["Grand Place"]@2001-01-01, ["Palais Royal"]@2001-01-02]
SELECT tjsonb '[{"location":"Point(1 1)", "PoIs": ["Grand Place", "La Bourse"]@2001-01-01,
  "location":"Point(2 2)", "PoIs": ["Palais Royal", "Manneken Pis"]@2001-01-02]' #-
  ARRAY[text "\"PoIs\"", "1"];
-- ["La Bourse"@2001-01-01, "Manneken Pis"@2001-01-02]
```

  Como se muestra a continuación, el resultado de una operación de eliminación puede ser un registro vacío o un arreglo vacío. Para eliminar estos valores, puede utilizarse la función `minusValue`.

```
SELECT tjsonb '{[{"unit": "km", "speed": 10, "light": true}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' - ARRAY[text 'unit', 'speed'];
-- {[{"light": true}@2001-01-01, {}@2001-01-02]}
SELECT tjsonb '[["Grand Place", "La Bourse"]@2001-01-01,
  ["Palais Royal"]@2001-01-02]' - 0;
-- [["La Bourse"]@2001-01-01, []@2001-01-02]
```

```
SELECT minusValues(tjsonb '{[{"unit": "km", "speed": 10, "light": true}@2001-01-01,
  {"unit": "km", "speed": 20}@2001-01-02]}' - ARRAY[text 'unit', 'speed'],
  jsonbset '{"[]","{}"}');
--  {[{"light": true}@2001-01-01, {"light": true}@2001-01-02)}
SELECT minusValues(tjsonb '[["Grand Place", "La Bourse"]@2001-01-01,
  ["Palais Royal"]@2001-01-02]' - 0,  jsonbset '{"[]","{}"}');
-- {[["La Bourse"]@2001-01-01, ["La Bourse"]@2001-01-02)}
```

- Existencia JSONB temporal

  `tjsonb ? jsonb → tbool`

  `tjsonb ?| jsonb[] → tbool`

  `tjsonb ?& jsonb[] → tbool`

  Como en PostgreSQL, los operadores `?|` y `?&` comprueban, respectivamente, si *alguna* o *todas* las cadenas del arreglo de texto existen como claves de nivel superior o elementos de arreglo.

```
SELECT tjsonb '"{\"geom\": \"Point(1 1)\"}"@2001-01-01' ? text 'geom';
-- t@2001-01-01
SELECT tjsonb '{{"geom": "Point(1 1)"}@2001-01-01, {"geom": "Point(2 2)"}@2001-01-02,
  {"geom": "Point(1 1)"}@2001-01-03}' ?| ARRAY[text 'geom'];
-- {t@2001-01-01, t@2001-01-02, t@2001-01-03}
SELECT tjsonb '[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20, "units": "km/h"}@2001-01-02]' ?& ARRAY['speed', 'units'];
-- {[t@2001-01-01, t@2001-01-02]}
```

- Contiene/contenido JSONB temporal

  `{tjsonb,jsonb} @> {tjsonb,jsonb} → tbool`

  `{tjsonb,jsonb} <@ {tjsonb,jsonb} → tbool`

```
SELECT tjsonb '"{\"geom\": \"Point(1 1)\"}"@2001-01-01' @> jsonb '{"geom": "Point(1 1)"}';
-- t@2001-01-01
SELECT tjsonb '{{"geom": "Point(1 1)"}@2001-01-01, {"geom": "Point(2 2)"}@2001-01-02,
  {"geom": "Point(1 1)"}@2001-01-03}' @> jsonb '{"geom": "Point(1 1)"}';
-- {t@2001-01-01, f@2001-01-02, t@2001-01-03}
```

```
SELECT jsonb '{"geom": "Point(1 1)"}' <@ tjsonb '{[{"geom": "Point(1 1)"}@2001-01-01,
  {"geom": "Point(1 1)"}@2001-01-02, {"geom": "Point(1 1)"}@2001-01-03],
  [{"geom": "Point(2 2)"}@2001-01-04, {"geom": "Point(2 2)"}@2001-01-05]}';
-- {[t@2001-01-01, t@2001-01-03], [f@2001-01-04, f@2001-01-05]}
```

- Asignación JSONB temporal

  `tjsonb_set(tjsonb,path text[],jsonb,create boolean=true)` → `tjsonb`

  `tjsonb_set_lax(tjsonb,path text[],jsonb,create boolean=true,handle_null text)` → `tjsonb`

  Devuelve el valor `tjsonb` con el elemento especificado por la ruta reemplazado por `jsonb`, o añadido si `create` es verdadero y el elemento no existe. Todos los pasos anteriores de la ruta deben existir, o se devuelve el objetivo sin cambios. Como ocurre con los operadores orientados a rutas, los enteros negativos que aparecen en la ruta cuentan desde el final de los arreglos JSON. Si el último paso de la ruta es un índice de arreglo que está fuera de rango, y create_if_missing es verdadero, el nuevo valor se añade al principio del arreglo si el índice es negativo, o al final del arreglo si es positivo.

  La función `tjsonb_set_lax` se comporta de forma idéntica a `tjsonb_set` si el valor dado no es NULL; de lo contrario, se comporta según el valor de `handle_null`, que debe ser uno de `'raise_exception'`, `'use_json_null'`, `'delete_key'` o `'return_source'`. Como se indicó anteriormente, mantuvimos el comportamiento de PostgreSQL para esta función habilitando el valor `'return_source'`, mientras que para todas las demás operaciones de *consulta*, este valor se ha reemplazado por `'return_null'`.

```
SELECT tjsonb_set(tjsonb '[{"speed":10}@2001-01-01, {"speed":20}@2001-01-02]',
  ARRAY['units'], '"km/h"'::jsonb);
-- [{"speed": 10, "units": "km/h"}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]
SELECT tjsonb_set(tjsonb '[{"speed":10}@2001-01-01, {"speed":20}@2001-01-02]',
  ARRAY['units'], '"km/h"'::jsonb, false);
-- [{"speed": 10}@2001-01-01, {"speed": 20}@2001-01-02]
SELECT tjsonb_set(tjsonb '[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20, "units": "km/h"}@2001-01-02]', ARRAY['units'], '"mi/h"'::jsonb);
-- [{"speed": 10, "units": "mi/h"}@2001-01-01, {"speed": 20, "units": "mi/h"}@2001-01-02]
SELECT tjsonb_set_lax(tjsonb '[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20}@2001-01-02]', ARRAY['units'], 'null'::jsonb, true, 'delete_key');
-- [{"speed": 10, "units": "km/h"}@2001-01-01, {"speed": 20}@2001-01-02]
```

- Inserción JSONB temporal

  `tjsonb_insert(tjsonb,path text[],jsonb,after boolean=false)` → `tjsonb`

  Devuelve el valor `tjsonb` con `jsonb` insertado. Si el elemento especificado por la ruta es un elemento de arreglo, el nuevo valor se insertará antes de ese elemento si `after` es falso, o después en caso contrario. Si el elemento especificado por la ruta es un campo de objeto, el nuevo valor se insertará solo si el objeto no contiene ya esa clave. Todos los pasos anteriores de la ruta deben existir, o se devuelve el objetivo sin cambios. Como ocurre con los operadores orientados a rutas, los enteros negativos que aparecen en la ruta cuentan desde el final de los arreglos JSON. Si el último paso de la ruta es un índice de arreglo que está fuera de rango, el nuevo valor se añade al principio del arreglo si el índice es negativo, o al final del arreglo si es positivo.

```
SELECT tjsonb_insert(tjsonb '[{"speed":10}@2001-01-01, {"speed":20}@2001-01-02]',
  ARRAY['units'], '"km/h"'::jsonb);
-- [{"speed": 10, "units": "km/h"}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]
SELECT tjsonb_insert(tjsonb '[{"speed":10}@2001-01-01, {"speed":20}@2001-01-02]',
  ARRAY['units'], '"km/h"'::jsonb, false);
-- [{"speed": 10}@2001-01-01, {"speed": 20}@2001-01-02]
SELECT tjsonb_insert(tjsonb '[{"speed": 10, "units": "km/h"}@2001-01-01,
  {"speed": 20, "units": "km/h"}@2001-01-02]', ARRAY['units'], '"mi/h"'::jsonb);
-- [{"speed": 10, "units": "mi/h"}@2001-01-01, {"speed": 20, "units": "mi/h"}@2001-01-02]
```

- Devolver un valor JSON temporal sin nulos

  `tjson_strip_nulls(ttext,strip_in_arrays bool=false)` → `ttext`

  `tjsonb_strip_nulls(tjsonb,strip_in_arrays bool=false)` → `tjsonb`

  El último argumento indica si los elementos de arreglo nulos también se eliminan. Los valores nulos simples nunca se eliminan.

```
SELECT tjson_strip_nulls(ttext '[{"speed": 10, "lights": null}@2001-01-01,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse", null]}@2001-01-02]');
/* [{"speed": 10}@2001-01-01,
   {"PoIs": ["Grand Place", "La Bourse", null], "speed": 20}@2001-01-02] */
SELECT tjsonb_strip_nulls(tjsonb '[{"speed": 10, "lights": null}@2001-01-01,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse", null]}@2001-01-02]', true);
/* [{"speed": 10}@2001-01-01,
   {"PoIs": ["Grand Place", "La Bourse"], "speed": 20}@2001-01-02] */
SELECT tjsonb_strip_nulls(tjsonb
  '{{"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-01,
  {"road": "Bvd de la Cambre", "category": "primary"}@2001-01-02,
  {"road": "rue de l''Abbaye", "category": null}@2001-01-03}');
/* [{"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-01,
   {"road": "Bvd de la Cambre", "category": "primary"}@2001-01-02,
   {"road": "rue de l'Abbaye"}@2001-01-03] */
```

Como se muestra a continuación, la función no elimina los valores nulos simples. Para este propósito puede utilizarse la función `minusValue`.

```
SELECT tjson_strip_nulls(ttext '[{"speed": 10}@2001-01-01, null@2001-01-02,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}@2001-01-03]');
/* [{"speed":10}@2001-01-01, null@2001-01-02,
   {"speed":20,"PoIs":["Grand Place","La Bourse"]}@2001-01-03] */
SELECT minusValues(ttext '[{"speed": 10}@2001-01-01, null@2001-01-02,
  {"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}@2001-01-03]', 'null');
/* {[{"speed": 10}@2001-01-01, {"speed": 10}@2001-01-02),
   [{"speed": 20, "PoIs": ["Grand Place", "La Bourse"]}@2001-01-03]}
```

- ¿Devuelve la ruta JSON algún elemento para el valor JSONB temporal especificado?

  `tjsonb @? jsonpath → tbool`

  `tjsonb_path_exists(tjsonb,vars jsonb='{}',silent boolean=false) → tbool`

  `tjsonb_path_exists_tz(tjsonb,vars jsonb='{}',silent boolean=false) → tbool`

  El operador `@?` suprime los siguientes errores: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Las funciones anteriores también pueden suprimir estos tipos de errores estableciendo el último argumento en verdadero. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT tjsonb '[{"speed":10}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]' @?
  '$.units ? (@ == "km/h")';
-- [f@2001-01-01, t@2001-01-02]
SELECT tjsonb_path_exists(tjsonb
  '{{"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-01,
  {"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-02,
  {"road": "Bvd de la Cambre", "category": "residential"}@2001-01-03}',
  '$.category ? (@ == "residential")');
-- {f@2001-01-01, f@2001-01-02, t@2001-01-03}
-- TODO, it works without ".datetime()"
SELECT tjsonb_path_exists_tz(tjsonb
  '[{"speed":10, "cameraId": "25", "lastInspection": "2000-08-01 12:00:00"}@2001-01-01,
    {"speed":20, "cameraId": "35", "lastInspection": "2000-03-01 12:00:00"}@2001-01-02]',
  '$.lastInspection ? (@.datetime() > "2000-06-01".datetime() )');
-- [t@2001-01-01, f@2001-01-02]
```

- Devolver el resultado de una comprobación de predicado de ruta JSON para un valor JSONB temporal

  `tjsonb @@ jsonpath → tbool`

  `tjsonb_path_match(tjsonb,vars jsonb='{}',silent boolean=false) → tbool`

  `tjsonb_path_match_tz(tjsonb,vars jsonb='{}',silent boolean=false) → tbool`

El operador `@@` suprime los siguientes errores: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Las funciones anteriores también pueden suprimir estos tipos de errores estableciendo el último argumento en verdadero. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT tjsonb '[{"speed":10}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]' @@
  '$.units == "km/h"';
-- [f@2001-01-01, t@2001-01-02]
SELECT tjsonb_path_match(tjsonb
  '{{"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-01,
  {"road": "Bvd Gén. Jacques", "category": "primary"}@2001-01-02,
  {"road": "Bvd de la Cambre", "category": "residential"}@2001-01-03}',
  '$.category == "residential"');
-- {f@2001-01-01, f@2001-01-02, t@2001-01-03}
-- TODO, it works without ".datetime()"
SELECT tjsonb_path_match_tz(tjsonb
  '[{"speed":10, "cameraId": "25", "lastInspection": "2000-08-01 12:00:00"}@2001-01-01,
    {"speed":20, "cameraId": "35", "lastInspection": "2000-03-01 12:00:00"}@2001-01-02]',
  '$.lastInspection.datetime() > "2000-06-01".datetime()');
-- [t@2001-01-01, f@2001-01-02]
```

- Devolver todos los elementos devueltos por una ruta JSON de un valor JSONB temporal, como un arreglo JSON

  `tjsonb_path_query_array(tjsonb,vars jsonb='{}',silent boolean=false) → tjsonb`

  `tjsonb_path_query_array_tz(tjsonb,vars jsonb='{}',silent boolean=false) → tjsonb`

  La función anterior puede suprimir los siguientes errores estableciendo el último argumento en verdadero: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT tjsonb_path_query_array(tjsonb
  '[{"speed":10}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]',
  '$ ? (@.speed >= $min && @.speed <= $max)', '{"min":10, "max":30}');
-- [[{"speed": 10}]@2001-01-01, [{"speed": 20, "units": "km/h"}]@2001-01-02]
-- TODO, it works without "_tz"
SELECT tjsonb_path_query_array_tz(tjsonb
  '[{"cameraId":25, "inspections":["2000-01-03", "2000-01-06", "2000-01-09"]}@2001-01-01,
   {"cameraId":35, "inspections":["2000-01-04", "2000-01-07", "2000-01-10"]}@2001-01-02]',
  '$.inspections ? (@.timestamp_tz() >= $min.timestamp_tz() &&
   @.timestamp_tz() <= $max.timestamp_tz())', '{"min":"2000-01-05", "max":"2000-01-09"}');
-- [["2000-01-06", "2000-01-09"]@2001-01-01, ["2000-01-07"]@2001-01-02]
```

- Devolver el primer elemento devuelto por una ruta JSON de un valor JSONB temporal

  `tjsonb_path_query_first(tjsonb,vars jsonb='{}',silent boolean=false) → tjsonb`

  `tjsonb_path_query_first_tz(tjsonb,vars jsonb='{}',silent boolean=false) → tjsonb`

  Las funciones anteriores pueden suprimir los siguientes errores estableciendo el último argumento en verdadero: campo de objeto o elemento de arreglo faltante, tipo de elemento JSON inesperado, errores de fecha/hora y numéricos. Este comportamiento puede ser útil al buscar en colecciones de documentos JSON de estructura variable.

```
SELECT tjsonb_path_query_first(tjsonb
  '[{"speed":10}@2001-01-01, {"speed": 20, "units": "km/h"}@2001-01-02]',
  '$.speed ? (@ >= $min && @ <= $max'), '{"min":10, "max":30}');
-- [[{"speed": 10}]@2001-01-01, [{"speed": 20, "units": "km/h"}]@2001-01-02]
-- TODO, it works without "_tz"
SELECT tjsonb_path_query_array_tz(tjsonb
  '[{"cameraId":25, "inspections":["2000-01-03", "2000-01-06", "2000-01-09"]}@2001-01-01,
   {"cameraId":35, "inspections":["2000-01-04", "2000-01-07", "2000-01-10"]}@2001-01-02]',
  '$.inspections ? (@.datetime() >= $min.datetime() && @.datetime() <= $max.datetime())',
  '{"min":"2000-01-05", "max":"2000-01-09"}');
-- [["2000-01-06", "2000-01-09"]@2001-01-01, ["2000-01-07"]@2001-01-02]
```

15.10. Restricciones

- Restringir a (el complemento de) un valor o un conjunto de valores

  `atValue(tjsonb,value) → tjsonb`

  `minusValue(tjsonb,value) → tjsonb`

  `atValues(tjsonb,valueset) → tjsonb`

  `minusValues(tjsonb,valueset) → tjsonb`

```
SELECT atValue(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03]',
  jsonb '{"vehicleId": 1, "location": "Point(2 2)"}');
-- {[{"location": "Point(2 2)", "vehicleId": 1}@2001-01-03]}
SELECT minusValue(tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03]',
  jsonb '{"vehicleId": 1, "location": "Point(2 2)"}');
/* {[{"location": "Point(1 1)", "vehicleId": 1}@2001-01-01,
    {"location": "Point(1 1)", "vehicleId": 1}@2001-01-03)} */
```

15.11. Comparaciones

- Comparaciones tradicionales

  `tjsonb {=, <>, <, >, <=, >=} tjsonb → boolean`

```
SELECT tjsonb '{[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02),
  [{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
   {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]}' =
  tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]';
-- true
SELECT tjsonb '{[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]}' <>
  tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]';
-- false
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]' <
  tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]';
-- true
```

- Comparaciones siempre y alguna vez

  `{jsonb,tjsonb} {?=,%=} {jsonb,tjsonb} → boolean`

```
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' ?=
  jsonb '{"vehicleId": 1, "location": "Point(1 1)"}';
-- true
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' %=
  jsonb '{"vehicleId": 1, "location": "Point(1 1)"}';
-- false
```

- Comparaciones temporales

  `{jsonb,tjsonb} {#=, #<>} {jsonb,tjsonb} → tbool`

```
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03)' #=
  jsonb '{"vehicleId": 1, "location": "Point(1 1)"}';
-- {[f@2001-01-01, t@2001-01-02], (f@2001-01-02, f@2001-01-03)}
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03)' #<>
  tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
  {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03)';
-- {[t@2001-01-01, f@2001-01-02], (t@2001-01-02, t@2001-01-03)}
```

15.12. Operaciones de caja delimitadora

- Operadores topológicos

  `{tjsonb,tstzspan} {&&, <@#, #@>, ~=, -|-} {tjsonb,tstzspan} → boolean`

  Los operadores temporales contiene y contenido se marcan con un # para distinguirlos de los operadores JSONB correspondientes.

```
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' &&
  tstzspan '[2001-01-01, 2001-03-01]';
-- true
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' #@>
  timestamptz '[2001-01-02]'::tstzspan;
-- true
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' -|-
  tjsonb '({"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02,
    {"vehicleId": 1, "location": "Point(3 3)"}@2001-01-03]';
-- true
```

- Operadores de posición

  `{tjsonb,tstzspan} {<<#, &<#, #>>, #&>} {tjsonb,tstzspan} → boolean`

```
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-02]' <<#
  tstzspan '[2001-01-01, 2001-03-01]';
-- false
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03]' #>>
  timestamptz '[2001-01-01]'::tstzspan;
-- true
SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
    {"vehicleId": 1, "location": "Point(2 2)"}@2001-01-03]' #&>
  tjsonb '[{"vehicleId": 1, "location": "Point(2 2)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(3 3)"}@2001-01-03]';
-- true
```

15.13. Agregaciones

Las funciones de agregación para valores JSONB temporales se ilustran a continuación.

- Conteo temporal

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

```
WITH Temp(temp) AS (
  SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]' UNION
  SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-04]' )
SELECT tCount(Temp)
FROM Temp;
-- {[1@2001-01-01, 2@2001-01-02, 2@2001-01-03], (1@2001-01-03, 1@2001-01-04]}
```

- Conteo en ventana

  `wCount(tjsonb) → {tintSeq,tintSeqSet}`

```
WITH Temp(temp) AS (
  SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-01,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-03]' UNION
  SELECT tjsonb '[{"vehicleId": 1, "location": "Point(1 1)"}@2001-01-02,
    {"vehicleId": 1, "location": "Point(1 1)"}@2001-01-04]' )
SELECT wCount(Temp, interval '2 days')
FROM Temp;
-- {[1@2001-01-01, 2@2001-01-02, 2@2001-01-05], (1@2001-01-05, 1@2001-01-06]}
```

15.14. Indexación

Pueden crearse índices GiST y SP-GiST para columnas de tabla de valores JSONB 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 la caja delimitadora de los valores JSONB temporales, que es un `stbox`, como todos los tipos espaciotemporales.

Un índice GiST o SP-GiST puede acelerar las consultas que involucran los siguientes operadores:

- `<<`, `&<`, `&>`, `>>`, `<<|`, `&<|`, `|&>`, `|>>`, que solo consideran la dimensión espacial en los valores JSONB temporales,
- `<<#`, `&<#`, `#&>`, `#>>`, que solo consideran la dimensión temporal en los valores JSONB temporales,
- `&&`, `@>`, `<@`, `~=`, `-|-` y `|=|` , que consideran tantas dimensiones como las que comparten la columna indexada y el argumento de la consulta.

Estos operadores trabajan sobre cajas delimitadoras, no sobre los valores completos.

Capítulo 16

Índices de Celdas H3 Temporales

El tipo de índice de celda H3 temporal `th3index` representa el movimiento de los objetos como una secuencia de identificadores de celdas hexagonales H3. H3 de Uber es un sistema de cuadrícula global discreta que tesela la esfera en 122 celdas base en la resolución 0 y subdivide cada celda en 7 hijos en cada resolución más fina (hasta la resolución 15). Cada celda tiene un identificador entero de 64 bits que codifica la celda base, la resolución y la posición jerárquica.

El tipo `th3index` comparte su representación en disco con `tbigint`: ambos almacenan un entero temporal de 64 bits. El tipo SQL distinto existe únicamente para que el verificador de tipos rechace las funciones específicas de H3 cuando se aplican a trayectorias `tbigint` arbitrarias (y viceversa). Las conversiones desde y hacia `tbigint` son *explícitas* (`AS ASSIGNMENT`) y de coerción binaria solamente — sin coste en tiempo de ejecución pero requieren un `::` explícito en SQL.

Como con otros tipos temporales, `th3index` tiene cuatro subtipos: `Instant`, secuencia discreta, secuencia continua con interpolación de pasos (las celdas H3 son discretas — la interpolación lineal carece de sentido), y `SequenceSet`.

La guía operativa sobre cuándo elegir `th3index` en lugar de `tgeompoint` o `tgeogpoint`; los peligros específicos de H3 que los consumidores deben anticipar (el orden int64 es arbitrario respecto a la geometría de la cuadrícula, la mezcla de resoluciones en las operaciones, las celdas pentagonales, la no preservación del orden de celdas en el ciclo de compactación, y el comportamiento en el antimeridiano y los polos a alta resolución); y las convenciones de durabilidad y almacenamiento se recogen en Sección 16.17.

16.1. Entrada y Salida

Un valor `th3index` se introduce utilizando la sintaxis temporal estándar con el identificador de celda H3 de 64 bits subyacente. El analizador acepta tanto la forma hexadecimal canónica (la que produce la CLI de H3 y `h3_cell_to_string` en el proyecto original) como el equivalente en decimal sin signo. El hexadecimal es idiomático en el ecosistema H3 y es lo que emite la función de salida — todos los ejemplos siguientes lo usan. Como referencia, el valor hexadecimal `831c02fffffffff` empleado en los ejemplos corresponde al valor decimal `590464338553208831`.

```
SELECT th3index '831c02fffffffff@2001-01-01';
SELECT th3index '{831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02}';
SELECT th3index '[831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02]';
SELECT th3index '{[831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02],
  [880326b885fffff@2001-01-03, 880326b88dfffff@2001-01-04]}';
```

El tipo acepta modificadores de tipo (`typmod`) para restringir una columna a un subtipo específico: `Instant`, `Sequence`, o `SequenceSet`.

```
SELECT th3index(Instant) '831c02fffffffff@2001-01-01';
SELECT th3index(SequenceSet) '{[831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02]}';
-- ERROR: el tipo temporal (Instant) no coincide con el tipo de columna (Sequence)
SELECT th3index(Sequence) '831c02fffffffff@2001-01-01';
```

Las funciones `asText`, `asBinary` y `asHexWKB` serializan un valor `th3index`, y `th3indexFromBinary` y `th3indexFromHex` lo reconstruyen. La forma hexadecimal WKB permite que una columna de índice H3 temporal se transfiera de ida y vuelta mediante un `COPY` basado en texto.

```
SELECT asHexWKB(th3index '831c02ffffffffff@2001-01-01');
SELECT th3indexFromHexWKB(asHexWKB(th3index '831c02ffffffffff@2001-01-01'));
-- 831c02ffffffffff@2001-01-01
```

16.2. Conversiones

Un valor `th3index` comparte la misma representación en disco que `tbigint`. La conversión por coerción binaria entre ellos es *explícita* — requiere `::` — para que las funciones específicas de H3 no puedan ser invocadas silenciosamente sobre trayectorias `tbigint` arbitrarias.

- Convertir entre un `th3index` y un `tbigint` (explícita, sin coste)

  `tbigint::th3index`

  `th3index::tbigint`

```
SELECT (tbigint '5904643385532088 31@2001-01-01')::th3index;
-- 831c02ffffffffff@2001-01-01
SELECT ((th3index '831c02ffffffffff@2001-01-01')::tbigint)::th3index
  = th3index '831c02ffffffffff@2001-01-01';
-- true
```

- Convertir una celda H3 temporal en un punto temporal geodésico o planar de los centroides de las celdas. Las dos sobrecargas dan al usuario una elección entre la representación geodésica (`tgeogpoint`) y la planar SRID 4326 (`tgeompoint`).

  `th3index::tgeogpoint`

  `th3index::tgeompoint`

```
SELECT (th3index '831c02ffffffffff@2001-01-01')::tgeogpoint;
SELECT (th3index '831c02ffffffffff@2001-01-01')::tgeompoint;
```

16.3. Cobertura de Geometrías Estáticas

Estas funciones convierten una geometría ordinaria (no temporal) en la celda o el conjunto de celdas H3 que la cubren a una resolución dada, y comprueban si una trayectoria H3 temporal entra alguna vez en ese conjunto de celdas. La salida de conjunto de celdas junto con el predicado de intersección «alguna vez» forman el prefiltro espacial multiplataforma usado por las consultas portables de BerlinMOD.

- Devuelve la celda H3 que contiene una geometría de punto a la resolución dada. La geometría debe ser un `POINT`.

  `geoToH3Cell(geometry,integer) → h3index`

```
SELECT geoToH3Cell(geometry 'SRID=4326;Point(4.35 50.85)', 7);
-- 871fa4418ffffff
```

- Devuelve el conjunto de celdas H3 que cubren una geometría a la resolución dada. Se admiten todos los tipos de geometría: un `POINT` produce una sola celda, una `LINESTRING` muestrea sus segmentos, un `POLYGON` rellena su interior, y los valores `MULTI*` / `GEOMETRYCOLLECTION` devuelven la unión de sus componentes.

  `geoToH3IndexSet(geometry,integer) → h3indexset`

```
SELECT geoToH3IndexSet(geometry 'SRID=4326;Point(4.35 50.85)', 7);
-- {"871fa4418ffffff"}
```

- Devuelve si una trayectoria H3 temporal toma alguna vez una de las celdas de un conjunto de celdas. Combinada con `geoToH3IndexSet` responde si un viaje pasa alguna vez por una región, sin materializar la geometría de la trayectoria — el prefiltro robusto y conservador para el predicado exacto `eIntersects(geometry,th3index)`.

  `eEq(h3indexset,th3index) → boolean`

  `h3indexset ?= th3index → boolean`

```
SELECT geoToH3IndexSet(geometry 'SRID=4326;Point(4.35 50.85)', 7) ?=
  th3index '871fa4418ffffff@2001-01-01';
-- true
```

16.4. Funciones de Acceso

- Devolver el primer o el último identificador de celda H3 de un valor temporal

  `startValue(th3index) → h3index`

  `endValue(th3index) → h3index`

```
SELECT startValue(th3index '831c02fffffffff@2001-01-01');
-- 831c02fffffffff
SELECT endValue(th3index '{831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02}');
-- 831c00fffffffff
```

- Devolver el `n`-ésimo identificador de celda H3 distinto (indexado desde 1). Devuelve `NULL` si está fuera del rango.

  `valueN(th3index,integer) → h3index`

```
SELECT valueN(th3index '{831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-02}', 1);
-- 831c02fffffffff
SELECT valueN(th3index '831c02fffffffff@2001-01-01', 99);
-- NULL
```

- Devolver el conjunto de identificadores de celda H3 distintos que toma la trayectoria

  `getValues(th3index) → h3indexset`

```
SELECT getValues(th3index '831c02fffffffff@2001-01-01');
-- {831c02fffffffff}
```

- Devolver el identificador de celda H3 en una marca de tiempo dada, utilizando interpolación de pasos

  `valueAtTimestamp(th3index,timestamptz) → h3index`

```
SELECT valueAtTimestamp(th3index
  '[831c02fffffffff@2001-01-01, 831c00fffffffff@2001-01-05]',
  '2001-01-03');
-- 831c02fffffffff
```

16.5. Inspección de Índices

- Devolver la resolución temporal (0–15) de cada celda en una trayectoria

  `th3GetResolution(th3index) → tint`

```
SELECT th3GetResolution(th3index '831c02fffffffff@2001-01-01');
-- 3@2001-01-01
```

- Devolver el número de celda base temporal (0–121) de cada celda en una trayectoria

  `th3GetBaseCellNumber(th3index) → tint`

```
SELECT th3GetBaseCellNumber(th3index '831c02fffffffff@2001-01-01');
```

- Devolver un booleano temporal que indica en cada instante si el valor es una celda H3 válida

  `isValidCell(th3index) → tbool`

```
SELECT isValidCell(th3index '0@2001-01-01');
-- f@2001-01-01
```

- Devolver un booleano temporal que indica si cada celda tiene orientación de Clase III. La Clase III se alterna con la resolución: las resoluciones impares son de clase III, las pares son de clase II.

  `th3IsResClassIii(th3index) → tbool`

```
SELECT th3IsResClassIii(th3index '871fa44a8ffffff@2001-01-01');
-- t@2001-01-01
```

- Devolver un booleano temporal que indica si cada celda es un pentágono (12 celdas por resolución son pentágonos; las restantes 110 en cada celda base descienden como hexágonos)

  `th3IsPentagon(th3index) → tbool`

```
SELECT th3IsPentagon(th3index '831c00fffffffff@2001-01-01');
-- t@2001-01-01
SELECT th3IsPentagon(th3index '831c02fffffffff@2001-01-01');
-- f@2001-01-01
```

16.6. Jerarquía

- Devolver la celda padre temporal en la resolución dada. La forma sin argumento desciende a la siguiente resolución más gruesa.

  `th3CellToParent(th3index,integer) → th3index`

  `th3CellToParent(th3index) → th3index`

```
SELECT th3GetResolution(
  th3CellToParent(th3index '8a2a1072b59ffff@2001-01-01', 5));
-- 5@2001-01-01
```

- Devolver la celda hijo central temporal en la resolución dada. La forma sin argumento desciende a la siguiente resolución más fina.

  `th3CellToCenterChild(th3index,integer) → th3index`

  `th3CellToCenterChild(th3index) → th3index`

```
SELECT th3CellToCenterChild(th3index '871fa44a8ffffff@2001-01-01', 8);
-- 881fa44a81fffff@2001-01-01
-- Without a resolution the child is one level finer.
SELECT th3CellToCenterChild(th3index '871fa44a8ffffff@2001-01-01');
-- 881fa44a81fffff@2001-01-01
```

- Devolver la posición ordinal (basada en 0) de cada celda entre los hijos de su padre en la resolución dada

  `th3CellToChildPos(th3index,integer) → tbigint`

```
SELECT th3CellToChildPos(th3index '871fa44a8ffffff@2001-01-01', 6);
-- 0@2001-01-01
```

- Devolver la celda hijo temporal en una posición ordinal dada del padre dado, en la resolución de hijo dada. Los dos operandos temporales se sincronizan sobre su eje de tiempo compartido.

  `th3ChildPosToCell(tbigint,th3index,integer) → th3index`

```
SELECT th3ChildPosToCell(tbigint '0@2001-01-01', th3index '871fa44a8ffffff@2001-01-01', 8) ↩
    ;
-- 881fa44a81fffff@2001-01-01
```

16.7. Conversiones de Latitud/Longitud

- Devolver la celda H3 temporal en la resolución dada que contiene cada punto de un punto temporal geodésico o planar (el SRID debe ser 4326 para la sobrecarga de `tgeompoint`)

  `th3index(tgeogpoint,integer) → th3index`

  `th3index(tgeompoint,integer) → th3index`

```
SELECT th3GetResolution(th3index(
  tgeogpoint 'POINT(-73.96 40.78)@2001-01-01', 9));
-- 9@2001-01-01
SELECT th3GetResolution(th3index(
  tgeompoint 'SRID=4326;POINT(-73.96 40.78)@2001-01-01', 9));
-- 9@2001-01-01
```

- Devolver el centroide geodésico temporal de cada celda en una trayectoria. Una sobrecarga planar (SRID 4326) está disponible bajo el nombre `th3CellToLatlngTgeompoint`.

  `th3CellToLatlng(th3index) → tgeogpoint`

  `th3CellToLatlngTgeompoint(th3index) → tgeompoint`

```
SELECT asText(th3CellToLatlng(th3index '871fa44a8ffffff@2001-01-01'), 6);
-- POINT(4.297401 50.8043)@2001-01-01
SELECT asText(th3CellToLatlngTgeompoint(th3index '871fa44a8ffffff@2001-01-01'), 6);
-- POINT(4.297401 50.8043)@2001-01-01
```

- Devolver el límite poligonal temporal de cada celda en una trayectoria, como una geografía temporal

  `th3CellToBoundary(th3index) → tgeography`

```
SELECT ST_GeometryType(getValue(th3CellToBoundary(th3index '871fa44a8ffffff@2001-01-01')) ↩
    ::geometry);
-- ST_Polygon
-- A hexagon closes on its first vertex, so it has seven points.
SELECT ST_NPoints(getValue(th3CellToBoundary(th3index '871fa44a8ffffff@2001-01-01')):: ↩
    geometry);
-- 7
```

16.8. Aristas Dirigidas

Las aristas dirigidas H3 son a su vez identificadores de 64 bits (codificados de manera diferente a los identificadores de celda — `isValidDirectedEdge` los distingue). El tipo `th3index` de MobilityDB transporta las aristas dirigidas por convención: el mismo tipo, interpretado a través de las funciones de acceso específicas de las aristas.

- Devolver un booleano temporal que indica si dos celdas temporales son vecinas en la cuadrícula en cada instante. Los dos operandos se sincronizan.

  `th3AreNeighborCells(th3index,th3index) → tbool`

```
SELECT th3AreNeighborCells(
  th3index '880326b885fffff@2001-01-01',
  th3index '880326b88dfffff@2001-01-01');
-- t@2001-01-01
```

- Devolver un identificador de arista dirigida temporal a partir de un par de celdas temporales origen/destino

  `th3CellsToDirectedEdge(th3index,th3index) → th3index`

```
SELECT th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001-01-01',
    th3index '871fa44aeffffff@2001-01-01');
-- 1671fa44a8ffffff@2001-01-01
```

- Devolver un booleano temporal que indica en cada instante si el valor es una arista dirigida H3 válida

  `isValidDirectedEdge(th3index) → tbool`

```
SELECT isValidDirectedEdge(th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001-01-01',
    th3index '871fa44aeffffff@2001-01-01'));
-- t@2001-01-01
SELECT isValidDirectedEdge(th3index '871fa44a8ffffff@2001-01-01');
-- f@2001-01-01
```

- Devolver la celda origen o destino temporal de una arista dirigida temporal

  `th3GetDirectedEdgeOrigin(th3index) → th3index`

  `th3GetDirectedEdgeDestination(th3index) → th3index`

```
SELECT th3GetDirectedEdgeOrigin(th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001 ↩
    -01-01',
    th3index '871fa44aeffffff@2001-01-01'));
-- 871fa44a8ffffff@2001-01-01
SELECT th3GetDirectedEdgeDestination(th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001 ↩
    -01-01',
    th3index '871fa44aeffffff@2001-01-01'));
-- 871fa44aeffffff@2001-01-01
```

- Devolver el límite poligonal temporal de cada arista dirigida en una trayectoria, como una geografía temporal

  `th3DirectedEdgeToBoundary(th3index) → tgeography`

```
SELECT ST_NPoints(getValue(th3DirectedEdgeToBoundary(
  th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001-01-01',
    th3index '871fa44aeffffff@2001-01-01')))::geometry);
-- 3
```

16.9. Vértices

- Devolver el vértice H3 temporal en el número de vértice dado (0–5 para hexágonos, 0–4 para pentágonos)

  `th3CellToVertex(th3index,integer) → th3index`

```
SELECT th3CellToVertex(th3index '871fa44a8ffffff@2001-01-01', 0);
-- 2071fa44a8ffffff@2001-01-01
```

- Devolver las coordenadas geodésicas temporales de cada vértice en una trayectoria

  `th3VertexToLatlng(th3index) → tgeogpoint`

```
SELECT asText(th3VertexToLatlng(th3CellToVertex(th3index '871fa44a8ffffff@2001-01-01', 0)) ↩
    , 6);
-- POINT(4.280027 50.807655)@2001-01-01
```

- Devolver un booleano temporal que indica en cada instante si el valor es un vértice H3 válido

  `isValidVertex(th3index) → tbool`

```
SELECT isValidVertex(th3CellToVertex(th3index '871fa44a8ffffff@2001-01-01', 0));
-- t@2001-01-01
```

16.10. Recorrido de la Cuadrícula

- Devolver la distancia temporal de saltos en la cuadrícula (número de pasos de una sola celda) entre dos celdas temporales. De forma equivalente, el operador <-> devuelve la misma distancia. Los dos operandos se sincronizan sobre su eje de tiempo compartido.

  `th3GridDistance(th3index,th3index) → tbigint`

  `th3index <-> th3index → tbigint`

```
SELECT th3index '880326b885fffff@2001-01-01'
   <-> th3index '880326b88dfffff@2001-01-01';
-- 1@2001-01-01
```

- Convertir entre una celda temporal y un par de coordenadas locales (I, J) temporales relativas a una celda temporal de origen. La coordenada local se expone como un `tgeompoint` con el eje I en el slot `x` y J en `y`.

  `th3CellToLocalIj(th3index,th3index) → tgeompoint`

  `th3LocalIjToCell(th3index,tgeompoint) → th3index`

```
SELECT asText(th3CellToLocalIj(th3index '871fa44a8ffffff@2001-01-01',
  th3index '871fa44aeffffff@2001-01-01'));
-- POINT(497 320)@2001-01-01
-- The two are inverse of each other.
SELECT th3LocalIjToCell(th3index '871fa44a8ffffff@2001-01-01',
  th3CellToLocalIj(th3index '871fa44a8ffffff@2001-01-01',
    th3index '871fa44aeffffff@2001-01-01'));
-- 871fa44aeffffff@2001-01-01
```

16.11. Métricas

Las tres funciones métricas aceptan un argumento opcional de unidad. Para áreas: `km2` (por defecto), `m2`, o `rads2`. Para longitudes: `km` (por defecto), `m`, o `rads`. Pasar una unidad de área a una función de longitud (o viceversa) provoca un error.

- Devolver el área temporal de cada celda en una trayectoria, en la unidad solicitada

  `th3CellArea(th3index,text='km2') → tfloat`

```
SELECT th3CellArea(th3index '871fa44a8ffffff@2001-01-01');
-- 4.441914270110932@2001-01-01
SELECT th3CellArea(th3index '871fa44a8ffffff@2001-01-01', 'm2');
-- 4441914.270110932@2001-01-01
```

- Devolver la longitud temporal de cada arista dirigida en una trayectoria, en la unidad solicitada

  `th3EdgeLength(th3index,text='km') → tfloat`

```
SELECT th3EdgeLength(th3CellsToDirectedEdge(th3index '871fa44a8ffffff@2001-01-01',
    th3index '871fa44aeffffff@2001-01-01'));
-- 1.272084901305088@2001-01-01
```

- Devolver la distancia temporal del gran círculo entre dos puntos geodésicos temporales (no entre celdas). Los dos operandos se sincronizan.

  `greatCircleDistance(tgeogpoint,tgeogpoint,text='km') → tfloat`

```
SELECT greatCircleDistance(tgeogpoint 'Point(0 0)@2001-01-01',
  tgeogpoint 'Point(1 0)@2001-01-01');
-- 111.195051975229 43@2001-01-01
```

16.12. Comparaciones

16.12.1. Comparaciones Siempre y Alguna Vez

El operando de prueba puede ser un identificador de celda H3 `h3index` simple o otra trayectoria `th3index`; ambos son soportados en cualquier posición de operando.

- Devolver `true` si una celda H3 temporal es alguna vez igual a la prueba en al menos un instante

  `eEq({h3index,th3index},th3index) → boolean`

  `eEq(th3index,h3index) → boolean`

  `{h3index,th3index} ?= th3index`

  `th3index ?= h3index`

```
SELECT th3index
  '[831c02fffffffff@2001-01-01, 880326b885fffff@2001-01-05]'
  ?= 880326b885fffff::h3index;
-- true
```

- Devolver `true` si una celda H3 temporal es siempre igual a la prueba a lo largo de todo su eje de tiempo

  `aEq({h3index,th3index},th3index) → boolean`

  `aEq(th3index,h3index) → boolean`

  `{h3index,th3index}%= th3index`

  `th3index%= h3index`

```
SELECT aEq(th3index '871fa44a8ffffff@2001-01-01', h3index '871fa44a8ffffff');
-- t
```

- Devolver `true` si una celda H3 temporal es alguna vez diferente de la prueba

  `eNe({h3index,th3index},th3index) → boolean`

  `eNe(th3index,h3index) → boolean`

```
SELECT eNe(th3index '871fa44a8ffffff@2001-01-01', h3index '871fa44aeffffff');
-- t
```

- Devolver `true` si una celda H3 temporal es siempre diferente de la prueba

  `aNe({h3index,th3index},th3index) → boolean`

  `aNe(th3index,h3index) → boolean`

```
SELECT aNe(th3index '871fa44a8ffffff@2001-01-01', h3index '871fa44aeffffff');
-- t
```

16.12.2. Comparaciones Temporales

- Devolver un booleano temporal que da la igualdad (o desigualdad) por instante entre una celda H3 temporal y una prueba

  `tEq({h3index,th3index},th3index) → tbool`

  `tEq(th3index,h3index) → tbool`

  `tNe({h3index,th3index},th3index) → tbool`

  `tNe(th3index,h3index) → tbool`

```
SELECT tEq(th3index '871fa44a8ffffff@2001-01-01', h3index '871fa44a8ffffff');
-- t@2001-01-01
SELECT tNe(th3index '871fa44a8ffffff@2001-01-01', h3index '871fa44a8ffffff');
-- f@2001-01-01
```

16.13. Operadores de Caja Envolvente

Los siguientes operadores comparan la *caja envolvente* espaciotemporal de las celdas H3 temporales, no la inclusión jerárquica de celdas. `th3index` es un tipo temporal espacial: su caja envolvente es un `stbox` que combina la extensión espacial geodésica de los límites de las celdas (su envolvente de longitud/latitud en SRID 4326) con el período temporal `tstzspan` de la trayectoria. Los operadores a continuación comparan esa caja a lo largo de los ejes espacial y/o temporal. Consulte la sección de indexación para las clases de operadores que hacen estos operadores indexables.

---

nota
El operador `@>` prueba la inclusión de la caja envolvente espaciotemporal (extensión espacial y tiempo juntos), *no* la inclusión jerárquica de H3. Para probar si una celda es el padre de otra en una resolución dada, use th3CellToParent.

- Solapamiento de caja envolvente, contiene, contenido por, igual y adyacente

  `th3index && {tstzspan,stbox,th3index} → boolean`

  `th3index @> {tstzspan,stbox,th3index} → boolean`

  `th3index <@ {tstzspan,stbox,th3index} → boolean`

  `th3index ~= {tstzspan,stbox,th3index} → boolean`

  `th3index -|- {tstzspan,stbox,th3index} → boolean`

```
SELECT th3index '871fa44a8ffffff@2001-01-01' && tstzspan '[2001-01-01, 2001-01-02]';
-- t
SELECT th3index '871fa44a8ffffff@2001-01-01' ~= th3index '871fa44a8ffffff@2001-01-01';
-- t
```

- Posición relativa a lo largo de los ejes espaciales (estrictamente a la izquierda, solapa a la izquierda, solapa a la derecha, estrictamente a la derecha; estrictamente abajo, solapa abajo, solapa arriba, estrictamente arriba)

  `th3index << {stbox,th3index} → boolean`

  `th3index &< {stbox,th3index} → boolean`

  `th3index &> {stbox,th3index} → boolean`

  `th3index >> {stbox,th3index} → boolean`

  `th3index <<| {stbox,th3index} → boolean`

  `th3index &<| {stbox,th3index} → boolean`

  `th3index |&> {stbox,th3index} → boolean`

  `th3index |>> {stbox,th3index} → boolean`

```
SELECT th3index '871fa44a8ffffff@2001-01-01' <<
  th3index '871fa44aeffffff@2001-01-01';
-- f
```

- Posición relativa a lo largo del eje de tiempo (estrictamente antes, solapa antes, estrictamente después, solapa después)

  `th3index <<# {tstzspan,stbox,th3index} → boolean`

  `th3index &<# {tstzspan,stbox,th3index} → boolean`

  `th3index #>> {tstzspan,stbox,th3index} → boolean`

  `th3index #&> {tstzspan,stbox,th3index} → boolean`

```
SELECT th3index '871fa44a8ffffff@2001-01-01' <<#
  th3index '871fa44aeffffff@2001-01-05';
-- t
SELECT th3index '871fa44a8ffffff@2001-01-01' #>>
  th3index '871fa44aeffffff@2001-01-05';
-- f
```

16.14. Funciones que devuelven conjuntos

Las siguientes funciones estáticas devuelven un `h3indexset` (o `intset` para `h3GetIcosahedronFaces`) — el tipo compañero de colección finita de `h3index`. Operan sobre un `h3index` estático (o un `h3indexset`) y no están elevadas a `th3index`; el levantamiento temporal está diferido a la espera de una primitiva `tset<T>`.

- Todas las celdas dentro de `k` pasos de cuadrícula del origen (incluyendo el origen cuando k=0).

  `h3GridDisk(h3index, integer) → h3indexset`

```
SELECT h3GridDisk(h3index '871fa44a8ffffff', 1);
-- {"871fa44a8ffffff", "871fa44a9ffffff", "871fa44aaffffff", "871fa44abffffff",
--  "871fa44acffffff", "871fa44adffffff", "871fa44aeffffff"}
```

- Las celdas a *exactamente* `k` pasos de cuadrícula del origen. Falla cerca de pentágonos.

  `h3GridRing(h3index, integer) → h3indexset`

```
SELECT h3GridRing(h3index '871fa44a8ffffff', 1);
-- {"871fa44a9ffffff", "871fa44aaffffff", "871fa44abffffff", "871fa44acffffff",
--  "871fa44adffffff", "871fa44aeffffff"}
```

- El camino inclusivo de celdas entre dos celdas de la misma resolución.

  `h3GridPathCells(h3index, h3index) → h3indexset`

```
SELECT h3GridPathCells(h3index '871fa44a8ffffff', h3index '871fa44b1ffffff');
-- {"871fa4484ffffff", "871fa44a3ffffff", "871fa44a8ffffff", "871fa44aeffffff",
--  "871fa44b1ffffff"}
```

- Todas las celdas hijas de una celda en la resolución objetivo.

  `h3CellToChildren(h3index, integer) → h3indexset`

```
SELECT h3CellToChildren(h3index '831c02fffffffff', 4);
-- {"841c021fffffff", "841c023fffffff", "841c025fffffff", "841c027fffffff",
--  "841c029fffffff", "841c02bfffffff", "841c02dfffffff"}
```

- Representación compactada de un conjunto de celdas — las celdas más finas se fusionan en su padre cuando el conjunto hexagonal completo de hermanas está presente.

  `h3CompactCells(h3indexset) → h3indexset`

```
SELECT h3CompactCells(h3CellToChildren(h3index '831c02fffffffff', 4));
-- {"831c02fffffffff"}
```

- Representación descompactada a la resolución dada — la inversa de `h3CompactCells`.

  `h3UncompactCells(h3indexset, integer) → h3indexset`

```
SELECT numValues(h3UncompactCells(
  h3CompactCells(h3CellToChildren(h3index '831c02fffffffff', 4)), 4));
-- 7
```

- Todas las aristas dirigidas salientes de una celda (6 para hexágonos, 5 para pentágonos).

  `h3OriginToDirectedEdges(h3index) → h3indexset`

```
SELECT h3OriginToDirectedEdges(h3index '871fa44a8ffffff');
-- {"1171fa44a8ffffff", "1271fa44a8ffffff", "1371fa44a8ffffff",
--  "1471fa44a8ffffff", "1571fa44a8ffffff", "1671fa44a8ffffff"}
```

- Todos los vértices de una celda (6 para hexágonos, 5 para pentágonos).

  `h3CellToVertexes(h3index) → h3indexset`

```
SELECT h3CellToVertexes(h3index '871fa44a8ffffff');
-- {"2071fa44a8ffffff", "2171fa44a8ffffff", "2271fa44a8ffffff",
--  "2371fa44a8ffffff", "2471fa44a8ffffff", "2571fa44a8ffffff"}
```

- Los índices de caras del icosaedro (0..19) intersectadas por una celda.

  `h3GetIcosahedronFaces(h3index) → intset`

```
SELECT h3GetIcosahedronFaces(h3index '831c02fffffffff');
-- {6, 11}
```

16.15. Relaciones Espaciales

Las relaciones espaciales evalúan un argumento `th3index` a través de la geometría de frontera de celda en cada instante —la huella hexagonal de cada celda, devuelta por `th3CellToBoundary`— y luego aplican la relación correspondiente de los tipos geométricos temporales. Cada relación tiene tres formas cuantificadas: la forma *alguna vez* con prefijo `e` devuelve `true` cuando la relación se cumple en algún instante, la forma *siempre* con prefijo `a` cuando se cumple en todos los instantes, y la forma *temporal* con prefijo `t` devuelve un `tbool` con el resultado instante a instante. Todas las formas aceptan los tres órdenes de argumentos `(geometry, th3index)`, `(th3index, geometry)` y `(th3index, th3index)`.

- Si una huella contiene a la otra (solo interiores)

  `eContains, aContains, tContains`

- Si una huella cubre a la otra (frontera incluida)

  `eCovers, aCovers, tCovers`

- Si las huellas no comparten ningún punto

  `eDisjoint, aDisjoint, tDisjoint`

- Si las huellas comparten al menos un punto

  `eIntersects, aIntersects, tIntersects`

- Si las huellas comparten un punto de frontera pero ningún punto interior

  `eTouches, aTouches, tTouches`

- Si las huellas se encuentran dentro de una distancia dada, pasada como tercer argumento

  `eDwithin, aDwithin, tDwithin (..., dist float)`

```
SELECT eContains(
  geometry 'SRID=4326;Polygon((4.20 50.70, 4.50 50.70, 4.50 50.90, 4.20 50.90, 4.20 50.70)) ↩
      ',
  th3index '871fa44a8ffffff@2001-01-01');
-- true
```

16.16. Indexación

Las clases de operadores GiST y SP-GiST están definidas para que los operadores de caja envolvente de la sección anterior puedan ser utilizados con los índices correspondientes.

- Clase de operadores GiST para celdas H3 temporales. Indexa una caja envolvente espaciotemporal `stbox` por fila (la extensión geodésica de los límites de las celdas junto con el período temporal).

```
CREATE INDEX ON trips USING gist(cells);       -- usa th3_rtree_ops por defecto
CREATE INDEX ON trips USING gist(cells th3_rtree_ops);
```

- Clases de operadores SP-GiST para celdas H3 temporales, indexadas por la caja envolvente espaciotemporal `stbox`. `th3_quadtree` es la predeterminada (partición por quadtree); `th3_kdtree_ops` utiliza partición por árbol k-d y debe ser nombrada explícitamente.

```
CREATE INDEX ON trips USING spgist(cells);     -- usa th3_quadtree_ops por defecto
CREATE INDEX ON trips USING spgist(cells th3_kdtree_ops);
```

16.17. Notas operativas

Esta sección recoge el conocimiento operativo que los consumidores necesitan además de la referencia de funciones: cuándo `th3index` es el tipo adecuado, las convenciones que asume la implementación y los peligros específicos de H3 que las cargas de trabajo reales deben tener en cuenta. La cuadrícula hexagonal jerárquica de H3 tiene algunas propiedades no evidentes que la codificación binaria `int64` oculta; esta sección las expone para que no sorprendan a los consumidores.

16.17.1. Cuándo usar th3index

Use `th3index` cuando el identificador natural de una ubicación discreta a una resolución dada sea una celda H3 — viajes de vehículos agrupados a la resolución H3 9, lecturas de sensores ambientales agrupadas por celda H3, eventos geo-cercados identificados por su celda contenedora. El tipo almacena una celda H3 por instante; la resolución puede variar entre los instantes de una misma trayectoria, pero la mayoría de las cargas de trabajo fijan una resolución por columna.

Elija `tgeompoint` / `tgeogpoint` en lugar de `th3index` cuando el valor de interés sea la posición geográfica real (precisión subcelular). Elija `tbigint` en lugar de `th3index` cuando los valores resulten ser enteros de 64 bits pero no tengan semántica H3 — las representaciones binarias son idénticas (se convierten sin pérdida mediante `th3index :: tbigint` y viceversa) pero el sistema de tipos SQL usa la distinción para rechazar funciones agnósticas a H3 sobre trayectorias específicas de H3.

16.17.2. Peligros específicos de H3

Cinco peligros afectan de forma fiable a las cargas de trabajo que usan celdas H3. Cada uno es una propiedad de la cuadrícula H3 en sí más que una decisión de implementación de MobilityDB; las mitigaciones siguientes explican cómo trabajar con ellos.

- *El orden int64 es arbitrario respecto a la geometría de la cuadrícula*. Los identificadores de celda H3 son enteros sin signo de 64 bits cuyo patrón de bits codifica resolución, celda base y posición jerárquica. El orden binario de dos celdas no tiene significado geográfico — dos celdas adyacentes en bits pueden estar en lados opuestos del planeta, y dos celdas espacialmente adyacentes pueden tener patrones de bits muy distintos.

  Mitigación: `th3index` deliberadamente no tiene tipos compañeros `h3span` / `h3spanset` (precedente: `geometry` no tiene `geometryspan`), y los índices de caja envolvente capturan la extensión espaciotemporal de las celdas (`stbox`), no el valor del identificador de celda. El filtrado por rango de valores debe pasar por funciones de inspección `h3_*` (resolución, celda base, comprobaciones de jerarquía) o la enumeración explícita de conjuntos mediante `h3indexset`. El código que asume que `cell_a < cell_b` refleja proximidad espacial producirá resultados silenciosamente erróneos.

- *Mezcla de resoluciones en las operaciones*. Las celdas H3 a distintas resoluciones (0–15) representan áreas de cobertura diferentes. Mezclar resoluciones en una misma trayectoria es válido pero semánticamente requiere justificación explícita — `th3CellToParent(cell, coarser_res)` engrosa, `h3CellToChildren(parent, finer_res)` refina, y `h3CompactCells` / `h3UncompactCells` realizan el ciclo de ida y vuelta correctamente solo cuando las resoluciones de entrada son compatibles.

  Mitigación: los consumidores deberían documentar el invariante de resolución por trayectoria (p. ej. "todas las celdas son de resolución 9") y validar las entradas en el límite de ingesta. El accesor `th3GetResolution` permite que una restricción CHECK lo imponga.

- *Celdas pentagonales*. La cuadrícula H3 tiene 12 celdas pentagonales (en lugar de hexagonales) por resolución — los duales de Eisenstein de los 12 vértices del icosaedro. `h3GridRing` puede fallar cerca de estos pentágonos; `h3GridPathCells` falla si el camino cruza uno. El error es explícito (libh3 lanza un fallo explícito), pero las cargas de trabajo que esperan que las operaciones de anillo / camino siempre tengan éxito necesitan una protección.

  Mitigación: el código defensivo debería envolver las llamadas sensibles a pentágonos y comprobar `h3_is_pentagon_cell` en las entradas y salidas clave. Para trayectorias que pasan cerca de un pentágono, recurra a `h3GridDisk` (que es seguro ante pentágonos) o calcule los segmentos del camino por partes alrededor del pentágono.

- *Ciclo de compactación / descompactación*. `h3CompactCells` produce la representación mixta de resoluciones más compacta de un conjunto de celdas; `h3UncompactCells` la expande de nuevo. El ciclo es sin pérdida en extensión espacial, pero *el orden de las celdas no se preserva* — libh3 no garantiza una salida ordenada. Las consultas que consumen la salida como si su orden fuera estable devolverán resultados distintos antes y después de la compactación aunque el conjunto espacial subyacente sea idéntico.

  Mitigación: reordene tras la compactación si el orden importa (p. ej. `ORDER BY cell` en SQL o `ARRAY_AGG ... ORDER BY 1` al materializar). Para conjuntos temporales, prefiera la semántica de igualdad de conjuntos de `h3indexset` sobre la de igualdad de arreglos.

- *Comportamiento en el antimeridiano y los polos*. Las celdas base de H3 se posicionan sobre un icosaedro; a altas resoluciones algunas celdas cerca del antimeridiano o los polos pueden tener una geometría poco intuitiva. `th3CellToLatlng` / `h3_latlng_to_cell` realizan el ciclo de ida y vuelta correctamente, pero las coordenadas cerca de $\pm 180°$ de longitud o cerca de los polos pueden resolverse a celdas en grupos de celdas base inesperados.

  Mitigación: las cargas de trabajo en latitudes polares o que cruzan el antimeridiano deberían validar celdas representativas contra la implementación de referencia de libh3 directamente y añadir fixtures para los casos límite específicos que la carga de trabajo encuentre.

16.17.3. Durabilidad y almacenamiento

La representación en disco de `th3index` es byte a byte idéntica a la de `tbigint` (cada instante lleva un ID de celda H3 de 64 bits más una marca de tiempo). Consecuencias para la planificación del almacenamiento:

- *WKT* mediante `th3_in` y `th3_out`. Las celdas se representan como cadenas hexadecimales canónicas (p. ej. `'8928308280ffff` se aceptan tanto decimal como hexadecimal en la entrada. El ciclo de ida y vuelta es estable a nivel de bits.

- *WKB / EWKB / HexWKB* mediante el patrón estándar de PostGIS de bandera de endianidad seguida de la carga útil. Estable entre versiones mayores de PostgreSQL.

- *MFJSON* mediante `asMFJSON(th3index)` y `th3indexFromMFJSON(text)`. La carga útil de la celda se representa como el identificador int64 de la celda en el arreglo `values`, la misma representación que utiliza `tbigint`; los consumidores que necesiten la forma hexadecimal canónica aplican `h3_cell_to_string`.
- *pg_dump* usa WKT en modo plano y WKB bajo `--binary-upgrade`; ambos realizan el ciclo de ida y vuelta limpiamente.
- *Conversión cruzada con `tbigint`*: las conversiones bidireccionales `th3index :: tbigint` son reinterpretaciones binarias sin coste. Úselas al trabajar con vistas tanto seguras por tipo (`th3index`) como aptas para aritmética (`tbigint`) de la misma trayectoria en una sola consulta.

Capítulo 17

Índices de Celdas QUADBIN Temporales

El tipo de índice de celda QUADBIN temporal `tquadbin` representa el movimiento de los objetos como una secuencia de identificadores de celdas cuadradas QUADBIN. QUADBIN de CARTO es un sistema de cuadrícula cuadrada jerárquica construido sobre el esquema de teselas slippy-map (web-mercator): una única celda mundial en la resolución 0 se subdivide en cuatro hijos iguales en cada resolución más fina (hasta la resolución 26). Cada celda tiene un identificador entero de 64 bits que codifica una etiqueta de cabecera, la resolución y las coordenadas de tesela de la celda.

El tipo `tquadbin` comparte su representación en disco con `tbigint`: ambos almacenan un entero temporal de 64 bits. El tipo SQL distinto existe únicamente para que el verificador de tipos rechace las funciones específicas de QUADBIN cuando se aplican a trayectorias `tbigint` arbitrarias (y viceversa). Las conversiones desde y hacia `tbigint` son *explícitas* (`AS ASSIGNMENT`) y de coerción binaria solamente — sin coste en tiempo de ejecución pero requieren un `::` explícito en SQL.

Como con otros tipos temporales, `tquadbin` tiene cuatro subtipos: `Instant`, secuencia discreta, secuencia continua con interpolación de pasos (las celdas QUADBIN son discretas — la interpolación lineal carece de sentido), y `SequenceSet`.

La guía operativa sobre cuándo elegir `tquadbin` en lugar de `tgeompoint` o `tgeogpoint`; los peligros específicos de QUADBIN que los consumidores deben anticipar (el orden int64 es arbitrario respecto a la geometría de la cuadrícula, la mezcla de resoluciones en las operaciones, el límite de latitud de web-mercator, y el comportamiento en el antimeridiano); y las convenciones de durabilidad y almacenamiento se recogen en Sección 17.12.

17.1. Entrada y Salida

Un valor `tquadbin` se introduce utilizando la sintaxis temporal estándar con el identificador de celda QUADBIN de 64 bits subyacente. El analizador acepta tanto la forma hexadecimal canónica como el equivalente en decimal sin signo. El hexadecimal es idiomático en el ecosistema QUADBIN y es lo que emite la función de salida — todos los ejemplos siguientes lo usan. Como referencia, el valor hexadecimal `480fffffffffffff` empleado en los ejemplos es la celda mundial de resolución 0, igual al valor decimal `5192650370358181887`.

```
SELECT tquadbin '480fffffffffffff@2001-01-01';
SELECT tquadbin '{480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02}';
SELECT tquadbin '[480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02]';
SELECT tquadbin '{[480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02],
  [48a6227affffffff@2001-01-03, 485623ffffffffff@2001-01-04]}';
```

El tipo acepta modificadores de tipo (`typmod`) para restringir una columna a un subtipo específico: `Instant`, `Sequence`, o `SequenceSet`.

```
SELECT tquadbin(Instant) '480fffffffffffff@2001-01-01';
SELECT tquadbin(SequenceSet) '{[480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02]}' ↩
    ;
-- ERROR: el tipo temporal (Instant) no coincide con el tipo de columna (Sequence)
SELECT tquadbin(Sequence) '480fffffffffffff@2001-01-01';
```

Las funciones `asText`, `asBinary` y `asHexWKB` serializan un valor `tquadbin`, y `tquadbinFromBinary` y `tquadbinFromHex` lo reconstruyen. La forma hexadecimal WKB permite que una columna de índice QUADBIN temporal se transfiera de ida y vuelta mediante un `COPY` basado en texto.

```
SELECT asHexWKB(tquadbin '480fffffffffffff@2001-01-01');
SELECT tquadbinFromHexWKB(asHexWKB(tquadbin '480fffffffffffff@2001-01-01'));
-- 480fffffffffffff@2001-01-01
```

La representación *MF-JSON* está disponible mediante `asMFJSON(tquadbin)` y `tquadbinFromMFJSON(text)`. La carga útil de la celda se representa como el identificador int64 de la celda en el arreglo `values`, la misma representación que utiliza `tbigint`.

```
SELECT asMFJSON(tquadbin '480fffffffffffff@2001-01-01');
SELECT tquadbinFromMFJSON(asMFJSON(tquadbin '480fffffffffffff@2001-01-01'));
-- 480fffffffffffff@2001-01-01
```

17.2. Conversiones

Un valor `tquadbin` comparte la misma representación en disco que `tbigint`. La conversión por coerción binaria entre ellos es *explícita* — requiere `::` — para que las funciones específicas de QUADBIN no puedan ser invocadas silenciosamente sobre trayectorias `tbigint` arbitrarias.

- Convertir entre un `tquadbin` y un `tbigint` (explícito, sin coste)

  `tbigint::tquadbin`

  `tquadbin::tbigint`

```
SELECT (tbigint '5192650370358181887@2001-01-01')::tquadbin;
-- 480fffffffffffff@2001-01-01
SELECT ((tquadbin '480fffffffffffff@2001-01-01')::tbigint)::tquadbin
  = tquadbin '480fffffffffffff@2001-01-01';
-- true
```

- Convertir una celda QUADBIN temporal a un punto planar temporal (SRID 4326) de los centroides de las celdas. La conversión delega en `tquadbinCellToPoint`.

  `tquadbin::tgeompoint`

```
SELECT (tquadbin '480fffffffffffff@2001-01-01')::tgeompoint;
```

17.3. Celdas Estáticas

El tipo estático complementario `quadbin` contiene un único identificador de celda QUADBIN (no temporal); `quadbinset` contiene un conjunto finito de ellos. Comparten la misma representación textual hexadecimal / decimal que el tipo temporal. El tipo estático lleva los operadores estándar de igualdad / orden (una clase de operadores btree y una clase hash), y expone los predicados de validez siguientes.

- Devuelve si un valor es un identificador QUADBIN estructuralmente válido (etiqueta de cabecera y campo de resolución correctos)

  `isValidIndex(quadbin) → boolean`

```
SELECT isValidIndex(quadbin '480fffffffffffff');
-- t
-- The input function already rejects malformed text, so a value that is
-- structurally invalid can only arrive through the bigint conversion.
SELECT isValidIndex(0::bigint::quadbin);
-- f
```

- Devuelve si un valor es una celda QUADBIN válida (un índice válido cuyas coordenadas de tesela están dentro de los límites de su resolución)

  `isValidCell(quadbin) → boolean`

```
SELECT isValidCell(quadbin '480fffffffffffff');
-- true
SELECT isValidCell(quadbin '0');
-- false
```

17.4. Accesores

- Devuelve el primer o el último identificador de celda QUADBIN de un valor temporal

  `startValue(tquadbin) → quadbin`

  `endValue(tquadbin) → quadbin`

```
SELECT startValue(tquadbin '480fffffffffffff@2001-01-01');
-- 480fffffffffffff
SELECT endValue(tquadbin '{480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02}');
-- 48427fffffffffff
```

- Devuelve el `n`-ésimo identificador de celda QUADBIN distinto (índice base 1). Devuelve `NULL` si está fuera de rango.

  `valueN(tquadbin,integer) → quadbin`

```
SELECT valueN(tquadbin '{480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02}', 1);
-- 480fffffffffffff
SELECT valueN(tquadbin '{480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02}', 2);
-- 48427fffffffffff
SELECT valueN(tquadbin '480fffffffffffff@2001-01-01', 99);
-- NULL
```

- Devuelve el conjunto de identificadores de celda QUADBIN distintos que toma la trayectoria

  `getValues(tquadbin) → quadbinset`

```
SELECT getValues(tquadbin '480fffffffffffff@2001-01-01');
-- {480fffffffffffff}
SELECT getValues(tquadbin
  '[480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02,
    48a6227affffffff@2001-01-03]');
-- {480fffffffffffff, 48427fffffffffff, 48a6227affffffff}
```

- Devuelve el identificador de celda QUADBIN en un instante dado, usando interpolación de pasos

  `valueAtTimestamp(tquadbin,timestamptz) → quadbin`

```
SELECT valueAtTimestamp(tquadbin
  '[480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-05]',
  '2001-01-03');
-- 480fffffffffffff
```

17.5. Inspección del Índice

- Devuelve el nivel de resolución (zoom) (0–26) de una celda. La forma estática devuelve un `integer`; la forma temporal devuelve la resolución de cada celda de una trayectoria como un `tint`.

  `quadbinGetResolution(quadbin) → integer`

  `tquadbinGetResolution(tquadbin) → tint`

```
SELECT quadbinGetResolution(quadbin '480fffffffffffff');
-- 0
SELECT tquadbinGetResolution(tquadbin '48427fffffffffff@2001-01-01');
-- 4@2001-01-01
```

- Devuelve un booleano temporal que indica en cada instante si el valor es una celda QUADBIN válida

  `isValidCell(tquadbin) → tbool`

```
SELECT isValidCell(tquadbin '480ffffffffffffff@2001-01-01');
-- t@2001-01-01
```

17.6. Jerarquía

- Devuelve la celda padre en la resolución más gruesa dada. La forma estática toma un único `quadbin`; la forma temporal devuelve el padre de cada celda de una trayectoria.

  `quadbinCellToParent(quadbin,integer) → quadbin`

  `tquadbinCellToParent(tquadbin,integer) → tquadbin`

```
SELECT quadbinCellToParent(quadbin '48a6227affffffff', 5);
-- 485623ffffffffff
SELECT tquadbinGetResolution(
  tquadbinCellToParent(tquadbin '48a6227affffffff@2001-01-01', 5));
-- 5@2001-01-01
```

- Devuelve las cuatro celdas hijas en la resolución más fina solicitada como un `quadbinset`. Es un auxiliar estático que devuelve un conjunto sobre un único `quadbin`; no tiene elevación temporal (los hijos son un conjunto por instante, diferido a la espera de una primitiva `tset<T>`).

  `quadbinCellToChildren(quadbin,integer) → quadbinset`

```
SELECT quadbinCellToChildren(quadbin '480ffffffffffffff', 1);
-- {4813fffffffffffff, 4817fffffffffffff, 481bfffffffffffff, 481ffffffffffffff}
```

- Devuelve la celda vecina en la misma resolución en la dirección dada (`up` / `down` / `left` / `right`)

  `quadbinCellSibling(quadbin,text) → quadbin`

```
SELECT quadbinCellSibling(
  quadbinCellSibling(quadbin '48427fffffffffff', 'right'), 'left')
  = quadbin '48427fffffffffff';
-- true
```

- Devuelve todas las celdas dentro de `k` pasos de cuadrícula del origen (inclusive del origen en k=0), como un `quadbinset`. En la cuadrícula cuadrada el disco en el paso `k` es el bloque cuadrado de `(2k+1)` × `(2k+1)` celdas centrado en el origen.

  `quadbinGridDisk(quadbin,integer) → quadbinset`

```
SELECT numValues(quadbinGridDisk(quadbin '48a6227affffffff', 0));  -- 1
SELECT numValues(quadbinGridDisk(quadbin '48a6227affffffff', 1));  -- 9
SELECT numValues(quadbinGridDisk(quadbin '48a6227affffffff', 2));  -- 25
```

17.7. Conversiones de Geometría

- Devuelve la celda QUADBIN en la resolución dada que contiene una geometría de punto. La geometría debe ser un `POINT`; las longitudes se interpretan en SRID 4326.

  `geoToQuadbinCell(geometry,integer) → quadbin`

```
SELECT geoToQuadbinCell(geometry 'SRID=4326;POINT(4.35 50.85)', 10)
  = quadbin '48a6227affffffff';
-- true
```

- Devuelve el centroide de una celda como un punto SRID 4326. La forma estática toma un único `quadbin`; la forma temporal devuelve el centroide de cada celda de una trayectoria como un `tgeompoint`.

  `quadbinCellToPoint(quadbin) → geometry`

  `tquadbinCellToPoint(tquadbin) → tgeompoint`

```
SELECT ST_AsText(quadbinCellToPoint(quadbin '48427fffffffffff'));
-- POINT(-101.25 48.92249926375824)
SELECT asText(tquadbinCellToPoint(tquadbin
  '{480fffffffffffff@2001-01-01, 48427fffffffffff@2001-01-02}'));
-- {POINT(0 0)@2001-01-01,
--  POINT(-101.25 48.92249926375824)@2001-01-02}
```

- Devuelve el contorno poligonal cuadrado de una celda como una geometría SRID 4326. La forma estática toma un único `quadbin`; la forma temporal devuelve el contorno de cada celda de una trayectoria como un `tgeometry`.

  `quadbinCellToBoundary(quadbin) → geometry`

  `tquadbinCellToBoundary(tquadbin) → tgeometry`

```
SELECT ST_AsText(quadbinCellToBoundary(quadbin '480fffffffffffff'));
-- POLYGON((-180 -85.0511287798066,180 -85.0511287798066,
--          180 85.0511287798066,-180 85.0511287798066,
--          -180 -85.0511287798066))
```

17.8. Teselas y Quadkeys

QUADBIN está construido sobre el esquema de teselas slippy-map (web-mercator), por lo que una celda se corresponde directamente con un triple de coordenadas de tesela `(x, y, z)` y con la cadena quadkey en base 4 usada por los servidores de teselas. Estas conversiones son el puente específico de QUADBIN con el ecosistema de teselado.

- Construye una celda a partir de coordenadas de tesela slippy-map: columna `x`, fila `y`, y zoom `z`

  `quadbinTileToCell(integer,integer,integer) → quadbin`

```
SELECT quadbinTileToCell(0, 0, 0) = quadbin '480fffffffffffff';
-- true
SELECT quadbinTileToCell(3, 5, 4) = quadbin '48427fffffffffff';
-- true
SELECT quadbinTileToCell(524, 343, 10) = quadbin '48a6227affffffff';
-- true
```

- Descompone una celda en su triple de coordenadas de tesela, devuelto como un arreglo de enteros `{x, y, z}` — la inversa de `quadbinTileToCell`

  `quadbinCellToTile(quadbin) → integer[]`

```
SELECT quadbinCellToTile(quadbin '480fffffffffffff');
-- {0,0,0}
SELECT quadbinCellToTile(quadbin '48a6227affffffff');
-- {524,343,10}
```

- Emite la cadena quadkey en base 4 de una celda (un dígito por nivel de zoom, primero el más significativo; la celda mundial de resolución 0 se corresponde con la cadena vacía). La forma estática toma un único `quadbin`; la forma temporal devuelve el quadkey de cada celda de una trayectoria como un `ttext`.

  `quadbinCellToQuadkey(quadbin) → text`

  `tquadbinCellToQuadkey(tquadbin) → ttext`

```
SELECT quadbinCellToQuadkey(quadbin '480fffffffffffff');
-- (cadena vacía)
SELECT quadbinCellToQuadkey(quadbin '48427fffffffffff');
-- 0213
SELECT quadbinCellToQuadkey(quadbin '48a6227affffffff');
-- 1202021322
```

17.9. Métricas

- Devuelve el área de una celda sobre la esfera, en metros cuadrados. La forma estática toma un único `quadbin`; la forma temporal devuelve el área de cada celda de una trayectoria como un `tfloat`. El área de la celda se reduce hacia los polos, ya que los cuadrados web-mercator cubren progresivamente menos terreno a latitudes más altas.

  `quadbinCellArea(quadbin) → float8`

  `tquadbinCellArea(tquadbin) → tfloat`

```
SELECT round(quadbinCellArea(quadbin '480fffffffffffff')::numeric, 0);
-- 508164135963886
SELECT round(quadbinCellArea(quadbin '48427fffffffffff')::numeric, 0);
-- 2726563416104
```

17.10. Comparaciones

Los operadores de comparación de árbol B (=, <>, <, <=, >, >=) y el operador hash están definidos sobre `tquadbin` a través de la maquinaria genérica de comparación temporal, ordenando por la secuencia de valores temporal subyacente. Soportan las clases de operadores `tquadbin_btree_ops` y `tquadbin_hash_ops` usadas por GROUP BY, DISTINCT, ORDER BY, y las uniones por mezcla / hash. Como con `tbigint`, el orden int64 de los identificadores QUADBIN no tiene significado geográfico — véase Sección 17.12.

17.11. Indexación

Se definen clases de operadores de árbol B y hash para que los operadores de comparación de la sección anterior puedan utilizarse con los índices correspondientes.

- Clase de operadores de árbol B por defecto para las celdas QUADBIN temporales, ordenando por el valor temporal subyacente.

```
CREATE INDEX ON trips USING btree(cells);        -- usa tquadbin_btree_ops por defecto
```

- Clase de operadores hash por defecto para las celdas QUADBIN temporales, usada por las uniones hash y la agregación hash.

```
CREATE INDEX ON trips USING hash(cells);         -- usa tquadbin_hash_ops por defecto
```

17.12. Notas operativas

Esta sección recoge el conocimiento operativo que los consumidores necesitan además de la referencia de funciones: cuándo `tquadbin` es el tipo adecuado, las convenciones que asume la implementación, y los peligros específicos de QUADBIN que las cargas de trabajo reales deben tener en cuenta. La cuadrícula cuadrada jerárquica QUADBIN tiene algunas propiedades no obvias que la codificación bit a bit en `int64` oculta; esta sección las expone para que no sorprendan a los consumidores.

17.12.1. Cuándo usar tquadbin

Use `tquadbin` cuando el identificador natural de una ubicación discreta a una resolución dada sea una tesela QUADBIN — viajes de vehículos agrupados a un nivel de zoom fijo, lecturas de sensores agrupadas por tesela, o eventos asociados a su celda contenedora, especialmente en flujos de trabajo que ya hablan el esquema de teselado slippy-map / web-mercator. El tipo almacena una celda QUADBIN por instante; la resolución puede variar entre instantes de una misma trayectoria pero la mayoría de las cargas de trabajo fijan una resolución por columna.

Elija `tgeompoint` / `tgeogpoint` en lugar de `tquadbin` cuando la posición geográfica real (precisión sub-celda) sea el valor de interés. Elija `tbigint` en lugar de `tquadbin` cuando los valores resulten ser enteros de 64 bits pero no lleven semántica QUADBIN — las representaciones binarias son idénticas (se convierten sin pérdidas vía `tquadbin :: tbigint` y de vuelta) pero el sistema de tipos SQL usa la distinción para rechazar funciones agnósticas a QUADBIN sobre trayectorias específicas de QUADBIN.

17.12.2. Peligros específicos de QUADBIN

Algunos peligros muerden de forma fiable a las cargas de trabajo que usan celdas QUADBIN. Cada uno es una propiedad de la propia cuadrícula QUADBIN y no una decisión de implementación de MobilityDB; las mitigaciones siguientes explican cómo trabajar con ellos.

- *El orden int64 es arbitrario respecto a la geometría de la cuadrícula*. Los identificadores de celda QUADBIN son enteros de 64 bits cuyo patrón de bits codifica una etiqueta de cabecera, la resolución y las coordenadas de tesela. El orden bit a bit de dos celdas no tiene significado de proximidad espacial — dos celdas adyacentes en bits pueden estar lejos, y dos celdas espacialmente adyacentes pueden tener patrones de bits diferentes.

  Mitigación: `tquadbin` deliberadamente no tiene tipos complementarios `quadbinspan` / `quadbinspanset` (precedente: `geometry` no tiene `geometryspan`). El filtrado por rango de valores debe pasar por las funciones de inspección `quadbin_*` (resolución, validez, jerarquía) o por la enumeración explícita de conjuntos vía `quadbinset`. El código que asume que `cell_a < cell_b` refleja proximidad espacial producirá resultados erróneos silenciosamente.

- *Mezcla de resoluciones en las operaciones*. Las celdas QUADBIN a distintas resoluciones (0–26) representan áreas de cobertura diferentes. Mezclar resoluciones en una misma trayectoria es válido pero semánticamente requiere una justificación explícita — `quadbinCellToParent(cell, res_mas_gruesa)` engruesa y `quadbinCellToChildren(parent, res_mas_fina)` refina.

  Mitigación: los consumidores deben documentar la invariante de resolución por trayectoria (por ejemplo, "todas las celdas son de resolución 10") y validar las entradas en el límite de ingesta. El accesor `quadbinGetResolution` permite que una restricción CHECK lo imponga.

- *Límite de latitud de web-mercator*. QUADBIN hereda la proyección web-mercator, que está definida únicamente entre aproximadamente ±85,05° de latitud. Los puntos fuera de esta banda no tienen celda QUADBIN, y el área de la celda se reduce hacia los polos — `quadbinCellArea` devuelve valores progresivamente menores a latitudes más altas para celdas de la misma resolución.

  Mitigación: las cargas de trabajo cercanas a los polos deben recortar o rechazar las latitudes fuera de la banda web-mercator en el límite de ingesta; no confíe en las celdas QUADBIN para la cobertura polar.

- *Comportamiento en el antimeridiano*. Las columnas de tesela se envuelven en el antimeridiano (±180° de longitud). Una trayectoria que cruza el antimeridiano se mueve entre celdas cuyas coordenadas de tesela `x` difieren en el ancho completo de la cuadrícula a ese zoom, aunque sean espacialmente adyacentes.

  Mitigación: las cargas de trabajo que cruzan el antimeridiano deben validar celdas representativas contra la implementación de referencia de QUADBIN y añadir fixtures para los casos límite específicos que la carga de trabajo encuentre.

17.12.3. Durabilidad y almacenamiento

La representación en disco de `tquadbin` es idéntica byte a byte a la de `tbigint` (cada instante lleva un ID de celda QUADBIN de 64 bits más una marca de tiempo). Consecuencias para la planificación del almacenamiento:

- *WKT* vía `tquadbin_in` y la función de salida temporal. Las celdas se representan como cadenas hexadecimales canónicas (por ejemplo, `'480fffffffffffff'`); se aceptan tanto decimal como hexadecimal en la entrada. El ciclo de ida y vuelta es estable a nivel de bits.
- *WKB / EWKB / HexWKB* vía el patrón estándar de PostGIS de bandera de endianness seguida de la carga útil. Estable entre versiones mayores de PostgreSQL.
- *pg_dump* usa WKT en modo plano y WKB bajo `--binary-upgrade`; ambos completan el ciclo de ida y vuelta limpiamente.
- *Conversión cruzada con `tbigint`*: las conversiones bidireccionales `tquadbin :: tbigint` son reinterpretaciones bit a bit sin coste. Úselas cuando trabaje con vistas seguras por tipo (`tquadbin`) y aptas para aritmética (`tbigint`) de la misma trayectoria en una sola consulta.

Capítulo 18

Nubes de Puntos Temporales

La extensión pgPointCloud almacena datos de nubes de puntos LIDAR en PostgreSQL mediante dos tipos estáticos: `pcpoint`, un único punto multidimensional cuyas dimensiones se describen mediante un *esquema* almacenado en la tabla de catálogo `pointcloud_formats`; y `pcpatch`, un lote comprimido de valores `pcpoint` que comparten el mismo `pcid` (identificador de esquema). Cada esquema fija las dimensiones presentes (`X`, `Y`, `Z`, `Intensity`, ...), su tipo numérico, escala y desplazamiento.

MobilityDB eleva estos tipos al mundo temporal mediante `tpcpoint` (un sensor LIDAR/GPS en movimiento) y `tpcpatch` (una serie temporal de grupos de puntos comprimidos). También añade los tipos de conjunto `pcpointset` / `pcpatchset` y el tipo de caja delimitadora espaciotemporal `tpcbox`. Una referencia completa de los tipos estáticos `pcpoint` y `pcpatch` se ofrece en la documentación de pgPointCloud; este capítulo cubre las adiciones de MobilityDB sobre pgPointCloud.

Para usar los tipos descritos aquí, MobilityDB debe compilarse con la opción de CMake `POINTCLOUD=ON`. En tal compilación, el archivo `mobilitydb.control` generado declara `requires = 'postgis, pointcloud'`, de modo que un único `CASCADE` crea la pila completa:

```
CREATE EXTENSION mobilitydb CASCADE;
-- NOTICE:  installing required extension "postgis"
-- NOTICE:  installing required extension "pointcloud"
```

18.1. Inicio rápido

Esta sección recorre un ejemplo mínimo de extremo a extremo: registrar un esquema, construir valores, elevar al tipo temporal, consultar con la superficie bbox e indexar. El objetivo es que cada sección posterior de este capítulo esté anclada a un uso concreto. Siga los pasos en orden sobre una base de datos nueva.

18.1.1. Registrar un esquema

Cada valor `pcpoint` y `pcpatch` lleva un `pcid` que se resuelve a través de `pointcloud_formats` en un esquema XML. El esquema declara las dimensiones (X, Y, Z, ...), su tipo numérico en disco y la escala por dimensión. Un esquema 3D mínimo con todas las dimensiones almacenadas como `int32` con escala unitaria:

```
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 4326, '
<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1">
  <pc:dimension><pc:position>1</pc:position><pc:size>4</pc:size>
    <pc:name>X</pc:name><pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>1</pc:scale></pc:dimension>
  <pc:dimension><pc:position>2</pc:position><pc:size>4</pc:size>
    <pc:name>Y</pc:name><pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>1</pc:scale></pc:dimension>
```

```
  <pc:dimension><pc:position>3</pc:position><pc:size>4</pc:size>
    <pc:name>Z</pc:name><pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>1</pc:scale></pc:dimension>
</pc:PointCloudSchema>');
```

Los esquemas son globales a la base de datos; normalmente se registran una sola vez durante la carga inicial de datos. El SRID es por esquema y es heredado por cada valor con ese `pcid`.

18.1.2. Construir valores base y temporales

MobilityDB incluye constructores ergonómicos sobre los `PC_MakePoint` / `PC_Patch` de pgPointCloud:

```
-- a single point at (1, 2, 3) under pcid=1
SELECT pcpoint(1, 1.0, 2.0, 3.0);

-- a patch holding two points
SELECT pcpatch(1, pcpoint(1, 1.0, 2.0, 3.0),
                  pcpoint(1, 4.0, 5.0, 6.0));

-- a single-instant tpcpoint at (10, 20, 30) at 2024-01-01
SELECT tpcpoint(pcpoint(1, 10, 20, 30), '2024-01-01'::timestamptz);

-- a 3-instant moving sensor track (step interpolation, the default for
-- pointcloud temporals because dimensions like Intensity don't interpolate
-- linearly the way coordinates do)
SELECT tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 0, 0, 0), '2024-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 1, 1, 1), '2024-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2024-01-03'::timestamptz)]);
```

Para sesiones largas, declare la columna con un *typmod* para que el pcid de cada fila se imponga en el momento del INSERT:

```
CREATE TABLE scans (id int, traj tpcpoint(1), full_scan tpcpatch(1));
INSERT INTO scans VALUES (1,
  tpcpointSeq(ARRAY[
    tpcpoint(pcpoint(1, 0, 0, 0), '2024-01-01'::timestamptz),
    tpcpoint(pcpoint(1, 1, 1, 1), '2024-01-02'::timestamptz)]),
  tpcpatch(pcpatch(1, pcpoint(1,1,1,1), pcpoint(1,2,2,2)),
           '2024-01-01'::timestamptz));
```

Insertar un valor cuyo pcid no coincide con el typmod de la columna genera `ERROR: Pcid of tpcpoint value (N) does not match column typmod pcid (M)`; mezclar pcids en una columna sin restricciones también se rechaza en el momento de la agregación con `extent`.

18.1.3. Consultar con la superficie bbox

Cada tpcpoint y tpcpatch lleva una `tpcbox` (espejo de `stbox` + `pcid`) calculada en el momento de la construcción. La superficie de operadores a nivel bbox — topológica (`&&`, `@>`, `<@`, `~=`, `-|-`), direccional en X/Y/Z/tiempo y distancia KNN `|=|` — funciona entre cualquier par de (`tpcbox`, `tpcpoint`, `tpcpatch`):

```
-- rows whose track ever entered a 3D-temporal box
SELECT id FROM scans
WHERE traj && tpcbox_zt(0, 0, 0, 1, 1, 1,
  tstzspan '[2024-01-01, 2024-01-02]', 1, 4326);

-- 5 nearest tracks to a query box (KNN, GiST-orderable)
SELECT id, traj |=| my_box AS dist FROM scans
ORDER BY traj |=| my_box LIMIT 5;
```

El filtrado a nivel de parche (descartar instantes completos cuyo parche no se superpone con una caja) se proporciona mediante `atTpcbox` / `minusTpcbox`. Véase Operaciones por punto para el ámbito de granularidad.

18.1.4. Índice para búsquedas impulsadas por bbox

Las clases de operadores R-tree GiST y quadtree / kd-tree SP-GiST aceleran todos los predicados bbox. Elija GiST para cargas de trabajo de propósito general (y para ordenación KNN); SP-GiST kd-tree es competitivo en datos de solo puntos con muchas lecturas:

```
CREATE INDEX scans_traj_gist ON scans USING gist(traj);
CREATE INDEX scans_full_spgist ON scans USING spgist(full_scan);
```

Las clases de operadores SP-GiST almacenan un `stbox` internamente y recuperan el filtro `pcid` en el momento de la reverificación, por lo que una columna con pcids mixtos devuelve un conjunto de candidatos ligeramente mayor para reverificar — no supone ningún problema cuando cada tabla contiene valores de un único esquema, que es lo que impone la receta typmod en Construir valores base y temporales.

18.1.5. Agregación

```
-- spatiotemporal extent of every track (one tpcbox)
SELECT extent(traj) FROM scans;

-- count of active tracks at each timestamp (tint)
SELECT tcount(traj) FROM scans;

-- merge per-source instants into a single tpcpoint
SELECT id, merge(traj) FROM scans GROUP BY id;
```

`extent` rechaza pcids mixtos (las dimensiones bbox serían ininterpretables entre esquemas); `merge` y `tcount` siguen las mismas reglas que sus equivalentes sin nube de puntos.

18.1.6. Acceso por punto

La función de retorno de conjuntos points expande el parche de cada instante en pcpoints individuales, lo que resulta útil para uniones con predicados por punto que la capa bbox no puede expresar:

```
-- explode every patch in the column into one row per (instant timestamp, point)
SELECT id, t, point FROM scans, points(full_scan);

-- total points across every instant of every track
SELECT id, numPoints(full_scan) FROM scans;
```

El *filtrado* por punto — construir un nuevo parche a partir de un subconjunto de los puntos del original — se proporciona mediante atTpcboxFine / `minusTpcboxFine` (contra una `tpcbox`) y atGeometry / `minusGeometry` (contra una geometría 2D); véase Operaciones por punto para la matriz de granularidad.

18.1.7. ¿Qué sigue?

El resto del capítulo profundiza en cada una de las superficies abordadas aquí: funciones de acceso de pcpoint / pcpatch, pcpointset / pcpatchset, la caja delimitadora tpcbox, tpcpoint, tpcpatch, índices y agregaciones.

18.2. Tipos estáticos `pcpoint` y `pcpatch`

Los tipos estáticos provienen de pgPointCloud. MobilityDB expone las funciones de acceso de coordenadas con conocimiento del esquema necesarias para proyectar los valores de pgPointCloud en el resto del ecosistema de tipos temporales.

18.2.1. Constructores ergonómicos

La función `pcpoint_in` de pgPointCloud solo acepta hex-WKB, por lo que el constructor canónico es `PC_MakePoint(pcid, ARRAY[...]::float[])`. MobilityDB incluye dos sobrecargas SQL ligeras sobre él para literales de prueba más cortos y consultas ad hoc; la validación subyacente de dimensiones del esquema sigue ocurriendo dentro de `PC_MakePoint`.

- Construye un pcpoint 2D o 3D directamente a partir de coordenadas x/y(/z)

  `pcpoint(pcid integer, x float, y float) → pcpoint`

  `pcpoint(pcid integer, x float, y float, z float) → pcpoint`

```
SELECT pcpoint(1, 10.0, 20.0, 30.0);
-- equivalent to PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]::float[])
```

- Construye un pcpatch a partir de una lista variádica de pcpoints que comparten el mismo pcid

  `pcpatch(pcid integer, VARIADIC points pcpoint[]) → pcpatch`

```
SELECT pcpatch(1, pcpoint(1, 1.0, 1.0, 1.0), pcpoint(1, 2.0, 2.0, 2.0));
-- equivalent to PC_Patch(ARRAY[PC_MakePoint(1, ARRAY[1.0, 1.0, 1.0]::float[]),
--                              PC_MakePoint(1, ARRAY[2.0, 2.0, 2.0]::float[])])
```

18.2.2. Funciones de acceso

- Devuelve el identificador de esquema de un pcpoint o pcpatch

  `pcid(pcpoint) → integer`

  `pcid(pcpatch) → integer`

```
SELECT pcid(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]));
-- 1
```

- Devuelve una coordenada de un pcpoint, NULL si la dimensión está ausente del esquema

  `getX(pcpoint) → float`

  `getY(pcpoint) → float`

  `getZ(pcpoint) → float`

```
WITH pt AS (SELECT PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]) p)
  SELECT getX(p), getY(p), getZ(p) FROM pt;
-- 10 | 20 | 30
```

- Devuelve cualquier dimensión con nombre de un pcpoint, NULL si el nombre es desconocido para el esquema

  `getDim(pcpoint,text) → float`

```
SELECT getDim(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]), 'X');
-- 10
```

- Devuelve el identificador de referencia espacial (heredado del esquema)

  `SRID(pcpoint) → integer`

  `SRID(pcpatch) → integer`

```
SELECT SRID(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]));
-- 0
```

18.3. Tipos de conjunto `pcpointset` y `pcpatchset`

Un `pcpointset` es un conjunto ordenado de valores `pcpoint` distintos; un `pcpatchset` es el tipo de conjunto análogo para `pcpatch`. Ambos siguen la semántica general de conjuntos de MobilityDB (véase Capítulo 2): los valores se deduplicen en la construcción, la representación binaria es canónica, y todos los operadores genéricos a nivel de conjunto (=, <>, <, <=, >, >=, @>, <@, &&, +, −, ⋆) y las funciones de acceso (`numValues`, `memSize`, `asText`, `asBinary`, `asHexWKB`, ...) están disponibles.

Todos los valores de un conjunto deben compartir el mismo `pcid`. Mezclar esquemas en un único conjunto genera un error en el momento de la construcción.

18.3.1. Constructores

- Construye un conjunto a partir de un array de valores, todos los cuales deben compartir el mismo pcid

  `set(pcpoint[])` → `pcpointset`

  `set(pcpatch[])` → `pcpatchset`

```
SELECT numValues(set(ARRAY[PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]),
  PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]),  -- duplicate, deduped
  PC_MakePoint(1, ARRAY[11.0, 21.0, 31.0]) ]));
-- 2
```

18.4. Caja Delimitadora `tpcbox`

El tipo `tpcbox` es la caja delimitadora espaciotemporal para los tipos de nube de puntos temporal. Refleja la estructura de `stbox` (véase Capítulo 3) — mismos ejes X/Y/Z/T, mismos bits de indicador para las dimensiones presentes — y añade un único campo: el `pcid`. Dos valores `tpcbox` con distintos valores de `pcid` nunca se consideran solapados, contenidos, iguales o adyacentes entre sí, independientemente de su extensión geométrica o temporal.

Propagar el `pcid` a través de la aritmética de cajas delimitadoras garantiza que la poda por caja delimitadora nunca descarte una fila cuyo esquema difiera del esquema de la consulta, y nunca devuelva un resultado entre esquemas incompatibles.

18.4.1. Constructores

- Construir un tpcbox a partir de límites X/Y/Z/T explícitos y un pcid

  `tpcbox(xmin,ymin,xmax,ymax,pcid,srid=0)` → `tpcbox`

  `tpcbox_z(xmin,ymin,zmin,xmax,ymax,zmax,pcid,srid=0)` → `tpcbox`

  `tpcbox_t(period,pcid)` → `tpcbox`

  `tpcbox_xt(xmin,ymin,xmax,ymax,period,pcid,srid=0)` → `tpcbox`

  `tpcbox_zt(xmin,ymin,zmin,xmax,ymax,zmax,period,pcid,srid=0)` → `tpcbox`

```
SELECT tpcbox(0, 0, 10, 10, 1, 0);
-- TPCBOX(X((0,0),(10,10)), 1)
SELECT tpcbox_zt(0, 0, 0, 10, 10, 10, tstzspan '[2024-01-01, 2024-01-02]', 1, 0);
-- TPCBOX(ZT(((0,0,0),(10,10,10)),[2024-01-01, 2024-01-02]), 1)
```

18.4.2. Conversiones

- Convertir un pcpoint a un tpcbox degenerado (de extensión cero); el SRID y el pcid provienen del esquema

  `tpcbox(pcpoint)` → `tpcbox`

```
SELECT tpcbox(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]));
-- TPCBOX(Z((10,20,30),(10,20,30)), 1)
```

- Convertir un pcpatch a un tpcbox usando los límites PCBOUNDS 2D integrados en el parche; la segunda forma acepta un SRID explícito en lugar de buscarlo en el esquema

  `tpcbox(pcpatch) → tpcbox`

  `tpcbox(pcpatch,integer) → tpcbox`

```
SELECT tpcbox(PC_Patch(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0])));
```

18.4.3. Accesores

- Devuelve si un tpcbox tiene establecidas las dimensiones X/Y, Z o T

  `hasX(tpcbox) → boolean`

  `hasZ(tpcbox) → boolean`

  `hasT(tpcbox) → boolean`

```
WITH b AS (SELECT tpcbox(0, 0, 10, 10, 1, 0) AS b)
  SELECT hasX(b), hasZ(b) FROM b;
-- t | f
```

- Devuelve un límite de coordenada, NULL si la dimensión correspondiente está ausente

  `xmin(tpcbox) → float`

  `xmax(tpcbox) → float`

  `ymin(tpcbox) → float`

  `ymax(tpcbox) → float`

  `zmin(tpcbox) → float`

  `zmax(tpcbox) → float`

```
WITH b AS (SELECT tpcbox(0, 0, 10, 10, 1, 0) AS b)
  SELECT xmin(b), xmax(b) FROM b;
-- 0 | 10
```

- Devuelve un límite temporal, NULL si la caja no tiene dimensión T

  `tmin(tpcbox) → timestamptz`

  `tmax(tpcbox) → timestamptz`

```
SELECT tmin(tpcbox_t(tstzspan '[2024-01-01, 2024-01-02]', 1));
-- 2024-01-01 00:00:00+00
```

- Devuelve el identificador de esquema y el SRID

  `pcid(tpcbox) → integer`

  `SRID(tpcbox) → integer`

```
WITH b AS (SELECT tpcbox(0, 0, 10, 10, 7, 4326) AS b)
  SELECT pcid(b), SRID(b) FROM b;
-- 7 | 4326
```

18.4.4. Transformaciones

- Devuelve un tpcbox con las coordenadas redondeadas al número de dígitos decimales indicado

  `round(tpcbox,integer=0) → tpcbox`

```
SELECT round(tpcbox(0.123456, 1.234567, 2.345678, 3.456789, 1, 0), 2);
-- TPCBOX(X((0.12,1.23),(2.35,3.46)), 1)
```

- Devuelve un tpcbox con el SRID sobreescrito. No reproyecta las coordenadas — para ello, proyecte primero el tpcpoint subyacente y tome la caja delimitadora del resultado

  `setSRID(tpcbox,integer) → tpcbox`

```
SELECT SRID(setSRID(tpcbox(0, 0, 10, 10, 1, 0), 4326));
-- 4326
```

18.4.5. Operaciones de conjuntos

- Devuelve la unión de dos tpcboxes; ambas deben compartir el mismo pcid

  `tpcbox_union(tpcbox,tpcbox) → tpcbox`

  `tpcbox + tpcbox → tpcbox`

```
SELECT tpcbox(0, 0, 5, 5, 1, 0) + tpcbox(3, 3, 10, 10, 1, 0);
-- TPCBOX(X((0,0),(10,10)), 1)
```

- Devuelve la intersección de dos tpcboxes, NULL si son disjuntas o el pcid no coincide

  `tpcbox_intersection(tpcbox,tpcbox) → tpcbox`

  `tpcbox * tpcbox → tpcbox`

```
SELECT tpcbox(0, 0, 5, 5, 1, 0) * tpcbox(3, 3, 10, 10, 1, 0);
-- TPCBOX(X((3,3),(5,5)), 1)
```

Nota: no se proporciona un `minus` genérico entre cajas, siguiendo la convención de `stbox` y `tbox`: la diferencia de dos cajas no es en general una caja.

18.4.6. Operadores Topológicos

Todos los operadores topológicos toman dos valores `tpcbox` con `pcid` coincidente. Una discrepancia de pcid siempre devuelve false (no un error) para que los escaneos de índice se comporten de forma intuitiva.

- Devuelve true si el primer tpcbox contiene al segundo

  `tpcbox @> tpcbox → boolean`

```
SELECT tpcbox(0, 0, 10, 10, 1, 0) @> tpcbox(2, 2, 8, 8, 1, 0);
-- t
```

- Devuelve true si el primer tpcbox está contenido en el segundo

  `tpcbox <@ tpcbox → boolean`

```
SELECT tpcbox(2, 2, 8, 8, 1, 0) <@ tpcbox(0, 0, 10, 10, 1, 0);
-- t
```

- Devuelve true si dos tpcboxes se solapan

  `tpcbox && tpcbox → boolean`

```
SELECT tpcbox(0, 0, 5, 5, 1, 0) && tpcbox(3, 3, 10, 10, 1, 0);  -- t
SELECT tpcbox(0, 0, 5, 5, 1, 0) && tpcbox(0, 0, 5, 5, 2, 0);  -- f (pcid mismatch)
```

- Devuelve true si dos tpcboxes son exactamente iguales

  `tpcbox ~= tpcbox → boolean`

```
SELECT tpcbox(0, 0, 2, 2) ~= tpcbox(0, 0, 2, 2);
-- t
```

- Devuelve true si dos tpcboxes son adyacentes (comparten un límite)

  `tpcbox -|- tpcbox → boolean`

```
SELECT tpcbox(0, 0, 2, 2) -|- tpcbox(5, 5, 7, 7);
-- f
```

18.4.7. Operadores de Posición

Dieciséis predicados de posición alineados por eje, paralelos a los de `stbox`. Cada uno comprueba la relación estricta o de solapamiento a lo largo de un eje. Todos requieren `pcid` coincidente; la discrepancia devuelve false. Los predicados del eje Z requieren que ambas cajas tengan dimensión Z; los del eje T requieren que ambas tengan dimensión T.

- Comprueba el orden en el eje X de dos tpcboxes

  `tpcbox << tpcbox → boolean` (estrictamente a la izquierda)

  `tpcbox &< tpcbox → boolean` (no se extiende a la derecha)

  `tpcbox >> tpcbox → boolean` (estrictamente a la derecha)

  `tpcbox &> tpcbox → boolean` (no se extiende a la izquierda)

```
SELECT tpcbox(0, 0, 2, 2) << tpcbox(5, 5, 7, 7);
-- t
```

- Comprueba el orden en el eje Y de dos tpcboxes

  `tpcbox <<| tpcbox → boolean` (estrictamente por debajo)

  `tpcbox &<| tpcbox → boolean` (no se extiende hacia arriba)

  `tpcbox |>> tpcbox → boolean` (estrictamente por encima)

  `tpcbox |&> tpcbox → boolean` (no se extiende hacia abajo)

```
SELECT tpcbox(0, 0, 2, 2) <<| tpcbox(5, 5, 7, 7);
-- t
```

- Comprueba el orden en el eje Z de dos tpcboxes (devuelve false si alguna carece de Z)

  `tpcbox <</ tpcbox → boolean` (estrictamente al frente)

  `tpcbox &</ tpcbox → boolean` (no se extiende hacia atrás)

  `tpcbox />> tpcbox → boolean` (estrictamente detrás)

  `tpcbox /&> tpcbox → boolean` (no se extiende hacia el frente)

```
-- Neither box carries a z dimension, so the front/back test is false.
SELECT tpcbox(0, 0, 2, 2) <</ tpcbox(5, 5, 7, 7);
-- f
```

- Comprueba el orden en el eje temporal de dos tpcboxes (devuelve false si alguna carece de T)

  `tpcbox <<# tpcbox → boolean` (estrictamente antes)

  `tpcbox &<# tpcbox → boolean` (no se extiende después)

  `tpcbox #>> tpcbox → boolean` (estrictamente después)

  `tpcbox #&> tpcbox → boolean` (no se extiende antes)

```
-- Neither box carries a time span, so the before/after test is false.
SELECT tpcbox(0, 0, 2, 2) <<# tpcbox(5, 5, 7, 7);
-- f
```

18.4.8. Comparaciones

- Compara dos tpcboxes; orden total sobre la codificación canónica (pcid, srid, flags, período y a continuación límites espaciales en orden XYZ-min/max)

  `tpcbox = tpcbox → boolean`

  `tpcbox <> tpcbox → boolean`

  `tpcbox < tpcbox → boolean`

  `tpcbox <= tpcbox → boolean`

  `tpcbox > tpcbox → boolean`

  `tpcbox >= tpcbox → boolean`

```
SELECT tpcbox(0, 0, 2, 2) = tpcbox(0, 0, 2, 2);
-- t
SELECT tpcbox(0, 0, 2, 2) <> tpcbox(5, 5, 7, 7);
-- t
SELECT tpcbox(0, 0, 2, 2) < tpcbox(5, 5, 7, 7);
-- t
```

18.5. Tipo Temporal `tpcpoint`

Un `tpcpoint` es el levantamiento de `pcpoint` a través de la maquinaria temporal de MobilityDB. Admite los mismos tres subtipos que los demás tipos temporales — instante, secuencia (con interpolación discreta o de pasos) y conjunto de secuencias. Todos los instantes de un único valor `tpcpoint` deben compartir el mismo `pcid`; el constructor impone esta restricción.

La caja delimitadora de un `tpcpoint` es un `tpcbox`, calculada en el momento de la construcción a partir de las dimensiones X/Y/Z de los valores `pcpoint` subyacentes junto con las marcas temporales de los instantes.

Una columna o dominio puede fijar el esquema añadiendo el `pcid` como un typmod de entero único, exactamente igual que el `geometry(Point, SRID)` de PostGIS. La mezcla de esquemas en dicha columna es rechazada en el momento de INSERT o de la conversión. El mismo constructor se aplica a `tpcpatch`.

```
CREATE TABLE scans (id int, traj tpcpoint(1), full_scan tpcpatch(1));
-- accepts pcid 1
INSERT INTO scans VALUES (1,
  tpcpoint(pcpoint(1, 1.0, 2.0, 3.0), '2024-01-01'::timestamptz),
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)),
           '2024-01-01'::timestamptz));
-- rejects any other pcid:
-- ERROR: Pcid of tpcpoint value (2) does not match column typmod pcid (1)
```

Omitir el typmod (`tpcpoint` sin pcid entre paréntesis) deja la columna sin restricción de esquema. La comprobación en tiempo de ejecución del agregado `extent` sigue rechazando entradas con pcid mixtos a mitad de consulta incluso en columnas sin restricciones.

18.5.1. Constructores

- Construir un pcpoint temporal de subtipo instante

  `tpcpoint(pcpoint,timestamptz) → tpcpoint`

```
SELECT tpcpoint(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]), '2024-01-01 12:00+00');
```

- Construir un pcpoint temporal de subtipo secuencia a partir de un array de instantes. La interpolación es una de `'discrete'`, `'step'`; `'step'` es el valor predeterminado para tpcpoint, ya que los pcpoints llevan dimensiones heterogéneas que no se interpolan de manera lineal

  `tpcpointSeq(tpcpoint[]) → tpcpoint`

  `tpcpointSeq(tpcpoint[],text) → tpcpoint`

```
SELECT tpcpointSeq(ARRAY[tpcpoint(PC_MakePoint(1, ARRAY[1.0, 2.0, 3.0]), '2024-01-01':: ↩
    timestamptz),
  tpcpoint(PC_MakePoint(1, ARRAY[4.0, 5.0, 6.0]), '2024-01-02'::timestamptz)], 'step');
```

- Construir un pcpoint temporal de subtipo conjunto de secuencias a partir de un array de secuencias

  `tpcpointSeqSet(tpcpoint[]) → tpcpoint`

```
SELECT numSequences(tpcpointSeqSet(ARRAY[tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)])]));
-- 1
SELECT numInstants(tpcpointSeqSet(ARRAY[tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)])]));
-- 2
```

18.5.2. Funciones de Acceso

Se aplican todas las funciones de acceso temporales genéricas (véase Capítulo 4): `numInstants`, `startInstant`, `endInstant`, `instantN`, `instants`, `startValue`, `endValue`, `getValues`, `numTimestamps`, `startTimestamp`, `endTimestamp`, `getTime`, `duration`, `memSize`, `interp`. Adicionalmente:

- Devolver el id de esquema de pgPointCloud compartido por todos los instantes

  `pcid(tpcpoint) → integer`

```
SELECT pcid(tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz));
-- 1
```

- Devolver el SRID heredado del esquema

  `SRID(tpcpoint) → integer`

```
-- The SRID is that of the schema registered for the pcid.
SELECT SRID(tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz));
-- 0
```

18.5.3. Proyecciones por Dimensión

- Proyectar un tpcpoint sobre una dimensión espacial, produciendo un tfloat

  `getX(tpcpoint) → tfloat`

  `getY(tpcpoint) → tfloat`

  `getZ(tpcpoint) → tfloat`

```
SELECT startValue(getX(tpcpoint(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]),
  '2024-01-01'::timestamptz)));
-- 10
```

- Proyectar un tpcpoint sobre cualquier dimensión de esquema con nombre, produciendo un tfloat

  `getDim(tpcpoint,text) → tfloat`

```
SELECT getDim(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)]), 'X');
-- Interp=Step;[1@2001-01-01, 2@2001-01-02]
```

18.5.4. Conversiones

- Convertir un tpcpoint a un punto de geometría temporal, proyectando todas las dimensiones no espaciales mientras se preservan las marcas temporales. El resultado tiene el mismo SRID que el esquema; su dimensionalidad (2D frente a 3D) sigue la presencia de Z en el esquema. No se proporciona una conversión inversa: reconstruir un pcpoint requiere un esquema destino que el valor temporal solo no puede suministrar

  `tgeompoint::tpcpoint` no está definida

  `tpcpoint::tgeompoint`

```
SELECT ST_AsText(startValue(tpcpoint(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0]),
  '2024-01-01'::timestamptz)::tgeompoint));
-- POINT Z (10 20 30)
```

18.5.5. Transformaciones

- Transformar un tpcpoint en otro subtipo

  `tpcpointInst(tpcpoint) → tpcpoint`

  `tpcpointSeq(tpcpoint,interp='step') → tpcpoint`

  `tpcpointSeqSet(tpcpoint) → tpcpoint`

```
SELECT numInstants(tpcpointInst(tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz))) ↩
    ;
-- 1
SELECT numInstants(tpcpointSeq(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)])));
-- 2
```

- Transformar un tpcpoint a otra interpolación

  `setInterp(tpcpoint,interp) → tpcpoint`

```
-- A tpcpoint carries step interpolation.
SELECT interp(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)]));
-- Step
SELECT interp(setInterp(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)], 'discrete'), 'step'));
-- Step
```

18.5.6. Comparaciones

- Comparar dos tpcpoints lexicográficamente sobre sus codificaciones canónicas

  `tpcpoint = tpcpoint → boolean`

  `tpcpoint <> tpcpoint → boolean`

  `tpcpoint < tpcpoint → boolean`

  `tpcpoint <= tpcpoint → boolean`

  `tpcpoint > tpcpoint → boolean`

  `tpcpoint >= tpcpoint → boolean`

```
SELECT tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz) = tpcpoint(pcpoint(1, 1, ↩
    1, 1), '2001-01-01'::timestamptz);
-- t
SELECT tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz) <> tpcpoint(pcpoint(1, 2, ↩
    2, 2), '2001-01-02'::timestamptz);
-- t
SELECT tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz) < tpcpoint(pcpoint(1, 2, ↩
    2, 2), '2001-01-02'::timestamptz);
-- t
```

- Comprobar si el valor en algún instante (alguna vez) o en todos los instantes (siempre) es igual a, o distinto de, un pcpoint u otro tpcpoint

  `eEq(pcpoint,tpcpoint) → boolean`

  `eEq(tpcpoint,pcpoint) → boolean`

  `eEq(tpcpoint,tpcpoint) → boolean`

  `aEq`, `eNe` y `aNe` reciben los mismos tres pares de argumentos

  `pcpoint ?= tpcpoint → boolean`

  `tpcpoint ?= pcpoint → boolean`

  `tpcpoint ?= tpcpoint → boolean`

  Los operadores `%=`, `?<>` y `%<>` reciben los mismos tres pares de operandos

```
SELECT eEq(pcpoint(1, 1, 1, 1), tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]));
-- t
SELECT aEq(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), pcpoint(1, 1, 1, 1));
-- f
```

- Devolver la igualdad o desigualdad temporal de un tpcpoint con un pcpoint o con otro tpcpoint, como un tbool

  `tEq(pcpoint,tpcpoint) → tbool`

  `tEq(tpcpoint,pcpoint) → tbool`

  `tEq(tpcpoint,tpcpoint) → tbool`

  `tNe` recibe los mismos tres pares de argumentos

  `pcpoint #= tpcpoint → tbool`

  `tpcpoint #= pcpoint → tbool`

  `tpcpoint #= tpcpoint → tbool`

  El operador `#<>` recibe los mismos tres pares de operandos

```
SELECT tEq(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), pcpoint(1, 1, 1, 1));
-- [t@2001-01-01, f@2001-01-02, f@2001-01-03]
```

18.5.7. Restricciones

- Restringir un tpcpoint a (al complemento de) un valor pcpoint o un conjunto de valores pcpoint

  `atValue(tpcpoint,pcpoint) → tpcpoint`

  `minusValue(tpcpoint,pcpoint) → tpcpoint`

  `atValues(tpcpoint,pcpointset) → tpcpoint`

  `minusValues(tpcpoint,pcpointset) → tpcpoint`

```
-- With step interpolation the matching value is held on
-- [2001-01-01, 2001-01-02); its closing bound is stored as a second instant.
SELECT numInstants(atValue(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), pcpoint(1, 1, 1, 1)));
-- 2
SELECT numInstants(minusValue(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), pcpoint(1, 1, 1, 1)));
-- 2
```

- Restringir un tpcpoint a un selector temporal (marca de tiempo, periodo, conjunto o spanset), o eliminarlo

  `atTime(tpcpoint,timestamptz) → tpcpoint`

  `atTime(tpcpoint,tstzspan) → tpcpoint`

  `atTime(tpcpoint,tstzset) → tpcpoint`

  `atTime(tpcpoint,tstzspanset) → tpcpoint`

  `minusTime(tpcpoint,timestamptz) → tpcpoint`

  `minusTime(tpcpoint,tstzspan) → tpcpoint`

  `minusTime(tpcpoint,tstzset) → tpcpoint`

  `minusTime(tpcpoint,tstzspanset) → tpcpoint`

```
SELECT numInstants(atTime(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), timestamptz '2001-01-02'));
-- 1
SELECT numInstants(atTime(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), tstzspan '[2001-01-01, ↩
      2001-01-02]'));
-- 2
```

- Restringir un tpcpoint al extent espacial / temporal de un tpcbox, o eliminarlo. Devuelve NULL cuando el pcid del tpcbox no coincide con el del tpcpoint. Internamente proyecta a un tgeompoint a través de la caché de esquemas, restringe la proyección mediante el stbox equivalente y devuelve el span temporal del resultado al tpcpoint original para preservar los valores pcpoint completos

`atTpcbox(tpcpoint,tpcbox,border_inc bool=TRUE) → tpcpoint`

`minusTpcbox(tpcpoint,tpcbox,border_inc bool=TRUE) → tpcpoint`

```
SELECT numInstants(atTpcbox(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]),
  tpcbox_zt(0, 0, 0, 2, 2, 2, tstzspan '[2001-01-01, 2001-01-03]', 1)));
-- 2
-- The box does not intersect the tpcpoint, so nothing is removed.
SELECT numInstants(minusTpcbox(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]),
  tpcbox_zt(10, 10, 10, 12, 12, 12, tstzspan '[2001-01-01, 2001-01-03]', 1)));
-- 3
```

- Restringir un tpcpoint a los instantes anteriores o posteriores a una marca de tiempo. El indicador strict excluye el instante en la marca de tiempo misma

`beforeTimestamp(tpcpoint,timestamptz,strict bool=TRUE) → tpcpoint`

`afterTimestamp(tpcpoint,timestamptz,strict bool=TRUE) → tpcpoint`

```
SELECT timeSpan(beforeTimestamp(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), timestamptz '2001-01-02'));
-- [2001-01-01, 2001-01-02)
SELECT timeSpan(afterTimestamp(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), timestamptz '2001-01-02'));
-- (2001-01-02, 2001-01-03]
```

18.5.8. Modificaciones

- Insertar un tpcpoint en otro, o actualizar otro con él

`insert(tpcpoint,tpcpoint,connect bool=TRUE) → tpcpoint`

`update(tpcpoint,tpcpoint,connect bool=TRUE) → tpcpoint`

```
SELECT numInstants(insert(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)]), tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)])));
-- 3
```

- Eliminar un selector de tiempo (marca de tiempo, conjunto, período o conjunto de spans) de un tpcpoint

`deleteTime(tpcpoint,timestamptz,connect bool=TRUE) → tpcpoint`

`deleteTime(tpcpoint,tstzset,connect bool=TRUE) → tpcpoint`

`deleteTime(tpcpoint,tstzspan,connect bool=TRUE) → tpcpoint`

`deleteTime(tpcpoint,tstzspanset,connect bool=TRUE) → tpcpoint`

```
-- Deleting the middle instant splits the sequence into
-- {[2001-01-01, 2001-01-02), (2001-01-02, 2001-01-03]}; each half stores
-- its bound at the deleted timestamp, giving four instants.
SELECT numInstants(deleteTime(tpcpointSeq(ARRAY[
```

```
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), timestamptz '2001-01-02'));
-- 4
```

- Añadir un instante o una secuencia a un tpcpoint

  `appendInstant(tpcpoint,tpcpoint) → tpcpoint`

  `appendSequence(tpcpoint,tpcpoint) → tpcpoint`

```
SELECT numInstants(appendInstant(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz)]),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)));
-- 3
```

18.5.9. Fragmentación

- Devolver los valores distintos de un tpcpoint con el conjunto de spans durante el cual toma cada uno de ellos

  `unnest(tpcpoint) → {(value,time)}`

```
SELECT (un).time
FROM (SELECT unnest(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2024-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2024-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 1, 1, 1), '2024-01-03'::timestamptz)])) AS un) t;
-- {[2024-01-01, 2024-01-02), [2024-01-03, 2024-01-03]}
-- {[2024-01-02, 2024-01-03)}
```

- Fragmentar un tpcpoint en fragmentos, uno por cada intervalo de tiempo

  `timeSplit(tpcpoint,bin_width interval,origin timestamptz='2000-01-03') → {(time,tpcpoint`

```
SELECT (ts).time, numInstants((ts).temp)
FROM (SELECT timeSplit(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2024-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2024-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2024-01-03'::timestamptz)]),
  interval '1 day', '2024-01-01'::timestamptz) AS ts) t;
-- 2024-01-01 | 2
-- 2024-01-02 | 2
-- 2024-01-03 | 1
```

18.5.10. Operadores de caja delimitadora

Los operadores de caja delimitadora evalúan contra el `tpcbox` del valor; véase Sección 18.4.6 y Sección 18.4.7 para la geometría por operador. Cada operador a continuación está conectado contra `tpcbox`, `tstzspan` y el propio `tpcpoint`.

- Predicados topológicos: `&&` solapa, `@>` contiene, `<@` contenido en, `~=` mismo, `-|-` adyacente.
- Predicados direccionales estrictos sobre el eje X (`<<`, `>>`), eje Y (`<<|`, `|>>`), eje Z (`<<\/`, `\/>>`) y eje temporal (`<<#`, `#>>`), más sus variantes "solapa o X": `&<`, `&>`, `&<|`, `|&>`, `&<\/`, `\/&>`, `&<#`, `#&>`.
- Distancia de aproximación más cercana, ordenable KNN mediante la clase de operadores GiST: `SELECT ... ORDER BY temp |=| query LIMIT N` se acelera con índice. El desajuste de pcid produce infinito.

  `nearestApproachDistance(tpcpoint,tpcbox) → float`

  `nearestApproachDistance(tpcpoint,tpcpoint) → float`

```
SELECT round(nearestApproachDistance(
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 10, 10, 10), '2001-01-01'::timestamptz))::numeric, 4);
-- 15.5885
-- A tpcbox without a time span measures the spatial approach only.
SELECT round(nearestApproachDistance(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]),
  tpcbox_z(10, 10, 10, 12, 12, 12, 1))::numeric, 4);
-- 12.1244
SELECT round((tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz) |=|
  tpcpoint(pcpoint(1, 10, 10, 10), '2001-01-01'::timestamptz))::numeric, 4);
-- 15.5885
```

18.5.11. Ordenación B-tree — qué significa `ORDER BY tpcpoint`

La clase de operadores `tpcpoint_btree_ops` produce un orden total sobre los valores `tpcpoint`, pero el orden es byte a byte sobre la serialización `pcpoint` subyacente, no geométrico. Se aplican las mismas advertencias que al B-tree de tpcpatch:

- **`pcid` es el discriminador primario.** Los puntos con `pcid` menor se ordenan antes que los puntos con `pcid` mayor, independientemente de las coordenadas. La mezcla de pcids en una única columna agrupa las filas por esquema primero al ordenar.
- **Dentro de un único `pcid`, el orden sigue la disposición de dimensiones en disco del esquema** — típicamente X, Y, Z codificados como int32 (tras aplicar la escala por dimensión del esquema). Dos puntos cuya distancia euclidiana es pequeña pueden ordenarse muy separados si sus dimensiones X difieren, ya que X se compara primero.
- **La igualdad es igualdad exacta de bytes.** Dos pcpoints con los mismos valores de dimensión en el mismo esquema son iguales; el mismo conjunto de coordenadas codificado en un esquema con escala u orden de dimensiones diferente no lo es.

Para una ordenación con significado espacial, proyecte a un `tgeompoint` mediante la conversión con conocimiento de esquema y use el KNN `<->` de PostGIS sobre la geometría resultante, o use el operador KNN GiST `|=|` directamente sobre el tpcpoint. Para una ordenación con significado temporal, proyecte a `startTimestamp` o `timespan` y ordene sobre ese valor.

18.5.12. Relaciones espaciales

Se evalúan proyectando el tpcpoint a un `tgeompoint` a través de la caché de esquemas y delegando en la relación tgeompoint correspondiente. Cada función está conectada en tres órdenes de argumentos: `(geometry, tpcpoint)`, `(tpcpoint, geometry)` y `(tpcpoint, tpcpoint)`. tpcpatch queda intencionalmente fuera del alcance — la intersección a nivel de parche requeriría la descompresión punto a punto y un diseño diferente.

- Alguna vez / siempre intersecta.

  `eIntersects(tpcpoint,geometry)` → `bool`, `aIntersects(tpcpoint,geometry)` → `bool`

  `eIntersects(tpcpoint,tpcpoint)` → `bool`, `aIntersects(tpcpoint,tpcpoint)` → `bool`

```
SELECT eIntersects(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (1 1 1)');
-- t
SELECT aIntersects(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (1 1 1)');
-- f
```

```
SELECT eIntersects(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]));
-- t
```

- Alguna vez / siempre disjunto.

  `eDisjoint(tpcpoint,geometry) → bool`, `aDisjoint(tpcpoint,geometry) → bool`

  `eDisjoint(tpcpoint,tpcpoint) → bool`, `aDisjoint(tpcpoint,tpcpoint) → bool`

```
SELECT eDisjoint(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (1 1 1)');
-- t
SELECT aDisjoint(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (1 1 1)');
-- f
```

- Alguna vez / siempre dentro de la distancia.

  `eDwithin(tpcpoint,geometry,float) → bool`, `aDwithin(tpcpoint,geometry,float) → bool`

  `eDwithin(tpcpoint,tpcpoint,float) → bool`, `aDwithin(tpcpoint,tpcpoint,float) → bool`

```
SELECT eDwithin(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (5 5 5)',  ↩
      4);
-- t
SELECT aDwithin(tpcpointSeq(ARRAY[
  tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz),
  tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz),
  tpcpoint(pcpoint(1, 3, 3, 3), '2001-01-03'::timestamptz)]), geometry 'POINT Z (5 5 5)',  ↩
      4);
-- f
```

18.6. Tipo Temporal `tpcpatch`

Un `tpcpatch` es la elevación temporal de `pcpatch`. Es el espejo de `tpcpoint` en todos los aspectos — mismos subtipos, mismas funciones de acceso, mismas conversiones y operadores — excepto que cada instante contiene un lote comprimido completo de puntos en lugar de un único punto. La caja delimitadora es de nuevo un `tpcbox`, calculada en O(1) por instante a partir del `PCBOUNDS` embebido en el parche.

18.6.1. Constructores

- Construir un pcpatch temporal de subtipo instante

  `tpcpatch(pcpatch,timestamptz) → tpcpatch`

```
SELECT tempSubtype(tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), ' ↩
    2001-01-01'::timestamptz));
-- Instant
```

- Construir un pcpatch temporal de subtipo secuencia

  `tpcpatchSeq(tpcpatch[]) → tpcpatch`

  `tpcpatchSeq(tpcpatch[],text) → tpcpatch`

```
SELECT numInstants(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )])), interp(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]));
-- 2 | Step
```

- Construir un pcpatch temporal de subtipo conjunto de secuencias

  `tpcpatchSeqSet(tpcpatch[]) → tpcpatch`

```
SELECT numSequences(tpcpatchSeqSet(ARRAY[tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )])]));
-- 1
```

18.6.2. Accesores

Se aplican todos los accesores temporales genéricos, además del `pcid` y el `SRID` al estilo de `tpcpoint`. Adicionalmente:

- Devolver el número de puntos del parche contenido en el primer o último instante

  `startNumPoints(tpcpatch) → integer`

  `endNumPoints(tpcpatch) → integer`

```
SELECT startNumPoints(tpcpatch(PC_Patch(PC_MakePoint(1, ARRAY[10.0, 20.0, 30.0])),
  '2024-02-01'::timestamptz));
-- 1
```

- Devolver el número total de puntos de todos los parches de todos los instantes. Se lee directamente de la cabecera de cada parche — sin descompresión.

  `numPoints(tpcpatch) → bigint`

```
SELECT numPoints(tpcpatchSeq(ARRAY[
  tpcpatch(PC_Patch(ARRAY[PC_MakePoint(1, ARRAY[1.0, 1.0, 1.0]),
                          PC_MakePoint(1, ARRAY[2.0, 2.0, 2.0])]),
           '2024-01-01'::timestamptz),
  tpcpatch(PC_Patch(ARRAY[PC_MakePoint(1, ARRAY[3.0, 3.0, 3.0])]),
           '2024-01-02'::timestamptz)]));
-- 3
```

- Función devuelve-conjunto: emite una fila por cada (marca de tiempo de instante, punto) recorriendo el parche de cada instante y descomponiéndolo mediante la API C en memoria de pgPointCloud. Útil para combinar una columna tpcpatch con predicados por punto que los operadores de nivel de caja delimitadora no pueden expresar. El coste es O(total de puntos) — una descompresión por instante más una emisión de fila por punto.

  `points(tpcpatch) → setof (t timestamptz, point pcpoint)`

```
SELECT t, point FROM points(tpcpatch(
  PC_Patch(ARRAY[PC_MakePoint(1, ARRAY[1.0, 1.0, 1.0]),
                 PC_MakePoint(1, ARRAY[2.0, 2.0, 2.0])]),
  '2024-01-01'::timestamptz));
-- ('2024-01-01', '01:0000000000000000000000F03F0000000000000000F03F00000000000000F03F':: ↩
    pcpoint)
-- ('2024-01-01', '01:0000000000000000000000040000000000000000400000000000000040'::pcpoint ↩
    )
```

Véase Sección 18.6.11 para conocer los límites de estos accesores y qué está y qué no está disponible a nivel de granularidad por punto.

18.6.3. Transformaciones

- Transformar un tpcpatch en otro subtipo

  `tpcpatchInst(tpcpatch) → tpcpatch`

  `tpcpatchSeq(tpcpatch,interp='step') → tpcpatch`

  `tpcpatchSeqSet(tpcpatch) → tpcpatch`

```
SELECT tempSubtype(tpcpatchSeq(tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2,  ↩
    2)), '2001-01-01'::timestamptz)));
-- Sequence
```

- Transformar un tpcpatch a otra interpolación

  `setInterp(tpcpatch,interp) → tpcpatch`

```
SELECT interp(setInterp(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )], 'discrete'), 'step'));
-- Step
```

18.6.4. Restricciones

- Restringir un tpcpatch a (al complemento de) un valor pcpatch o un conjunto de valores pcpatch

  `atValue(tpcpatch,pcpatch) → tpcpatch`

  `minusValue(tpcpatch,pcpatch) → tpcpatch`

  `atValues(tpcpatch,pcpatchset) → tpcpatch`

  `minusValues(tpcpatch,pcpatchset) → tpcpatch`

```
-- The matching value is held up to the next instant; the closing bound
-- is stored as a second instant.
SELECT numInstants(atValue(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2))));
-- 2
SELECT numInstants(minusValue(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
```

```
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2))));
-- 1
```

- Restringir un tpcpatch a los instantes cuyo parche supera una prueba de superposición aproximada de PCBOUNDS frente al tpcbox, o eliminarlos. La granularidad es a nivel de parche: cada instante superviviente conserva su payload pcpatch textualmente, sin descompresión por punto. Como el `PCBOUNDS` de pgPointCloud es 2D, la dimensión Z de la caja se ignora a esta granularidad. Devuelve NULL cuando el pcid del tpcbox no coincide.

  `atTpcbox(tpcpatch,tpcbox,border_inc bool=TRUE) → tpcpatch`

  `minusTpcbox(tpcpatch,tpcbox,border_inc bool=TRUE) → tpcpatch`

```
SELECT numInstants(atTpcbox(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  tpcbox_xt(0, 0, 2, 2, tstzspan '[2001-01-01, 2001-01-03]', 1)));
-- 1
-- The dropped patch stays present on the open interval before its instant,
-- so the complement keeps two instants.
SELECT numInstants(minusTpcbox(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  tpcbox_xt(0, 0, 2, 2, tstzspan '[2001-01-01, 2001-01-03]', 1)));
-- 2
```

- Variante por punto de la anterior: cada instante superviviente contiene un pcpatch recién construido que incluye solo los puntos dentro (o fuera, en el caso de minus) del tpcbox en 2D — y en 3D cuando la caja tiene dimensión Z. Los instantes cuyo parche filtra hasta cero puntos se descartan. Más lenta que la variante aproximada porque cada parche se descomprime y reconstruye; úsela cuando se necesite fidelidad a nivel de punto.

  `atTpcboxFine(tpcpatch,tpcbox,border_inc bool=TRUE) → tpcpatch`

  `minusTpcboxFine(tpcpatch,tpcbox,border_inc bool=TRUE) → tpcpatch`

```
SELECT numInstants(atTpcboxFine(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  tpcbox_xt(0, 0, 1, 1, tstzspan '[2001-01-01, 2001-01-03]', 1)));
-- 1
```

- Restringir un tpcpatch a los puntos cuya proyección XY intersecta (o no intersecta, en el caso de minus) una geometría 2D. Z se ignora. El llamador debe garantizar la compatibilidad de SRID entre el esquema del parche y la geometría.

  `atGeometry(tpcpatch,geometry) → tpcpatch`

  `minusGeometry(tpcpatch,geometry) → tpcpatch`

```
-- The polygon boundary touches the point (3 3) of the patch at
-- 2001-01-02, so that patch survives both the at and the minus side.
SELECT numInstants(atGeometry(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  geometry 'POLYGON((0 0, 0 3, 3 3, 3 0, 0 0))'));
-- 2
```

```
SELECT numInstants(minusGeometry(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  geometry 'POLYGON((0 0, 0 3, 3 3, 3 0, 0 0))'));
-- 1
```

- Restringir un tpcpatch a los instantes anteriores o posteriores a una marca de tiempo. El indicador strict excluye el instante en la marca de tiempo misma

  `beforeTimestamp(tpcpatch,timestamptz,strict bool=TRUE) → tpcpatch`

  `afterTimestamp(tpcpatch,timestamptz,strict bool=TRUE) → tpcpatch`

```
SELECT timeSpan(beforeTimestamp(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5)), '2001-01-03'::timestamptz)]), timestamptz ' ↩
      2001-01-02'));
-- [2001-01-01, 2001-01-02)
SELECT timeSpan(afterTimestamp(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5)), '2001-01-03'::timestamptz)]), timestamptz ' ↩
      2001-01-02'));
-- (2001-01-02, 2001-01-03]
```

18.6.5. Modificaciones

- Insertar un tpcpatch en otro, o actualizar otro con él

  `insert(tpcpatch,tpcpatch,connect bool=TRUE) → tpcpatch`

  `update(tpcpatch,tpcpatch,connect bool=TRUE) → tpcpatch`

```
SELECT numInstants(insert(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]), tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5)), '2001-01-03'::timestamptz)])));
-- 3
```

- Eliminar un selector de tiempo (marca de tiempo, conjunto, período o conjunto de spans) de un tpcpatch

  `deleteTime(tpcpatch,timestamptz,connect bool=TRUE) → tpcpatch`

  `deleteTime(tpcpatch,tstzset,connect bool=TRUE) → tpcpatch`

  `deleteTime(tpcpatch,tstzspan,connect bool=TRUE) → tpcpatch`

  `deleteTime(tpcpatch,tstzspanset,connect bool=TRUE) → tpcpatch`

```
-- Deleting the middle instant splits the sequence into
-- {[2001-01-01, 2001-01-02), (2001-01-02, 2001-01-03]}; each half stores
-- its bound at the deleted timestamp, giving four instants.
SELECT numInstants(deleteTime(tpcpatchSeq(ARRAY[
```

```
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5)), '2001-01-03'::timestamptz)]), timestamptz ' ↩
      2001-01-02'));
-- 4
```

- Añadir un instante o una secuencia a un tpcpatch

  `appendInstant(tpcpatch,tpcpatch) → tpcpatch`

  `appendSequence(tpcpatch,tpcpatch) → tpcpatch`

```
SELECT numInstants(appendInstant(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5)), '2001-01-03'::timestamptz)));
-- 3
```

18.6.6. Fragmentación

- Devolver los valores distintos de un tpcpatch con el conjunto de spans durante el cual toma cada uno de ellos

  `unnest(tpcpatch) → {(value,time)}`

```
SELECT (un).time
FROM (SELECT unnest(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2024-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5), pcpoint(1, 6, 6, 6)), '2024-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2024-01-03'::timestamptz ↩
      )])) AS un) t;
-- {[2024-01-02, 2024-01-03)}
-- {[2024-01-01, 2024-01-02), [2024-01-03, 2024-01-03]}
```

- Fragmentar un tpcpatch en fragmentos, uno por cada intervalo de tiempo

  `timeSplit(tpcpatch,bin_width interval,origin timestamptz='2000-01-03') → {(time,tpcpatch`

```
SELECT (ts).time, numInstants((ts).temp)
FROM (SELECT timeSplit(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2024-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 5, 5, 5), pcpoint(1, 6, 6, 6)), '2024-01-02'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2024-01-03'::timestamptz ↩
      )]),
  interval '1 day', '2024-01-01'::timestamptz) AS ts) t;
-- 2024-01-01 | 2
-- 2024-01-02 | 2
-- 2024-01-03 | 1
```

18.6.7. Relaciones espaciales

- Devolver verdadero si y solo si al menos un punto de alguno de los instantes del tpcpatch intersecta la geometría (solo XY).

  `eIntersects(tpcpatch,geometry) → boolean`

```
SELECT eIntersects(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]), geometry 'POINT(1 1)');
-- t
```

18.6.8. Operadores de caja delimitadora

La misma superficie de operadores de caja que en Sección 18.5.10 se aplica a `tpcpatch`. El predicado se evalúa sobre el `tpcbox` del valor, calculado en O(1) por instante a partir del `PCBOUNDS` embebido en el parche.

- Predicados topológicos: `&&`, `@>`, `<@`, `~=`, `-|-`. Véase Sección 18.4.6.
- Predicados direccionales estrictos sobre los ejes X / Y / Z / tiempo (`<<`, `>>`, `<<|`, `|>>`, `<<\/`, `\/>>`, `<<#`, `#>>`) y sus variantes «solapa-o-X» (`&<`, `&>`, `&<|`, `|&>`, `&<\/`, `\/&>`, `&<#`, `#&>`). Véase Sección 18.4.7.
- La distancia de aproximación más cercana (`|=|`) es ordenable KNN a través de la clase de operador GiST.

  `nearestApproachDistance(tpcpatch,tpcbox) → float`

  `nearestApproachDistance(tpcpatch,tpcpatch) → float`

18.6.9. Comparaciones

- Comprobar si el valor en algún instante (alguna vez) o en todos los instantes (siempre) es igual a, o distinto de, un pcpatch u otro tpcpatch

  `eEq(pcpatch,tpcpatch) → boolean`

  `eEq(tpcpatch,pcpatch) → boolean`

  `eEq(tpcpatch,tpcpatch) → boolean`

  `aEq`, `eNe` y `aNe` reciben los mismos tres pares de argumentos

  `pcpatch ?= tpcpatch → boolean`

  `tpcpatch ?= pcpatch → boolean`

  `tpcpatch ?= tpcpatch → boolean`

  Los operadores `%=`, `?<>` y `%<>` reciben los mismos tres pares de operandos

```
SELECT eEq(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]));
-- t
SELECT aEq(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]), pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)));
-- f
```

- Devolver la igualdad o desigualdad temporal de un tpcpatch con un pcpatch o con otro tpcpatch, como un tbool

  `tEq(pcpatch,tpcpatch) → tbool`

  `tEq(tpcpatch,pcpatch) → tbool`

  `tEq(tpcpatch,tpcpatch) → tbool`

`tNe` recibe los mismos tres pares de argumentos

`pcpatch #= tpcpatch → tbool`

`tpcpatch #= pcpatch → tbool`

`tpcpatch #= tpcpatch → tbool`

El operador `#<>` recibe los mismos tres pares de operandos

```
SELECT tEq(tpcpatchSeq(ARRAY[
  tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01'::timestamptz ↩
      ),
  tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02'::timestamptz ↩
      )]), pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)));
-- [t@2001-01-01, f@2001-01-02]
```

18.6.10. Ordenación B-tree — qué significa `ORDER BY tpcpatch`

La clase de operador `tpcpatch_btree_ops` produce un orden total sobre los valores `tpcpatch`, pero ese orden no es espacial. El comparador sobre el `pcpatch` subyacente es byte a byte (`memcmp` sobre los bytes varlena significativos); el comparador al nivel temporal es la combinación lexicográfica de las marcas de tiempo por instante y el orden de bytes de `pcpatch` por instante. Las consecuencias observables son:

- **`pcid` es el discriminador primario.** Un parche con un `pcid` menor se ordena antes que uno con un `pcid` mayor, independientemente de las coordenadas de los puntos, su cantidad o la compresión. Esto hace que `ORDER BY` sobre una columna de pcid mixto agrupe las filas primero por esquema, lo que a veces resulta útil como paso de agrupación gruesa «por esquema de origen».
- **Dentro de un mismo `pcid`, el orden sigue la disposición de bytes en disco**: esquema de `compression`, luego `npoints`, luego el `PCBOUNDS` 2D (xmin, xmax, ymin, ymax) y finalmente el payload `data` comprimido. Este orden está bien definido y es estable para una versión dada de pgPointCloud, pero *no* es un orden geométrico o espacial — dos parches cuyos cuadros delimitadores se superponen en el espacio 3D pueden ordenarse de forma arbitraria entre sí si difieren en su esquema de compresión o en el número de puntos.
- **La igualdad es igualdad exacta de bytes** sobre los bytes significativos (se excluye el relleno de ceros al final que reserva el varlena de pgPointCloud). Dos parches construidos a partir del mismo conjunto de puntos en el mismo orden, con la misma compresión, son iguales; reordenar los puntos o cambiar la compresión los hace desiguales.

Para un orden con sentido espacial, use el operador KNN de GiST `|=|` (véase distancia de aproximación más cercana en tpcpatch); para un orden con sentido temporal, proyecte a `startTimestamp` o `timespan` y ordene sobre ese valor.

18.6.11. Operaciones por punto: qué está y qué no está disponible

Las operaciones sobre valores `tpcpatch` se realizan a dos granularidades: *nivel de parche* y *por punto*. La distinción importa porque los parches de pgPointCloud almacenan sus puntos en un payload comprimido que la capa de caja delimitadora no puede inspeccionar.

- **Nivel de parche — soportado.** El parche de cada instante se trata como opaco y se conserva o descarta como un todo, usando únicamente la cabecera `PCBOUNDS` de 4 valores double (xmin / xmax / ymin / ymax) y la marca de tiempo del instante. Esta es la granularidad de atTpcbox / minusTpcbox y de los operadores de caja delimitadora en Sección 18.6.8, así como del agregado `tpcbox` basado en caja. No se produce descompresión del payload; las consultas son O(número de instantes).

  Como `PCBOUNDS` es 2D, la dimensión Z de cualquier argumento `tpcbox` se ignora a esta granularidad, incluso cuando el esquema subyacente tiene dimensión Z.
- **Inspección por punto — soportada.** La función devuelve-conjunto points emite una fila por cada (marca de tiempo de instante, punto) descomponiendo el parche de cada instante mediante la API C en memoria de pgPointCloud. El coste es O(total de puntos) — una descompresión por instante más una emisión de fila por punto.

- **Filtrado / construcción por punto — soportado.** atTpcboxFine / minusTpcboxFine, atGeometry / minusGeometry y eIntersects(tpcpatch, geometry) recorren en C cada punto de cada instante, aplicando su predicado sobre las coordenadas reales del punto en lugar de limitarse al cuadro delimitador del parche. Los instantes supervivientes contienen un parche recién construido que incluye solo los puntos que superaron el predicado; los instantes cuyos parches filtran hasta cero puntos se descartan.

El flujo de trabajo recomendado cuando ambas granularidades están disponibles es: (a) podar al nivel de parche con `atTpcbox` o un escaneo de índice SP-GiST/GiST para descartar los instantes completos cuyo `PCBOUNDS` no se superpone con el área de interés, y (b) refinar sobre los supervivientes con `atTpcboxFine` / `atGeometry` cuando se necesite fidelidad a nivel de punto.

18.7. Índices

Las clases de operadores B-tree (`tpcpoint_btree_ops`, `tpcpatch_btree_ops`, `tpcbox_btree_ops`) y hash (`tpcpoint_`
`tpcpatch_hash_ops`) admiten predicados de igualdad y ordenación. Las clases de operadores GiST R-tree y SP-GiST quadtree / kd-tree aceleran los predicados basados en la caja delimitadora (`&&`, `@>`, `<@`, `~=`, `-|-`) sobre los tres tipos `tpcbox`, `tpcpoint` y `tpcpatch`:

```
-- GiST (R-tree)
CREATE INDEX ON my_table USING gist(my_tpcpoint_column);
CREATE INDEX ON my_table USING gist(my_tpcpatch_column);

-- SP-GiST quadtree (default) or kd-tree
CREATE INDEX ON my_table USING spgist(my_tpcpoint_column);
CREATE INDEX ON my_table USING spgist(my_tpcpoint_column tpcpoint_kdtree_ops);
```

Las clases de operadores SP-GiST almacenan internamente un `stbox` (el filtro `pcid` se recupera mediante la revisión del operador), por lo que una consulta contra un índice sobre una columna cuyos valores mezclan múltiples `pcid` devolverá un conjunto ligeramente mayor de filas candidatas para revisión — algo irrelevante cuando cada tabla contiene valores de un único esquema, que es el caso común.

18.8. Agregaciones

- Agregado de caja delimitadora sobre una columna de valores tpcpoint, tpcpatch o tpcbox. Devuelve el tpcbox más pequeño que contiene todas las entradas. Agregar a través de pcid distintos genera un error — la caja delimitadora sería ininterpretable ya que las dimensiones son relativas al pcid.

  `extent(tpcpoint) → tpcbox`

  `extent(tpcpatch) → tpcbox`

  `extent(tpcbox) → tpcbox`

```
SELECT extent(v) FROM (VALUES
  (tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz)),
  (tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz))) t(v);
-- TPCBOX(ZT(((1,1,1),(2,2,2)),[2001-01-01, 2001-01-02]), 1)
```

- Conteo temporal y conteo por ventana de filas superpuestas en cada marca de tiempo. El resultado es un tint. Todas las filas de entrada deben compartir el mismo subtipo temporal (instante / secuencia / conjunto de secuencias).

  `tcount(tpcpoint) → tint, tcount(tpcpatch) → tint`

  `wcount(tpcpoint,interval) → tint, wcount(tpcpatch,interval) → tint`

```
SELECT tCount(v) FROM (VALUES
  (tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz)),
  (tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz))) t(v);
-- {1@2001-01-01, 1@2001-01-02}
```

```
SELECT wCount(v, interval '1 day') FROM (VALUES
  (tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz)),
  (tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz))) t(v);
-- {[1@2001-01-01, 2@2001-01-02], (1@2001-01-02, 1@2001-01-03]}
```

- Conteo de puntos de pcpatch por instante, sumado a través de las filas. Distinto de `tcount(tpcpatch)`, que cuenta el número de parches (siempre 1 por instante). `tnpoints` se lee como «cuántos puntos había en la nube en el tiempo t», la primitiva natural para el control de calidad de la ingesta y los paneles de densidad a lo largo del tiempo.

  `tnpoints(tpcpatch) → tint`

```
SELECT tnpoints(v) FROM (VALUES
  (tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01':: ↩
      timestamptz)),
  (tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02':: ↩
      timestamptz))) t(v);
-- {2@2001-01-01, 2@2001-01-02}
```

- Densidad de puntos por instante (`numPoints / área-bbox-xy`) sumada a través de las filas. Cada pcpatch lleva el `PCBOUNDS` en línea (xmin / xmax / ymin / ymax) por lo que el término de área de la caja delimitadora se lee directamente — sin recálculo. Un parche de 1 punto o colineal produce `+Infinity` para ese instante; filtre con `isfinite()` en consultas posteriores si es necesario.

  `tdensity(tpcpatch) → tfloat`

```
-- Each patch holds 2 points over a 1x1 xy-bbox.
SELECT tdensity(v) FROM (VALUES
  (tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01':: ↩
      timestamptz)),
  (tpcpatch(pcpatch(1, pcpoint(1, 3, 3, 3), pcpoint(1, 4, 4, 4)), '2001-01-02':: ↩
      timestamptz))) t(v);
-- {2@2001-01-01, 2@2001-01-02}
```

- Combina múltiples valores temporales en uno solo con todos los instantes de entrada fusionados en orden temporal. Todas las entradas deben compartir el mismo pcid y subtipo.

  `merge(tpcpoint) → tpcpoint, merge(tpcpatch) → tpcpatch`

  `mergeAgg(tpcpoint) → tpcpoint, mergeAgg(tpcpatch) → tpcpatch`

```
SELECT numInstants(mergeAgg(v)) FROM (VALUES
  (tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz)),
  (tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz))) t(v);
-- 2
```

- Agregados de construcción en flujo: `appendInstant` ensambla instantes en una secuencia; `appendSequence` ensambla secuencias en un conjunto de secuencias. Útil para cargadores de trayectorias que consumen filas en orden temporal.

  `appendInstant(tpcpoint[,interp,maxt]) → tpcpoint, appendInstant(tpcpatch) → tpcpatch`

  `appendInstantAgg(tpcpoint[,interp,maxt]) → tpcpoint, appendInstantAgg(tpcpatch) → tpcpatc`

  `appendSequence(tpcpoint) → tpcpoint, appendSequence(tpcpatch) → tpcpatch`

  `appendSequenceAgg(tpcpoint) → tpcpoint, appendSequenceAgg(tpcpatch) → tpcpatch`

```
SELECT numInstants(appendInstant(v ORDER BY startTimestamp(v))) FROM (VALUES
  (tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz)),
  (tpcpoint(pcpoint(1, 2, 2, 2), '2001-01-02'::timestamptz))) t(v);
-- 2
```

18.9. Representación binaria y portabilidad

Tanto `tpcpoint` como `tpcpatch` admiten la E/S binaria estándar de MobilityDB a través de `asBinary` / `tpcpointFromBinary` / `tpcpatchFromBinary`, y a través de `COPY ... (FORMAT BINARY)` de PostgreSQL. La codificación incrusta el `pcid` de cada valor junto a la carga de dimensiones, por lo que el blob binario es inequívoco sobre bajo qué esquema se escribieron los valores.

Siempre que el backend de codificación tenga el XML de esquema de un `pcid` dado registrado en su caché de esquemas a nivel de MEOS (que la extensión de PG rellena de forma perezosa en el primer uso, escaneando `pointcloud_formats`), el codificador incrusta además el XML de esquema en el blob WKB. El bit `MEOS_WKB_PCSCHEMAFLAG` (0x80) en el byte de banderas temporales lo señala. Un receptor que aún no conoce el `pcid` — por ejemplo, un clúster diferente tras un `pg_dump --format=binary` — analiza y registra automáticamente el esquema incrustado, por lo que la importación tiene éxito sin un paso de precarga de esquema fuera de banda.

Si el backend de codificación solo tiene un `PCSCHEMA` analizado (sin XML de origen) para un `pcid`, el bit de esquema se deja sin establecer y el blob es portable solo entre backends que comparten el mismo catálogo `pointcloud_formats`. Esto coincide con el comportamiento de la codificación anterior sin esquema y es la ruta de reserva cuando la caché se rellenó a mano desde un programa MEOS independiente.

El receptor también necesita que `mobilitydb_init` se haya ejecutado — automático en cualquier backend que haya cargado la extensión `mobilitydb`, ya que el módulo de información de funciones instala allí el hook de análisis de XML.

18.10. Salida de texto

- Devolver la representación textual de un tpcpoint, o de un arreglo de ellos, y convertir un tpcpoint a texto

  `asText(tpcpoint) → text`

  `asText(tpcpoint[]) → text[]`

  `tpcpoint::text → text`

```
-- The pcpoint payload renders as hex-encoded WKB.
SELECT asText(tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz));
-- 5C0000000100000064000000640000006400000000000000@2001-01-01
```

- Devolver la representación textual de un tpcpatch, o de un arreglo de ellos, y convertir un tpcpatch a texto

  `asText(tpcpatch) → text`

  `asText(tpcpatch[]) → text[]`

  `tpcpatch::text → text`

```
-- The pcpatch payload renders as hex-encoded WKB.
SELECT left(asText(tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), ' ↩
    2001-01-01'::timestamptz)), 24);
-- EC0100000100000000000000
```

A diferencia de los demás tipos temporales espaciales, aquí `asText` no recibe un argumento `maxdecimaldigits`: un `pcpoint` o un `pcpatch` se imprime como el hex-WKB de su forma serializada, que no tiene decimales que redondear. Lo mismo ocurre con `th3index`.

La salida de texto es el subconjunto seguro de la superficie textual: no existe `tpcpointFromText` / `tpcpatchFromText`, porque `pcpoint_in` / `pcpatch_in` de pgPointCloud solo aceptan hex-WKB y el esquema no se puede reconstruir a partir de la forma textual. Para la ida y vuelta use `asBinary` / `asHexWKB` con sus funciones de análisis correspondientes.

18.11. Salida MF-JSON

- Para `tpcpoint`, el tipo temporal se emite como `{"type":"MovingPCPoint", ... "coordinates":[[X,Y,Z], ...], "datetimes":[...]}`. X/Y/[Z] se extraen del `pcpoint` de cada instante mediante la caché de esquemas; si el esquema carece de X/Y el array de coordenadas está vacío para ese instante.

  `asMFJSON(tpcpoint, options int=0, flags int=0, maxdecimaldigits int=15) → text`

```
SELECT asMFJSON(tpcpoint(pcpoint(1, 1, 1, 1), '2001-01-01'::timestamptz));
-- {"type":"MovingPCPoint","coordinates":[[1,1,1]],
--  "datetimes":["2001-01-01T00:00:00+00"],"interpolation":"None"}
```

- Para `tpcpatch`, la etiqueta de tipo es `"MovingPCPatch"` y cada instante emite un pequeño objeto `{"pcid":N, "npoints":N, "bounds":[xmin,xmax,ymin,ymax]}`. La carga de puntos comprimida no está intencionadamente en el JSON — use `asBinary` / `asHexWKB` para el ida y vuelta.

  `asMFJSON(tpcpatch, options int=0, flags int=0, maxdecimaldigits int=15) → text`

```
SELECT asMFJSON(tpcpatch(pcpatch(1, pcpoint(1, 1, 1, 1), pcpoint(1, 2, 2, 2)), '2001-01-01 ↩
    '::timestamptz));
-- {"type":"MovingPCPatch","values":[{"pcid":1,"npoints":2,"bounds":[1,2,1,2]}],
--  "datetimes":["2001-01-01T00:00:00+00"],"interpolation":"None"}
```

Cuando `options` se establece en 1 la salida también incluye una clave `"bbox"` — para `tpcpoint` / `tpcpatch` esto es un objeto JSON TPCBox que añade un campo `"pcid"` junto al array `"bbox"` con forma de stbox estándar.

MF-JSON es solo de salida para `tpcpoint` / `tpcpatch`. La forma JSON no lleva el esquema (pcid + disposición de dimensiones) para tpcpoint, y es un resumen por instante que descarta la carga por punto para tpcpatch — por lo que un analizador inverso no puede reconstruir el valor original. Para un ida y vuelta sin pérdidas use `asBinary` / `tpcpointFromBinary` / `tpcpatchFromBinary`, o `asHexWKB` / `tpcpointFromHexWKB` / `tpcpatchFromHexWKB`.

18.12. Integración con PDAL: `readers.tpcpatch` y `writers.tpcpatch`

MobilityDB incluye complementos PDAL nativos bajo `contrib/pdal/` que leen y escriben valores `tpcpatch` directamente sin pasar por `readers.pgpointcloud` + `tpcpatchSeq()`. Compílelos con:

```
cd contrib/pdal
cmake -B build && cmake --build build -j
sudo cmake --install build
```

PDAL descubre los complementos mediante `PDAL_DRIVER_PATH` o, tras la instalación, en su ruta de búsqueda por defecto. Verifique con `pdal --drivers | grep tpcpatch`.

18.12.1. `readers.tpcpatch`

Lee cualquier consulta SQL que devuelva filas de `(timestamp, pcpatch, pcid)` y emite un `PointView` de PDAL con todas las dimensiones del esquema más una dimensión `time_t` que lleva microsegundos desde la época. La decodificación se delega a `pc_patch_from_wkb` de pgPointCloud, por lo que se manejan los tres tipos de compresión (`PC_NONE`, `PC_DIMENSIONAL`, `PC_LAZPERF`).

Para descomprimir un `tpcpatch` almacenado en filas por instante que el lector pueda consumir:

```
{
  "type": "readers.tpcpatch",
  "connection": "host=/tmp dbname=mydb",
  "query": "SELECT (EXTRACT(EPOCH FROM timestampN(traj, n))*1000000)::bigint AS t,
                   valueN(traj, n)              AS pcp,
                   PC_PCId(valueN(traj, n))     AS pcid
```

```
            FROM trajectories,
                 generate_series(1, numInstants(traj)) n
            WHERE id = 42
            ORDER BY n"
}
```

18.12.2. writers.tpcpatch

Recibe un PointView con una dimensión de tiempo configurable, agrupa los puntos por marca de tiempo en valores pcpatch por instante, luego despacha un INSERT, UPDATE, fusión-append o upsert contra una columna tpcpatch usando los constructores tpcpatch(pcpatch, timestamptz) + tpcpatchSeq(pcpatch[]) de MobilityDB. La codificación se delega a pc_patch_to_wkb de pgPointCloud con pc_patch_compress opcional. La escala / desplazamiento del esquema se aplica simétricamente — los valores fluyen como sus unidades semánticas (metros, grados, . . .) a través de PDAL incluso cuando el esquema los almacena como int32 escalado.

Opciones:

- connection (posicional) — cadena de conexión libpq.
- table (posicional) — nombre de la tabla de destino.
- column — nombre de la columna tpcpatch (por defecto traj).
- pcid (requerido) — entrada en pointcloud_formats cuyo esquema deben coincidir las dimensiones de entrada.
- id_column, id_value — identifican la fila de destino para cualquier modo distinto de insert.
- mode — insert (por defecto — nueva fila por ejecución de la tubería), update (sobrescribe la fila identificada por id_column = id_value), append (concatena mediante merge() sobre la fila existente), o upsert (inserta si está ausente, añade si está presente). append / update generan un error claro «matched no row» si no se encuentra id_value, por lo que no pueden ocurrir escrituras silenciosas de cero filas.
- time_dim — nombre de la dimensión de tiempo de origen (por defecto time_t; establézcalo en GpsTime para consumir marcas de tiempo LAS directamente sin una etapa filters.assign).
- time_format — interpretación de time_dim: unix_microseconds (por defecto), unix_seconds, o gps_adjusted (Tiempo GPS Estándar Ajustado LAS-1.4 — el escritor aplica el desplazamiento de época GPS-Unix y el offset de segundo intercalar de 2024).
- compression — none (por defecto), dimensional, o laz. MobilityDB normaliza el almacenamiento en disco de tpcpatch a PC_NONE, por lo que la opción solo afecta al tamaño de transporte libpq — la reducción típica es ~3x para dimensional en datos con dimensiones redundantes.
- flush_threshold (por defecto 1024) — número de parches por instante completados almacenados en memoria antes de emitir un INSERT/append. Se respeta para mode=append/upsert; mode=insert/update difieren a un único vaciado final ya que cada ejecución de la tubería produce una única fila.

El escritor emite líneas de progreso periódicas de nivel Info mediante el registrador de PDAL y un resumen final en done(): total de puntos / parches escritos, columna de destino, pcid, compresión y modo. Visible con pdal pipeline -v 5.

18.12.3. Esquemas de compresión soportados

El lector y el escritor admiten los tres tipos de compresión de pgPointCloud: PC_NONE, PC_DIMENSIONAL y PC_LAZPERF. La regresión PC_LAZPERF en contrib/pdal/examples/test_lazperf.sql requiere que la instalación de pgPointCloud en ejecución esté compilada con --with-lazperf=DIR; el fixture SQL aborta con un mensaje claro en caso contrario. Los complementos se mantienen bajo contrib/pdal/ en lugar de en el árbol de compilación principal de MobilityDB para que puedan compilarse y actualizarse de forma independiente.

18.12.4. Limitaciones

- *Endianness de WKB.* `readers.tpcpatch` acepta solo pcpatch WKB little-endian (NDR). Un parche big-endian (primer byte `0x00`) se rechaza con un error claro en lugar de decodificarse mal silenciosamente. pgPointCloud emite NDR en toda plataforma común; el soporte XDR sería aditivo.
- *Compresión del tpcpatch almacenado.* La opción `compression` de `writers.tpcpatch` controla solo la forma de transporte libpq. El `tpcpatch` de MobilityDB normaliza su almacenamiento de pcpatch en disco a `PC_NONE`.
- *Escritura en flujo.* `writers.tpcpatch` implementa la interfaz `Streamable::processOne` de PDAL, por lo que una tubería `readers.las` → `writers.tpcpatch` se ejecuta en modo de flujo de PDAL automáticamente y nunca mantiene toda la nube de puntos en memoria. La cola de parches completados está limitada por la opción del escritor `flush_threshold` (por defecto 1024); en `mode=append` / `mode=upsert` cada vaciado activado por umbral emite un lote SQL incremental mediante `merge()`, mientras que `mode=insert` / `mode=update` difieren a un único vaciado final en `done()` (semántica de fila única). Esta es la ruta que hace tratable la ingesta de LAS de varios GB.

18.13. Puente PCL: filtros invocables desde SQL y registro

Una extensión complementaria opcional `mobilitydb_pcl` vive bajo `contrib/pcl/`. Enlaza el backend de PostgreSQL en ejecución con la Point Cloud Library y expone un pequeño conjunto de operaciones por pcpatch como funciones SQL. El despacho de tipo de punto consciente del esquema elige `pcl::PointXYZ`, `pcl::PointXYZI`, o `pcl::PointXYZRGB` en el momento de la llamada según la disposición de dimensiones del pcid de entrada.

- `pcpatch_to_pcd(pcpatch)` → `bytea` y `pcpatch_from_pcd(bytea, pcid)` → `pcpatch` — ida y vuelta a través del formato de cable PCD de PCL. El ida y vuelta preserva todos los valores de dimensión de forma bit-exacta.
- `pcpatch_voxel_grid(pcpatch, leaf double precision)` → `pcpatch` — submuestreo VoxelGrid, leaf en las unidades espaciales del esquema. Preserva Intensity / RGB.
- `pcpatch_sor(pcpatch, k integer DEFAULT 50, stddev_mul double precision DEFAULT 1.0)` → `pcpatch` — Eliminación de valores atípicos estadística (Statistical Outlier Removal). Descarta los puntos cuya distancia media a sus k vecinos más cercanos cae fuera de (media ± stddev_mul × stddev). Preserva Intensity / RGB.
- `pcpatch_icp(source pcpatch, target pcpatch, max_iter int DEFAULT 50, max_corr double DEFAULT 1.0)` → `double[]` y `pcpatch_gicp(...)` con la misma firma — registro rígido IterativeClosestPoint y GeneralizedICP. Ambos devuelven un array de 8 dobles `[tx, ty, tz, qw, qx, qy, qz, fitness]` componible en una pose de MobilityDB mediante `pose_make_3d`. El coste de Mahalanobis de GICP sobre covarianzas locales por punto es notablemente más robusto que ICP en superficies no planas / con vegetación (escaneos de corredores viales, fachadas patrimoniales fotogramétricas).
- `pcpatch_normals(pcpatch, k int DEFAULT 10)` → `double[]` — normal de superficie + curvatura por punto mediante `NormalEstimation<PointXYZ, Normal>` de PCL sobre los `k` vecinos más cercanos. Devuelve un array plano de longitud `4 × npoints` dispuesto como `[nx_0, ny_0, nz_0, curv_0, nx_1, ny_1, nz_1, curv_1, ...]`. Casos de uso: detección de cambios de superficie orientada entre estudios repetidos, exportación Potree `--normal`, siembra de mallas fotogramétricas. Para parches muy grandes (>50 k puntos) considere submuestrear primero con `pcpatch_voxel_`

El puente está condicionado a `CREATE EXTENSION mobilitydb_pcl` en bases de datos que ya han cargado `pointcloud` + `mobilitydb`. Las instrucciones de compilación y una suite de regresión determinista están en `contrib/pcl/README.md`.

18.14. Advertencias

- La extensión `pointcloud` debe existir en la base de datos antes de que las definiciones SQL de `mobilitydb` hagan referencia a `pcpoint` / `pcpatch`. En una compilación `POINTCLOUD=ON` el `mobilitydb.control` generado declara esta dependencia en su cláusula `requires =`, por lo que `CREATE EXTENSION mobilitydb CASCADE` resuelve `pointcloud` automáticamente. La creación manual en el orden incorrecto sin `CASCADE` sigue generando un error de tipo ausente.

- La mezcla de esquemas se rechaza en el momento de la construcción: no se puede construir una secuencia `tpcpoint` cuyos instantes tengan valores `pcid` diferentes, y la misma restricción se aplica a `pcpointset` y `pcpatchset`. El esquema es lo que da significado a las dimensiones, y mezclar esquemas silenciosamente produciría valores cuyas coordenadas son ininterpretables.
- El flag de CMake `POINTCLOUD=ON` es obligatorio. Las compilaciones sin él producen un `libMobilityDB` que no declara ninguno de los tipos de este capítulo; los archivos SQL para ellos no se generan.

Capítulo 19

Muestreo de Ráster

La extensión ráster de PostGIS almacena datos de cobertura en cuadrícula (elevación, temperatura, imágenes satelitales, ...) en el tipo `raster`. Cada ráster tiene una o más bandas; una banda contiene un arreglo 2D de valores de píxel, una extensión espacial, un tamaño de píxel y un valor centinela opcional para datos nulos (`nodata`).

El operador de muestreo de ráster de MobilityDB evalúa una banda de ráster en los instantes de una trayectoria de punto móvil y devuelve un `tfloat`. Los instantes cuya posición cae fuera de la extensión del ráster o sobre un píxel `nodata` se descartan silenciosamente. Esto permite consultas como *¿qué perfil de elevación siguió cada vehículo?* o *¿qué viajes entraron en una banda de umbral de temperatura?*

Para usar los operadores descritos aquí, MobilityDB debe ser compilado con `-DRASTER=ON`. En tal compilación, el archivo `mobilitydb.control` generado declara `requires = 'postgis, postgis_raster'`, por lo que un único `CASCADE` crea la pila completa:

```
CREATE EXTENSION mobilitydb CASCADE;
-- NOTICE:  installing required extension "postgis"
-- NOTICE:  installing required extension "postgis_raster"
```

19.1. Operador de Muestreo

- Muestrear una banda de ráster en cada instante de una trayectoria

  `raster_value(raster,tgeompoint,band integer DEFAULT 1) → tfloat`

  Evalúa el píxel de la `band` en cada instante de `traj` usando muestreo bilineal-vecino más próximo (`ST_Value` con `resample=false`). Devuelve un `tfloat` de conjunto de instantes cuyos valores se redondean a cuatro decimales. Devuelve `NULL` cuando ningún instante se interseca con el ráster.

```
-- Build a 3$\times$3 synthetic elevation raster (SRID 4326, 1\textdegree{} pixels)
WITH rast AS (
  SELECT ST_SetValues(
    ST_AddBand(
      ST_MakeEmptyRaster(3, 3, 0.0, 3.0, 1.0, -1.0, 0.0, 0.0, 4326),
      '32BF'::text, 0.0::float8, NULL::float8),
    1, 1, 1,
    ARRAY[[10.0::float4, 20.0::float4, 30.0::float4],
          [40.0::float4, 50.0::float4, 60.0::float4],
          [70.0::float4, 80.0::float4, 90.0::float4]]) AS r
)
SELECT raster_value(r,
  tgeompoint 'SRID=4326;[POINT(0.5 2.5)@2000-01-01 00:00:00+00,
                         POINT(2.5 2.5)@2000-01-02 00:00:00+00,
                         POINT(0.5 0.5)@2000-01-03 00:00:00+00]'::text
FROM rast;
```

Aplicado al conjunto de datos de trayectorias BerlinMOD con un ráster de elevación del terreno:

```
-- Elevation profile per trip (requires a loaded elevation raster)
SELECT tripid, raster_value(elev, trip4326) AS elevation
FROM trips, elevation_raster;

-- Average elevation per trip
SELECT tripid, avgValue(raster_value(elev, trip4326)) AS mean_elev
FROM trips, elevation_raster;

-- Trips that crossed terrain above 200 m
SELECT tripid
FROM trips, elevation_raster
WHERE atSpan(raster_value(elev, trip4326),
             floatspan '[200, 9999]') IS NOT NULL;
```

19.2. Muestreo Raquet / QUADBIN

Raquet es un formato ráster nativo en la nube que almacena teselas ráster como filas en un archivo Apache Parquet. Cada fila contiene los bytes de píxel de una tesela Web-Mercator junto con su identificador de celda QUADBIN — un entero de 64 bits que codifica el nivel de zoom y las coordenadas x/y intercaladas mediante Morton. No se requieren metadatos espaciales adicionales: el rectángulo delimitador y el mapeo de píxel a coordenadas quedan completamente determinados por el valor QUADBIN.

El patrón de consulta típico une una trayectoria con una tabla Raquet en dos pasos:

```
-- Step 1: identify which tiles the trajectory overlaps
SELECT DISTINCT q
FROM unnest(trajectory_quadbins(traj, 8)) AS q;

-- Step 2: sample each matching tile
SELECT raster_tile_value_quadbin(
    band_data, width, height, quadbin, 'FLOAT32', nodata, true, traj)
FROM elevation_raquet
WHERE quadbin = ANY(trajectory_quadbins(traj, 8));
```

- Devolver las celdas QUADBIN distintas en un nivel de zoom cubiertas por una trayectoria

  `trajectory_quadbins(tgeompoint,integer) → bigint[]`

  Devuelve el conjunto de celdas QUADBIN únicas (deduplicadas) cuyas envolventes de tesela Web-Mercator contienen al menos un instante de `traj`. El resultado es adecuado como clave de unión en la cláusula WHERE contra una tabla Raquet. El parámetro `zoom` debe estar en [0, 15].

```
-- Three instants spanning three tiles at zoom 3
SELECT trajectory_quadbins(
  tgeompointFromText('SRID=4326;{Point(-60 45)@2024-01-01,
                                 Point(0 45)@2024-01-02,
                                 Point(60 45)@2024-01-03}'), 3);

-- Count of Raquet tiles a ship trajectory overlaps at zoom 8
SELECT array_length(trajectory_quadbins(trip4326, 8), 1) AS tile_count
FROM trips
ORDER BY tile_count DESC;
```

- Muestrear un chip ráster de Raquet a lo largo de una trayectoria usando una celda QUADBIN para la georreferenciación

  `raster_tile_value_quadbin(bytea,integer,integer,bigint,text,float8,boolean,tgeompoint) → tfloat`

  Evalúa un arreglo de píxeles de banda única en orden de fila mayor en cada instante de `traj` (SRID 4326). La georreferenciación de la tesela (rectángulo delimitador, mapeo de píxel a coordenadas) se deriva únicamente de `quadbin` mediante la

transformada de tesela deslizante Web-Mercator. El argumento `pixtype` debe ser uno de `UINT8`, `INT16`, `INT32`, `FLOAT32` o `FLOAT64`. Cuando `has_nodata` es verdadero, los instantes cuyo píxel equivale a `nodata` se descartan silenciosamente. Devuelve `NULL` cuando ningún instante sobrevive.

```
-- Sample elevation tiles from a Raquet Parquet table
-- for all ship trajectories
SELECT t.tripid,
       raster_tile_value_quadbin(
           r.band_data, r.width, r.height, r.quadbin,
           'FLOAT32', r.nodata, true, t.trip4326)
FROM trips t
JOIN elevation_raquet r
  ON r.quadbin = ANY(trajectory_quadbins(t.trip4326, 8));

-- Average sea-surface temperature per trajectory
SELECT t.tripid,
       avgValue(raster_tile_value_quadbin(
           r.band_data, 256, 256, r.quadbin,
           'INT16', -9999, true, t.trip4326)) AS mean_sst
FROM trips t
JOIN sst_raquet r
  ON r.quadbin = ANY(trajectory_quadbins(t.trip4326, 6));
```

- Construir una tesela ráster Raquet a partir de sus bytes de píxeles, dimensiones, celda QUADBIN y tipo de píxel

  `raquet(bytea,integer,integer,bigint,text,float8) → raquet`

  Empaqueta un arreglo de píxeles `pixels` de banda única en orden de fila mayor de `width` × `height` píxeles de tipo `pixtype` (uno de `UINT8`, `INT16`, `INT32`, `FLOAT32` o `FLOAT64`), georreferenciado por la celda `quadbin`, en un único valor `raquet` autodescriptivo. El argumento `nodata` es opcional; cuando se omite (`NULL`) la tesela no lleva valor centinela de nodata. Se genera un error cuando el arreglo `pixels` es menor que `width` × `height` × el tamaño del píxel. Un `raquet` es un valor de longitud variable, libre de GDAL, con una representación portátil en texto Hex-WKB y binaria, distinto de y coexistente con el tipo `raster` de PostGIS usado por raster_value.

```
-- Empaquetar las columnas sueltas de una tabla Raquet en valores raquet
SELECT raquet(band_data, width, height, quadbin, 'FLOAT32', nodata) AS tile
FROM elevation_raquet;

-- Una tesela UINT8 de 2 x 2 sin nodata
SELECT raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8');
```

- Decodificar un archivo ráster en memoria en una tesela ráster Raquet mediante GDAL

  `raquet_read(bytea,bigint) → raquet`

  Lee los bytes de `rasterfile` — un ráster en cualquier formato compatible con GDAL (GeoTIFF, PNG, . . .) — a través del sistema de archivos virtual `/vsimem/` de GDAL, y empaqueta su primera banda, en orden de fila mayor, en un único valor `raquet` autodescriptivo georreferenciado por la celda `quadbin`. El tipo de datos de la banda debe ser uno de `Byte`, `Int16`, `Int32`, `Float32` o `Float64`; el valor de nodata de la banda, cuando está definido, se traslada a la tesela. Como el archivo se suministra como bytes, no se requiere acceso a archivos del lado del servidor. GDAL realiza la decodificación, por lo que el controlador de formato del ráster debe estar permitido por la configuración `postgis.gdal_enabled_drivers` de PostGIS (que deshabilita todos los controladores de forma predeterminada); habilítelo con, por ejemplo, `SET postgis.gdal_enable = 'ENABLE_ALL'`. Esta es la contraparte de ingesta del constructor raquet, que construye una tesela a partir de bytes de píxeles ya decodificados.

  Cuando se omite `quadbin` o es `NULL`, el identificador de la tesela se deriva de la georreferenciación del ráster: el ráster debe llevar una referencia espacial `EPSG:3857` (Web-Mercator) y una geotransformación alineada con los ejes que describa una única tesela QUADBIN. Un ráster sin referencia espacial, uno que no esté en `EPSG:3857`, o una geotransformación rotada o no alineada con la cuadrícula de teselas provoca un error en lugar de ser forzado.

```
-- Decodificar un GeoTIFF almacenado en una columna bytea en una tesela raquet
SELECT raquet_read(file_bytes, quadbin) AS tile
FROM elevation_files;
```

```
-- Derivar la celda QUADBIN de la georreferenciación de un GeoTIFF EPSG:3857
SELECT raquet_read(file_bytes) AS tile
FROM elevation_files;
```

- Muestrear una tesela ráster `raquet` a lo largo de una trayectoria

  `raster_tile_value(raquet,tgeompoint) → tfloat`

  Evalúa la tesela en cada instante de `traj` (SRID 4326), devolviendo los valores de píxel muestreados como un `tfloat`. Es el equivalente tipado de raster_tile_value_quadbin: las dimensiones, la celda QUADBIN, el tipo de píxel y el tratamiento de nodata se toman del valor `raquet` en lugar de argumentos separados. Devuelve `NULL` cuando ningún instante cae dentro de la tesela o sobrevive al filtrado de nodata.

```
-- Muestrear teselas raquet preempaquetadas a lo largo de trayectorias de barcos
SELECT t.tripid, raster_tile_value(r.tile, t.trip4326)
FROM trips t
JOIN elevation_tiles r
  ON r.quadbin = ANY(trajectory_quadbins(t.trip4326, 8));
```

- Muestrear un arreglo de teselas ráster `raquet` a lo largo de una trayectoria

  `raster_tile_value(raquet[],tgeompoint) → tfloat`

  Evalúa cada tesela de `rast` en cada instante de `traj` (SRID 4326) y devuelve los valores de píxel muestreados como un único `tfloat`. Una trayectoria que sale de una tesela queda cubierta por varias, y cada una aporta los instantes que caen dentro de ella. Las teselas de un mismo nivel de zoom particionan el plano, de modo que aportan instantes disjuntos. Las teselas de niveles de zoom distintos se solapan, y cuando dos de ellas muestrean el mismo instante se conserva el valor de la tesela de mayor zoom, que es la que tiene la resolución más fina. Devuelve `NULL` cuando ningún instante cae dentro de una tesela o sobrevive al filtrado de nodata.

```
-- Muestrear una trayectoria en todas las teselas que cruza, en un solo valor
SELECT t.tripid, raster_tile_value(array_agg(r.tile), t.trip4326)
FROM trips t
JOIN elevation_tiles r
  ON r.quadbin = ANY(trajectory_quadbins(t.trip4326, 8))
GROUP BY t.tripid, t.trip4326;
```

19.3. Accesores y Comparación de Raquet

Un valor `raquet` es autodescriptivo: los accesores siguientes leen la georreferenciación y la disposición que transporta, de modo que una tesela empaquetada con el constructor raquet o decodificada con raquet_read puede inspeccionarse sin conservar las columnas sueltas a partir de las cuales se construyó.

Las teselas también admiten el conjunto completo de operadores de comparación. El orden es total y se define primero por la celda QUADBIN, luego por el tipo de píxel, el ancho, el alto y finalmente los bytes de píxeles; está pensado para indexación y deduplicación, no para una interpretación espacial. Una clase de operadores btree por defecto hace que `ORDER BY`, `DISTINCT` y las uniones por mezcla funcionen sobre columnas `raquet`, y una clase de operadores hash por defecto habilita las uniones y agregaciones hash.

- Devolver la celda QUADBIN de una tesela Raquet

  `quadbin(raquet) → bigint`

```
SELECT quadbin(raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8'));
-- 5193776270265024512
```

- Devolver el ancho en píxeles de una tesela Raquet

  `width(raquet) → integer`

```
SELECT width(raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8'));
-- 2
```

- Devolver el alto en píxeles de una tesela Raquet

  `height(raquet) → integer`

```
SELECT height(raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8'));
-- 2
```

- Devolver el valor centinela nodata de una tesela Raquet

  `nodata(raquet) → float8`

```
SELECT nodata(raquet('\x01020304'::bytea, 2, 2,
  5193776270265024512, 'UINT8', 255));
-- 255
-- The sentinel defaults to 0 when the constructor omits it.
SELECT nodata(raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8'));
-- 0
```

- Devolver el nombre del tipo de dato de píxel de una tesela Raquet

  `pixtype(raquet) → text`

  El nombre devuelto es el que aceptan los constructores, es decir, uno de `UINT8`, `INT16`, `INT32`, `FLOAT32` o `FLOAT64`.

```
-- Leer la disposición de una tesela empaquetada
SELECT quadbin(tile), width(tile), height(tile), pixtype(tile), nodata(tile)
FROM (SELECT raquet('\x01020304'::bytea, 2, 2, 5193776270265024512, 'UINT8') AS tile) t;
-- 5193776270265024512 | 2 | 2 | UINT8 | 0
```

- Convertir una tesela Raquet en una caja espaciotemporal

  `stbox(raquet) → stbox`

  Devuelve la huella de la tesela: la envolvente de longitud y latitud de su celda QUADBIN, en SRID 4326 y sin dimensión temporal. La huella se deriva únicamente de la celda, por lo que teselas que difieren en píxeles, dimensiones o tipo de píxel la comparten. La conversión también está disponible como conversión de tipo.

```
-- The footprint of a tile covering the north-eastern quadrant at zoom 1
SELECT round(stbox(raquet('\x01020304'::bytea, 2, 2, 5193776270265024512,
  'UINT8')), 6);
-- SRID=4326;STBOX X((0,0),(180,85.051129))

-- Keep the tiles whose footprint overlaps a region of interest
SELECT * FROM elevation_tiles
WHERE tile::stbox && stbox 'SRID=4326;STBOX X((0,0),(30,30))';
```

  La huella admite los operadores, agregados y clases de operadores de `stbox`, por lo que una tabla de teselas se consulta por solapamiento espacial en cualquier nivel de zoom en lugar de por una prueba de igualdad sobre la celda QUADBIN en uno fijo. Indexar la conversión dota a la tabla de teselas de un índice espacial, y la extensión de un conjunto de teselas es la extensión de sus huellas.

```
-- A spatial index over the tile footprints
CREATE INDEX elevation_tiles_gist ON elevation_tiles USING gist (stbox(tile));
CREATE INDEX elevation_tiles_spgist ON elevation_tiles USING spgist (stbox(tile));

-- The extent covered by a set of tiles
SELECT extent(stbox(tile)) FROM elevation_tiles;

-- Tiles a trajectory passes through, at whatever zoom levels the table holds
SELECT t.tripid, r.tile
FROM trips t, elevation_tiles r
WHERE stbox(r.tile) && stbox(t.trip4326);
```

- Comparar dos teselas Raquet

  `raquet = raquet → boolean`

  `raquet <> raquet → boolean`

  `raquet < raquet → boolean`

  `raquet <= raquet → boolean`

  `raquet >= raquet → boolean`

  `raquet > raquet → boolean`

  Las funciones que respaldan estos operadores son `eq`, `ne`, `lt`, `le`, `ge` y `gt`, junto con `cmp(raquet,raquet) → integer`, que devuelve -1, 0 o 1.

```
-- Dos teselas con la misma celda, disposición y píxeles son iguales
SELECT raquet('\x0102'::bytea, 2, 1, 5193776270265024512, 'UINT8') =
       raquet('\x0102'::bytea, 2, 1, 5193776270265024512, 'UINT8');
-- true

-- Deduplicar teselas, lo que permiten las clases de operadores btree y hash
SELECT DISTINCT tile FROM elevation_tiles;
```

19.4. Operadores de Restricción y Predicado

Los siguientes operadores definidos en SQL componen `raster_value` con las funciones estándar de restricción temporal y predicado de MobilityDB. Aceptan un argumento opcional `band` (por defecto 1) e invocan `raster_value` exactamente una vez por invocación.

- Devolver los instantes de una trayectoria donde el valor ráster muestreado cae dentro de un rango de flotante

  `atRasterValue(tgeompoint,raster,floatspan,band integer DEFAULT 1) → tgeompoint`

  Evalúa `raster_value` en cada instante de `traj` y devuelve la sub-trayectoria restringida a los tiempos en que el valor del píxel está dentro de `vspan`. Devuelve `NULL` cuando ningún instante satisface la condición.

```
WITH rast AS (
  SELECT ST_SetValues(
    ST_AddBand(
      ST_MakeEmptyRaster(3, 3, 0.0, 3.0, 1.0, -1.0, 0.0, 0.0, 4326),
      '32BF'::text, 0.0::float8, NULL::float8),
    1, 1, 1,
    ARRAY[[10.0::float4, 20.0::float4, 30.0::float4],
          [40.0::float4, 50.0::float4, 60.0::float4],
          [70.0::float4, 80.0::float4, 90.0::float4]]) AS r
)
SELECT asText(atRasterValue(
  tgeompoint 'SRID=4326;[POINT(0.5 2.5)@2000-01-01,
                         POINT(1.5 1.5)@2000-01-02,
                         POINT(0.5 0.5)@2000-01-03]',
  r, floatspan '[40, 90]'))
FROM rast;
```

- Devolver los instantes de una trayectoria donde el valor ráster muestreado cae fuera de un rango de flotante

  `minusRasterValue(tgeompoint,raster,floatspan,band integer DEFAULT 1) → tgeompoint`

  El complemento de `atRasterValue`: devuelve la sub-trayectoria restringida a los tiempos en que el valor del píxel está fuera de `vspan`. Devuelve `NULL` cuando todos los instantes satisfacen la condición.

```
-- Keep only the instant whose raster value (10) is below 40
SELECT asText(minusRasterValue(traj, elev_raster, floatspan '[40, 9999]'))
FROM trips, elevation_raster;
```

- Devolver verdadero si la trayectoria alguna vez muestrea un valor de píxel ráster dentro de un rango de flotante

  `eRasterValue(raster,tgeompoint,floatspan,band integer DEFAULT 1) → boolean`

  Devuelve `true` cuando al menos un instante dentro de la extensión de `traj` muestrea un valor de píxel dentro de `vspan`, `false` en caso contrario. Devuelve `NULL` únicamente cuando la entrada es `NULL`.

```
-- Trips that ever crossed terrain above 200 m
SELECT tripid
FROM trips, elevation_raster
WHERE eRasterValue(elev_raster, trip4326, floatspan '[200, 9999]');
```

- Devolver verdadero si cada instante de la trayectoria dentro de la extensión del ráster muestrea un valor de píxel dentro de un rango de flotante

  `aRasterValue(raster,tgeompoint,floatspan,band integer DEFAULT 1) → boolean`

  Devuelve `true` cuando cada instante dentro de la extensión de `traj` muestrea un valor de píxel dentro de `vspan`, `false` cuando al menos uno no lo hace. Devuelve `NULL` únicamente cuando la entrada es `NULL`.

```
-- Trips that stayed entirely below 100 m elevation
SELECT tripid
FROM trips, elevation_raster
WHERE aRasterValue(elev_raster, trip4326, floatspan '[0, 100]');
```

19.5. Compilación e Instalación

El ráster de PostGIS forma parte del paquete estándar `postgresql-N-postgis-3` en Debian/Ubuntu; no se requiere ningún paquete adicional. Pase `-DRASTER=ON` a CMake:

```
cmake -DRASTER=ON ..
make -j$(nproc)
sudo make install
```

Para incluir el ráster en la compilación de cobertura de CI junto con otras familias opcionales, añada `-DRASTER=ON` a la invocación de cmake en `.github/workflows/pgversion.yml`.

19.6. Hoja de Ruta de Producción

Las dos vías descritas en este capítulo sirven propósitos distintos y ambas se mantienen. La vía `raquet` es un tipo de valor MEOS autocontenido: transporta sus píxeles y su georreferenciación QUADBIN en un único valor y no necesita PostgreSQL para evaluarse, por lo que está disponible en todos los enlaces de lenguaje derivados de MEOS. La vía `raster` de PostGIS ofrece el procesamiento ráster mucho más rico de la extensión `postgis_raster` — álgebra de mapas, estadísticas, remuestreo, recorte, conversión a polígonos — únicamente dentro de PostgreSQL.

Ninguna vía reemplaza a la otra. El trabajo sobre la vía `raquet` amplía lo que una tesela nativa de la nube puede hacer a lo largo de una trayectoria; no pretende reimplementar el procesamiento ráster.

Las extensiones siguientes requieren un modelo de datos ráster general, es decir, sistemas de referencia espacial arbitrarios, georreferenciación afín arbitraria y varias bandas por tesela. Un valor `raquet` es una tesela de una sola banda cuya extensión y rejilla de píxeles se derivan de su celda QUADBIN, por lo que quedan fuera de ese modelo y siguen disponibles a través de la vía `raster` de PostGIS:

- *Teselas multibanda* — un `tfloat` por banda para imágenes multiespectrales como RGB o escenas satelitales multicanal.
- *Sistemas de referencia espacial arbitrarios* — teselas fuera de la rejilla Web-Mercator y reproyección entre sistemas de referencia.
- *Procesamiento ráster* — álgebra de mapas, estadísticas de resumen, remuestreo y conversión a polígonos, junto con el recorte de una tesela a una región, como la extensión que un conjunto de trayectorias visita durante un intervalo de tiempo.

Capítulo 20

Dialecto Portable de Funciones con Nombre

Todo operador de MobilityDB puede invocarse también como una función con nombre. Una consulta escrita únicamente con funciones con nombre, sin símbolos de operador, es portable: el mismo SQL se ejecuta sin cambios en todo el ecosistema MobilityDB — los motores de base de datos MobilityDB (PostgreSQL), MobilityDuck (DuckDB) y MobilitySpark (Apache Spark), y los motores de *streaming* —, varios de los cuales no pueden registrar símbolos de operador de PostgreSQL. Cada alias con nombre reutiliza la propia implementación del operador, por lo que devuelve exactamente el mismo resultado que el operador.

Este capítulo es la referencia canónica, a nivel de todo el ecosistema, de esa correspondencia. Un motor que no acepte un operador dado puede consultarlo aquí y usar la forma de función en su lugar. Por ejemplo, DuckDB no acepta ningún operador que contenga `#` (reserva `#` para las referencias posicionales de columna), de modo que en MobilityDuck las comparaciones temporales (`#=`, ...) se escriben `tEq`, ... y las pruebas de posición temporal (`<<#`, `#>>`, `&<#`, `#&>`) se escriben `before`, `after`, `overbefore`, `overafter`, mientras que los operadores que sí acepta pueden usarse directamente.

Las funciones de relación espacial (`eIntersects`, `aIntersects`, `eTouches`, `eDwithin`, ...) y las funciones de restricción (`atTime`, `atGeometry`, `atValue`, ...) son siempre funciones con nombre (no tienen operador). El comparador de ordenación de árbol B es asimismo la función con nombre `cmp(a, b)`, que devuelve `-1`, `0` o `1` y no tiene operador. Cada operador se corresponde con un nombre simple de la manera siguiente; para los operadores aritméticos y de teoría de conjuntos el nombre de la función lleva el prefijo de la familia de tipos del operando (`set`, `span` o `spanset`, y `t` para los números temporales).

Categoría	**Operador**	**Función portable**
Topología	&&	overlaps(a, b)
	@>	contains(a, b)
	<@	contained(a, b)
	-|-	adjacent(a, b)
Posición temporal	<<#	before(a, b)
	#>>	after(a, b)
	&<#	overbefore(a, b)
	#&>	overafter(a, b)
Espacio, eje X	<<	left(a, b)
	>>	right(a, b)
	&<	overleft(a, b)
	&>	overright(a, b)
Espacio, eje Y	<<|	below(a, b)
	|>>	above(a, b)
	&<|	overbelow(a, b)
	|&>	overabove(a, b)
Espacio, eje Z	<</	front(a, b)
	/>>	back(a, b)
	&</	overfront(a, b)
	/&>	overback(a, b)

Categoría	Operador	Función portable
Comparación temporal	#=	tEq(a, b)
	#<>	tNe(a, b)
	#<	tLt(a, b)
	#<=	tLe(a, b)
	#>	tGt(a, b)
	#>=	tGe(a, b)
Distancia	<->	tDistance(a, b)
	|=|	nearestApproachDistance(a, b)
Igual	~=	same(a, b)
Comparación tradicional	=	eq(a, b)
	<>	ne(a, b)
	<	lt(a, b)
	<=	le(a, b)
	>	gt(a, b)
	>=	ge(a, b)
Comparación alguna vez	?=	eEq(a, b)
	?<>	eNe(a, b)
	?<	eLt(a, b)
	?<=	eLe(a, b)
	?>	eGt(a, b)
	?>=	eGe(a, b)
Comparación siempre	%=	aEq(a, b)
	%<>	aNe(a, b)
	%<	aLt(a, b)
	%<=	aLe(a, b)
	%>	aGt(a, b)
	%>=	aGe(a, b)
Booleano (booleano temporales)	&	tAnd(a, b)
	|	tOr(a, b)
	~	tNot(a)
Aritmética (números temporales)	+	tAdd(a, b)
	-	tSub(a, b)
	*	tMul(a, b)
	/	tDiv(a, b)
Unión (set / span / span set)	+	setUnion / spanUnion / spansetUnion (a, b)
Intersección (set / span / span set)	*	setIntersection / spanIntersection / spansetIntersection (a, b)
Diferencia (set / span / span set)	-	setMinus / spanMinus / spansetMinus (a, b)

Por ejemplo, la superposición de cajas envolventes `t1.trip && t2.trip` se escribe de forma portable como `overlaps(t1.trip, t2.trip)`, y la prueba temporal «antes» `p1.period <<# p2.period` como `before(p1.period, p2.period)`.

Las funciones de agregación siguen el mismo principio y se enumeran en su propia tabla a continuación, de modo que un motor pueda localizar la maquinaria que necesita sin rastrear el texto. Una agregación SQL se compone de hasta tres funciones de la maquinaria de PostgreSQL, cada una expuesta bajo un nombre canónico formado a partir del nombre de la agregación y su rol: una función de transición `<agregación>Transition` (siempre presente), una función de combinación `<agregación>Combine` usada para la agregación en paralelo, y una función final `<agregación>Final`. Los roles de combinación y final son opcionales: una agregación cuyo estado de transición ya es su resultado no tiene función final (por ejemplo `extent` y `minDistance`), y una agregación que no admite la agregación parcial no tiene función de combinación (por ejemplo las agregaciones de anexado). Los roles se mantienen distintos y con un nombre uniforme para que cada *binding* pueda reconstruir la agregación a partir del catálogo. La siguiente tabla enumera cada agregación y sus funciones de maquinaria; un guion (—) indica un rol que la agregación no define.

Categoría	Transición	Combinación	Final
Conteo	`tCountTransition`	`tCountCombine`	`tCountFinal`
	`wCountTransition`	`wCountCombine`	`wCountFinal`

Categoría	Transición	Combinación	Final
Suma	`tSumTransition`	`tSumCombine`	`tSumFinal`
	`wSumTransition`	`wSumCombine`	`wSumFinal`
Promedio	`tAvgTransition`	`tAvgCombine`	`tAvgFinal`
	`wAvgTransition`	`wAvgCombine`	`wAvgFinal`
Mínimo / máximo	`tMinTransition`	`tMinCombine`	`tMinFinal`
	`tMaxTransition`	`tMaxCombine`	`tMaxFinal`
	`wMinTransition`	`wMinCombine`	`wMinFinal`
	`wMaxTransition`	`wMaxCombine`	`wMaxFinal`
Booleano	`tAndTransition`	`tAndCombine`	`tAndFinal`
	`tOrTransition`	`tOrCombine`	`tOrFinal`
Centroide	`tCentroidTransition`	`tCentroidCombine`	`tCentroidFinal`
Nube de puntos	`tnpointsTransition`	`tnpointsCombine`	`tnpointsFinal`
	`tdensityTransition`	`tdensityCombine`	`tdensityFinal`
Caja delimitadora	`extentTransition`	`extentCombine`	—
Distancia	`minDistanceTransition`	`minDistanceCombine`	—
Unión	`setUnionTransition`	`setUnionCombine`	`setUnionFinal`
	`spanUnionTransition`	`spanUnionCombine`	`spanUnionFinal`
	`spansetUnionTransition`	`spansetUnionCombine`	`spansetUnionFinal`
Fusión	`mergeTransition`	`mergeCombine`	`mergeFinal`
Anexado	`appendInstantTransition`	—	`appendInstantFinal`
	`appendSequenceTransition`	—	`appendSequenceFinal`

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	tbigint
float	floatset	floatspan	floatspanset	tfloat
date	dateset	datespan	datespanset	
timestamptz	tstzset	tstzspan	tstzspanset	
geometry	geomset			tgeompoint
geography	geogset			tgeogpoint, tgeography
pose	poseset			tpose
npoint	npointset			tnpoint
cbuffer	cbufferset			tcbuffer
h3index	h3indexset			th3index
quadbin	quadbinset			tquadbin
pcpoint	pcpointset			tpcpoint
pcpatch	pcpatchset			tpcpatch

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: Devuelve la representación textual conocida (Well-Known Text o WKT)
- asBinary, asHexWKB: Devuelve la representación binaria conocida (Well-Known Binary o WKB) o la representación hexadecimal binaria conocida (HexWKB)

- settypeFromBinary, spantypeFromBinary, spansettypeFromBinary, settypeFromHexWKB, spantypeFromHexWKB, spansettypeFromHexWKB: Entrar a partir de la representación binaria conocida (WKB) o 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: Devuelve el tamaño de la memoria en bytes
- lower, upper: Devuelve el límite inferior o superior
- lowerInc, upperInc: ¿Es el límite inferior o superior inclusivo?
- width: Devuelve el ancho del rango como un número de punto flotante
- duration: Devuelve el intervalo de tiempo
- span: Devuelve el lapso de tiempo delimitador ignorando las posibles brechas de tiempo
- numValues, getValues: Devuelve el número de valores o los valores
- startValue, endValue, valueN: Devuelve el valor inicial, final o enésimo
- numSpans, spans: Devuelve el número de rangos o los rangos
- startSpan, endSpan, spanN: Devuelve el rango inicial, final o enésimo
- numDates, dates: Devuelve el número de fechas o las fechas diferentes
- startDate, endDate, dateN: Devuelve la fecha inicial, final o enésima
- numTimestamps, timestamps: Devuelve el número de marcas de tiempo o las marcas de tiempo diferentes
- startTimestamp, endTimestamp, timestampN: Devuelve la marca de tiempo inicial, final o enésima

A.2.5. Transformaciones

- expand: Extender o reducir los límites con un valor o intervalo de tiempo
- shift: Desplazar con un valor o intervalo de tiempo
- scale: Escalar con un valor o intervalo de tiempo
- shiftScale: Desplazar y escalar con los valores o intervalos de tiempo
- floor, ceil: Redondear al entero inferior o superior
- round: Redondear a un número de decimales
- degrees, radians: Convertir a grados o radianes
- lower, upper, initcap: Transformar en minúsculas, majúsculas o initcap
- ||: Concatenación de texto
- tprecision: Establecer la precisión temporal al intervalo con respecto al origen

A.2.6. Sistema de referencia espacial

- SRID, setSRID: Devuelve o especifica el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.2.7. Operaciones de conjuntos

- +, -, *: Unión, diferencia e intersección de conjuntos o de rangos

A.2.8. Operaciones de cuadro delimitador

A.2.8.1. 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.2.8.2. 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.2.8.3. Operaciones de división

- splitNSpans: Devuelve una matriz de N rangos obtenida fusionando los elementos de un conjunto o los rangos de un conjunto de rangos
- splitEachNSpans: Devuelve una matriz de rangos obtenida fusionando N elementos consecutivos de un conjunto o N rangos consecutivos de un conjunto de rangos

A.2.9. Operaciones de distancia

- <->: Devuelve la distancia mínima

A.2.10. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

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: Devuelve la representación textual conocida (Well-Known Text o WKT)
- asBinary, asHexWKB: Devuelve la representación binaria conocida (Well-Known Binary o WKB) o la representación hexadecimal binaria conocida (HexWKB)
- boxFromBinary, boxFromHexWKB: Entrar a partir de la representación binaria conocida (WKB) o a partir de la representación hexadecimal binaria conocida (HexWKB)

A.3.2. Constructores

- tbox: Constructor para `tbox`
- stboxX, stboxZ, stboxT, stboxXT, stboxZT, 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: Devuelve el valor mínimo de X/Y/Z/T
- xMax, yMax, zMax, tMax: Devuelve 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?
- area, volume, perimeter: Área, volumen, perímetro

A.3.5. Transformaciones

- shiftValue, scaleValue, shiftScaleValue: Desplazar y/o escalear el rango de valores de un cuadro delimitador con uno o dos valores
- shiftTime, scaleTime, shiftScaleTime: Desplazar y/o escalear el período de un cuadro delimitador con uno o dos intervalo
- getSpace: Devuelve la dimensión espacial de un cuadro delimitador, eliminando la dimensión temporal, si existe
- expandValue, expandSpace, expandTime: Extender la dimensión numérica, espacial, o temporal de un cuadro delimitador con un valor o un intervalo
- 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, setSRID: Devuelve o especifica el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.3.7. Operaciones de división

- quadSplit: Dividir un cuadro delimitador en cuadrantes u octantes

A.3.8. Operaciones de conjuntos

- +, *: Union e intersección de dos 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

- =, <>, <, >, <=, >=: Comparaciones tradicionales

A.3.11. Agregaciones

- extent: Extensión del cuadro delimitador

A.4. Tipos temporales

A.4.1. Entrada y salida

- asText: Devuelve la representación Well-Known Text (WKT)
- asMFJSON: Devuelve la representación Moving Features JSON (MF-JSON)
- asBinary, asHexWKB: Devuelve la representación Well-Known Binary (WKB) o la representación Hexadecimal Well-Known Binary (HexWKB)
- ttypeFromMFJSON: Entrar a partir de la representación Moving Features JSON (MF-JSON)
- ttypeFromBinary, ttypeFromHexWKB: Entrar a partir de la representación Well-Known Binary (WKB) o a partir de la representación Hexadecimal Well-Known Binary (HexWKB)

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: Convertir un valor temporal a un rango de marcas de tiempo

A.4.4. Accesores

- memSize: Devuelve el tamaño de la memoria en bytes
- tempSubtype: Devuelve el subtipo temporal
  tempBasetype: Devuelve el tipo base
- interp: Devuelve la interpolación
- getValue, getTimestamp: Devuelve el valor o el tiempo del instante
- getValues, getTime: Devuelve los valores o el tiempo en el que se define el valor temporal
- timeSpan: Devuelve el período delimitador en el que se define el valor temporal
- startValue, endValue, valueN: Devuelve el valor inicial, final o enésimo
- valueAtTimestamp: Devuelve el valor en una marca de tiempo
- duration: Devuelve el intervalo de tiempo

- lowerInc, upperInc: ¿Es el instante inicial/final inclusivo?
- numInstants, instants: Devuelve el número o los instantes diferentes
- startInstant, endInstant, instantN: Devuelve el instante inicial, final o enésimo
- numTimestamps, timestamps: Devuelve el número o las marcas de tiempo diferentes
- startTimestamp, endTimestamp, timestampN: Devuelve la marca de tiempo inicial, final o enésima
- numSequences, sequences: Devuelve el número o las secuencias
- startSequence, endSequence, sequenceN: Devuelve la secuencia inicial, final o enésima
- segments: Devuelve los segmentos

A.4.5. Transformaciones

- ttypeInst, ttypeSeq, ttypeSeqSet: Transformar un valor temporal en otro subtipo
- setInterp: Transformar un valor temporal a otra interpolación
- shiftTime, scaleTime, shiftScaleTime: Desplazar y/o escalear el intervalo de tiempo del valor temporal con uno o dos intervalos
- 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, appendInstantAgg: Anexar un instante temporal a un valor temporal
- appendSequence, appendSequenceAgg: Anexar una secuencia temporal a un valor temporal
- merge, mergeAgg: Fusionar los valores temporales

A.4.7. Restricciones

- atValue, minusValue, atValues, minusValues: Restringir a (al complemento de) un valor o un conjunto de valores
- atTime, minusTime: Restringir a (al complemento de) un valor de tiempo

A.4.8. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- ?=, %=, ?<>, %<>, ?<, %<, ?>, %>, ?<=, %<=, ?>=, %>=: Comparaciones alguna vez y siempre
- #=, #<>, #<, #>, #<=, #>=: Comparaciones temporales

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 alfanuméricos

A.5.1. Conversión de tipos

- tnumber::{span,tbox}, valueSpan(tnumber), timeSpan(tnumber), tbox(tnumber): Convertir un número temporal en un rango o un cuadro delimitador temporal
- tbool::tint, tint(tbool): Convertir entre un booleano temporal y un entero temporal
- tint::tfloat, tfloat::tint, tfloat(tint), tint(tfloat): Convertir entre un entero temporal y un flotante temporal

A.5.2. Accessores

- valueSet: Devuelve el conjunto de valores de un número temporal
- minValue, maxValue, avgValue: Devuelve el valor mínimo, máximo, o promedio
- minInstant, maxInstant: Devuelve el instante con el valor mínimo o máximo

A.5.3. Transformaciones

- shiftValue, scaleValue, shiftScaleValue: Desplazar y/o escalear el rango de valores de un número temporal con uno o dos valores
- stops: Extraer de un flotante temporal con interpolación lineal las subsecuencias donde los valores permanecen dentro de un rango con un ancho dado durante al menos una duración dada

A.5.4. Restricciones

- atMin, atMax, minusMin, minusMax: Restringir al (complemento del) valor mínimo o máximo
- atTbox, minusTbox: Restringir a (al complemento de) un `tbox`

A.5.5. Operaciones booleanas

- &, |: Y temporal, o temporal
- ~: No temporal
- whenTrue: Devuelve el tiempo cuando un booleano temporal toma el valor verdadero

A.5.6. Operaciones matemáticas

- +, -, *, /: Adición, resta, multiplicación y división temporal
- abs: Devuelve el valor absoluto de un número temporal
- floor, ceil: Redondear al entero inferior o superior
- round: Redondear a un número de posiciones decimales
- degrees, radians: Convertir a grados o radianes
- deltaValue: Devuelve la diferencia de valor entre instantes consectivos de un número temporal
- trend: Devuelve la tendencia de un flotante temporal con interpolación lineal, que indica si su valor es creciente, constante o decreciente, representado, respectivamente, por 1, 0 y -1.

- derivative: Devuelve la derivada sobre el tiempo de un número flotante temporal en unidades por segundo
- integral: Devuelve el area baja la curva
- twAvg: Devuelve el promedio ponderado en el tiempo
- ln, log10: Devuelve el logaritmo natural y el logaritmo base 10 de un número flotante temporal
- exp: Devuelve el exponencial (e elevado a la potencia dada) de un número flotante temporal

A.5.7. Operaciones de texto

- ||: Concatenación de texto
- lower, upper, initcap: Transformar en minúsculas, mayúsculas o initcap

A.6. Tipos temporales geométricos

A.6.1. Entrada y salida

- asText, asEWKT: Devuelve la representación de texto conocido (Well-Known Text o WKT) o la representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- asMFJSON: Devuelve la representación JSON de características móviles (Moving Features JSON o MF-JSON)
- asBinary, asEWKB, asHexEWKB: Devuelve la representación binaria conocida (Well-Known Binary o WKB), la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB) en formato texto
- tspatialFromText, tspatialFromEWKT: Entrar a partir de la representación de texto conocido (Well-Known Text o WKT) o de la representación extendida de texto conocido (Extended Well-Known Text o EWKT)
- tspatialFromMFJSON: Entrar a partir de la representación JSON de características móviles (Moving Features JSON o MF-JSON)
- tspatialFromBinary, tspatialFromEWKB, tspatialFromHexEWKB: Entrar a partir de la representación binaria conocida (Well-Known Binary o WKB), de la representación extendida binaria conocida (Extended Well-Known Binary o EWKB), o de la representación hexadecimal extendida binaria conocida (Hexadecimal Extended Well-Known Binary o EWKB)

A.6.2. Conversión de tipos

- ::, stbox(tspatial): Convertir un valor espaciotemporal a un cuadro delimitador espaciotemporal
- tgeometry::tgeography, tgeography::tgeometry, tgeompoint::tgeogpoint, tgeogpoint::tgeompoint: Convertir entre una geometría temporal y una geografía temporal
- tgeompoint::geometry, tgeogpoint::geography, geometry::tgeompoint, geography::tgeogpoint: Convertir entre un punto temporal y una trayectoria PostGIS

A.6.3. Sistema de referencia espacial

- SRID, setSRID: Devuelve o especifica el identificador de referencia espacial
- transform, transformPipeline: Transformar a una referencia espacial diferente

A.6.4. Accesores

- trajectory, traversedArea: Devuelve la trayectoria or el área atravesada
- twCentroid: Devuelve el centroide ponderado en el tiempo
- getX, getY, getZ: Devuelve los valores de las coordenadas X/Y/Z como un número flotante temporal
- isSimple: Devuelve verdadero si el punto temporal no se auto-intersecta espacialmente
- length: Devuelve la longitud atravesada por el punto temporal
- cumulativeLength: Devuelve la longitud acumulada atravesada por el punto temporal
- speed: Devuelve la velocidad del punto temporal en unidades por segundo
- direction: Devuelve la dirección
- azimuth: Devuelve el acimut temporal
- angularDifference: Devuelve la diferencia angular temporal
- bearing: Devuelve el rumbo temporal

A.6.5. Transformaciones

- round: Redondear los valores de las coordenadas a un número de decimales
- makeSimple: Devuelve 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
- affine: Devuelve la transformación afín 3D de un punto temporal para hacer cosas como trasladar, rotar y escalar en un solo paso
- rotate: Devuelve un punto temporal rotado en sentido antihorario alrededor del punto de origen
- scale: Devuelve un punto temporal escalado por factores dados
- asMVTGeom: Transformar un punto geométrico temporal en el espacio de coordenadas de un Mapbox Vector Tile
- stops: Extraer de un punto temporal con interpolación lineal las subsecuencias donde el punto permanece dentro de un área con un tamaño máximo especificado durante al menos la duración dada

A.6.6. Restricciones

- atGeometry, minusGeometry: Restringir a (al complemento de) una geometría
- atStbox, minusStbox: Restringir a (al complemento de) un `stbox`
- atElevation, minusElevation: Restringir a (al complemento de) un rango de altitud

A.6.7. Operaciones de distancia

- |=|: Devuelve la distancia mínima que haya existido
- minDistance: Devuelve la distancia espacial mínima entre dos geometrías temporales sobre la unión de sus extensiones temporales
- nearestApproachInstant: Devuelve el instante del primer punto temporal en el que los dos argumentos están a la distancia más cercana
- shortestLine: Devuelve la línea que conecta el punto de aproximación más cercano
- <->: Devuelve la distancia temporal

A.6.8. Relaciones espaciales

A.6.8.1. Relaciones 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.6.8.2. Relaciones espaciotemporales

- 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, frechetDistance, dynTimeWarpDistance, averageHausdorffDistance, lcssDistance: Devuelve la distancia de Hausdorff discreta, la distancia de Fréchet discreta, la distancia de distorsión de tiempo dinámica (Dynamic Time Warp), la distancia de Hausdorff promedio o la distancia de subsecuencia común más larga (Longest Common SubSequence o LCSS) entre dos valores temporales
- frechetDistancePath, dynTimeWarpPath: Devuelve las parejas de correspondencia entre dos valores temporales con respecto a la distancia de Fréchet discreta o a la distancia de distorsión de tiempo dinámica (Dynamic Time Warp)

A.7.4. Operaciones de división de cuadro delimitador

- splitNSpans: Devuelve una matriz de N rangos de tiempo obtenida fusionando los instantes o segmentos de un valor temporal
- splitEachNSpans: Devuelve una matriz de rangos de tiempo obtenida fusionando N instantes o segmentos consecutivos de un valor temporal
- splitNTboxes: Devuelve una matriz de N cuadros temporales obtenida fusionando los instantes o segmentos de un número temporal
- splitEachNTboxes: Devuelve una matriz de cuadros temporales obtenida fusionando N instantes o segmentos consecutivos de un número temporal
- splitNStboxes: Devuelve ya sea una matriz de N cuadros espaciales obtenida fusionando los segmentos de una (multi)línea o una matriz de N cuadros espaciotemporales obtenida fusionando los instantes o segmentos de un punto temporal
- splitEachNStboxes: Devuelve ya sea una matriz de cuadros espaciales obtenida fusionando N segmentos consecutivos de una (multi)línea o una matriz de cuadros espaciotemporales obtenida fusionando N instantes o segmentos consecutivos de un punto temporal

A.7.5. Mosaicos multidimensionales

A.7.5.1. Operaciones de intervalos

- bins: Devuelve un conjunto de intervalos que cubre un rango de valores o de tiempo con intervalos de la misma amplitud or duración
- getBin: Devuelve el intervalo que contiene un número o una marca de tiempo

A.7.5.2. Operaciones de mosaico

- valueTiles, timeTiles, valueTimeTiles: Devuelve el conjunto de mosaicos que cubre un cuadro delimitador temporal con mosaicos del mismo tamaño y/o duración
- spaceTiles, timeTiles, spaceTimeTiles: Devuelve el conjunto de mosaicos que cubre un cuadro delimitador espaciotemporal con mosaicos del mismo tamaño y/o duración
- getValueTile, getTboxTimeTile, getValueTimeTile: Devuelve el mosaico temporal que cubre un valor y/o una marca de tiempo
- getSpaceTile, getStboxTimeTile, getSpaceTimeTile: Devuelve el mosaico espaciotemporal que cubre un punto y/o una marca de tiempo

A.7.5.3. Operaciones de cuadro delimitador

- valueBins, timeBins: Devuelve una matriz de intervalos de valores or de tiempo obtenidos a partir de los instantes o segmentos de un número temporal con respecto a un mosaico de valores o de tiempo
- valueBoxes, timeBoxes, valueTimeBoxes: Devuelve una matriz de cuadros temporales obtenidos a partir de los instantes o segmentos de un número temporal con respecto a un mosaico de valores y/o tiempo
- spaceBoxes, timeBoxes, spaceTimeBoxes: Devuelve una matriz de cuadros espaciotemporales obtenidos a partir de los instantes o segmentos de un punto temporal con respecto a un mosaico espacial y/o temporal

A.7.5.4. Operaciones de división

- valueSplit, timeSplit, valueTimeSplit: Fragmentar un número temporal con respecto a intervalos de valores y/o de tiempo
- spaceSplit, spaceTimeSplit: Fragmentar un punto temporal con respecto a los mosaicos de una malla espacial o espaciotemporal

A.7.6. Agregaciones

- tCount: Conteo temporal
- extent: Extensión del cuadro delimitador
- tAnd, tOr: Y, o temporal
- tMin, tMinAgg, tMax, tMaxAgg, tSum, tAvg: Mínimo, máximo, suma y promedio temporal
- wCount: Conteo de ventana
- wMin, wMax, wSum, wAvg: Mínimo, máximo, suma y promedio de ventana
- tCentroid: Centroide temporal

A.8. Poses temporales

A.8.1. Poses estáticas

A.8.1.1. Entrada y salida

- asText, asEWKT: Devuelve la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- asBinary, asEWKB, asHexEWKB: Devuelve la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB)
- poseFromText, poseFromEWKT: Entrada desde la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- poseFromBinary, poseFromEWKB, poseFromHexEWKB: Entrada desde la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB)

A.8.1.2. Constructores

- pose: Constructores para poses

A.8.1.3. Conversiones de tipo

- pose::stbox, stbox(pose): Convertir una pose y, opcionalmente, una marca de tiempo o un período, en un cuadro espaciotemporal
- pose::geometry: Convertir una pose en un punto de geometría

A.8.1.4. Accesores

- point: Devuelve el punto
- rotation: Devuelve la rotación
- orientation: Devuelve la orientación

A.8.1.5. Transformaciones

- round: Redondea el punto y la orientación de la pose al número de decimales

A.8.1.6. Sistema de referencia espacial

- SRID, setSRID: Devuelve o establece el identificador de referencia espacial
- transform, transformPipeline: Transforma a un identificador de referencia espacial
- same: Similitud espacial

A.8.1.7. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

A.8.2. Poses temporales

A.8.2.1. Entrada y salida

- asText, asEWKT: Devuelve la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- asMFJSON: Devuelve la representación en JSON de características móviles (MF-JSON)
- asBinary, asEWKB, asHexEWKB: Devuelve la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB)
- tposeFromText, tposeFromEWKT: Entrada desde la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- tposeFromMFJSON: Entrada desde la representación en JSON de características móviles (MF-JSON)
- tposeFromBinary, tposeFromEWKB , tposeFromHexEWKB: Entrada desde la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB)

A.8.2.2. Constructores

- tpose: Constructor para poses temporales a partir de un valor base y un valor de tiempo
- tposeSeq: Constructores para poses temporales del subtipo secuencia
- tposeSeqSet, tposeSeqSetGaps: Constructor para poses temporales del subtipo conjunto de secuencias
- tpose: Construye una pose temporal 2D a partir de un punto de geometría temporal y un flotante temporal

A.8.2.3. Conversiones de tipo

- tpose::tgeompoint: Convertir una pose temporal en un punto de geometría temporal

A.8.2.4. Accesores

- getValues: Devuelve los valores
- points: Devuelve los puntos
- rotation: Devuelve la rotación
- valueAtTimestamp: Devuelve el valor en una marca de tiempo

A.8.2.5. Transformaciones

- tposeInst, tposeSeq , tposeSeqSet: Transforma a otro subtipo
- setInterp: Transforma a otra interpolación
- round: Redondea los puntos y las orientaciones de la pose temporal al número de decimales

A.8.2.6. Modificaciones

- appendInstant, appendInstantAgg, appendSequence, appendSequenceAgg: Añade un instante temporal o una secuencia temporal
- merge, mergeAgg: Combina las poses temporales

A.8.2.7. Restricciones

- atValue, minusValue, atValues, minusValues: Restringe a (el complemento de) un valor o un conjunto de valores
- atGeometry, minusGeometry: Restringe a (el complemento de) una geometría

A.8.2.8. Sistema de referencia espacial

- SRID, setSRID: Devuelve o establece el identificador de referencia espacial
- transform, transformPipeline: Transforma a un identificador de referencia espacial

A.8.2.9. Operaciones de cuadro delimitador

- &&, <@, @>, ~=, -|-: Operadores topológicos
- <<, &<, >>, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, |&>: Operadores de posición

A.8.2.10. Operadores de distancia

- |=|: Devuelve la menor distancia en cualquier momento
- nearestApproachInstant: Devuelve el instante de la primera pose temporal en el que los dos argumentos están a la distancia más cercana
- shortestLine: Devuelve la línea que conecta el punto de aproximación más cercano
- <->: Devuelve la distancia temporal

A.8.2.11. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- ?=, &=: Comparaciones siempre y en algún momento
- #=, #<>: Comparaciones temporales

A.8.2.12. Agregaciones

- tCount: Cuenta temporal
- wCount: Cuenta de ventana

A.9. Operaciones para puntos de red temporales

A.9.1. Tipos de red estáticos

A.9.1.1. Constructores

- npoint, nsegment: Constructores para puntos y segmentos de red

A.9.1.2. Conversiones de tipo

- npoint::geometry, nsegment::geometry, geometry::npoint, geometry::nsegment: Convertir entre un punto o un segmento de red y una geometría

A.9.1.3. Accesores

- route: Devuelve el identificador de ruta
- getPosition: Devuelve la fracción de posición
- startPosition, endPosition: Devuelve la posición inicial o final

A.9.1.4. Transformaciones

- round: Redondear la(s) posición(es) del punto de red or el segmento de red en el número de posiciones decimales

A.9.1.5. Operaciones espaciales

- SRID: Devuelve el identificador de referencia espacial
- equals: Igualdad espacial para puntos de red

A.9.1.6. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

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, tnpointSeqSetGaps: Constructor para puntos de red temporal de subtipo conjunto de secuencias

A.9.2.2. Conversión de tipos

- tnpoint::tgeompoint, tgeompoint::tnpoint: Convertir entre un punto de red temporal en un punto de geometría temporal

A.9.2.3. Accesores

- getValues: Devuelve los valores
- routes: Devuelve los identificadores de ruta
- valueAtTimestamp: Devuelve el valor en una marca de tiempo
- length: Devuelve la longitud atravesada por el punto de red temporal
- cumulativeLength: Devuelve la longitud acumulada atravesada por el punto de red temporal
- speed: Devuelve la velocidad del punto de red temporal en unidades por segundo

A.9.2.4. Transformaciones

- 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

A.9.2.5. Modifications

- appendInstant, appendSequence: Agregar un instante temporal o una secuencia temporal
- merge, mergeAgg: Fusionar los puntos de red temporales

A.9.2.6. Restricciones

- atValue, minusValue, atValues, minusValues: Restringir a (al complemento de) un valor o un conjunto de valores
- atGeometry, minusGeometry: Restringir a (al complemento de) una geometría

A.9.2.7. Operadores de distancia

- |=|: Devuelve la distancia más cercana
- nearestApproachInstant: Devuelve el instante del primer punto de red temporal en el que los dos argumentos están a la distancia más cercana
- shortestLine: Devuelve la línea que conecta el punto de aproximación más cercano entre los dos argumentos
- <->: Devuelve la distancia temporal

A.9.2.8. Operaciones espaciales

- twCentroid: Devuelve el centroide ponderado en el tiempo

A.9.2.9. Operaciones de cuadro delimitador

- &&, <@, @>, ~=, -|-: Operadores topológicos
- <<, &<, >>, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, |&>: Operadores de posición relativa

A.9.2.10. Relaciones espaciales

- 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.9.2.11. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- ?=, &=: Comparaciones siempre y alguna vez
- #=, #<>: Comparaciones temporales

A.9.2.12. Agregaciones

- tCount: Conteo temporal
- wCount: Conteo de ventana
- tCentroid: Centroide temporal

A.10. Búferes circulares temporales

A.10.1. Búferes circulares estáticos

A.10.1.1. Constructores

- cbuffer: Constructor para búferes circulares

A.10.1.2. Conversiones de tipo

- cbuffer::geometry, geometry::cbuffer: Convertir entre un búfer circular y un punto de geometría
- stbox: Convertir un búfer circular y, opcionalmente, una marca de tiempo o un período, en un cuadro espaciotemporal

A.10.1.3. Accesores

- point: Devuelve el punto
- radius: Devuelve el radio

A.10.1.4. Transformaciones

- round: Redondea el punto y el radio del búfer circular al número de decimales

A.10.1.5. Operaciones espaciales

- SRID, setSRID: Devuelve o establece el identificador de referencia espacial
- transform, transformPipeline: Transforma a un identificador de referencia espacial

A.10.1.6. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

A.10.2. Búferes circulares temporales

A.10.2.1. Entrada y salida

- asText, asEWKT: Devuelve la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- asMFJSON: Devuelve la representación en JSON de entidades móviles (MF-JSON)
- asBinary, asEWKB: Devuelve la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB) como texto
- tcbufferFromText, tcbufferFromEWKT: Entrada desde la representación en texto conocido (WKT) o en texto conocido extendido (EWKT)
- tcbufferFromMFJSON: Entrada desde la representación en JSON de entidades móviles (MF-JSON)
- tcbufferFromBinary, tcbufferFromEWKB, tcbufferFromHexEWKB: Entrada desde la representación en binario conocido (WKB), en binario conocido extendido (EWKB) o en binario conocido extendido hexadecimal (HexEWKB)

A.10.2.2. Constructores

- tcbuffer: Constructor para búferes circulares temporales con un valor constante
- tcbufferSeq: Constructores para búferes circulares temporales del subtipo secuencia
- tcbufferSeqSet, tcbufferSeqSetGaps: Constructor para búferes circulares temporales del subtipo conjunto de secuencias
- tcbuffer: Construye un búfer circular temporal a partir de un punto de geometría temporal y un flotante temporal

A.10.2.3. Conversiones de tipo

- tcbuffer::tgeompoint: Convertir un búfer circular temporal a un punto de geometría temporal
- tcbuffer::tfloat: Convertir un búfer circular temporal a un flotante temporal

A.10.2.4. Accesores

- getValues: Devuelve los valores
- points, radius: Devuelve los puntos o los radios
- valueAtTimestamp: Devuelve el valor en una marca de tiempo

A.10.2.5. Transformaciones

- tcbufferInst, tcbufferSeq, tcbufferSeqSet: Transforma un búfer circular temporal a otro subtipo
- setInterp: Transforma un búfer circular temporal a otra interpolación
- round: Redondea los puntos y los radios del búfer circular temporal al número de decimales

A.10.2.6. Modificaciones

- appendInstant, appendInstantAgg, appendSequence, appendSequenceAgg: Añade un instante temporal o una secuencia temporal
- merge, mergeAgg: Combina los búferes circulares temporales

A.10.2.7. Restricciones

- atValue, minusValue, atValues, minusValues: Restringe a (el complemento de) un valor o un conjunto de valores
- atGeometry, minusGeometry: Restringe a (el complemento de) una geometría

A.10.2.8. Operaciones de distancia

- |=|: Devuelve la menor distancia que existió en algún momento entre los dos argumentos
- nearestApproachInstant: Devuelve el instante del primer búfer circular temporal en el que los dos argumentos están a la menor distancia
- shortestLine: Devuelve la línea que conecta el punto de aproximación más cercana entre los dos argumentos
- <->: Devuelve la distancia temporal

A.10.2.9. Operaciones espaciales

- SRID, setSRID: Devuelve o establece el identificador de referencia espacial
- transform, transformPipeline: Transforma a un identificador de referencia espacial
- traversedArea: Devuelve el área recorrida por el búfer circular temporal

A.10.2.10. Operaciones de cuadro delimitador

- &&, <@, @>, ~=, -|-: Operadores topológicos
- <<, &<, >>, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, |&>: Operadores de posición

A.10.2.11. Relaciones espaciales

- eContains, eDisjoint, eIntersects, eTouches, eDwithin: Relaciones espaciales posibles
- tContains, tDisjoint, tIntersects, tTouches, tDwithin: Relaciones espaciotemporales

A.10.2.12. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- ?=, &=: Comparaciones siempre y alguna vez
- #=, #<>: Comparaciones temporales

A.10.2.13. Agregaciones

- tCount: Conteo temporal
- wCount: Conteo de ventana
- tCentroid: Centroide temporal

A.11. Geometrías rígidas temporales

A.11.1. Entrada y salida

- asText: Devuelve la representación en formato Well-Known Text (WKT)
- asEWKT: Devuelve la representación en formato Extended Well-Known Text (EWKT), precedida por el SRID
- asMFJSON: Devuelve la representación en formato Moving Features JSON
- asEWKB, asHexEWKB: Devuelve la representación en formato Extended Well-Known Binary (EWKB), o su texto codificado en hexadecimal
- trgeometryFromText, trgeometryFromEWKT, trgeometryFromMFJSON, trgeometryFromEWKB, trgeometryFromHexEWKB: Analiza una trgeometry a partir de su forma WKT, EWKT, MFJSON, EWKB o hex-EWKB

A.11.2. Constructores

- trgeometry: Construir una geometría rígida temporal a partir de una geometría de referencia y una pose temporal
- appendInstant, appendInstantAgg: Añade un único instante a una trgeometry existente
- appendSequence, appendSequenceAgg: Añade una secuencia completa a una trgeometry existente

A.11.3. Conversiones de tipo

- tpose: Devuelve la pose temporal subyacente
- tgeompoint: Devuelve la trayectoria del centroide (antena) como un punto temporal

A.11.4. Accesores

- startValue, endValue, valueAtTimestamp: Devuelve la geometría materializada en el inicio, el final o un instante de tiempo elegido
- numInstants, numSequences, numTimestamps: Devuelve el número de instantes, secuencias o instantes de tiempo distintos
- instants, sequences, segments: Devuelve el arreglo de instantes, secuencias o segmentos entre instantes constituyentes
- points: Devuelve el conjunto de puntos distintos del centroide (antena)
- rotation: Devuelve el ángulo de rotación como un flotante temporal
- geom: Devuelve la geometría de referencia estática

A.11.5. Área recorrida

- traversedArea: Devuelve la unión de los polígonos materializados a lo largo del dominio temporal

A.11.6. Funciones espaciales

- centroid: Devuelve el centroide del cuerpo en movimiento como un punto temporal
- convexHull: Devuelve la envolvente convexa del área recorrida por el cuerpo en movimiento
- bodyPointTrajectory: Devuelve la trayectoria en el marco global de un punto del marco del cuerpo transportado por la geometría

A.11.7. Métricas de movimiento

- length, cumulativeLength: Devuelve la longitud total o acumulada recorrida por el centroide
- speed: Devuelve la velocidad del centroide como un flotante temporal
- twCentroid: Devuelve el centroide ponderado en el tiempo de la trayectoria del centroide como una geometría

A.11.8. Modificaciones

- round: Redondea todos los componentes numéricos a un número dado de decimales
- setInterp: Convierte entre las interpolaciones lineal y escalonada
- shiftTime, scaleTime, shiftScaleTime: Desplaza, escala, o desplaza y escala a la vez el dominio temporal

A.11.9. Restricciones

- atGeometry, minusGeometry: Restringe una trgeometry a (el complemento de) una geometría
- atStbox, minusStbox: Restringe una trgeometry a (el complemento de) un cuadro espaciotemporal

A.11.10. Operadores de distancia

- <->, tDistance: Devuelve la distancia temporal entre dos trgeometries
- |=|, nearestApproachDistance: Devuelve la menor distancia entre dos trgeometries a lo largo de su dominio temporal compartido
- nearestApproachInstant: Devuelve el instante en el que los argumentos están a su menor distancia mutua
- shortestLine: Devuelve la línea que conecta los dos argumentos en su punto de máxima aproximación

A.11.11. Similitud

- hausdorffDistance, frechetDistance, dynTimeWarpDistance: Devuelve la distancia de Hausdorff, de Frechet o de Dynamic Time Warp
- frechetDistancePath, dynTimeWarpPath: Devuelve los pares de correspondencia de Frechet o de Dynamic Time Warp

A.11.12. Comparaciones

- =, <>: Prueba la (des)igualdad exacta de dos trgeometries
- ?=, ?<>: Prueba si la geometría materializada es alguna vez (no) igual al operando
- %=, %<>: Prueba si la geometría materializada es siempre (no) igual al operando

A.11.13. Operaciones de cuadro delimitador

- &&, @>, <@, ~=, -|-: Operadores topológicos estándar sobre cuadros delimitadores
- <<, >>, &<, &>, <<|, &<|, |>>, |&>, <<#, &<#, #>>, #&>: Operadores de posición espacial y temporal sobre cuadros delimitadores

A.11.14. Teselas

- spaceBoxes, timeBoxes, spaceTimeBoxes: Devuelve los cuadros de una cuadrícula espacial, temporal o espaciotemporal intersecados

A.11.15. Cajas delimitadoras

- stboxes, spans: Devuelve los cuadros espaciotemporales o los lapsos de tiempo de los segmentos
- splitNStboxes, splitEachNStboxes, splitNSpans, splitEachNSpans: Devuelve los cuadros o lapsos divididos en un número dado de particiones

A.11.16. Agregaciones

- tCount: Número de trgeometries que se solapan en cada instante
- extent: Cuadro delimitador agregado de una columna de trgeometries
- merge, mergeAgg: Combina dos trgeometries con la misma geometría de referencia en un conjunto de secuencias

A.12. JSON temporal

A.12.1. Entrada y salida

- asText: Devuelve la representación Well-Known Text (WKT)
- asBinary, asWKB: Devuelve la representación Well-Known Binary (WKB) o la representación Hexadecimal Well-Known Binary (HexEWKB) como texto
- asMFJSON: Devuelve la representación Moving Features JSON (MF-JSON) como texto
- tjsonbFromText, tjsonbFromWKT: Entrar a partir de la representación Well-Known Text (WKT)
- tjsonbFromBinary, tjsonbFromHexWKB: Entrar a partir de la representación Well-Known Binary (WKB) o a partir de la representación Hexadecimal Well-Known Binary (HexEWKB)
- tjsonbFromMFJSON: Entrar a partir de la representación Moving Features JSON (MF-JSON)

A.12.2. Constructores

- tjsonb: Constructor para un JSONB temporal con un valor constante
- tjsonbSeq: Constructores para un JSONB temporal de subtipo secuencia
- tjsonbSeqSet, tjsonbSeqSetGaps: Constructor para un JSONB temporal de subtipo conjunto de secuencias

A.12.3. Conversiones de tipo

- tjsonb::tstzspan: Convertir un JSONB temporal a un punto geométrico temporal
- tjsonb::ttext: Convertir un JSONB temporal a un texto temporal

A.12.4. Accesores

- getValues: Devuelve los valores
- valueAtTimestamp: Devuelve el valor en una marca de tiempo

A.12.5. Transformaciones

- tjsonbInst, tjsonbSeq, tjsonbSeqSet: Transformar un JSONB temporal en otro subtipo
- setInterp: Transformar un JSONB temporal a otra interpolación

A.12.6. Operaciones JSON temporales

- ->, ->>, tjson_object_field, tjsonb_object_field, tjsonb_object_field_text: Extraer un campo de un valor JSON temporal especificado por una clave
- #>, #>>: Extraer un campo de un valor JSON temporal especificado por una ruta
- tjson_array_element, tjsonb_array_element, tjsonb_array_element_text: Extraer un elemento de un arreglo JSON temporal
- ||: Concatenación de JSONB temporal
- -, #-: Eliminación de JSONB temporal
- tjsonb_set, tjsonb_set_lax: Asignación de JSONB temporal
- ?, ?|, ?&: Existencia de JSONB temporal
- @>, <@: JSONB temporal contiene/contenido
- tbool, tint, tfloat, ttext: Extraer un valor alfanumérico temporal de un valor JSONB temporal dado por una clave
- tjson_strip_nulls, tjsonb_strip_nulls: Devuelve un valor JSON temporal sin valores nulos
- @?, tjsonb_path_exists, tjsonb_path_exists_tz: ¿Devuelve la expresión de ruta JSON algún elemento del valor JSONB?
- @@, tjsonb_path_match, tjsonb_path_match_tz: Devuelve el resultado de la comprobación de una expresión de ruta JSON para un valor JSONB temporal.
- tjsonb_path_query, tjsonb_path_query_tz: Devuelve todos los elementos retornados por una expresión de ruta JSON de un valor JSONB temporal, como un arreglo JSON
- tjsonb_path_query_first, tjsonb_path_query_first_tz: Devuelve el primer elemento retornado por una expresión de ruta JSON de un valor JSONB temporal

A.12.7. Restricciones

- atValue, minusValue, atValues, minusValues: Restringir a (al complemento de) un valor o un conjunto de valores

A.12.8. Operaciones de cuadro delimitador

- &&, <@, @>, ~=, -|-: Operadores topológicos
- <<#, &<#, #>>, |&>: Operadores de posición relativa

A.12.9. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- ?=, &=: Comparaciones siempre y alguna vez
- #=, #<>: Comparaciones temporales

A.12.10. Agregaciones

- tCount: Conteo temporal
- wCount: Conteo de ventana

A.13. Índices de celda H3 temporales

A.13.1. Conversiones de tipo

- th3index::tbigint, tbigint::th3index: Convertir entre una celda H3 temporal y un bigint temporal (explícito, sin coste)
- th3index::tgeogpoint, th3index::tgeompoint: Convertir una celda H3 temporal en un punto temporal de los centroides de las celdas

A.13.2. Cobertura geométrica estática

- geoToH3Cell: Devuelve la celda H3 que contiene una geometría de punto en la resolución dada
- geoToH3IndexSet: Devuelve el conjunto de celdas H3 que cubren una geometría en la resolución dada

A.13.3. Accesores

- startValue, endValue: Devuelve el primer o el último identificador de celda H3 de un valor temporal
- valueN: Devuelve el n-ésimo identificador de celda H3 distinto
- getValues: Devuelve el conjunto de identificadores de celda H3 distintos que toma la trayectoria
- valueAtTimestamp: Devuelve el identificador de celda H3 en un instante dado

A.13.4. Inspección del índice

- th3GetResolution: Devuelve el nivel de resolución temporal (0-15) de cada celda
- th3GetBaseCellNumber: Devuelve el número de celda base temporal (0-121) de cada celda
- isValidCell: Devuelve un booleano temporal que indica si cada valor es una celda válida
- th3IsResClassIii: Devuelve un booleano temporal que indica si cada celda tiene orientación de Clase III
- th3IsPentagon: Devuelve un booleano temporal que indica si cada celda es un pentágono

A.13.5. Jerarquía

- th3CellToParent: Devuelve la celda padre temporal en la resolución dada
- th3CellToCenterChild: Devuelve la celda hija central temporal en la resolución dada
- th3CellToChildPos: Devuelve la posición ordinal de cada celda entre las hijas de su padre
- th3ChildPosToCell: Devuelve la celda hija temporal en una posición ordinal dada

A.13.6. Conversiones de latitud/longitud

- tgeompoint_to_th3, tgeogpoint_to_th3: Devuelve la celda H3 temporal que contiene cada punto en la resolución dada
- th3CellToLatlng: Devuelve el centroide geodésico temporal de cada celda
- th3CellToBoundary: Devuelve la frontera poligonal temporal de cada celda como una geografía

A.13.7. Aristas dirigidas

- th3AreNeighborCells: Devuelve un booleano temporal que indica si dos celdas son vecinas en la malla
- th3CellsToDirectedEdge: Devuelve un identificador de arista dirigida temporal a partir de un par de celdas origen/destino
- isValidDirectedEdge: Devuelve un booleano temporal que indica si cada valor es una arista dirigida válida
- th3GetDirectedEdgeOrigin, th3GetDirectedEdgeDestination: Devuelve la celda origen o destino temporal de una arista dirigida
- th3DirectedEdgeToBoundary: Devuelve la frontera poligonal temporal de cada arista dirigida como una geografía

A.13.8. Vértices

- th3CellToVertex: Devuelve el vértice H3 temporal en el número de vértice dado
- th3VertexToLatlng: Devuelve las coordenadas geodésicas temporales de cada vértice
- isValidVertex: Devuelve un booleano temporal que indica si cada valor es un vértice válido

A.13.9. Recorrido de la malla

- th3GridDistance, <->: Devuelve la distancia temporal en saltos de malla entre dos celdas temporales
- th3CellToLocalIj, th3LocalIjToCell: Convertir entre una celda temporal y una coordenada local (I, J) temporal

A.13.10. Métricas

- th3CellArea: Devuelve el área temporal de cada celda en la unidad solicitada
- th3EdgeLength: Devuelve la longitud temporal de cada arista dirigida en la unidad solicitada
- greatCircleDistance: Devuelve la distancia temporal de círculo máximo entre dos puntos geodésicos temporales

A.13.11. Comparaciones ever y always

- ever_eq, ?=: Devuelve si una celda H3 temporal es alguna vez igual al valor de prueba
- always_eq, %=: Devuelve si una celda H3 temporal es siempre igual al valor de prueba
- ever_ne, ?<>: Devuelve si una celda H3 temporal es alguna vez distinta del valor de prueba
- always_ne, %<>: Devuelve si una celda H3 temporal es siempre distinta del valor de prueba

A.13.12. Comparaciones temporales

- #=, #<>: Devuelve la igualdad o desigualdad instante a instante entre una celda H3 temporal y un valor de prueba

A.13.13. Operaciones de cuadro delimitador

- &&, @>, <@, ~=, -|-: Solapamiento, contiene, contenido en, igual y adyacente del cuadro delimitador
- <<, &<, &>, >>, <<|, &<|, |&>, |>>: Posición relativa a lo largo de los ejes espaciales
- <<#, &<#, #>>, #&>: Posición relativa a lo largo del eje temporal

A.13.14. Funciones que devuelven conjuntos

- h3GridDisk: Todas las celdas dentro de k pasos de malla desde el origen
- h3GridRing: Las celdas situadas exactamente a k pasos de malla del origen
- h3GridPathCells: El camino inclusivo de celdas entre dos celdas
- h3CellToChildren: Todas las hijas de una celda en la resolución objetivo dada
- h3CompactCells: Representación compactada de resolución mixta de un conjunto de celdas
- h3UncompactCells: Representación descompactada en la resolución dada
- h3OriginToDirectedEdges: Todas las aristas dirigidas salientes de una celda
- h3CellToVertexes: Todos los vértices de una celda
- h3GetIcosahedronFaces: Los índices de cara del icosaedro intersecados por una celda

A.13.15. Relaciones espaciales

- eContains, aContains, tContains: Si una huella contiene a la otra (solo interiores)
- eCovers, aCovers, tCovers: Si una huella cubre a la otra (incluida la frontera)
- eDisjoint, aDisjoint, tDisjoint: Si las huellas no comparten ningún punto
- eIntersects, aIntersects, tIntersects: Si las huellas comparten al menos un punto
- eTouches, aTouches, tTouches: Si las huellas comparten un punto de frontera pero ningún punto interior
- eDwithin, aDwithin, tDwithin: Si las huellas se encuentran dentro de una distancia dada

A.13.16. Índices

- th3index_rtree_ops: Clase de operadores GiST que indexa el cuadro delimitador espaciotemporal
- th3index_quadtree_ops, th3index_kdtree_ops: Clases de operadores SP-GiST basadas en el cuadro delimitador espaciotemporal

A.14. Índices de celda Quadbin temporales

A.14.1. Conversiones de tipo

- tquadbin::tbigint, tbigint::tquadbin: Convertir entre una celda quadbin temporal y un bigint temporal
- tquadbin::tgeompoint: Convertir una celda quadbin temporal a un punto geométrico temporal de los centroides de las celdas

A.14.2. Celdas estáticas

- isValidIndex: Devuelve si un valor es un identificador quadbin estructuralmente válido
- isValidCell: Devuelve si un valor es una celda quadbin válida

A.14.3. Accesores

- startValue, endValue: Devuelve la primera o la última celda
- valueN: Devuelve la enésima celda distinta
- getValues: Devuelve el conjunto de celdas distintas
- valueAtTimestamp: Devuelve la celda en una marca de tiempo

A.14.4. Inspección del índice

- quadbinGetResolution, tquadbinGetResolution: Devuelve el nivel de resolución (zoom) de una celda
- isValidCell: Devuelve un booleano temporal de validez de la celda

A.14.5. Jerarquía

- quadbinCellToParent, tquadbinCellToParent: Devuelve la celda padre en una resolución más gruesa
- quadbinCellToChildren: Devuelve las celdas hijas en una resolución más fina
- quadbinCellSibling: Devuelve la celda vecina en una dirección
- quadbinGridDisk: Devuelve las celdas dentro de k pasos de malla de un origen

A.14.6. Conversiones geométricas

- geoToQuadbinCell: Devuelve la celda que contiene un punto en una resolución
- quadbinCellToPoint, tquadbinCellToPoint: Devuelve el centroide de una celda
- quadbinCellToBoundary, tquadbinCellToBoundary: Devuelve el polígono de frontera de una celda

A.14.7. Teselas y quadkeys

- quadbinTileToCell: Devuelve la celda a partir de coordenadas de tesela slippy-map
- quadbinCellToTile: Devuelve las coordenadas de tesela de una celda
- quadbinCellToQuadkey, tquadbinCellToQuadkey: Devuelve la cadena quadkey de una celda

A.14.8. Métricas

- quadbinCellArea, tquadbinCellArea: Devuelve el área de una celda

A.14.9. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

A.14.10. Índices

- tquadbin_btree_ops: Clase de operadores B-tree por defecto
- tquadbin_hash_ops: Clase de operadores hash por defecto

A.15. Nubes de Puntos Temporales

A.15.1. Tipos estáticos `pcpoint` y `pcpatch`

A.15.1.1. Constructores

- pcpoint: Construir un pcpoint 2D o 3D a partir de coordenadas x/y(/z)
- pcpatch: Construir un pcpatch a partir de una lista variádica de pcpoints

A.15.1.2. Accesores

- pcid: Devuelve el identificador de esquema de un pcpoint o pcpatch
- getX, getY, getZ: Devuelve una coordenada de un pcpoint
- getDim: Devuelve cualquier dimensión nombrada de un pcpoint
- SRID: Devuelve el identificador de referencia espacial de un pcpoint o pcpatch

A.15.2. Tipos de conjunto `pcpointset` y `pcpatchset`

A.15.2.1. Constructores

- set: Construir un conjunto a partir de un arreglo de valores pcpoint o pcpatch con el mismo pcid

A.15.3. Cuadro delimitador `tpcbox`

A.15.3.1. Constructores

- tpcbox, tpcbox_z, tpcbox_t, tpcbox_xt, tpcbox_zt: Construir un tpcbox a partir de cotas X/Y/Z/T explícitas y un pcid

A.15.3.2. Conversiones de tipo

- tpcbox(pcpoint): Convertir un pcpoint en un tpcbox degenerado (de extensión cero)
- tpcbox(pcpatch): Convertir un pcpatch en un tpcbox usando los PCBOUNDS embebidos del parche

A.15.3.3. Accesores

- hasX, hasZ, hasT: Devuelve si un tpcbox tiene las dimensiones X/Y, Z o T definidas
- xmin, xmax, ymin, ymax, zmin, zmax: Devuelve una cota de coordenada de un tpcbox
- tmin, tmax: Devuelve una cota temporal de un tpcbox
- pcid, SRID: Devuelve el identificador de esquema y el SRID de un tpcbox

A.15.3.4. Transformaciones

- round: Devuelve un tpcbox con las coordenadas redondeadas al número de dígitos decimales dado
- setSRID: Devuelve un tpcbox con el SRID sobreescrito

A.15.3.5. Operaciones de conjunto

- tpcbox_union, +: Devuelve la unión de dos tpcboxes
- tpcbox_intersection, *: Devuelve la intersección de dos tpcboxes

A.15.3.6. Operadores topológicos

- &&: Devuelve verdadero si dos tpcboxes se solapan
- @>: Devuelve verdadero si el primer tpcbox contiene al segundo
- <@: Devuelve verdadero si el primer tpcbox está contenido en el segundo
- ~=: Devuelve verdadero si dos tpcboxes son exactamente iguales
- -|-: Devuelve verdadero si dos tpcboxes son adyacentes

A.15.3.7. Operadores de posición relativa

- <<, &<, >>, &>: Operadores de posición relativa en el eje X
- <<|, &<|, |>>, |&>: Operadores de posición relativa en el eje Y
- <</, &</, />>, /&>: Operadores de posición relativa en el eje Z
- <<#, &<#, #>>, #&>: Operadores de posición relativa en el eje temporal

A.15.3.8. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales

A.15.4. Tipo temporal `tpcpoint`

A.15.4.1. Entrada y salida

- asText, tpcpoint::text: Devolver la representación textual, o convertir a texto

A.15.4.2. Constructores

- tpcpoint: Constructor para tpcpoint de subtipo instante
- tpcpointSeq: Constructor para tpcpoint de subtipo secuencia
- tpcpointSeqSet: Constructor para tpcpoint de subtipo conjunto de secuencias

A.15.4.3. Accesores

- pcid: Devuelve el identificador de esquema pgPointCloud compartido por todos los instantes
- SRID: Devuelve el SRID heredado del esquema
- getX, getY, getZ: Proyectar un tpcpoint sobre una dimensión espacial, devolviendo un tfloat
- getDim: Proyectar un tpcpoint sobre cualquier dimensión del esquema, devolviendo un tfloat

A.15.4.4. Conversiones de tipo

- tpcpoint::tgeompoint: Convertir un tpcpoint en un punto de geometría temporal

A.15.4.5. Transformaciones

- tpcpointInst, tpcpointSeq, tpcpointSeqSet: Transformar un tpcpoint en otro subtipo
- setInterp: Transformar un tpcpoint a otra interpolación

A.15.4.6. Comparaciones

- =, <>, <, >, <=, >=: Comparaciones tradicionales
- eEq, aEq, eNe, aNe, ?=, %=, ?<>, %<>: Comparaciones de igualdad y desigualdad alguna vez y siempre
- tEq, tNe, #=, #<>: Comparaciones de igualdad y desigualdad temporales

A.15.4.7. Restricciones

- atValue, minusValue, atValues, minusValues: Restringir a (al complemento de) un valor o un conjunto de valores
- atTime, minusTime: Restringir a (al complemento de) un selector de tiempo
- atTpcbox, minusTpcbox: Restringir a (al complemento de) la extensión de un tpcbox
- beforeTimestamp, afterTimestamp: Restringir a los instantes anteriores o posteriores a una marca de tiempo

A.15.4.8. Modificaciones

- insert, update: Insertar un tpcpoint en otro, o actualizar otro con él
- deleteTime: Eliminar un selector de tiempo de un tpcpoint
- appendInstant, appendSequence: Añadir un instante o una secuencia a un tpcpoint

A.15.4.9. Fragmentación

- unnest: Devolver los valores distintos de un tpcpoint con el conjunto de spans durante el cual toma cada uno de ellos
- timeSplit: Fragmentar un tpcpoint en fragmentos, uno por cada intervalo de tiempo

A.15.4.10. Operaciones de cuadro delimitador

- &&, @>, <@, ~=, -|-: Operadores topológicos
- <<, >>, <<|, |>>, <<\/, \/>>, <<#, #>>: Operadores de posición relativa
- |=|, nearestApproachDistance: Distancia de aproximación más cercana, orderable por KNN

A.15.4.11. Relaciones espaciales

- eIntersects, aIntersects: Alguna vez / siempre intersecta
- eDisjoint, aDisjoint: Alguna vez / siempre disjunto
- eDwithin, aDwithin: Alguna vez / siempre dentro de una distancia

A.15.5. Tipo temporal `tpcpatch`

A.15.5.1. Entrada y salida

- asText, tpcpatch::text: Devolver la representación textual, o convertir a texto

A.15.5.2. Constructores

- tpcpatch: Constructor para tpcpatch de subtipo instante
- tpcpatchSeq: Constructor para tpcpatch de subtipo secuencia
- tpcpatchSeqSet: Constructor para tpcpatch de subtipo conjunto de secuencias

A.15.5.3. Accesores

- startNumPoints, endNumPoints: Devuelve el número de puntos en el parche del primer o último instante
- numPoints: Devuelve el número total de puntos en todos los instantes
- points: Función que devuelve conjuntos, emitiendo una fila por (marca de tiempo, pcpoint)

A.15.5.4. Transformaciones

- tpcpatchInst, tpcpatchSeq, tpcpatchSeqSet: Transformar un tpcpatch en otro subtipo
- setInterp: Transformar un tpcpatch a otra interpolación

A.15.5.5. Restricciones

- atValue, minusValue, atValues, minusValues: Restringir a (al complemento de) un valor o un conjunto de valores
- atTpcbox, minusTpcbox: Restringir a los instantes cuyo parche se solapa con un tpcbox (nivel de parche)
- atTpcboxFine, minusTpcboxFine: Restringir a (al complemento de) los puntos dentro de un tpcbox (por punto)
- atGeometry, minusGeometry: Restringir a (al complemento de) los puntos que intersectan una geometría 2D
- beforeTimestamp, afterTimestamp: Restringir a los instantes anteriores o posteriores a una marca de tiempo

A.15.5.6. Modificaciones

- insert, update: Insertar un tpcpatch en otro, o actualizar otro con él
- deleteTime: Eliminar un selector de tiempo de un tpcpatch
- appendInstant, appendSequence: Añadir un instante o una secuencia a un tpcpatch

A.15.5.7. Fragmentación

- unnest: Devolver los valores distintos de un tpcpatch con el conjunto de spans durante el cual toma cada uno de ellos
- timeSplit: Fragmentar un tpcpatch en fragmentos, uno por cada intervalo de tiempo

A.15.5.8. Relaciones espaciales

- eIntersects: Devuelve verdadero si al menos un punto de algún instante intersecta una geometría

A.15.5.9. Operaciones de cuadro delimitador

- &&, @>, <@, ~=, -|-: Operadores topológicos
- <<, >>, <<|, |>>, <<\/, \/>>, <<#, #>>: Operadores de posición relativa
- |=|, nearestApproachDistance: Distancia de aproximación más cercana, orderable por KNN

A.15.6. Agregaciones

- extent: Agregación de cuadro delimitador sobre valores tpcpoint, tpcpatch o tpcbox
- tcount, wcount: Conteo temporal y conteo de ventana
- tnpoints: Recuento de puntos por instante del pcpatch sumado entre filas
- tdensity: Densidad de puntos por instante sumada entre filas
- merge, mergeAgg: Combinar múltiples valores temporales en uno solo
- appendInstant, appendInstantAgg, appendSequence, appendSequenceAgg: Agregaciones de construcción en flujo

A.15.7. Índices

- Clases de operador GiST, SP-GiST, árbol-B y hash para tpcpoint, tpcpatch y tpcbox

A.15.8. Representación binaria

- asBinary, tpcpointFromBinary, tpcpatchFromBinary: Entrada/salida binaria con esquema embebido para portabilidad

A.15.9. Salida MF-JSON

- asMFJSON: Devuelve la representación Moving Features JSON de un tpcpoint
- asMFJSON: Devuelve la representación Moving Features JSON de un tpcpatch

A.15.10. Integración con PDAL

- readers.tpcpatch: Lector PDAL que lee valores tpcpatch desde una consulta SQL
- writers.tpcpatch: Escritor PDAL que escribe valores tpcpatch en una tabla PostgreSQL

A.15.11. Puente PCL

- pcpatch_to_pcd, pcpatch_from_pcd, pcpatch_voxel_grid, pcpatch_sor, pcpatch_icp, pcpatch_normals: Funciones SQL respaldadas por PCL para procesamiento de nubes de puntos

A.16. Muestreo de Ráster

A.16.1. Operador de Muestreo

- raster_value: Muestrear una banda de ráster en cada instante de una trayectoria

A.16.2. Muestreo Raquet / QUADBIN

- trajectory_quadbins: Devolver las celdas QUADBIN distintas en un nivel de zoom cubiertas por una trayectoria
- raster_tile_value_quadbin: Muestrear un chip ráster de Raquet a lo largo de una trayectoria usando una celda QUADBIN para la georreferenciación
- raquet: Construir una tesela ráster Raquet a partir de sus bytes de píxeles, dimensiones, celda QUADBIN y tipo de píxel
- raquet_read: Decodificar un archivo ráster en memoria en una tesela ráster Raquet mediante GDAL
- raster_tile_value: Muestrear una tesela ráster raquet a lo largo de una trayectoria
- raster_tile_value: Muestrear un arreglo de teselas ráster raquet a lo largo de una trayectoria, conservando la resolución más fina donde se solapan

A.16.3. Accesores y Comparación de Raquet

- quadbin: Devolver la celda QUADBIN de una tesela Raquet
- width: Devolver el ancho en píxeles de una tesela Raquet
- height: Devolver el alto en píxeles de una tesela Raquet
- nodata: Devolver el valor centinela nodata de una tesela Raquet
- pixtype: Devolver el nombre del tipo de dato de píxel de una tesela Raquet
- stbox: Convertir una tesela Raquet en una caja espaciotemporal
- comparación de raquet: Comparar dos teselas Raquet

A.16.4. Operadores de Restricción y Predicado

- atRasterValue: Devolver los instantes de una trayectoria donde el valor ráster muestreado cae dentro de un rango de flotante
- minusRasterValue: Devolver los instantes de una trayectoria donde el valor ráster muestreado cae fuera de un rango de flotante
- eRasterValue: Devolver verdadero si la trayectoria alguna vez muestrea un valor de píxel ráster dentro de un rango de flotante
- aRasterValue: Devolver verdadero si cada instante de la trayectoria dentro de la extensión del ráster muestrea un valor de píxel dentro de un rango de flotante

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.