Activity 1: MobilityDB Data Model and Type System

An instance of a temporal type represents a function from time to the base type. Temporal types are defined based on their base type and time type.

The MobilityDB database defines six built-in temporal types based on top of the base and **spatial** types: bool, int, float, text, **geometry point (uses planar coordinate system), and geography point (uses spherical coordinate system)** data types. Formally, the temporal data types are: tbool, tint, tfloat, ttext, **tgeompoint, and tgeogpoint**.

Instants represent **valid time**, i.e. the time in the real world when the event occurred.

For instance, we can create a table with a temporal data type as follows:

CREATE TABLE ranking(star **tint**);

CREATE TABLE IoT(temperature **tfloat**);

CREATE TABLE gps(position **tgeompoint**);

In addition, temporal data types have four subtypes: instant, sequence and sequence set. Any temporal type column can store an instance of any of these four sybtypes.

1. **Instant Subtype**: represents a value at a time instant.

As an example of a **tfloat instant**, temperature sensors measure the amount of heat energy in a source at different instants:

17@2018-01-01 08:00:00

This can be inserted in the IoT table as follows (note that it is not necessary to cast to tfloat):

INSERT INTO IoT VALUES (tfloat '17@2018-01-01 08:00:00');

INSERT INTO IoT VALUES ('18.4@2018-01-01 08:00:05');

MobilitDB transforms the literal (the @ symbol separates the value from the time instant) to the corresponding temporal data type.

We can also insert these data using the appropriate type constructor which has two parameters: base type and the timestamptz data type (timestamp with time zone). The constructor for creating an instant is defined as follows:

tbase(base, timestamptz): tbase

Note that timestampz is an abbreviation for timestamp with time zone. Standard SQL also requires timestamp to be equivalent to timestamp with zone. You can use any of them.

Thus, we can insert both previous tuples by using the following alternative expressions:

```
INSERT INTO IoT VALUES (tfloat(17, '2018-01-01 08:00:00'::timestamp with time zone) );

INSERT INTO IoT VALUES (tfloat(18.4, '2018-01-01 08:00:05'::timestamptz) );
```

When we have instant values, the 'Discrete Interpolation method' is applied, i.e. we cannot infer data types beyond the data provided.

If we run

```
SELECT tempSubType(temperature), interp(temperature), temperature
FROM IoT;
```

We obtain

tempsubtype text	interp text	temperature tfloat
Instant	None	17@2018-01-01 08:00:00+00
Instant	None	18.4@2018-01-01 08:00:05+00

Since the attribute temperature corresponds to an **instant** subtype (necessarily with discrete interpolation), if we ask for its value at a moment different to this instant, we obtain null value

SELECT valueattimestamp(temperature, '2018-01-01 08:00:00'), *

from IoT;

valueattimestamp double precision	temperature tfloat
17	17@2018-01-01 08:00:00+00
[null]	18.4@2018-01-01 08:00:05+00

GPS sensors provide geolocation positions at different instants. For storing these data in the GPS table, **tgeompoint** or **tgeogpoint** would be more appropriate.

Notice that if locations are provided as latitude-longitude pairs, the use of a constructor instead of literals is more convenient when inserting data into the table.

For instance, consider a gps table defined as follows:

CREATE TABLE gps(position tgeompoint, lat float, lon float, t timestamp);

If the three attributes lat, lon and t contain data, in order to populate the "position" column we can run the following query

UPDATE gps SET position=tgeompoint(ST_MakePoint(lat, lon), t);

It is also possible to generate the expected literal string from scratch, but this method is error-prone:

UPDATE gps SET position = tgeompoint ('POINT(' || lat || ' ' || lon || ')@' || t) ;

2. **Sequence Subtype**: represent the evolution of the value during a sequence of time instants where **the values between these instants are interpolated using discrete (i.e. none)/stepwise/linear manner**.

A temporal sequence is composed of N instants in ascending order, i.e. t1 < t2 <… < tN

The expression of a sequence is a comma-separated list of instants enclosed in curly brackets, squared brackets or parentheses as explained next.

2.1 Set (Curly brackets): represents the evolution of a value at a **set** of time instants where **the values between these instants are unknown, i.e. the discrete (or none) interpolation method is used.**

We next show an example of a **tfloat sequence** value with discrete interpolation method.

{17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00};

We can insert a tuple containing a sequence value with discrete interpolation method as follows:

INSERT INTO IoT VALUES (' {17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00} ';

It is also possible to insert these data using the appropriate constructor.

If we have same value at different instants we can use:

tbase (base, tstzset): tbaseSeq

For instance,

```
INSERT INTO IoT VALUES (tfloat ( 17,
                        tstzset('{2018-01-01 08:00:00,
                                  2018-01-01 08:05:00,
                                  2018-01-01 08:10:00}')) );
```

Where “tstzset” refers to a set of timestamps.

After both insertions, if we run

SELECT tempSubType(temperature), interp(temperature),

temperature

FROM IoT;

The last two tuples show the sequences.

tempsubtype text	interp text	temperature tfloat
Instant	None	17@2018-01-01 08:00:00+00
Instant	None	18.4@2018-01-01 08:00:05+00
Sequence	Discrete	{17@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00, 18@2018-01-01 08:10:00+00}
Sequence	Discrete	{17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00, 17@2018-01-01 08:10:00+00}

A more general form is the usage of an array of tbase instances. The second parameter is the text 'Discrete' since we are building a sequence with none interpolation. In next subsection, we show that this parameter change according to the interpolation method used.

tbaseSeq(tbase[], 'Discrete'): tbaseSeq

or

tbaseSeq(tbase[], 'Discrete', boolean, boolean): tbaseSeq

As previously discussed, the use of a constructor could be more convenient. Here, we show the alternative expression for inserting the tuple with different values shown above:

```
INSERT INTO IoT VALUES (tfloatseq( ARRAY [
                tfloat (17,  '2018-01-01 08:00:00'::timestamp),
                tfloat (17.4,  '2018-01-01 08:05:00'::timestamp),
                tfloat (18,  '2018-01-01 08:10:00'::timestamp)
                        ], 'Discrete') );
```

Or

```
INSERT INTO IoT VALUES (tfloatseq( ARRAY [
                tfloat ( 17,  '2018-01-01 08:00:00'::timestamp ),
                tfloat ( 17.4,  '2018-01-01 08:05:00'::timestamp ),
                tfloat ( 18,  '2018-01-01 08:10:00'::timestamp )
                        ], 'Discrete', true, true) );
```

Notice that the last two Boolean parameters will be ignored.

Since we have an tbase set, if we ask for a value at a timestamp different for those present in the (discrete) sequences, we obtain the *null* value.

```
SELECT valueattimestamp(temperature, '2018-01-01 08:05:00'), *
FROM  IoT;
```

valueattimestamp double precision	temperature tfloat
[null]	17@2018-01-01 08:00:00+00
[null]	18.4@2018-01-01 08:00:05+00
17.4	{17@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00, 18@2018-01-01 08:10:00+00}
17	{17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00, 17@2018-01-01 08:10:00+00}

SELECT valueattimestamp(temperature, '2018-01-01 08:04:00'), *

FROM IoT;

valueattimestamp double precision	temperature tfloat
[null]	17@2018-01-01 08:00:00+00
[null]	18.4@2018-01-01 08:00:05+00
[null]	{17@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00, 18@2018-01-01 08:10:00+...
[null]	{17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00, 17@2018-01-01 08:10:00+00}

2.2 Continuous Sequence (parentheses or squared brackets): represents a range of values. The parenthesis is used when the boundary of the interval is exclusive. On the contrary, the squared bracket is used when the boundary is inclusive. Both can be combined in the same expression. In any case, we can use **stepwise** or **linear** interpolation depending on the tbase.

In the case of a continuous temporal sequence, the N instants t1 < t2 <… < tN produce a partition.

For instance, [v1@t1, v2@t2, v3@t3, v4@t4) defines a temporal partition composed of three intervals: [t1, t2), [t2, t3) and [t3, t4).

This is an example of a tint sequence:

(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]

For these continuous sequences, MobilityDB provides **step** and/or **linear interpolation**, depending on the subjacent (base/spatial) data type:

2.2.1. Stepwise Interpolation: the temporal value remains constant within each interval and this constant value is the one defined in the left bound of the interval.

Discrete data (**tbool, tint, ttext)** data can only use *stepwise interpolation*. On the contrary, continuous data (**tfloat, tgeompoint, tgeogpoint**) can choose between stepwise or linear (default) interpolation (see next subsection).

Consider the following **tint sequence** (10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]. A sequence with discrete base type only accepts stepwise interpolation. Thus, we can infer that $\forall$ t $\in$ (2018-01-01 08:00:00, 2018-01-01 08:05:00) the value is 10, $\forall$ t $\in$ [2018-01-01 08:05:00, 01-01 08:10:00] the value is 20 and the value is 15 for t=01-01 08:10:00

We can insert a tuple with a **tint sequence** as a String:

INSERT INTO ranking VALUES ('(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');

We can also explicitly indicate the interpolation method:

INSERT INTO ranking VALUES ('**interp=step**;(10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');

An error is raised when we indicate an invalid interpolation method for a tint data type, which accepts only stepwise interpolation, as in the following statement:

INSERT INTO ranking VALUES ('interp=linear; (10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]');

It is also possible to insert data using the appropriate constructor.

If we have same value at different instants we can use:

tbase (base, tstzspan, '**step**'): tbaseSeq

or

tbase (base, tstzspan): tbaseSeq

but, in this case you need to use a timestamp span (i.e. a time interval given by its opened/closed lower and opened/closed upper bounds). A temporal span, i.e. tstzspan, is special temporal sequence defined by its two extremes (produces a single partition).

Note that, if you omit the '**step**' literal and you are using a tbase which admits both interpolation methods, you are building a sequence with **linear** interpolation, since it is the default.

For instance,

```
INSERT INTO ranking VALUES (tint (17,
        tstzspan('[2018-01-01 08:00:00,
                    2018-01-01 08:05:00) ') )  );
```

Nevertheless, if you have different values at different intervals you can provide an array of instants and the literal 'step'. Temporal limits are inclusive unless you provide explicitly, by using the last two parameters.

tbaseSeq(tbase[],'step', lower boolean, upper boolean): tbaseSeq

Here we show the alternative expression for the same tuple insertion with values (10@2018-01-01 08:00:00, 20@2018-01-01 08:05:00, 15@2018-01-01 08:10:00]. The left bound is not included but you can omit the last parameter.

```
INSERT INTO ranking VALUES (tintseq(ARRAY [
tint(10,  '2018-01-01 08:00:00'::timestamptz ),
tint(20,  '2018-01-01 08:05:00'::timestamptz ),
tint(15,  '2018-01-01 08:10:00'::timestamptz ) ],
'step', false));
```

In the case that the first value has a closed left bound you should use

INSERT INTO ranking VALUES (**tintseq**(ARRAY [

tint(10, '2018-01-01 08:00:00'::timestamptz),

tint(20, '2018-01-01 08:05:00'::timestamptz),

tint(15, '2018-01-01 08:10:00'::timestamptz)],

'step'));

After these two insertions, we can show both the sequence and the interpolation method

SELECT tempSubType(star), interp(star), *

FROM ranking;

tempsubtype text	interp text	star tint
Sequence	Step	(10@2018-01-01 08:00:00+00, 20@2018-01-01 08:05:00+00, 15@2018-01-01 08:10:00+00]
Sequence	Step	[17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00)

Since the sequence has stepwise interpolation, we can ask for values different from the instants used.

SELECT valueattimestamp(star, '2018-01-01 08:00:00'), *

FROM ranking

valueattimestamp integer	star tint
[null]	(10@2018-01-01 08:00:00+00, 20@2018-01-01 08:05:00+00, 15@2018-01-01 08:10:00+00]
17	[17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00)

The null value obtained is because the left instant is not included in the sequence.

If, instead, we ask for a value at a time in the interval of the sequence, we obtain either the value or the interpolated value.

SELECT valueattimestamp(star, '2018-01-01 08:02:00'), *

FROM ranking

valueattimestamp integer	star tint
10	(10@2018-01-01 08:00:00+00, 20@2018-01-01 08:05:00+00, 15@2018-01-01 08:10:00+00]
17	[17@2018-01-01 08:00:00+00, 17@2018-01-01 08:05:00+00)

In this case, the values obtained are valid $\forall$ t $\in$ [2018-01-01 08:00:00, 2018-01-01 08:05:00)

In the case that we want to use a stepwise interpolation with a tbase data that admits both, stepwise and linear interpolation, we should indicate that explicitly, since the latter is the default.

Let us try with a tfloat.

We first delete the information in table IoT:

DELETE FROM IoT;

Now, we insert some **tfloats** in different formats.

From text:

INSERT INTO IoT VALUES ('interp=step;(17@2018-01-01 08:00:00, 17.4@2018-01-01 08:05:00, 18@2018-01-01 08:10:00]');

It also possible to insert these data using the appropriate constructor.

If we have same value at different instants we should use

tbase (base, tstzspan, '**step**'): tbaseSeq

(do not omit step interpolation)

```
INSERT INTO IoT VALUES (tfloat (17.4,
        tstzspan('[2018-01-01 08:00:00,
                2018-01-01 08:05:00) '), 'step' )  );
```

Or by an array of instants, as shown previously.

```
INSERT INTO IoT VALUES (tfloatseq( ARRAY [
tfloat(10,  '2018-01-01 08:00:00'::timestamptz ),
tfloat(17.4,  '2018-01-01 08:05:00'::timestamptz ),
tfloat(18,  '2018-01-01 08:10:00'::timestamptz ) ],
'step', false ) );
```

After these three insertions, we can show both the sequence and the interpolation method

```
SELECT  tempSubType(temperature),  interp(temperature), *
FROM IoT;
```

tempsubtype text	interp text	temperature tfloat
Sequence	Step	Interp=Step;(17@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00, 18@2018-01-01 08:10:00+00]
Sequence	Step	Interp=Step;[17.4@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00)
Sequence	Step	Interp=Step;(10@2018-01-01 08:00:00+00, 17.4@2018-01-01 08:05:00+00, 18@2018-01-01 08:10:00+00]

2.2.2. Linear Interpolation: the temporal value evolves continuously in a linear way within each interval. Only **tfloat, tgeompoint or tgeogpoint** can use linear interpolation and, if not specified, it is the default. We show some linear interpolation examples.

The following sentence inserts a tuple with a tgeompoint sequence as a String:

```
INSERT INTO gps VALUES ('(Point(1 1)@2021-01-01 08:00:00, Point(2 3)@2021-01-01 08:05:00,  Point(3 3)@2021-01-01 08:10:00]');
```

linear interpolation is used in each of the 3-temporal partitions (2021-01-01 08:00:00, 2021-01-01 08:05:00) , [2021-01-01 08:05:00, 2021-01-01 08:10:00) and [2021-01-01 08:10:00, 2021-01-01 08:10:00].

It is also possible to insert these data using the appropriate constructor.

If we have only one value valid during a temporal span, we can use:

tbaseSeq(base, period, linear=true): tbaseSeq

For instance,

```
INSERT INTO gps VALUES
(tgeompoint ('Point(1 1)'::geometry,
 tstzspan('[2018-01-01 08:00:00,
                    2018-01-01 08:05:00) '), 'linear')  );
```

The third parameter is used for setting the interpolation method (by default it is true and corresponds for linear interpolation).

In a more general form, we can use an array of instants. Notice that second parameter is used for indicating linear interpolation method. The last two Boolean parameters indicate whether the temporal limits are exclusive (false) or inclusive (by default it is true when not specified). If the last two parameters are not omitted, you should provide 'linear' parameter.

tbaseSeq(tbase[],'linear', lower boolean, upper boolean): tbaseSeq

Next we show the alternative expression for the String

(Point(1 1)@2021-01-01 08:00:00, Point(2 3)@2021-01-01 08:05:00, Point(3 3)@2021-01-01 08:10:00]

INSERT INTO gps VALUES (**tgeompointseq**(ARRAY [

tgeompoint(ST_MakePoint(1, 1), '2021-01-01 08:00:00'::timestamptz),

tgeompoint(ST_MakePoint(2, 3), '2021-01-01 08:05:00'::timestamptz),

tgeompoint(ST_MakePoint(3, 3), '2021-01-01 08:10:00'::timestamptz)], 'linear', false, true));

We can also omit the last parameter:

INSERT INTO gps VALUES (**tgeompointSeq**(ARRAY [

tgeompoint(ST_MakePoint(1, 1), '2021-01-01 08:00:00'::timestamptz),

tgeompoint(ST_MakePoint(2, 3), '2021-01-01 08:05:00'::timestamptz),

tgeompoint(ST_MakePoint(3, 3), '2021-01-01 08:10:00'::timestamptz)], 'linear', false));

We show now how to generate an array of tgeompoints from scratch. Nevertheless, a very useful alternative consists of building an array of tgeompoints that comes from some column of another table. We next explain this option.

Consider a table with three columns: x, y and date. The table (called **rawdata**) contains:

RawData		
X (int)	Y (int)	Date (timestamp)
1	1	2021-01-01 08:00:00
2	3	2021-01-01 08:05:00
3	3	2021-01-01 08:10:00
…	…	…

Postgres offers a function that returns an array from an expression:

array_agg(expression): array

Thus, instead of ARRAY[] we can use array_agg(), which is used to generate a **tgeompoint_inst** in temporal ascending order. To dynamically insert dynamically tuples in the **gps table** we write:

```
INSERT INTO gps
  SELECT
  tgeompointSeq(array_agg(
  tgeompoint(ST_MakePoint(?? ??), ??) ORDER BY ??, 'linear', false )
  FROM rawdata
```

where we need to replace every "??" symbol with the correct column of the source table. Since we need to generate a sequence, these values need to be inserted sorted by the **date column**. Summarizing, the expression should be:

```
INSERT INTO gps
   SELECT
      tgeompointSeq(array_agg(
```

tgeompoint (ST_MakePoint(x, y) , date) ORDER BY date

), 'linear', false)

FROM rawdata

We showed different ways of populating a **tsequence** column: building a string, from a static array or from an array generated dynamically from other table. Now, we can ask:

SELECT tempSubType(position), interp(position), *

FROM gps;

Temp subtype Text	Interpol. text	Position tgeompoint
Sequence	Linear	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 0101000000000000000000004000000000000000840@2021-01-01 08:05:00-03, 0101000000000000000000084000000000000000840@2021-01-01 08:10:00-03]

Since we have linear interpolation, we can ask for values different from the three instants used.

SELECT valueattimestamp(position, '2021-01-01 08:09:00'), *

FROM gps

Valueattimestamp geometry	Position tgeompoint
0101000000666666666666064 0000000000000840	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 0101000000000000000000004000000000000000840@2021-01-01 08:05:00-03, 0101000000000000000000084000000000000000840@2021-01-01 08:10:00-03]

We reformulate the query for a human readable result

SELECT

ST_AsText(valueattimestamp(position, '2021-01-01 08:09:00')), *

FROM gps

St_astext text	Position tgeompoint
POINT(2.8 3)	(0101000000000000000000F03F000000000000F03F@2021-01-01 08:00:00-03, 0101000000000000000000004000000000000000840@2021-01-01 08:05:00-03, 0101000000000000000000084000000000000000840@2021-01-01 08:10:00-03]

This point is obtained by linear interpolation within the corresponding subinterval ['2018-01-01 08:05:00', '2018-01-01 08:10:00')

3. **Continuous Sequence Set (set of continuous sequences, i.e. curly brackets enclosing parentheses or squared brackets):** represents a **set of range of values**. More precisely, represents the evolution of the value at a set of continuous sequences where **the values between theses sequences are unknown, i.e., there exist temporal gaps where there is no value information.** The expression of a sequence is a comma-separated list of continuous sequences enclosed in curly brackets. This is an example of tfloat sequence set:

{

[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],

(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]

}

We can insert a tuple with a **tfloat** sequence set as a String, for instance, with linear interpolation method. First delete tuples.

```
DELETE FROM IoT;

INSERT INTO IoT VALUES (
'{[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],
(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]}');
```

It is also possible to insert these data using the constructor:

tbaseSeqset(tbaseSeq[]): tbaseSeqset

where the array parameter can be built on top of elements of ttypeSeq.

Here we show the alternative expression for the same tuple insertion with values:

```
{
[10.3@2018-01-01 08:00:00, 20.4@2018-01-01 08:05:00],
(15.1@2018-01-01 08:10:00, 3.5@2018-01-01 08:12:00]
}
```

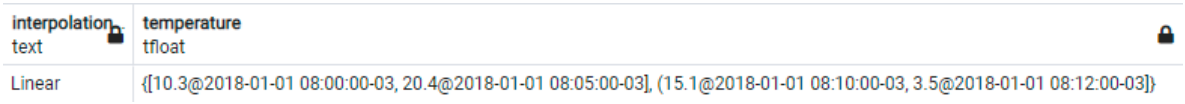
```
DELETE FROM IoT;

INSERT INTO IoT VALUES (
 tfloatSeqset( ARRAY [
      tfloatSeq( ARRAY [
         tfloat(10.3 ,  '2018-01-01 08:00:00'::timestamptz),
         tfloat( 20.4,  '2018-01-01 08:05:00'::timestamptz)
                          ], 'linear', true, true),
      tfloatSeq( ARRAY [
       tfloat ( 15.1,  '2018-01-01 08:10:00'::timestamptz),
       tfloat( 3.5,  '2018-01-01 08:12:00'::timestamptz)
                  ], 'linear', false, true)  ] )    );
```

Sequences with different interpolation methods in the same sequence set, are not allowed.

We can show both the sequence and the interpolation method:

```
SELECT  tempSubType(temperature),  interp(temperature), *
FROM IoT;
```

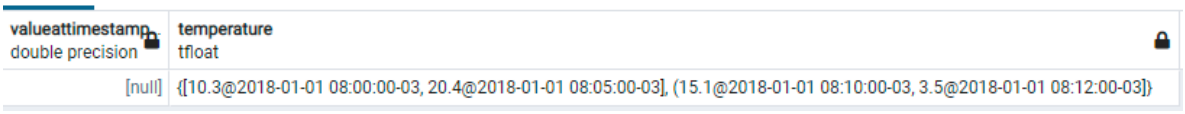


interpolation text	temperature tfloat
Linear	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (15.1@2018-01-01 08:10:00-03, 3.5@2018-01-01 08:12:00-03]}

Tempsubtype Text	Interp text	Position tgeompoint
SequenceSet	Linear	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (15.1@2018-01-01 08:10:00-03, 3.5@2018-01-01 08:12:00-03]}

Asking for values of instant outside any of the sequences belonging to the set returns a *null* value.

```
SELECT valueattimestamp(temperature, '2018-01-01 08:07:00'), *
FROM Iot
```



valueattimestamp double precision	temperature tfloat
[null]	{[10.3@2018-01-01 08:00:00-03, 20.4@2018-01-01 08:05:00-03], (15.1@2018-01-01 08:10:00-03, 3.5@2018-01-01 08:12:00-03]}

Important Whenever possible, MobilityDB merges data.