MobilityDB

A PostgreSQL-PostGIS Extension for Mobility Data Management
What is Mobility Data?
# Mobility Data: Constructing Trajectories

<table>
<thead>
<tr>
<th>moid</th>
<th>tripid</th>
<th>astext text</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>163</td>
<td>POINT (2997192.88890412 5839689.91506735)@2007-05-28 06:00:00.001+00, POI</td>
</tr>
<tr>
<td>6</td>
<td>165</td>
<td>POINT (2985654.50641456 5848965.14626724)@2007-05-28 16:09:12.824+00, POI</td>
</tr>
<tr>
<td>8</td>
<td>235</td>
<td>POINT (3010311.09650771 5836055.09743228)@2007-05-28 07:19:01.864+00, POI</td>
</tr>
<tr>
<td>8</td>
<td>237</td>
<td>POINT (2997958.79103681 5837131.44989043)@2007-05-28 16:05:50.982+00, POI</td>
</tr>
<tr>
<td>8</td>
<td>241</td>
<td>POINT (2997958.79103681 5837131.44989043)@2007-05-29 17:11:03.19+00, POI</td>
</tr>
<tr>
<td>8</td>
<td>247</td>
<td>POINT (3010311.09650771 5836055.09743228)@2007-05-30 07:02:57.848+00, POI</td>
</tr>
<tr>
<td>9</td>
<td>280</td>
<td>POINT (3001526.14852942 5837101.46991784)@2007-05-31 21:15:07.6+00, POI</td>
</tr>
<tr>
<td>9</td>
<td>290</td>
<td>POINT (3008321.78980041 5845720.9362808)@2007-05-31 22:47:38.444+00, POI</td>
</tr>
<tr>
<td>10</td>
<td>323</td>
<td>POINT (2993181.49144001 5853123.75533338)@2007-05-30 17:09:18.5+00, POI</td>
</tr>
<tr>
<td>10</td>
<td>325</td>
<td>POINT (2995709.23953211 5838172.58057013)@2007-05-31 07:01:19.697+00, POI</td>
</tr>
<tr>
<td>13</td>
<td>422</td>
<td>POINT (3020510.76271993 5835681.48725136)@2007-05-28 06:32:00.131+00, POI</td>
</tr>
<tr>
<td>13</td>
<td>424</td>
<td>POINT (2998220.90876918 5842741.02120682)@2007-05-28 17:21:02.64+00, POI</td>
</tr>
</tbody>
</table>
But also Temporal Alphanumeric Types

t\texttt{float}: \text{speed(Trip)}

t\texttt{bool}: \text{speed(Trip)} > 90
Also Instantaneous Events

Instant: UK road accidents 2012-14


InstantSet: foursquare check-ins

https://support.foursquare.com/
MobilityDB

- A mainstream moving object database (MOD)
- Builds on PostgreSQL and PostGIS
- Developed by a team in Université Libre de Bruxelles
- Meant to be OPEN SOURCE
- Compliant with Open Geospatial Consortium (OGC) standards, in particular the OGC Moving Features Access
Quick Example: Spatial Projection

TABLE Bus ( LineNo integer, TripNo integer, Trip tgeompoint(Sequence, Point, 3812) );
TABLE POI ( POINo integer, Name text, Geo GEOMETRY(3812) );

List the bus lines that traverse Place Louise

SELECT TripNo
FROM Bus B, (SELECT P.Geo FROM POI P WHERE P.Name = 'Place Louise' LIMIT 1) T
WHERE intersects(B.Trip, T.Geo)

The intersects function is index supported, i.e., it is defined as follows

'SELECT $1 OPERATOR(@extschema@.&&) $2 AND @extschema@._intersects($1,$2)'

The & & operator performs a bounding box overlaps index filtering
Quick Example: Spatial Filtering

TABLE Bus ( LineNo integer, TripNo integer, Trip tgeompoint(Sequence, Point, 3812) );

TABLE Network ( LineNo integer, Route GEOMETRY(LINestring, 3812) );

Find all the trips that deviated from their line routes

SELECT TripNo
FROM Bus B, Network N
WHERE B.LineNo = N.LineNo AND NOT contains(st_buffer(N.Route, 20), B.Trip)
Quick Example: Traditional Aggregation

TABLE Bus ( LineNo integer, TripNo integer, Trip tgeompoint(Sequence, Point, 3812) );

Total distance per week travelled by the buses

SELECT SUM( length(Trip) ) travelled, date_part('week', startTimestamp(Trip)) AS week
FROM Bus
GROUP BY week;
Quick Example: Temporal Aggregation

```
TABLE Bus ( LineNo integer, TripNo integer, Trip tgeompoint(Sequence, Point, 3812) );
```

Cumulative distance travelled by the buses at each instant during one week

```
SELECT tsum( cumulativeLength(Trip) ) travelled, date_part('week', startTimestamp(Trip)) AS week
FROM Bus
GROUP BY week;
```

Partial output for 3 days
Quick Example: Spatio-temporal Join

TABLE Bus ( LineNo integer, TripNo integer, Trip tgeompoint(Sequence, Point, 3812) );
TABLE Stops ( StopNo integer, Geo GEOMETRY(POLYGON, 3812) );

List all transit possibilities, i.e., when two buses from different lines meet at a station, so the passenger have the opportunity to change the line

WITH AllStops AS ( SELECT ST_Union(S.Geo) AS Geo FROM Stops S ),
    BusStops AS ( SELECT TripNo, atGeometry(B.Trip, S.Geo) RestrictedRoute
                   FROM Bus B, AllStops S )
SELECT A.TripNo, B.TripNo FROM BusStops A, BusStops B
WHERE A.LineNo < B.LineNo AND A.TripNo < B.TripNo AND
toverlaps(A.RestrictedRoute, B.RestrictedRoute) &= TRUE

The &= (ever equals) operator tests whether a temporal type ever has a given value and results in a Boolean value
MobilityDB Components

- Time types
- Temporal types
- Query functions
- GiST and SP-GiST indexes
- Aggregation functions
Time Types

- In addition to `TimestampTz` we needed 3 additional time types
  - **Period** is a specialized version of `tstzrange`
    
    ```sql
    SELECT period '[2012-01-01 08:00:00, 2012-01-03 09:30:00]';
    ```
  - Similar functionality, more efficient implementation
    - fixed length while `tstzrange` is of variable length
    - empty periods and infinite bounds not allowed
  - **TimestampSet** represents a set of distinct and ordered `timestamptz` values
    
    ```sql
    SELECT timestampset '{2012-01-01 08:00:00, 2012-01-03 09:30:00}';
    ```
  - **PeriodSet** represents a set of disjoint and ordered `period` values
    
    ```sql
    SELECT periodset '{[2012-01-01 08:00:00, 2012-01-01 08:10:00],
        [2012-01-01 08:20:00, 2012-01-01 08:40:00]}';
    ```
Time Types

- **Accessor Functions**: lower, upper, duration, startTimestamp, ...
  
  ```sql
  SELECT timestampN(periodset '{[2012-01-01, 2012-01-03],
  (2012-01-03, 2012-01-05)}', 3);
  -- "2012-01-04"
  ```

- **Operators**: =, <, ..., @>, &&, ..., <<#, &<#, ..., -|-, +, -, *
  
  ```sql
  ```

- **Indexing**: GiST and SP-GiST indexes are supported
  
  ```sql
  CREATE TABLE reservation (ResID integer, RoomID integer, During period);
  CREATE INDEX reservation_during_idx ON reservation USING GIST (during);
  ```
Temporal Types

- Currently tint, tfloat, tbool, ttext, tgeompoint, tgeogpoint
- Come in four durations

![Diagram showing different temporal types](image-url)
Temporal Types

CREATE TABLE Department ( DeptNo integer, DeptName varchar(25), NoEmps tint(Sequence) );
CREATE TABLE Flight ( FlightNo integer, Route tgeogpoint(Sequence,PointZ,4326) );
CREATE TABLE Trips ( CarId integer, TripId integer, Trip tgeompoint );
INSERT INTO Trips VALUES
  (10, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:00:00, Point(2 0)@2012-01-01 08:10:00,
                  Point(2 1)@2012-01-01 08:15:00)}'),
(20, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:05:00, Point(1 1)@2012-01-01 08:10:00,
                  Point(3 3)@2012-01-01 08:20:00]}');
Why not PostGIS Trajectories

- Moving objects represented using LinestringM
- Measure M is not designed to specifically represent time, so total ordering cannot be assumed
- This prevents efficient implementation
  - e.g., binary search cannot be used on general LinestringM values
- Besides temporal point, we also need temporal numbers, temporal Booleans, temporal strings, etc.
From Static to Temporal Types: Lifting

- **Lifted functions**: functions that have static counterparts, but because some of the arguments are temporal, the return is also temporal
  - Static: \( st\text{-intersects}: \text{geometry} \times \text{geometry} \rightarrow \text{bool} \)
  - Lifted: \( t\text{intersects}: \text{tgeompoint} \times \text{geometry} \rightarrow \text{tbool} \)
  \( t\text{intersects}: \text{tgeompoint} \times \text{tgeompoint} \rightarrow \text{tbool} \)

- **Semantics**: result of a lifted function obtained by applying static function to each instant of a temporal value

- **Spatial support** delegated to PostGIS
- We developed a novel generic method for lifting static functions
Temporal Types: Functions

Constructor Functions: easier than input literals

```sql
SELECT tgeompointinst('Point(0 0)', '2001-01-01 08:00:00');
SELECT tintseq(ARRAY[tintinst(2,'2001-01-01 08:00:00'),
                   tintinst(2,'2001-01-01 08:10:00')], true, false);
```

Accessor Functions: startValue, startTimestamp, Instants, ...

```sql
SELECT instantN(tfloat '{[1@2012-01-01, 2@2012-01-02),
                       [3@2012-01-03, 3@2012-01-04 , 5@2012-01-05]}', 3);
-- "3@2012-01-03"
```
Temporal Types: Functions

Spatiotemporal functions: twCentroid, nearestApproachInstant, ...

```
SELECT nearestApproachDistance( tgeompoint '[[Point(0 0)@2012-01-02, Point(1 1)@2012-01-04, Point(0 0)@2012-01-06)',
geometry 'Linestring(2 2,2 1,3 1)');
-- "1"
```

Projection functions: atValue, atRange, atMax, atTimestamp, ...

```
SELECT astext(atGeometry(
tgeompoint '[[Point(0 0)@2012-01-01, Point(3 3)@2012-01-04)',
geometry 'Polygon((1 1,1 2,2 2,2 1,1 1))']));
-- "{"[POINT(1 1)@2012-01-02, POINT(2 2)@2012-01-03]"}"
```
Temporal Types: Functions

- **Difference Functions**: minusValue, minusMax, minusPeriod, …
- **Comparison Operators**: =, <, …, (B-Tree), #=, #<, … (temporal comparison), &=, @= (temporal type to Boolean)
- **Temporal Operators**: +, -, *, / for temporal integers and floats
- **Bounding Box Operators**
  - <<, >>, &<, &>: value dimension for tint and tfloat, x-dimension for temporal points
  - ||, |>, |<, and |&>, y-dimension
  - |<, />, &\<, and /&> z-dimension
  - <#, #>, #<&, and #&> time dimension
- **Distance Operators**: |=|, <-
- **Casting**: tfloat::tint, tgeogpoint::tgeompoint
- **Spatial Relationships**: intersects, relate, …, tintersects, trelate, …
GiST and SP-GiST Indexes

Temporal types support both GiST and SP-GiST indexes

```
CREATE INDEX Department_NoEmps_Gist_Idx ON Department USING Gist(NoEmps);
CREATE INDEX Trips_Trip_SPGist_Idx ON Trips USING SPGist(Trip);
```

Indexes store the **bounding box** for the temporal types

- period for tbool and ttext (1D)
- box for tint and tfloat (2D)
- gbox for tgeompoint and tgeogpoint (4D)

Indexes can accelerate queries involving the following operators

- `<<, &<, ..., <<|, &<|, ..., &</, <</, ..., for the value/spatial dimension`
- `&<#, <<#, .., for the time dimension`
- `&&, @>, <@, ~=, consider as many shared dimensions`
SP-GiST Indexes

- To implement SP-GiST, the bounding box is transformed into a higher dimensional point
  - 2D point to represent a period
  - 4D point to represent a box
  - 8D point to represent a gbox
- We reused approach from SP-GiST indexes for BOX type in PostgreSQL
- After that we proposed patches for SP-GiST indexes for 2D/3D Geometry (PostGIS V2.5) and ND Geometry (PostGIS V3.0)
Aggregation Functions

Three types of aggregations

- Regular aggregation functions
  
  ```sql
  SELECT COUNT(Trip) FROM Bus;
  ```

- Temporal aggregation functions: result in a temporal type
  
  ```sql
  SELECT TCOUNT(Trip) FROM Bus;
  ```

- Sliding window aggregation functions: interval parameter, result in a temporal type
  
  ```sql
  SELECT WMAX(speed(Trip), interval '10 minutes') FROM Bus;
  ```
Temporal Aggregation: Parallel Execution

Compute how many cars were active at each period in table Periods

```
EXPLAIN ANALYZE SELECT P.PeriodID, COUNT(*), TCOUNT(atPeriod(T.Trip, P.Period))
FROM Trips T, Periods P
WHERE T.Trip & P.Period
GROUP BY P.PeriodID
ORDER BY P.PeriodID;
```

Finalize GroupAggregate  
(cost=382307.54..382359.43 rows=100 width=20) (actual time=174195.681..174195.869 rows=100 loops=1)

<table>
<thead>
<tr>
<th>Group Key: p.periodid</th>
</tr>
</thead>
</table>
| Gather Merge  
  (cost=382307.54..382355.43 rows=400 width=20) (actual time=174195.672..174198.738 rows=500 loops=1)  |
|  Workers Planned: 4 |
|  Workers Launched: 4 |
|  Sort  
  (cost=381307.48..381307.73 rows=100 width=20) (actual time=174187.012..174187.017 rows=100 loops=5)  |
|  Sort Key: p.periodid |
|  Sort Method: quicksort Memory: 32kB |
|  Worker 0: Sort Method: quicksort Memory: 32kB |
|  Worker 1: Sort Method: quicksort Memory: 32kB |
|  Worker 2: Sort Method: quicksort Memory: 32kB |
|  Worker 3: Sort Method: quicksort Memory: 32kB |
|  Partial HashAggregate  
  (cost=381303.16..381304.16 rows=100 width=20) (actual time=174186.953..174186.968 rows=100 loops=5)  |
|  Group Key: p.periodid |
|  Nested Loop  
  (cost=0.00..373995.06 rows=730810 width=193) (actual time=4.044..154010.621 rows=207994 loops=5)  |
|  Join Filter: (t.trip & p.period)  |
|  Rows Removed by Join Filter: 5650806  |
|  Parallel Seq Scan on trips t  
  (cost=0.00..63400.81 rows=73081 width=165) (actual time=0.091..120.061 rows=58588 loops=5)  |
|  Seq Scan on periods p  
  (cost=0.00..3.00 rows=100 width=28) (actual time=0.001..0.010 rows=100 loops=292940)  |

Planning Time: 0.158 ms
Execution Time: 174198.888 ms
**Temporal Aggregation: Partitioning**

```sql
CREATE TABLE Trips
(
    CarId integer NOT NULL,
    TripId integer NOT NULL,
    TripDate date,
    Trip tgeompoint,
    Traj geometry,
    PRIMARY KEY (CarId, TripId, TripDate),
    FOREIGN KEY (CarId) REFERENCES Cars (CarId)
) PARTITION BY LIST(TripDate);

CREATE TABLE Trips_2007_05_27 PARTITION OF Trips FOR VALUES IN ('2007-05-27');
CREATE TABLE Trips_2007_05_28 PARTITION OF Trips FOR VALUES IN ('2007-05-28');
CREATE TABLE Trips_2007_05_29 PARTITION OF Trips FOR VALUES IN ('2007-05-29');
CREATE TABLE Trips_2007_05_30 PARTITION OF Trips FOR VALUES IN ('2007-05-30');
...```
Temporal Aggregation: Partitioning

Finalize GroupAggregate (cost=7914.89..7966.78 rows=100 width=20) (actual time=8.084..8.084 rows=0 loops=1)
  Group Key: p.periodid
  -> Gather Merge (cost=7914.89..7962.78 rows=400 width=20) (actual time=8.083..8.798 rows=0 loops=1)
     Workers Planned: 4
     Workers Launched: 4
     -> Sort (cost=6914.83..6915.08 rows=100 width=20) (actual time=0.019..0.019 rows=0 loops=5)
        Sort Key: p.periodid
        Sort Method: quicksort  Memory: 25kB
        Worker 0: Sort Method: quicksort  Memory: 25kB
        Worker 1: Sort Method: quicksort  Memory: 25kB
        Worker 2: Sort Method: quicksort  Memory: 25kB
        Worker 3: Sort Method: quicksort  Memory: 25kB
  -> Partial HashAggregate (cost=6910.51..6911.51 rows=100 width=20) (actual time=0.002..0.002 rows=0 loops=5)
     Group Key: p.periodid
     -> Nested Loop (cost=0.00..6754.51 rows=15600 width=60) (actual time=0.001..0.001 rows=0 loops=5)
        Join Filter: (t.trip & p.period)
           -> Parallel Append (cost=0.00..124.51 rows=1560 width=32) (actual time=0.001..0.00 1 rows=0 1)
              -> Parallel Seq Scan on trips_2007_05_27 t (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_05_28 t_1 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_05_29 t_2 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_05_30 t_3 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_06_01 t_4 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_06_02 t_5 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_06_03 t_6 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Parallel Seq Scan on trips_2007_06_04 t_7 (cost=0.00..14.59 rows=459 width=32) (actual)
              -> Seq Scan on periods p (cost=0.00..3.00 rows=100 width=28) (never executed)

Planning Time: 7.231 ms
Execution Time: 9.016 ms
3D Temporal Points

- Builds upon PostGIS 3D Geometry and Geography
- Many functions: length, cumulativeLength, speed, expandSpatial, expandTemporal, twCentroid, azimuth, tintersects, …
- gbox is used for representing the bounding box
- Supported in both GiST and SP-GiST
MobilityDB Roadmap

- Development started locally at ULB
  - 2,374 functions, 62,825 lines of C, 19,070 lines of SQL, 19,939 lines of regression tests
- Promising results from robustness tests, performance tests, benchmarking wrt PostGIS trajectories and Secondo
- Currently finalizing planner statistics and selectivity functions
- Development planned to move to github soon
- Stable release delivered open source as soon as it is ready
- Several extensions are being explored
  - Network-constrained trajectories with pgRouting
  - Visualization with QGIS
  - Big mobility data with Postgres-XL
  - Split lists for optimized temporal aggregation
MobilityDB Roadmap

Thanks for listening!

Questions?