Query Plan Analysis for Selected BerlinMOD/R Queries

Christian Düntgen*

January 24, 2011

Abstract

This article shows how executable Secondo query plans for a subset of the BerlinMOD/R benchmark queries relate to different representation schemas for moving object data. It covers the *Compact Representation* (CR), *Unit Representation* (UR), *Hybrid Represenation* (with ($HR^{\oplus}$) and without ($HR^{\ominus}$) summary fields), each for both object semantics, the *Object Based Approach* (OBA) and the *Trip Based Approach* (TBA), introduced by the BerlinMOD benchmark.

1 Introduction

This article analyses several Secondo query plans used in the experiments reported in [1]. The plans implement queries from the BerlinMOD benchmark [2, 3] using different representation schemas for the moving object data (MOD), as introduced in [1]. You can freely download preliminary versions of all mentioned reports and articles from the BerlinMOD website [4].

All experiments have been executed using the Secondo DBMS [5]. Hence, in the remainder of this article, all query plans are given in Secondo executable language. While the articles [6–10] provide an introduction to the Secondo extensible DBMS and give further examples of its use, the "Secondo User's Guide" and the system's self-explanatory facilities (commands `list types`, `list operators`) provide detailed information on how to use the system and its extension modules (called "algebras"), and on all the data types and operators used in the queries presented here. You can download Secondo for free from the according project website.

For each query, we provide one section (Sections 2–6). Each section provides three subsections:

1. The query in literal English, followed by the according SQL-statements for the OBA and TBA.
2. Eight query plans in following order: OBA/CR, OBA/UR, OBA/$HR^{\oplus}$, OBA/$HR^{\ominus}$; TBA/CR, TBA/UR, TBA/$HR^{\oplus}$, TBA/$HR^{\ominus}$.
3. A literal description of the eight plans including an analyses helping to understand the query response times.

We analyze the query plans to show, how the different representations affect the query plans and the observed response times. Within the analyses subsections, we use the following abbreviations: `QL` for `QueryLicences`, `QP` for `QueryPeriods`, and `QI` for `QueryInstants`. We also shorten all relation names starting with "`data`", so "`dataXY`" becomes "`XY`".

The MBRs used as keys within R-tree indexes have been scaled in order to avoid skew from diffenent range sizes on the 3 dimension scales. The function scaleBox3D(Rect3: R), for a given 3D rectangle R returns the properly scaled 3D-rectangle to be used with the R-tree indexes.

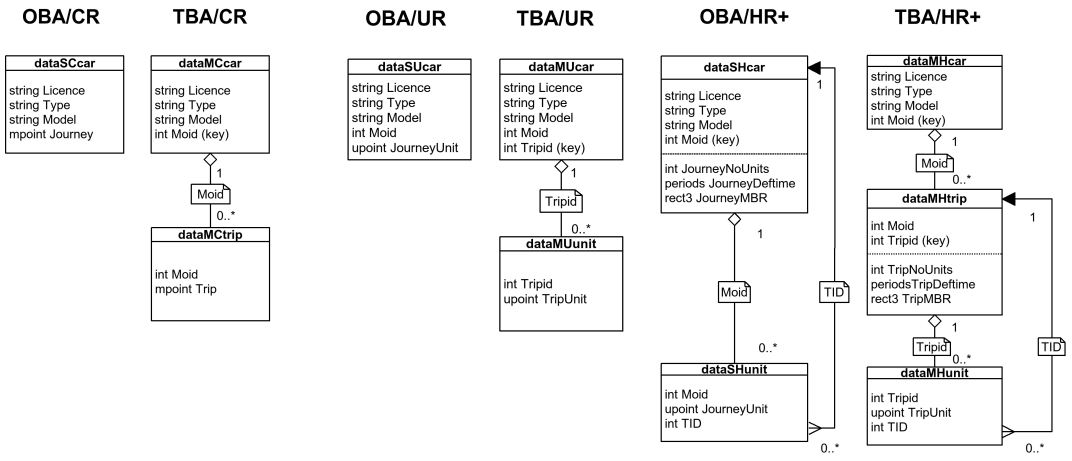
Within the queries and query plans, we use database schemas as visualized in Figure 1.

2 BerlinMOD/R Query Q3

2.1 Query Q3

Q3: Report positions of all cars from `QueryLicences1` at each instant from `QueryInstants1`.

*The author is with the Faculty of Mathematics and Computer Science, University of Hagen.
E-Mail: christian.duentgen@fernuni-hagen.de



(a) Compact Representation (b) Unit Representation (c) Hybrid Representation

Figure 1: BerlinMOD conceptual database schemas for the different MOD semantics and representation models.

2.1.1 SQL-Query for OBA

```
SELECT LL.Licence AS Licence, II.Instant AS Instant,
       val(C.Trip atinstant II.Instant) AS Pos
FROM dataSCcar C, QueryLicences1 LL, QueryInstants1 II
WHERE C.Licence = LL.Licence AND C.Trip present II.Instant;
```

2.1.2 SQL-Query for TBA

```
SELECT LL.Licence AS Licence, II.Instant AS Instant,
       val(C.Trip atinstant II.Instant) AS Pos
FROM dataMCcar C, QueryLicences1 LL, QueryInstants1 II
WHERE C.Licence = LL.Licence AND C.Trip present II.Instant;
```

2.2 Executable Secondo Plans for Q3

2.2.1 OBA/CR

```
let OBACRres003 =
  QueryLicences feed {LL} head[10]
  loopjoin[dataSCcar_Licence_btree dataSCcar exactmatch[.Licence_LL]]
  QueryInstants feed {II} head[10]
  product
  projectextend[; Licence: .Licence_LL, Instant: .Instant_II,
                  Pos: val(.Journey atinstant .Instant_II)]
  consume;
```

2.2.2 OBA/UR

```
let OBAURres003 =
  QueryLicences feed {LL} head[10]
  loopjoin[dataSUcar_Licence_btree dataSUcar exactmatch[.Licence_LL]]
  QueryInstants feed {II} head[10]
  symmjoin[.JourneyUnit present ..Instant_II]
  projectextend[; Licence: .Licence_LL, Instant: .Instant_II,
                  Pos: val(.JourneyUnit atinstant .Instant_II)]
  consume;
```

2.2.3 OBA/HR$^{\oplus}$

```
let OBAHRres023 =
  QueryLicences feed {LL} head[10]
  loopsel[fun(t1: TUPLE)
    QueryInstants feed {II} head[10]
    dataSHcar_Licence_btree dataSHcar exactmatch[attr(t1,Licence_LL)]
    symmjoin[.Instant_II inside ..JourneyDeftime]
  ]
  loopjoin[fun(t2: TUPLE)
    dataSHunit_Moid_btree dataSHunit exactmatch[attr(t2,Moid)]
    filter[.JourneyUnit present attr(t2,Instant_II)]
    project[JourneyUnit]
  ]
  projectextend[Licence; Instant: .Instant_II,
                Pos: val(.JourneyUnit atinstant .Instant_II)]
  consume;
```

2.2.4 OBA/HR$^{\ominus}$

```
let OBAHRres003 =
  QueryLicences feed {LL} head[10]
  loopsel[dataSHcar_Licence_btree dataSHcar exactmatch[.Licence_LL]]
  loopjoin[
    dataSHunit_Moid_btree dataSHunit exactmatch[.Moid]
    project[JourneyUnit]]
  QueryInstants feed {II} head[10]
  symmjoin[.JourneyUnit present ..Instant_II]
  projectextend[Licence; Instant: .Instant_II,
                Pos: val(.JourneyUnit atinstant .Instant_II)]
  consume;
```

2.2.5 TBA/CR

```
let TBACRres003 =
  QueryLicences feed head[10] {LL}
  loopsel[dataMCcar_Licence_btree dataMCcar exactmatch[.Licence_LL]
          project[Licence, Moid] {LL}]
  loopjoin[dataMCtrip_Moid_btree dataMCtrip exactmatch[.Moid_LL]]
  QueryInstants feed head[10] {II}
  symmjoin[.Trip present ..Instant_II]
  projectextend[; Licence: .Licence_LL, Instant: .Instant_II,
                Pos: val(.Trip atinstant .Instant_II)]
  consume;
```

2.2.6 TBA/UR

```
let TBAURres003 =
  QueryLicences feed head[10] {LL}
  loopjoin[ fun(t: TUPLE)
    dataMUcar_Licence_btree dataMUcar exactmatch[attr(t,Licence_LL)]
    loopsel[ fun(t1: TUPLE)
             dataMUunit_Tripid_btree dataMUunit exactmatch[attr(t1,Tripid)]
           ]
  ]
  QueryInstants feed head[10] {II}
  symmjoin[.TripUnit present ..Instant_II]
  projectextend[; Licence: .Licence_LL, Instant: .Instant_II,
                Pos: val(.TripUnit atinstant .Instant_II)]
  consume;
```

2.2.7 TBA/HR$^{\oplus}$

```
let TBAHRres003 =
  QueryLicences feed head[10] {LL}
  loopsel[
    dataMHcar_Licence_btree dataMHcar exactmatch[.Licence_LL]
    project[Moid, Licence] ]
  loopjoin[ dataMHtrip_Moid_btree dataMHtrip exactmatch[.Moid] remove[Moid] ]
  QueryInstants feed head[10] {II}
  symmjoin[..Instant_II inside .TripDeftime]
  loopsel[ fun(t1: TUPLE)
           dataMHunit_Tripid_btree dataMHunit exactmatch[attr(t1,Tripid)]
           filter[.TripUnit present attr(t1,Instant_II)]
           projectextend[; Licence: attr(t1,Licence),
              Instant: attr(t1,Instant_II),
              Pos: val(.TripUnit atinstant attr(t1,Instant_II))
           ]
  ]
  consume;
```

2.2.8 TBA/HR$^{\ominus}$

```
let TBAHRres023 =
  QueryLicences feed head[10] {LL}
  loopsel[ dataMHcar_Licence_btree dataMHcar exactmatch[.Licence_LL]
           project[Moid, Licence] ]
  loopjoin[ dataMHtrip_Moid_btree dataMHtrip exactmatch[.Moid]
            remove[Moid] ]
  QueryInstants feed head[10] {II}
  product
  loopsel[ fun(t1: TUPLE)
    dataMHunit_Tripid_btree dataMHunit exactmatch[attr(t1,Tripid)]
    filter[.TripUnit present attr(t1,Instant_II)]
    projectextend[; Licence: attr(t1,Licence),
                    Instant: attr(t1,Instant_II),
                    Pos: val(.TripUnit atinstant attr(t1,Instant_II))
    ]
   ]
  consume;
```

2.3 Analyses for Q3

We used the following plans for this temporal point query on known objects: *OBA/CR:* For `QL1`, select according tuples from `SCcar` using a B-tree index. Then for each instant from `QI1` calculate the vehicles' position. *OBA/UR:* For `QL1`, select the JourneyUnits from `SUcar` using a B-tree index. Join each unit with all instants from `QI1` selecting those pairs, where the unit is present at the given instant. *OBA/HR$^{\ominus}$:* For `QL1`, select trips from `SHcar`, then units from `SHunit` using the B-tree indexes, joins units with `QI1`, selecting TripUnits present at an query instant and evaluate the temporal function. *OBA/HR$^{\oplus}$:* For `QL1`, select trips from `SHcar` and join with `QL1`, selecting pairs whose query instant is contained by the TripDeftime summary (but as the summary is for the complete journey, this is always the case). Then fetch all units from `SHunit` to refine the temporal containment and evaluate the temporal function. *TBA/CR:* For `QL1` retrieve all Moids from `MCcar` using a B-tree index, then retrieve all TripUnits belonging to each found Moid from `MCtrip` using a B-tree index. Join with all instants from `QL1`, where the TripUnit is present at the given instant. *TBA/UR:* For `QL1` retrieve all TripIds from `MUcar` and then all TripUnits belonging to the according TripIds (both times using B-tree indexes). Join with all instants from `QL1`, where the TripUnit is present at the given instant. *TBA/HR$^{\ominus}$:* For `QL1` select the Moids from `MHcar`, then retrieve all according TripIds from `MHtrip` and finally the TripUnits from `MHunit` using the three according B-trees. Then join with all instants from `QL1`, where the TripUnit is present at the given instant. *TBA/HR$^{\oplus}$:* For `QL1` select all Moids from `MHcar` and get all according trips descriptions from `MHtrip` using two B-trees. Then join with the instants from `QL1` selecting TripIds whose TripDeftime contains the given query instant. Retrieve the TripUnits belonging to the remaining TripIds using a B-tree and restrict to those tuples, where the TripUnit is present at the query instant

(this is necessary as a trip within `MHtrip` theoretically may contain definition gaps).

Q3 shows some interesting results. First, it is simple to understand, that the CR is superior to the UR and HR, as for each given licence, one just extracts the car's position at each of the query instants. Second, as basically the same access sequence (first id, then time) is also used for the UR and HR, also the superiority of the ID/TMP to the SPTMP is clear. Third, using summary fields is good for all HR-variants except the OBA/HR, where the prefiltering adds complexity, but does not drop any candidates (because prefiltering is always positive).

3 BerlinMOD/R Query Q6

3.1 Query Q6

Q6: Report the licences of "trucks", that ever have approached each other to 10m or below.

3.1.1 SQL-Query for OBA

```
SELECT V1.Licence AS Licence1, V2.Licence AS Licence2
FROM dataSCcar V1, dataSCcar V2
WHERE V1.Licence < V2.Licence AND V1.Type = "truck" AND V2.Type = "truck"
  AND sometimes(distance(V1.Journey,V2.Journey) <= 10.0);
```

3.1.2 SQL-Query for TBA

```
SELECT DISTINCT T1.Licence AS Licence1, T2.Licence AS Licence2
FROM dataMCtrip T1, dataMCcar C1, dataMtrip T2, dataMCcar C2
WHERE T1.Licence < T2.Licence AND T1.Licence = C1.Licence
      AND T2.Licence = C2.Licence AND C1.Type = "truck"
      AND C2.Type = "truck"
      AND sometimes(distance(T1.Trip, T2.Trip) <= 10.0);
```

3.2 Executable Secondo Plans for Q6

3.2.1 OBA/CR

```
let OBACRres006 =
  dataSCcar feed {V1} filter[.Type_V1 = "truck"]
  dataSCcar feed {V2} filter[.Type_V2 = "truck"]
  symmjoin[.Licence_V1 < ..Licence_V2]
  filter[ minimum(distance(.Journey_V1, .Journey_V2)) <= 10.0 ]
  projectextend[ ; Licence1: .Licence_V1,
                   Licence2: .Licence_V2 ]
  consume;
```

3.2.2 OBA/UR

```
let OBAURres086tmpTrucks =
  dataSUcar feed
  filter[.Type = "truck"]
  extend[MBR: enlargeRect(bbox(.JourneyUnit), 5.0, 5.0, 0.0)]
  consume;

let OBAURres086tmpTrucks_MBR_sptmpobj =
  OBAURres086tmpTrucks feed
  extend[TID: tupleid(.)]
  sortby[MBR]
  bulkloadrtree[MBR];

let OBAURres086 =
  OBAURres086tmpTrucks feed
  loopsel[fun(t1: TUPLE)
    OBAURres086tmpTrucks_MBR_sptmpobj windowintersectsS[scaleBox3D(attr(t1,MBR))]
```

```
    sort rdup
    OBAURres086tmpTrucks gettuples
    filter[attr(t1,Moid) < .Moid]
    filter[minimum(distance(attr(t1,JourneyUnit), .JourneyUnit)) <= 10.0]
    projectextend[ ; Licence1: attr(t1,Licence), Licence2: .Licence,
                     Moid1: attr(t1,Moid), Moid2: .Moid ]
    sortby[Moid1, Moid2] krdup[Moid1, Moid2]
    project[Licence1, Licence2]
  ]
  consume;

delete OBAURres086tmpTrucks;
delete OBAURres086tmpTrucks_MBR_sptmpobj;
```

3.2.3 OBA/HR$^{\oplus}$

```
let OBAHRres006 =
  dataSHcar feed filter[.Type = "truck"] {V1}
  extend[QRectMBR: enlargeRect(.JourneyMBR_V1, 10.0, 10.0, 0.0)]
  loopsel[ fun(t1: TUPLE)
    dataSHcar_JourneyMBR_sptmptrp
      windowintersectsS[scaleBox3D(attr(t1,QRectMBR))]
    sort rdup dataSHcar gettuples {V2}
    filter[(.Type_V2 = "truck") and (.Moid_V2 > attr(t1,Moid_V1))]
    loopsel[ fun(t2: TUPLE)
      dataSHunit_Moid_btree dataSHunit exactmatch[attr(t1,Moid_V1)]
      projectextend[JourneyUnit; QRect:
                                  enlargeRect(bbox(.JourneyUnit), 10.0, 10.0, 0.0)]
      {VV1}
      dataSHunit_Moid_btree dataSHunit exactmatch[attr(t2,Moid_V2)]
      projectextend[JourneyUnit; QRect:   bbox(.JourneyUnit)]
      {VV2}
      symmjoin[.QRect_VV1 intersects ..QRect_VV2]
      filter[sometimes(distance(.JourneyUnit_VV1,.JourneyUnit_VV2) <= 10.0)]
      head[1]
      projectextend[; Moid1: attr(t1,Moid_V1),
                      Moid2: attr(t2,Moid_V2),
                      Licence1: attr(t1,Licence_V1),
                      Licence2: attr(t2,Licence_V2)]
    ]
  ]
  sortby[Moid1, Moid2] krdup[Moid1, Moid2]
  project[Licence1, Licence2] consume;
```

3.2.4 OBA/HR$^{\ominus}$

```
let OBAHRres026 =
  dataSHcar feed filter[.Type = "truck"] {V1}
  loopsel[ fun(t1: TUPLE)
    dataSHunit_Moid_btree dataSHunit exactmatch[attr(t1,Moid_V1)]
    extend[MBR: enlargeRect(bbox(.JourneyUnit), 10.0, 10.0, 0.0)] {V2}
    loopsel[ fun(t2: TUPLE)
      dataSHunit_JourneyUnit_sptmpobj
        windowintersectsS[scaleBox3D(attr(t2,MBR_V2))]
      dataSHunit gettuples {V3}
      filter[.Moid_V3 # attr(t2,Moid_V2)]
      filter[sometimes(distance(attr(t2,JourneyUnit_V2),
                                .JourneyUnit_V3) <= 10.0)]
      dataSHcar gettuples2[TID_V3]
      filter[.Type = "truck"]
      projectextend[;Moid1: attr(t1,Moid_V1), Moid2: .Moid,
```

```
                    Licence1: attr(t1,Licence_V1), Licence2: .Licence]
      sort rdup
    ]
  ]
  sortby[Moid1, Moid2] krdup[Moid1, Moid2]
  project[Licence1, Licence2] consume;
```

3.2.5 TBA/CR

```
let TBACRres006BBoxMtrip =
  dataMCcar feed filter[.Type = "truck"]
  project[Licence, Moid]
  loopsel[fun(t: TUPLE)
    dataMCtrip_Moid_btree dataMCtrip exactmatch[attr(t,Moid)]
    projectextend[Trip, Moid; BBox: bbox(.Trip), Licence: attr(t,Licence)]
    projectextend[Moid, Licence, Trip, BBox; Box:
    rectangle2((minD(.BBox,1) - 5.0),
      (maxD(.BBox,1) + 5.0),
      (minD(.BBox,2) - 5.0),
      (maxD(.BBox,2) + 5.0))
    ]
  ]
  consume;

let TBACRres006BBoxMtrip_BBox_sptmpobj =
  TBACRres006BBoxMtrip feed
  projectextend[Moid, Licence, Trip ; TID: tupleid(.),
    Box: scaleBox3D(enlargeRect(.BBox, 5.0, 5.0, 0.0))]
  sortby[Box]
  bulkloadrtree[Box];

let TBACRres006 =
  TBACRres006BBoxMtrip feed
  loopsel[ fun(t1: TUPLE)
    TBACRres006BBoxMtrip_BBox_sptmpobj
      windowintersectsS[scaleBox3D(enlargeRect(attr(t1,BBox),5.0,5.0,0.0))]
    sort rdup
    TBACRres006BBoxMtrip gettuples
    filter[ attr(t1,Moid) < .Moid ]
    filter[ everNearerThan(attr(t1,Trip), .Trip, 10.0) ]
    projectextend[ ; Licence1: attr(t1,Licence),
                     Licence2: .Licence ]
  ]
  sort rdup consume;

delete TBACRres006BBoxMtrip;
delete TBACRres006BBoxMtrip_BBox_sptmpobj;
```

3.2.6 TBA/UR

```
let TBAURres006BBoxMtrip =
  dataMUcar feed filter[.Type = "truck"]
  project[Moid, Licence, Tripid]
  loopsel[ fun(t: TUPLE)
    dataMUunit_Tripid_btree dataMUunit exactmatch[attr(t,Tripid)]
    projectextend[TripUnit ; Moid: attr(t, Moid), Licence: attr(t, Licence),
                             MBR: enlargeRect(bbox(.TripUnit), 5.0, 5.0, 0.0)]
  ]
  consume;

let TBAURres006BBoxMtrip_MBR_sptmpobj =
```

```
  TBAURres006BBoxMtrip feed
  extend[TID: tupleid(.)]
  sortby[MBR]
  bulkloadrtree[MBR];

let TBAURres006 =
  TBAURres006BBoxMtrip feed
  loopsel[fun(t1: TUPLE)
    TBAURres006BBoxMtrip_MBR_sptmpobj windowintersectsS[scaleBox3D(attr(t1,MBR))]
    sort rdup
    TBAURres006BBoxMtrip gettuples
    filter[sometimes(distance(attr(t1,TripUnit),.TripUnit)<10.0)]
    projectextend[; Moid_C1: attr(t1,Moid), Moid_C2: .Moid,
                    Licence1: attr(t1,Licence), Licence2: .Licence ]
    sortby[Moid_C1, Moid_C2] krdup[Moid_C1, Moid_C2]
    project[Licence1, Licence2]
  ]
  consume;

delete TBAURres006BBoxMtrip;
delete TBAURres006BBoxMtrip_MBR_sptmpobj;
```

3.2.7 TBA/HR$^{\oplus}$

```
let TBAHRres026Candidates =
  dataMHcar feed filter[.Type = "truck"]
  project[Moid, Licence]
  loopjoin[ dataMHtrip_Moid_btree dataMHtrip
    exactmatch[.Moid] project[Tripid, TripMBR] ]
  projectextend[Moid, Licence, Tripid; Ext5TripMBR:
                                        enlargeRect(.TripMBR, 5.0, 5.0, 0.0)]
  consume;

# could create an rtree index for spatial selfjoin here!
# however, cardinalities are not too worse (trip level: ~29200 trips for trucks)
let TBAHRres026CandidatePairs =
  TBAHRres026Candidates feed {C1}
  TBAHRres026Candidates feed {C2}
  symmjoin[      (.Moid_C1 < ..Moid_C2)
           and (.Ext5TripMBR_C1 intersects ..Ext5TripMBR_C2)]
  consume;

let TBAHRres026 =
  TBAHRres026CandidatePairs feed
  loopsel[fun(t1: TUPLE)
    dataMHunit_Tripid_btree dataMHunit exactmatch[attr(t1,Tripid_C1)]
    extend[MBR: enlargeRect(bbox(.TripUnit), 5.0, 5.0, 0.0)] {C1}
    dataMHunit_Tripid_btree dataMHunit exactmatch[attr(t1,Tripid_C2)]
    extend[MBR: enlargeRect(bbox(.TripUnit), 5.0, 5.0, 0.0)] {C2}
    symmjoin[.MBR_C1 intersects ..MBR_C2]
    filter[sometimes(distance(.TripUnit_C1,.TripUnit_C2)<10.0)]
    projectextend[; Moid1: attr(t1,Moid_C1), Moid2: attr(t1,Moid_C2),
                    Licence1: attr(t1,Licence_C1), Licence2: attr(t1,Licence_C2)]
  ]
  sortby[Moid1, Moid2]
  krdup[Moid1, Moid2]
  project[Licence1, Licence2]
  consume;

delete TBAHRres026Candidates;
delete TBAHRres026CandidatePairs;
```

3.2.8 TBA/HR$^{\ominus}$

```
let TBAHRres006BBoxMtrip =
  dataMHcar feed filter[.Type = "truck"]
  project[Moid, Licence]
  loopsel[ fun(t: TUPLE)
    dataMHtrip_Moid_btree dataMHtrip exactmatch[attr(t,Moid)]
    loopsel[ dataMHunit_Tripid_btree dataMHunit exactmatch[.Tripid]
             projectextend[ TripUnit; MBR:
                                    enlargeRect(bbox(.TripUnit), 5.0, 5.0, 0.0),
                            Moid: attr(t,Moid), Licence: attr(t,Licence)
                          ]
           ]
  ]
  consume;

let TBAHRres006BBoxMtrip_MBR_sptmpobj =
  TBAHRres006BBoxMtrip feed
  extend[TID: tupleid(.)]
  sortby[MBR]
  bulkloadrtree[MBR];

let TBAHRres006 =
  TBAHRres006BBoxMtrip feed
  loopsel[fun(t1: TUPLE)
    TBAHRres006BBoxMtrip_MBR_sptmpobj windowintersectsS[scaleBox3D(attr(t1,MBR))]
    sort rdup
    TBAHRres006BBoxMtrip gettuples
    filter[ attr(t1,Moid) < .Moid ]
    filter[sometimes(distance(attr(t1,TripUnit),.TripUnit) < 10.0)]
    projectextend[; Moid_C1: attr(t1,Moid), Moid_C2: .Moid,
                    Licence1: attr(t1,Licence), Licence2: .Licence ]
    sortby[Moid_C1, Moid_C2]
    krdup[Moid_C1, Moid_C2]
    project[Licence1, Licence2]
  ]
  consume;

delete TBAHRres006BBoxMtrip;
delete TBAHRres006BBoxMtrip_MBR_sptmpobj;
```

3.3 Analyses for Q6

The plans used for this unbounded spatio-temporal join query with unknown identities and without aggregation are: *OBA/CR:* Select trucks from `SCcar`, self-join if minimum distance ever becomes <10m. *OBA/UR:* Create a materialized view with all trucks' JourneyUnits from `SUcar`. Create a spatio-temporal index for the view, with keys buffered by 5m in both spatial dimensions. Do a spatial self-join on this R-Tree, selecting pairs of units from different trucks and that getting nearer than 10m. Remove duplicates. *OBA/HR$^{\ominus}$:* Select trucks from `SHcar`, retrieve according units from `SUunit` using the moid B-tree. Join with all units that intersect with their distance-buffered MBR using the spatio-temporal index. Drop pairs of units belonging to the same truck, evaluate the spatio-temporal condition and drop pairs with non-truck join partners. *OBA/HR$^{\oplus}$:* Select trucks from `SHcar`, use the spatio-temporal index for a distance-buffered self-join on the MBR summaries, filtering pairs of different trucks. Retrieve units for the first and second truck truck from `SHunit`, using the moid index. Do a distance-buffered spatial join between them and evaluate the spatio-temporal condition. Remove dublicates. *TBA/CR:* Select trucks from `MCcar`, retrieve all their trips from `MCtrip` and do a buffered spatial self-join select pairs that have different licences and whose trips come closer than 10m and remove duplicates. *TBA/UR:* Select all trucks from `MUcar` and retrieve the according TripUnits from `MUunit` using a B-tree, store them as a materialized view. Create a spatio-temporal index with keys buffered by 5m in both spatial dimensions. Do a spatial self-join on this R-Tree, filter pairs of units, that have different moids and belong to different

trucks and ever get nearer than 10m. Finally, remove duplicates. *TBA/HR$^{\ominus}$:* An R-tree-indexed relation with distance-buffered MBRs of all trucks' units is created which is used to evaluate a spatial self-join as a prefilter. Then the spatio-temporal condition is applied and duplicate results are removed. *TBA/HR$^{\oplus}$:* Create a relation with the extended TripMBRs for all trucks' trips. Apply a self-join to find all trip pairs with different Moids having overlapping distance-buffered TripMBRs. Then, join in the units for each pair of trips using the B-tree index. Filter the tuples having overlapping buffered MBRs and the evaluate the spatio-temporal predicate. Remove all duplicate results.

Q6 strongly benefits the TBA/CR (small product size for self-join, short mpoint values for parallel scan), and the OBA/UR (high selectivity for the spatial self-join). TBA is superior to OBA (single trips can be retrieved instead of the complete history of an object, reducing the cardinality of intermediate results by orders of magnitude). Maybe surprisingly, the clustering schema clearly is the dominating factor in the TBA, ID/TMP being superior to SPTMP (because unit access is always ordered by Moid (TripId)).

4 BerlinMOD/R Query Q8

4.1 Query for Q8

Q8: What are the overall traveled distances of the vehicles stated by `QueryLicences1` during each period from `QueryPeriods1`?

4.1.1 SQL-Query for OBA

```
SELECT V1.Licence AS Licence, PP.Period AS Period,
  length(V1.Journey atperiods PP.Period) AS Dist
FROM dataSCcar V1, QueryPeriods1 PP, QueryLicences1 LL
WHERE V1.Licence = LL.Licence AND V1.Journey present PP.Period;
```

4.1.2 SQL-Query for TBA

```
SELECT V1.Licence AS Licence, PP.Period AS Period,
       SUM(length(V1.Trip atperiods PP.Period)) AS Dist
FROM dataMCtrip V1, QueryPeriods1 PP, QueryLicences1 LL
WHERE V1.Licence = LL.Licence AND V1.Trip present PP.Period
GROUP BY V1.Licence, PP.Period;
```

4.2 Executable Secondo Plans Q8

4.2.1 OBA/CR

```
let OBACRres008 =
  QueryLicences feed {LL} head[10]
  loopsel[ dataSCcar_Licence_btree dataSCcar exactmatch[.Licence_LL] ]
  QueryPeriods feed project[Period] head[10] {PP}
  product
  projectextend[Licence; Period: .Period_PP,
    Dist: round(length(.Journey atperiods .Period_PP),3)]
  project[Licence, Period, Dist] consume;
```

4.2.2 OBA/UR

```
let OBAURres008 =
  QueryLicences feed {LL} head[10]
  loopsel[ dataSUcar_Licence_btree dataSUcar
    exactmatch[.Licence_LL] ]
  QueryPeriods feed head[10] {PP}
  symmjoin[.JourneyUnit present ..Period_PP]
  projectextendstream[ Moid, Licence, Id_PP, Period_PP
    ; Unit1: .JourneyUnit atperiods .Period_PP ]
  projectextend[Moid, Licence, Id_PP; Period: .Period_PP,
```

```
    Dist1: length(.Unit1)]
  sortby[Moid, Licence, Id_PP, Period]
  groupby[Moid, Licence, Id_PP, Period; Dist:
    round(group feed sum[Dist1],3)]
  project[Licence, Period, Dist]
  consume;
```

4.2.3 OBA/HR$^{\oplus}$

```
let OBAHRres008 =
  QueryLicences feed head[10] project[Licence]
  QueryPeriods feed head[10] {PP}
  product
  loopjoin[ fun(t1: TUPLE)
    dataSHcar_Licence_btree dataSHcar exactmatch[attr(t1,Licence)]
    filter[.JourneyDeftime intersects attr(t1,Period_PP)]
    project[Moid]
    loopjoin[ fun(t2: TUPLE)
      dataSHunit_Moid_btree dataSHunit
      exactmatch[attr(t2,Moid)]
      filter[.JourneyUnit present attr(t1,Period_PP)]
      project[JourneyUnit]
    ]
  ]
  projectextendstream[ Moid, Licence, Id_PP, Period_PP
    ; Unit1: .JourneyUnit atperiods .Period_PP ]
  projectextend[Moid, Licence, Id_PP; Period: .Period_PP,
    Dist1: length(.Unit1)]
  sortby[Moid, Licence, Id_PP, Period]
  groupby[Moid, Licence, Id_PP, Period; Dist:
    round(group feed sum[Dist1],3)]
  project[Licence, Period, Dist]
  consume;
```

4.2.4 OBA/HR$^{\ominus}$

```
let OBAHRres028 =
  QueryLicences feed head[10] project[Licence]
  loopjoin[ fun(t1: TUPLE)
    dataSHcar_Licence_btree dataSHcar exactmatch[attr(t1,Licence)]
    project[Moid]
    loopjoin[ fun(t2: TUPLE)
      dataSHunit_Moid_btree dataSHunit
      exactmatch[attr(t2,Moid)]
      project[JourneyUnit]
    ]
  ]
  QueryPeriods feed head[10] {PP}
  symmjoin[.JourneyUnit present ..Period_PP]
  projectextendstream[ Moid, Licence, Id_PP, Period_PP
    ; Unit1: .JourneyUnit atperiods .Period_PP ]
  projectextend[Moid, Licence, Id_PP; Period: .Period_PP,
    Dist1: length(.Unit1)]
  sortby[Moid, Licence, Id_PP, Period]
  groupby[Moid, Licence, Id_PP, Period; Dist:
    round(group feed sum[Dist1],3)]
  project[Licence, Period, Dist]
  consume;
```

4.2.5 TBA/CR

```
let TBACRres008 =
  QueryPeriods  feed head[10]
  QueryLicences feed head[10] project[Licence]
  product
  loopsel[ fun(t:TUPLE)
    dataMCcar_Licence_btree dataMCcar exactmatch[attr(t,Licence)] {CAR}
    extend[Dist: round(
      dataMCtrip_Moid_btree dataMCtrip exactmatch[.Moid_CAR]
      filter[.Trip present attr(t,Period)]
      projectextend[;L: length(.Trip atperiods attr(t,Period))]
      sum[L], 3)]
    projectextend[; Licence: attr(t,Licence),
                    Period: attr(t,Period), Dist: .Dist]
  ]
  consume;
```

4.2.6 TBA/UR

```
let TBAURres008 =
  QueryPeriods  feed head[10]
  QueryLicences feed head[10] project[Licence]
  product
  loopsel[ fun(t:TUPLE)
    dataMUcar_Licence_btree dataMUcar exactmatch[attr(t,Licence)]
    project[Moid,Tripid]
    sortby[Moid]
    groupby[Moid; Dist:
      round(
        (group feed
          loopsel[fun(t1:TUPLE)
            dataMUunit_Tripid_btree dataMUunit exactmatch[attr(t1,Tripid)]
            filter[.TripUnit present attr(t,Period)]
            extendstream[U: .TripUnit atperiods attr(t,Period)]
            projectextend[; L: length(.U)]
          ]
          sum[L]
         ),
         3
      )
    ]
    extend[Licence: attr(t,Licence), Period: attr(t,Period)]
    project[Licence, Period, Dist]
  ]
  consume;
```

4.2.7 TBA/HR$^{\oplus}$

```
let TBAHRres008 =
  QueryPeriods  feed head[10]
  QueryLicences feed head[10] project[Licence]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
  project[Moid, Licence] sort rdup]
  product
  loopsel[ fun(t:TUPLE)
    round(
      dataMHtrip_Moid_btree  dataMHtrip exactmatch[attr(t,Moid)]
      filter[.TripDeftime intersects attr(t,Period)]
      loopsel[ dataMHunit_Tripid_btree dataMHunit exactmatch[.Tripid] ]
      filter[.TripUnit present attr(t,Period)]
      projectextendstream[Tripid; U: .TripUnit atperiods attr(t,Period)]
      projectextend[; L: length(.U)]
```

```
      sum[L], 3)
    feed namedtransformstream[Dist]
    extend[Id_PR: attr(t,Id), Licence: attr(t,Licence), Period: attr(t,Period)]
  ]
  project[Licence, Period, Dist] consume;
```

4.2.8 TBA/HR$^{\ominus}$

```
delete TBAHRres028;
let TBAHRres028 =
  QueryPeriods  feed head[10]
  QueryLicences feed head[10] project[Licence]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
  project[Moid, Licence] sort rdup]
  product
  loopsel[ fun(t:TUPLE)
    round(
          dataMHtrip_Moid_btree  dataMHtrip exactmatch[attr(t,Moid)]
          loopsel[ dataMHunit_Tripid_btree dataMHunit exactmatch[.Tripid] ]
          filter[.TripUnit present attr(t,Period)]
          projectextendstream[Tripid; U: .TripUnit atperiods attr(t,Period)]
          projectextend[; L: length(.U)]
          sum[L], 3)
    feed namedtransformstream[Dist]
    extend[Id_PR: attr(t,Id), Licence: attr(t,Licence), Period: attr(t,Period)]
  ]
  project[Licence, Period, Dist] consume;
```

4.3 Analyses for Q8

Q8 is a temporal range query with a concluding spatial aggregation. The plans are as follows: *OBA/CR:* For each Licence from `QL1` retrieve the according tuple from `SCcar`. Create the product with `QP1` and for each pair restrict the Journey to the given period and then calculate the distance traveled. *OBA/UR:* For each Licence from `QL1` retrieve all according tuples from `SUcar`. Join with the periods where the JourneyUnit is present and restrict to the given period. Then calculate the traveled distance for each restricted JourneyUnit, group by Moid, Licence and query Period and aggregate the total distances. *OBA/HR$^{\ominus}$:* Selects all cars having a licence from `QL1` and get the according units using a B-tree nested loop join. Then join the units with `QP1` and restrict to the according period before computing the traveled distance by grouping for each Licence and Period. *OBA/HR$^{\oplus}$:* Create the product of `QL1` and `QP1`. Select the cars by B-tree nested loop join on Licence, and eliminate all JourneyDeftimes not intersecting Period. Then, fetch the units for each Moid using B-tree nested loop join. Restricted each unit to Period, group the the units by Licence and Period, and aggregate each group's traveled distance. *TBA/CR:* Create the product of `QP1` and `QL1`. For each combination fetch all trips belonging to a given licence from `MCcar` using the B-tree. For each trip, retrieve the TripUnits from `MCtrip` using the B-tree on TripId, restrict them to the query period and sum up their travel distances. *TBA/UR:* Create the product of `QP1` and `QL1`. For each combination fetch all trips belonging to a given licence from `MUcar` using the Licence B-tree. Use the TripId to retrieve all according TripUnits from `MUunit` using the TripId B-tree. Restrict the units to the query period and aggregate the lengths first for each trip, then for each vehicle. *TBA/HR$^{\ominus}$:* Create the product of `QP1` and `QL1`. For each combination, first fetch all trips belonging to a given licence from `MHcar` using the B-tree, then for each trip the TripUnits from `MHtrip` using the TripId B-tree, restrict the unit to the query period, aggregate a trip's units' lengths and sum up the legths of a vehicle's trips. *TBA/HR$^{\oplus}$:* Create the product of `QP1` and `QL1`. For each combination, first fetch all trips belonging to a given licence from `MHcar` using the B-tree. Select those whose TripDeftime intersects with the query period. Using the TripId fetch the TripUnits from `MHtrip` using the TripId B-tree, restrict the unit to the query period, aggregate a trip's units' lengths and sum up the legths of a vehicle's trips.

Here, the OBA variants are generally faster than the TBA queries. The reason is, that they generally avoid one additional retrieval and aggregation step. While the OBA/HR$^{\oplus}$ cannot profit from summary fields (since almost all cars' Journeys are defined at all times from `QP`, so that the prefiltering step

using the JourneyDeftime summary field does not help much), the TBA/HR$^{\oplus}$ can (because trips can be efficiently prefiltered using TripDeftime). Since data is accessed vehicle by vehicle, the ID/TMP is superior again.

5 BerlinMOD/R Query Q9

5.1 Query Q9

Q9: What is the longest distance traveled by a vehicle during each period from `QueryPeriods`?

5.1.1 SQL-Query for OBA

```
SELECT PP.Period AS Period, MAX(length(V1.Journey atperiods PP)) AS Dist
FROM dataSCcar V1, QueryPeriods PP
WHERE V1.Journey present PP.Period GROUP BY PP.Period;
```

5.1.2 SQL-Query for TBA

```
CREATE VIEW Distances AS
  SELECT SUM(length(V1.Trip atperiods PP.Periods)) AS Length,
         PP.Period AS Period, V1.Licence AS Licence
  FROM dataMCtrip V1, QueryPeriods PP
  WHERE V1.Trip present PP.Period
  GROUP BY PP.Period, V1.Licence;

SELECT Period, MAX(Length) AS Dist
FROM Distances
GROUP BY Period;
```

5.2 Executable Secondo Plans for Q9

5.2.1 OBA/CR

```
let OBACRres009 =
  dataSCcar feed project[Journey] {V1}
  QueryPeriods feed {PP}
  product
  projectextend[Id_PP ; Period: .Period_PP, D:
    length(.Journey_V1 atperiods .Period_PP)]
  sortby[Id_PP, Period, D desc]
  groupby[ Id_PP, Period; Dist: round(group feed max[D],3) ]
  project[Period, Dist] consume;
```

5.2.2 OBA/UR

```
let OBAURres009 =
  QueryPeriods feed extend[MBR: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period))]
  loopjoin[fun(t1: TUPLE)
    dataSUcar_JourneyUnit_tmpobj windowintersectsS[scaleTimeBox2D(attr(t1,MBR))]
    dataSUcar gettuples
    filter[.JourneyUnit present attr(t1,Period)]
    projectextend[Moid; Dist1: length(simplify(
      ((.JourneyUnit atperiods attr(t1,Period)) the_mvalue),
      0.000001))
    ]
    sortby[Moid]
    groupby[Moid; Dist2: round((group feed sum[Dist1]),3)]
  ]
  sortby[Id asc, Period asc, Dist2 desc]
  groupby[Id, Period; Dist: group feed max[Dist2]]
  project[Period, Dist] consume;
```

5.2.3 OBA/HR$^{\oplus}$

```
let OBAHRres009 =
  QueryPeriods feed
  dataSHcar feed
  symmjoin[.Period intersects ..JourneyDeftime]
  loopjoin[ fun(t1: TUPLE)
    dataSHunit_Moid_btree dataSHunit exactmatch[attr(t1,Moid)]
    sortby[JourneyUnit]
    filter[.JourneyUnit present attr(t1,Period)]
    groupby[; Dist2:
      round(length(simplify
      (
        group feed
        projectextendstream[; Unit: (.JourneyUnit atperiods attr(t1,Period))]
        transformstream the_mvalue,
        0.000001
      )),3)
    ]
  ]
  sortby[Id asc, Period asc, Dist2 desc]
  groupby[Id, Period; Dist: group feed max[Dist2]]
  project[Period, Dist] consume;
```

5.2.4 OBA/HR$^{\ominus}$

```
let OBAHRres029 =
  QueryPeriods feed extend[MBR: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period))]
  loopjoin[fun(t1: TUPLE)
    dataSHunit_JourneyUnit_tmpobj
    windowintersectsS[scaleTimeBox2D(attr(t1,MBR))]
    dataSHunit gettuples
    filter[.JourneyUnit present attr(t1,Period)] project[Moid,JourneyUnit]
    sortby[Moid, JourneyUnit]
    groupby[Moid; Dist2:
      round(length(simplify
      (
        group feed
        projectextendstream[; Unit: (.JourneyUnit atperiods attr(t1,Period))]
        transformstream the_mvalue,
        0.000001
      )),3)
    ]
  ]
  sortby[Id asc, Period asc, Dist2 desc]
  groupby[Id, Period; Dist: group feed max[Dist2]]
  project[Period, Dist] consume;
```

5.2.5 TBA/CR

```
let TBACRres009 =
  QueryPeriods feed extend[ PeriodBox: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period))]
  loopjoin[ fun(t:TUPLE)
    dataMCtrip_Trip_tmpuni windowintersectsS[ scaleTimeBox2D(attr(t,PeriodBox)) ]
    sort rdup
    dataMCtrip gettuples
    projectextend[Moid; TripOdo:
      length(.Trip atperiods attr(t,Period))]
    filter[.TripOdo > 0]
```

```
    sortby[Moid asc]
    groupby[Moid; Length: round(group feed sum[TripOdo],3)]
  ]
  groupby[Id, Period; Dist: group feed max[Length]]
  project[Period, Dist] consume;
```

5.2.6 TBA/UR

```
let TBAURres009 =
  QueryPeriods feed extend[ PeriodBox: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period)) ]
  loopjoin[ fun(t:TUPLE)
    dataMUunit_TripUnit_tmpobj
      windowintersectsS[ scaleTimeBox2D(attr(t,PeriodBox)) ]
    sort rdup dataMUunit gettuples
    loopjoin[ dataMUcar_Tripid_btree dataMUcar exactmatch[.Tripid]
      project[Moid] ]
    projectextendstream[Moid; U:
      (.TripUnit atperiods attr(t,Period))]
    projectextend[Moid; Dist: length(.U)]
    sortby[Moid asc]
    groupby[Moid; Length: round(group feed sum[Dist],3)]
  ]
  sortby[Id, Period]
  groupby[Id, Period; Dist: group feed max[Length]]
  project[Period, Dist] consume;
```

5.2.7 TBA/HR$^{\oplus}$

```
let TBAHRres009 =
  QueryPeriods feed extend[ PeriodBox: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period))]
  loopjoin[ fun(t:TUPLE)
    dataMHtrip_TripMBR_tmptrp
      windowintersectsS[ scaleTimeBox2D(attr(t,PeriodBox)) ]
    sort rdup dataMHtrip gettuples
    filter[.TripDeftime intersects attr(t,Period)]
    project[Moid, Tripid]
    loopsel[ fun(t2: TUPLE)
             dataMHunit_Tripid_btree dataMHunit exactmatch[attr(t2,Tripid)]
             projectextend[TripUnit; Moid: attr(t2,Moid)]
             projectextendstream[Moid; Dist:
                ( (.TripUnit atperiods attr(t,Period))
                  use[fun(U: upoint) length(U)]
                  )
             ]
           ]
    sortby[Moid asc]
    groupby[Moid; Length: round(group feed sum[Dist],3)]
  ]
  sortby[Id, Period]
  groupby[Id, Period; Dist: group feed max[Length]]
  project[Period, Dist] consume;
```

5.2.8 TBA/HR$^{\ominus}$

```
let TBAHRres029 =
  QueryPeriods feed extend[ PeriodBox: queryrect2d(
    minimum(.Period)) union queryrect2d(maximum(.Period))]
  loopjoin[ fun(t:TUPLE)
    dataMHunit_TripUnit_tmpobj
      windowintersectsS[ scaleTimeBox2D(attr(t,PeriodBox)) ]
```

```
    dataMHunit gettuples
    filter[deftime(.TripUnit) intersects attr(t,Period)]
    projectextendstream[TID; Dist:
                ( (.TripUnit atperiods attr(t,Period))
                  use[fun(U: upoint) length(U)]
                )
    ]
    sortby[TID asc]
    groupby[TID; Length: round(group feed sum[Dist],3)]
    dataMHtrip gettuples2[TID]
    project[Moid, Length]
]
sortby[Id, Period, Moid]
groupby[Id, Period, Moid; DistV: group feed sum[Length]]
groupby[Id, Period; Dist: group feed max[DistV]]
project[Id, Period, Dist] consume;
```

5.3 Analyses for Q9

This is a temporal range query for unknown objects using an aggregation. *OBA/CR:* Create the product of `SCcar` and `QP`. Extend each tuple with the traveled distance of Journey restricted to the query period. Group by query period to extract the maximum traveled distance. *OBA/UR:* Retrieve all tuples from `SUcar` whose JourneyUnit is present at a query period from `QP` using the temporal R-tree index. Extend each tuple with the length of JourneyUnit restricted to the query period. Group by Moid sum up the total traveled distance for each vehicle. Finally, group by query period to extract the maximum total distance. *OBA/HR$^{\ominus}$:* Use the temporal index to retrieve all units from `SHunit` present at a given Period from `QP`. Restrict the units to the query period and group by Moid and query period to calculate the traveled distance for each vehicle and period. After that, grouping is used to select the maximum traveled distance for each period. *OBA/HR$^{\oplus}$:* Join `QP` with all cars from `SHcar` selecting pairs where the JourneyDeftime summary field intersects the query period. Then load the journey units using the Moid B-tree from `SHunit`. Apply the refined temporal condition, restrict the units to the query periods and sum up the total length for each combination of query period and Moid. Group by query period to extract the maximum value. *TBA/CR:* For each query period from `QP` load all trips from `MCtrip`, whose deftime intersect the query period (using the temporal index). Calculate the length of the Trip restricted to the query period. Group by Moid to sum up the total traveled distance for each vehicle. The group by query periods to extract the maximum traveled distance. *TBA/UR:* For each query period from `QP` load all units from `MUunit`, whose deftime intersect the query period (using the temporal index). Calculate the length of the TripUnit restricted to the query period. Group by Moid to sum up the total length for each vehicle. Group by query period to extract the maximum traveled distance. *TBA/HR$^{\ominus}$:* For each query period from `QP` load all trips from `MCtrip`, whose deftime intersect the query period (using the temporal index). Calculate the length of the Trip restricted to the query period. Group by Moid to sum up the total traveled distance for each vehicle. The group by query periods to extract the maximum traveled distance. *TBA/UR:* For each query period from `QP` load all units from `MHunit`, whose deftime intersect the query period (using the temporal index). Calculate the length of the TripUnit restricted to the query period. Group by TID to sum up the total length for each trip. Retrieve the trip tuple from `MHtrip` "containing" the grouped units dereferencing the TID link. Group by Moid to calculate the total length for each vehicle. Group by query period to extract the maximum traveled distance. *TBA/HR$^{\oplus}$:* For each query period from `QP` load all trips from `MHtrip`, whose deftime intersect the query period (using the temporal index). Using the Tripid B-tree index, load all according units from `MHunit` using the TripId B-tree. Calculate the length of the TripUnit restricted to the query period. Group by Moid to sum up the total length for each vehicle. Group by query period to extract the maximum traveled distance.

The query clearly favors the OBA/CR, because during query processing, cardinalities remain small and the temporal restrictions are fast and easy to perform. Only a single grouping is required to get the result. All other variants are more than 6 times slower. The second rank is for the TBA/HR$^{\oplus}$ with IDTMP clustering, where candidate trips can be selected fast using the TripDeftime summary. Then, the unit data is accessed ordered by Tripid and time, perfectly matching the clustering schema. Using SPTMP more than doubles the response time, This also holds for the remaining TBA variants.

While still existent, the benefits of using the Deftime summary becomes much smaller for OBA, since the definition time summary field is more accurate for single trips. For the OBA/UR and OBA/HR$^{\ominus}$ variants, the SPTMP clustering is better than the ID/TMP due to the access pattern created by the result sequence returned by the temporal index, which does not reflect the object identity.

6 BerlinMOD/R Query Q16

6.1 Query Q16

Q16: List the pairs of licences for vehicles, from `QueryLicences1`, resp. `QueryLicences2`, where both are present within each region from `QueryRegions1` during each period from `QueryPeriod1`, but do not meet each other there and then.

In our query translations, we accept two objects as "meeting", if their spatial distance is 0.1m or below.

6.1.1 SQL-Query for OBA

```
SELECT PP.Period AS Period, RR.Region AS Region, C1.Licence AS Licence1,
  C2.Licence AS Licence2
FROM dataSCcar C1, dataSCcar C2, QueryRegions1 RR, QueryPeriods1 PP,
  QueryLicences1 LL1, QueryLicences2 LL2
WHERE C1.Licence = LL1.Licence AND C2.Licence = LL2.Licence
  AND LL1.Licence <> LL2.Licence
  AND (C1.Journey at PP.Period) passes RR.Region
  AND (C2.Journey at PP.Period) passes RR.Region
  AND isempty((intersection(C1.Journey, C2.Journey) atperiods PP.Period)
        at RR.Region);
```

6.1.2 SQL-Query for TBA

```
(SELECT RR.Region AS Region, PP.Period AS Period,
        C1.Licence AS Licence1, C2.Licence AS Licence2
 FROM dataMtrip C1, dataMtrip C2, QueryRegions1 RR,
       QueryPeriods1 PP, QueryLicences1 LL1, QueryLicences2 LL2
 WHERE C1.Licence < C2.Licence
       AND C1.Licence = LL1.Licence
       AND (C1.Trip at PP.Period) passes RR.Region
       AND C2.Licence = LL2.Licence
       AND (C2.Trip at PP.Period) passes RR.Region
 GROUP BY RR.Region, PP.Period, C1.Licence, C2.Licence )
EXCEPT
(SELECT RR.Region AS Region, PP.Period AS Period,
        C1.Licence AS Licence1, C2.Licence AS Licence2
 FROM dataMtrip C1, dataMtrip C2, QueryRegions1 RR,
      QueryPeriods1 PP, QueryLicences1 LL1, QueryLicences2 LL2
 WHERE C1.Licence < C2.Licence
       AND C1.Licence = LL1.Licence
       AND (C1.Trip at PP.Period) passes RR.Region
       AND C2.Licence = LL2.Licence
       AND (C2.Trip at PP.Period) passes RR.Region
       AND NOT(isempty(deftime((intersection(C1.Trip, C2.Trip)
                    atperiods PP.Period) at RR.Region)))
 GROUP BY RR.Region, PP.Period, C1.Licence, C2.Licence )
```

6.2 Executable Secondo Plans for Q16

6.2.1 OBA/CR

```
let OBACRres016Candidates1 = QueryLicences feed head[10]
  loopsel[ fun(t:TUPLE)
    dataSCcar_Licence_btree dataSCcar exactmatch[attr(t,Licence)]]
  QueryPeriods1 feed {PP}
  QueryRegions1 feed {RR}
  product
```

```
  product
  projectextend[Licence, Region_RR, Period_PP, Id_RR, Id_PP
    ; Journey: (.Journey atperiods .Period_PP) at .Region_RR]
  filter[no_components(.Journey) > 0] consume;

let OBACRres016Candidates2 = QueryLicences feed head[20] filter[.Id > 10]
  loopsel[ fun(t:TUPLE)
    dataSCcar_Licence_btree dataSCcar exactmatch[attr(t,Licence)]]
  QueryPeriods1 feed {PP}
  QueryRegions1 feed {RR}
  product
  product
  projectextend[Licence, Region_RR, Period_PP, Id_RR, Id_PP
    ; Journey: (.Journey atperiods .Period_PP) at .Region_RR]
  filter[no_components(.Journey) > 0] consume;

query
  OBACRres016Candidates1 feed {C1}
  OBACRres016Candidates2 feed {C2}
  symmjoin[ (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
    and (.Licence_C1 # ..Licence_C2)
  ]
  filter[ not(everNearerThan(.Journey_C1, .Journey_C2, 0.1)) ]
  projectextend[; Licence1: .Licence_C1,
    Licence2: .Licence_C2, Region:
    .Region_RR_C1, Period: .Period_PP_C1, Id_RR:
    .Id_RR_C1, Id_PP: .Id_PP_C1
  ]
  sortby[Id_RR, Id_PP, Licence1, Licence2]
  project[Region, Period, Licence1, Licence2] consume;

delete OBACRres016Candidates1;
delete OBACRres016Candidates2;
```

paragraphOBA/UR

```
let OBAURres016Candidates1 =
  QueryLicences feed head[10]
  QueryPeriods1 feed {PP}
  QueryRegions1 feed {RR}
  product
  product
  loopjoin[ fun(t:TUPLE)
    dataSUcar_Licence_btree dataSUcar exactmatch[attr(t,Licence)]
    projectextendstream[Moid
     ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                    use[fun(U: upoint) U at attr(t,Region_RR)]
    ]
    filter[not(isempty(.JourneyUnit))]
    extend[starttime: inst(initial(.JourneyUnit))]
    sortby[Moid asc, starttime asc]
    groupby[Moid; Trip: group feed
    projecttransformstream[JourneyUnit] the_mvalue]
  ]
  filter[no_components(.Trip)>0]
  project[ Moid, Licence, Region_RR, Period_PP, Id_RR, Id_PP, Trip]
  consume;

let OBAURres016Candidates2 =
  QueryLicences2 feed
  QueryPeriods1 feed {PP}
```

```
  QueryRegions1 feed {RR}
  product
  product
  loopjoin[ fun(t:TUPLE)
    dataSUcar_Licence_btree dataSUcar exactmatch[attr(t,Licence)]
    projectextendstream[Moid
     ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                   use[fun(U: upoint) U at attr(t,Region_RR)]
    ]
    filter[not(isempty(.JourneyUnit))]
    extend[starttime: inst(initial(.JourneyUnit))]
    sortby[Moid asc, starttime asc]
    groupby[Moid; Trip:
      group feed projecttransformstream[JourneyUnit] the_mvalue]
  ]
  filter[no_components(.Trip)>0]
  project[ Moid, Licence, Region_RR, Period_PP, Id_RR, Id_PP, Trip]
  consume;

query
  OBAURres016Candidates1 feed {C1}
  OBAURres016Candidates2 feed {C2}
  symmjoin[  (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
    and (.Moid_C1 # ..Moid_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  remove[Trip_C1,Trip_C2]
  sortby[Id_RR_C1, Id_PP_C1, Moid_C1, Moid_C2]
  krdup[Id_RR_C1, Id_PP_C1, Moid_C1, Moid_C2]
  projectextend[; Region: .Region_RR_C1, Period: .Period_PP_C1,
    Licence1: .Licence_C1, Licence2: .Licence_C2 ]
  consume;

delete OBAURres016Candidates1;
delete OBAURres016Candidates2;
```

6.2.2 OBA/HR$^{\oplus}$ (With summary Fields)

```
let OBAHRres016Candidates1 =
  QueryPeriods feed head[10] {PP}
  QueryRegions feed head[10] {RR}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[10]
  loopjoin[ dataSHcar_Licence_btree dataSHcar exactmatch[.Licence]
            project[Moid, JourneyMBR]
          ]
  symmjoin[..JourneyMBR intersects .QBox]
  projectextend[Moid, Id_RR, Id_PP, Licence, Region_RR, Period_PP
    ; Trip: fun(t: TUPLE)
              dataSHunit_Moid_btree dataSHunit exactmatch[attr(t,Moid)]
              projectextendstream[
                ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                               use[fun(U: upoint) U at attr(t,Region_RR)]
                               filter[not(isempty(.))]
              ]
              extend[starttime: inst(initial(.JourneyUnit))]
              sortby[starttime asc]
              makemvalue[JourneyUnit]
  ]
```

```
  filter[no_components(.Trip)>0]
  consume;

let OBAHRres016Candidates2 =
  QueryPeriods feed head[10] {PP}
  QueryRegions feed head[10] {RR}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[20] filter[.Id > 10]
  loopjoin[ dataSHcar_Licence_btree dataSHcar exactmatch[.Licence]
            project[Moid, JourneyMBR]
          ]
  symmjoin[..JourneyMBR intersects .QBox]
  projectextend[Moid, Id_RR, Id_PP, Licence, Region_RR, Period_PP
    ; Trip: fun(t: TUPLE)
              dataSHunit_Moid_btree dataSHunit exactmatch[attr(t,Moid)]
              projectextendstream[
                ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                               use[fun(U: upoint) U at attr(t,Region_RR)]
                               filter[not(isempty(.))]
              ]
              extend[starttime: inst(initial(.JourneyUnit))]
              sortby[starttime asc]
              makemvalue[JourneyUnit]
  ]
  filter[no_components(.Trip)>0]
  consume;

query
  OBAHRres016Candidates1 feed {C1}
  OBAHRres016Candidates2 feed {C2}
  symmjoin[ (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
    and (.Moid_C1 # ..Moid_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[; Moid1: .Moid_C1, Moid2: .Moid_C2,
    Licence1: .Licence_C1, Licence2: .Licence_C2,
    Region: .Region_RR_C1, Id_RR: .Id_RR_C1,
    Period: .Period_PP_C1, Id_PP: .Id_PP_C1
  ]
  sortby[Id_RR, Id_PP, Moid1, Moid2]
  krdup[Id_RR, Id_PP, Moid1, Moid2]
  project[Region, Period, Licence1, Licence2] consume;

delete OBAHRres016Candidates1;
delete OBAHRres016Candidates2;
```

6.2.3 OBA/HR$^{\ominus}$ (Without summary Fields)

```
let OBAHRres036Candidates1 =
  QueryLicences feed head[10]
  QueryPeriods feed head[10] {PP}
  QueryRegions feed head[10] {RR}
  product
  product
  loopjoin[
    dataSHcar_Licence_btree dataSHcar exactmatch[.Licence] project[ Moid] ]
  projectextend[Moid, Id_RR, Id_PP, Licence, Region_RR, Period_PP
    ; Trip: fun(t: TUPLE)
              dataSHunit_Moid_btree dataSHunit exactmatch[attr(t,Moid)]
```

```
              projectextendstream[
                ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                               use[fun(U: upoint) U at attr(t,Region_RR)]
                               filter[not(isempty(.))]
              ]
              extend[starttime: inst(initial(.JourneyUnit))]
              sortby[starttime asc]
              makemvalue[JourneyUnit]
  ]
  filter[no_components(.Trip)>0]
  consume;

let OBAHRres036Candidates2 =
  QueryLicences feed head[20] filter[.Id > 10]
  QueryPeriods feed head[10] {PP}
  QueryRegions feed head[10] {RR}
  product
  product
  loopjoin[
    dataSHcar_Licence_btree dataSHcar exactmatch[.Licence] project[Moid] ]
  projectextend[Moid, Id_RR, Id_PP, Licence, Region_RR, Period_PP
    ; Trip: fun(t: TUPLE)
              dataSHunit_Moid_btree dataSHunit exactmatch[attr(t,Moid)]
              projectextendstream[
                ; JourneyUnit: (.JourneyUnit atperiods attr(t,Period_PP))
                               use[fun(U: upoint) U at attr(t,Region_RR)]
                               filter[not(isempty(.))]
              ]
              extend[starttime: inst(initial(.JourneyUnit))]
              sortby[starttime asc]
              makemvalue[JourneyUnit]
  ]
  filter[no_components(.Trip)>0]
  consume;

query
  OBAHRres036Candidates1 feed {C1}
  OBAHRres036Candidates2 feed {C2}
  symmjoin[ (.Moid_C1 < ..Moid_C2)
    and (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[; Moid1: .Moid_C1, Moid2: .Moid_C2,
    Licence1: .Licence_C1, Licence2: .Licence_C2,
    Region: .Region_RR_C1, Id_RR: .Id_RR_C1,
    Period: .Period_PP_C1, Id_PP: .Id_PP_C1
  ]
  sortby[Id_RR, Id_PP, Moid1, Moid2]
  krdup[Id_RR, Id_PP, Moid1, Moid2]
  project[Region, Period, Licence1, Licence2] consume;

delete OBAHRres036Candidates1;
delete OBAHRres036Candidates2;
```

6.2.4 TBA/CR

```
let TBACRres016CandidateTrips1 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
```

```
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[10] {LL}
  product
  loopjoin[ fun(tt1:TUPLE)
    dataMCcar_Licence_btree dataMCcar exactmatch[attr(tt1,Licence_LL)]
    project[Moid]
    loopsel[dataMCtrip_Moid_btree exactmatchS[.Moid]]
    sort {L}
    dataMCtrip_Trip_sptmpuni windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMCtrip gettuples2[id_L]
    filter[ .Trip present attr(tt1,Period_PP) ]
    filter[ .Trip passes  attr(tt1,Region_RR) ]
    projectextend[Moid; Trip: (.Trip atperiods
      attr(tt1,Period_PP)) at attr(tt1,Region_RR)]
    filter[no_components(.Trip) > 0]
    projectextend[Moid, Trip; starttime: inst(initial(.Trip))]
    sortby[ Moid asc, starttime asc ]
    groupby[ Moid
      ; Trip: group feed aggregateB[Trip
      ; fun(mp1: mpoint, mp2: mpoint)
        mp1 translateappend[ mp2 , [const duration value (0 0)] ]
      ; [const mpoint value ()] ]
    ]
  ]
  projectextend[Id_RR, Id_PP, Region_RR, Period_PP,
    Trip, Moid; Licence: .Licence_LL]
  consume;

let TBACRres016CandidateTrips2 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[20] filter[.Id > 10] {LL}
  product
  loopjoin[ fun(tt1:TUPLE)
    dataMCcar_Licence_btree dataMCcar exactmatch[attr(tt1,Licence_LL)]
    project[Moid]
    loopsel[dataMCtrip_Moid_btree exactmatchS[.Moid]]
    sort {L}
    dataMCtrip_Trip_sptmpuni
    windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMCtrip gettuples2[id_L]
    filter[ .Trip present attr(tt1,Period_PP) ]
    filter[ .Trip passes  attr(tt1,Region_RR) ]
    projectextend[Moid; Trip: (.Trip atperiods
      attr(tt1,Period_PP)) at attr(tt1,Region_RR)]
    filter[no_components(.Trip) > 0]
    projectextend[Moid, Trip; starttime: inst(initial(.Trip))]
    sortby[ Moid asc, starttime asc ]
    groupby[ Moid
      ; Trip: group feed aggregateB[Trip
      ; fun(mp1: mpoint, mp2: mpoint)
        mp1 translateappend[ mp2 , [const duration value (0 0)] ]
      ; [const mpoint value ()] ]
    ]
  ]
```

```
  projectextend[Id_RR, Id_PP, Region_RR, Period_PP,
    Trip, Moid; Licence: .Licence_LL]
  consume;

query
  TBACRres016CandidateTrips1 feed {C1}
  TBACRres016CandidateTrips2 feed {C2}
  symmjoin[ (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
    and (.Moid_C1 < ..Moid_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[Moid_C1, Moid_C2 ; Licence1: .Licence_C1,
    Licence2: .Licence_C2, Region: .Region_RR_C1, Period:
    .Period_PP_C1, Id_RR: .Id_RR_C1, Id_PP: .Id_PP_C1 ]
  sortby[Id_RR, Id_PP, Moid_C1, Moid_C2]
  krdup[Id_RR, Id_PP, Moid_C1, Moid_C2]
  project[Region, Period, Licence1, Licence2] consume;

delete TBACRres016CandidateTrips1;
delete TBACRres016CandidateTrips2;
```

6.2.5 TBA/UR

```
let TBAURres016CandidateTrips1 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: scaleBox3D(box3d(bbox(.Region_RR), .Period_PP))]
  QueryLicences feed head[10] {LL}
  product
  loopjoin[ fun(t2:TUPLE)
    dataMUcar_Licence_btree dataMUcar exactmatch[attr(t2,Licence_LL)]
    loopsel[ fun(t1:TUPLE)
      dataMUunit_Tripid_btree exactmatchS[attr(t1,Tripid)]
      sort {L}
      dataMUunit_TripUnit_sptmpobj windowintersectsS[attr(t2,QBox)]
      sort rdup {W}
      mergejoin[id_L, id_W] dataMUunit gettuples2[id_L]
      filter[ .TripUnit present attr(t2,Period_PP) ]
      filter[ trajectory(.TripUnit) intersects attr(t2,Region_RR) ]
      extend[Moid: attr(t1,Moid)]
      projectextendstream[Moid; TripUnit:
        (.TripUnit atperiods attr(t2,Period_PP))
        use[ fun(U: upoint) U at attr(t2,Region_RR)]
      ]
      filter[not(isempty(.TripUnit))]
    ]
    projectextend[Moid, TripUnit; starttime: inst(initial(.TripUnit))]
    sortby[ Moid asc, starttime asc ]
    groupby[ Moid ; Trip:
      group feed projecttransformstream[TripUnit] the_mvalue]
  ]
  projectextend[Moid, Id_RR, Id_PP, Trip, Region_RR, Period_PP
    ; Licence: .Licence_LL ]
  consume;

let TBAURres016CandidateTrips2 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
```

```
  extend[QBox: scaleBox3D(box3d(bbox(.Region_RR), .Period_PP))]
  QueryLicences feed head[10] {LL}
  product
  loopjoin[ fun(t2:TUPLE)
    dataMUcar_Licence_btree dataMUcar exactmatch[attr(t2,Licence_LL)]
    loopsel[ fun(t1:TUPLE)
      dataMUunit_Tripid_btree exactmatchS[attr(t1,Tripid)]
      sort {L}
      dataMUunit_TripUnit_sptmpobj windowintersectsS[attr(t2,QBox)]
      sort rdup {W}
      mergejoin[id_L, id_W] dataMUunit gettuples2[id_L]
      filter[ .TripUnit present attr(t2,Period_PP) ]
      filter[ trajectory(.TripUnit) intersects attr(t2,Region_RR) ]
      extend[Moid: attr(t1,Moid)]
      projectextendstream[Moid; TripUnit:
        (.TripUnit atperiods attr(t2,Period_PP))
        use[ fun(U: upoint) U at attr(t2,Region_RR)]
      ]
      filter[not(isempty(.TripUnit))]
    ]
    projectextend[Moid, TripUnit; starttime: inst(initial(.TripUnit))]
    sortby[ Moid asc, starttime asc ]
    groupby[ Moid ; Trip: group feed projecttransformstream[TripUnit]
    the_mvalue]
  ]
  projectextend[Moid, Id_RR, Id_PP, Trip, Region_RR, Period_PP
    ; Licence: .Licence_LL ]
  consume;

query
  TBAURres016CandidateTrips1 feed {C1}
  TBAURres016CandidateTrips2 feed {C2}
  symmjoin[ (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
    and (.Moid_C1 # ..Moid_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[ ; Moid1:    .Moid_C1,        Moid2:    .Moid_C2,
                   Licence1: .Licence_C1,     Licence2: .Licence_C2,
                   Region:   .Region_RR_C1,   Id_RR:    .Id_RR_C1,
                   Period:   .Period_PP_C1,   Id_PP:    .Id_PP_C1 ]
  sortby[Id_RR, Id_PP, Moid1, Moid2]
  krdup[Id_RR, Id_PP, Moid1, Moid2]
  project[Region, Period, Licence1, Licence2] consume;

delete TBAURres016CandidateTrips1;
delete TBAURres016CandidateTrips2;
```

6.2.6 TBA/HR$^{\oplus}$ (With summary Fields)

```
let TBAHRres016CandidateTrips1 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[10]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
          project[Moid, Licence]
  ]
  product
```

```
  loopjoin[ fun(tt1:TUPLE)
    dataMHtrip_Moid_btree dataMHtrip exactmatch[attr(tt1,Moid)]
    project[Moid, Tripid, TripMBR]
    filter[.TripMBR intersects attr(tt1,QBox)]
    loopsel[
      dataMHunit_Tripid_btree exactmatchS[.Tripid]
      sort {L}
    ]
    dataMHunit_TripUnit_sptmpobj
      windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMHunit gettuples2[id_L]
    filter[ .TripUnit present attr(tt1,Period_PP) ]
    filter[ trajectory(.TripUnit) intersects attr(tt1,Region_RR) ]
    projectextendstream[; TripUnit:
      (.TripUnit atperiods attr(tt1,Period_PP))
      use[ fun(U: upoint) U at attr(tt1,Region_RR)]
    ]
    filter[not(isempty(.TripUnit))]
    projectextend[TripUnit; starttime: inst(initial(.TripUnit))]
    sortby[ starttime asc ]
    project[TripUnit]
    groupby[ ; Trip: group feed transformstream the_mvalue]
  ]
  project[Id_RR, Id_PP, Region_RR, Period_PP, Trip, Moid, Licence]
  consume;

let TBAHRres016CandidateTrips2 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[20] filter[.Id > 10]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
          project[Moid, Licence]
  ]
  product
  loopjoin[ fun(tt1:TUPLE)
    dataMHtrip_Moid_btree dataMHtrip exactmatch[attr(tt1,Moid)]
    project[Moid, Tripid, TripMBR]
    filter[.TripMBR intersects attr(tt1,QBox)]
    loopsel[
      dataMHunit_Tripid_btree exactmatchS[.Tripid]
      sort {L}
    ]
    dataMHunit_TripUnit_sptmpobj
      windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMHunit gettuples2[id_L]
    filter[ .TripUnit present attr(tt1,Period_PP) ]
    filter[ trajectory(.TripUnit) intersects attr(tt1,Region_RR) ]
    projectextendstream[; TripUnit:
      (.TripUnit atperiods attr(tt1,Period_PP))
      use[ fun(U: upoint) U at attr(tt1,Region_RR)]
    ]
    filter[not(isempty(.TripUnit))]
    projectextend[TripUnit ; starttime: inst(initial(.TripUnit))]
    sortby[ starttime asc ]
    project[TripUnit]
```

```
    groupby[ ; Trip: group feed transformstream the_mvalue]
  ]
  project[Id_RR, Id_PP, Region_RR, Period_PP, Trip, Moid, Licence]
  consume;

query
  TBAHRres016CandidateTrips1 feed {C1}
  TBAHRres016CandidateTrips2 feed {C2}
  symmjoin[ (.Moid_C1 # ..Moid_C2)
    and (.Id_RR_C1 = ..Id_RR_C2)
    and (.Id_PP_C1 = ..Id_PP_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[Moid_C1, Moid_C2 ; Licence1: .Licence_C1,
    Licence2: .Licence_C2, Region: .Region_RR_C1, Period:
    .Period_PP_C1, Id_RR: .Id_RR_C1, Id_PP: .Id_PP_C1 ]
  sortby[Id_RR, Id_PP, Moid_C1, Moid_C2]
  krdup[Id_RR, Id_PP, Moid_C1, Moid_C2]
  project[Region, Period, Licence1, Licence2] consume;

delete TBAHRres016CandidateTrips1;
delete TBAHRres016CandidateTrips2;
```

6.2.7 TBA/HR$^{\ominus}$ (Without summary Fields)

```
let TBAHRres016CandidateTrips1 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[10]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
  project[Moid, Licence]]
  product
  loopjoin[ fun(tt1:TUPLE)
    dataMHtrip_Moid_btree dataMHtrip exactmatch[attr(tt1,Moid)]
    project[Moid, Tripid]
    loopsel[
      dataMHunit_Tripid_btree exactmatchS[.Tripid]
      sort {L}
    ]
    dataMHunit_TripUnit_sptmpobj
      windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMHunit gettuples2[id_L]
    filter[ .TripUnit present attr(tt1,Period_PP) ]
    filter[ trajectory(.TripUnit) intersects attr(tt1,Region_RR) ]
    projectextendstream[; TripUnit:
      (.TripUnit atperiods attr(tt1,Period_PP))
      use[ fun(U: upoint) U at attr(tt1,Region_RR)]
    ]
    filter[not(isempty(.TripUnit))]
    projectextend[TripUnit ; starttime:inst(initial(.TripUnit))]
    sortby[ starttime asc ]
    project[TripUnit]
    groupby[ ; Trip:  group feed projecttransformstream[TripUnit]
                      the_mvalue
    ]
```

```
  ]
  project[Id_RR, Id_PP, Region_RR, Period_PP, Trip, Moid, Licence]
  consume;

let TBAHRres016CandidateTrips2 =
  QueryRegions feed head[10] {RR}
  QueryPeriods feed head[10] {PP}
  product
  extend[QBox: box3d(bbox(.Region_RR), .Period_PP)]
  QueryLicences feed head[20] filter[.Id > 10]
  loopsel[dataMHcar_Licence_btree dataMHcar exactmatch[.Licence]
  project[Moid, Licence]]
  product
  loopjoin[ fun(tt1:TUPLE)
    dataMHtrip_Moid_btree dataMHtrip exactmatch[attr(tt1,Moid)]
    project[Moid, Tripid]
    loopsel[
      dataMHunit_Tripid_btree exactmatchS[.Tripid]
      sort {L}
    ]
    dataMHunit_TripUnit_sptmpobj
      windowintersectsS[scaleBox3D(attr(tt1,QBox))]
    sort rdup {W}
    mergejoin[id_L, id_W]
    dataMHunit gettuples2[id_L]
    filter[ .TripUnit present attr(tt1,Period_PP) ]
    filter[ trajectory(.TripUnit) intersects attr(tt1,Region_RR) ]
    projectextendstream[; TripUnit:
      (.TripUnit atperiods attr(tt1,Period_PP))
      use[ fun(U: upoint) U at attr(tt1,Region_RR)]
    ]
    filter[not(isempty(.TripUnit))]
    projectextend[TripUnit ; starttime:inst(initial(.TripUnit))]
    sortby[ starttime asc ]
    project[TripUnit]
    groupby[ ; Trip: group feed projecttransformstream[TripUnit] the_mvalue]
  ]
  project[Id_RR, Id_PP, Region_RR, Period_PP, Trip, Moid, Licence]
  consume;

query
  TBAHRres016CandidateTrips1 feed {C1}
  TBAHRres016CandidateTrips2 feed {C2}
  symmjoin[ (.Id_RR_C1 = ..Id_RR_C2)
            and (.Id_PP_C1 = ..Id_PP_C2)
            and (.Moid_C1 # ..Moid_C2)
  ]
  filter[ not(everNearerThan(.Trip_C1, .Trip_C2, 0.1)) ]
  projectextend[Moid_C1, Moid_C2 ; Licence1: .Licence_C1,
                Licence2: .Licence_C2, Region: .Region_RR_C1, Period:
                .Period_PP_C1, Id_RR: .Id_RR_C1, Id_PP: .Id_PP_C1 ]
  sortby[Id_RR, Id_PP, Moid_C1, Moid_C2]
  krdup[Id_RR, Id_PP, Moid_C1, Moid_C2]
  project[Region, Period, Licence1, Licence2] consume;

delete TBAHRres016CandidateTrips1;
delete TBAHRres016CandidateTrips2;
```

6.3 Analyses for Q16

This is a quite complex spatio-temporal range join query without aggregation. The general design for the query plans is as follows: For ID/TMP clustering: First, create two materialized views (candidate views), each containing the product from `QL1` (resp. `QL2`), `QueryRegions1`, and `QP1`, and extend each tuple with the spatially restricted and temporally restricted MOD for the referenced vehicle using the respective query period and query region. Second, the tuples of both candidate views are joined if they have the same query region and query period, but different licences (i.e. represent journeys for different vehicles). After this, the tuples whose Journey never come nearer than 0.1m are selected. Only the plan for the first step, that also dominates the execution time, is representation specific. For SPTMP clustering, the candidate views do not contain mpoint, but upoint data (increasing cardinalities, but avoiding grouping and aggregation). The second step is the same as for the ID/TMP variant, with the difference, that dublicates are removed from the result.

OBA/CR: Two materialized views with candidates are created. For each licence in `QL1` (resp. `QL2`), the according tuples from `SCcar` are retrieved using a B-tree. Then, the product with `QP1` and `QueryRegions1` is formed and for each tuple the Journey is restricted to the query region and query period. Empty Journeys are removed. *OBA/UR:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed and for each licence all according units from `SUcar` are fetched using a B-tree. Each JourneyUnit is restricted to the query period and query region. Grouping by Moid is used to aggregate the restricted upoint values to a single mpoint. Tuples with an empty mpoint are dropped. *OBA/HR$^{\ominus}$:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed and for each licence first all according trips from `SHcar` and then all units belonging to a trip `SHunit` are fetched using the Licence and the Moid B-tree. Each JourneyUnit is restricted to the query period and query region. Grouping by Moid is used to aggregate the restricted upoint values to a single mpoint. Tuples with an empty mpoint are dropped. *OBA/HR$^{\oplus}$:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed and for each licence the according tuple from `SHcar` are fetched using the Licence B-tree. For those tuples, whose query period intersects the JourneyDeftime summary, all units are retrieved from`SHunit`. Each JourneyUnit is restricted to the query period and query region. Grouping by Moid is used to aggregate the restricted upoint values to a single mpoint. Tuples with an empty mpoint are dropped. *TBA/CR:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed and for each licence the according tuples from from `MCcar` are fetched using a B-tree. The, the trips for each car are retrieved from `MCtrip` using the Tripid B-tree. Each Trip is restricted to the query period and query region, empty Trips are dropped. Grouping by Moid is used to concatenate all Trips of a vehicle to a single mpoint. *TBA/UR:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed and for each licence the according trips from `MUcar` are fetched using a B-tree. For each trip, the according units are loaded from `MUunit` using the Tripid B-tree. Each TripUnit is restricted to the query period and query region, dropping empty units. Grouping by Moid, all TripUnits belonging to a vehicle are aggregated into a single mpoint. *TBA/HR$^{\ominus}$:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed. For each licence the according car descriptor from `MHcar`, for each car all trips from `MHtrip`, and for each trip the tuple ids referencing `MHunit` are fetched using the according Licence, Moid and Tripid B-trees. The tuple ids are joined with the tuple ids returned by the spatio-temporal index for a 3D-range query using a MBR created by combining the query region and query period. Only for matching tuple ids, the references tuples are rtrieved from `MHunit`. Duplicate tuples are removed. Each TripUnit is restricted to the query period and query region, empty TripUnits are dropped. For each combination (Licence,Period,Region), all TripUnits are then concatenate to a single mpoint. *TBA/HR$^{\oplus}$:* Two materialized views with candidates are created. The product of `QL1` (resp. `QL2`), `QP1` and `QueryRegions1` is formed. For each licence the according car descriptor from `MHcar`, for each car all trips from `MHtrip` are fetched using the according Licence and Moid B-trees. For all trips, whose TripMBR summary intersects the MBR created by combining the query region and query period, the Tripid B-tree is used to retrieve the tuple ids referencing according tuples in `MHunit`. The tuple ids are joined with the tuple ids returned by the spatio-temporal index for a 3D-range query using a MBR created by combining the query region and query period. Only for matching tuple ids, the references tuples are rtrieved from `MHunit`. Duplicate tuples are removed. Each TripUnit is restricted to the query period and query region, empty TripUnits are dropped. For each

combination (Licence,Period,Region), all TripUnits are then concatenate to a single mpoint.

Generally, the CR and the TBA are better suited for Q16: In the CR, loading the MOD does not increase the cardinalities, and temporal restrictions to MOD can be handled very fast. In the TBA, the amount of MOD loaded can be more effectively pruned to the data really required. On the other hand, for the UR and HR, the construction of the spatio-temporally restricted MOD inflicts overhead. Summary fields allow to reduce this drastically, as shown by TBA/HR$^{\oplus}$ (second and third best time for ID/TMP and SPTMP variants).

The clustering schema has significant influence only in the OBA/UR (SPTMP (606s/719s) better) and OBA/HR (OBA/HR$^{\ominus}$: SPTMP (206s/733s) better, OBA/HR$^{\oplus}$: ID/TMP (385s/869s) better, because the clustering of the unit relation matches the access pattern). With SPTMP, OBA/HR$^{\oplus}$ becomes slower than OBAHR$^{\ominus}$ due to the overhead of testing against the summary, which almost always is passed in the OBA.

References

[1] C. Düntgen, T. Behr, and R. H. Güting, "Assessing representations for moving object histories," Faculty of Mathematics and Computer Science, Fernuniversität in Hagen, Tech. Rep. 357, 2010.

[2] C. Düntgen, T. Behr, and R. H. Güting, "BerlinMOD - A Benchmark for Moving Object Databases," Faculty of Mathematics and Computer Science, Fernuniversität in Hagen, Tech. Rep. 340, 2007.

[3] C. Düntgen, T. Behr, and R. H. Güting, "BerlinMOD: a benchmark for moving object databases," *The VLDB Journal*, vol. 18, no. 6, pp. 1335–1368, 2009.

[4] "BerlinMOD Benchmark Web Site," http://dna.fernuni-hagen.de/secondo/BerlinMOD/BerlinMOD.html, 2010.

[5] "Secondo Web Site," http://www.informatik.fernuni-hagen.de/secondo, 2010.

[6] R. H. Güting, S. Dieker, C. Freundorfer, L. Becker, and H. Schenk, "Secondo/qp: Implementation of a generic query processor." in *DEXA*, 1999, pp. 66–87.

[7] S. Dieker and R. H. Güting, "Plug and play with query algebras: SECONDO - a generic DBMS development environment," in *Proc. of the International Symposium on Database Engineering & Applications*, 2000, pp. 380–392.

[8] R. H. Güting, T. Behr, V. Almeida, Z. Ding, F. Hoffmann, and M. Spiekermann, "SECONDO: An Extensible DBMS Architecture and Prototype," FernUniversität Hagen, Tech. Rep., 2004.

[9] R. H. Güting, V. T. de Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle, "Secondo: An extensible DBMS platform for research prototyping and teaching." in *ICDE*, 2005, pp. 1115–1116.

[10] R. H. Güting, T. Behr, and C. Düntgen, "Secondo: A platform for moving objects database research and for publishing and integrating research implementations," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 56–63, 2010.