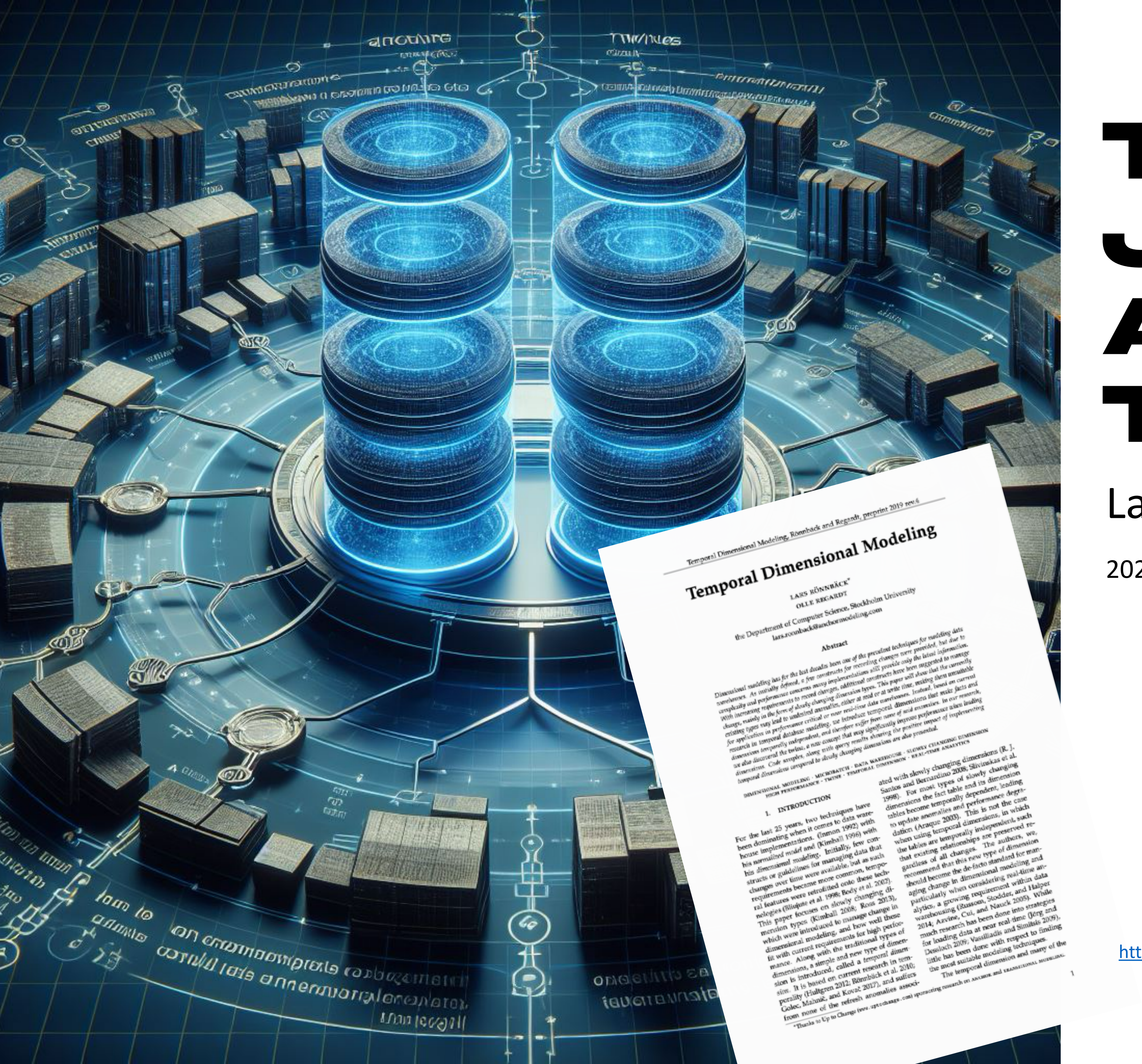


Temporal Dimensional Modeling, Rönnbäck and Regardt, preprint 2019 rev.4

Temporal Dimensional Modeling

LARS RÖNNBÄCK*
OLLE REGARDT
the Department of Computer Science, Stockholm University
lars.ronnback@anchormodeling.com

Abstract

Dimensional modeling has for the last decades been one of the prevalent techniques for modeling data warehouses. As initially defined, a few constructs for recording changes were provided, but due to complexity and performance concerns many implementations still provide only the latest information. With increasing requirements to record changes, additional constructs have been suggested to manage change, mainly in the form of slowly changing dimension types. This paper will show that the currently existing types may lead to undesired anomalies, either at read or at write time, making them unsuitable for application in performance critical or near real-time data warehouses. Instead, based on current research in temporal database modeling, we introduce temporal dimensions that make facts and dimensions temporally independent, and therefore suffer from none of said anomalies. In our research, we also discovered the twine, a new concept that may significantly improve performance when loading dimensions. Code samples, along with query results showing the positive impact of implementing temporal dimensions compared to slowly changing dimensions are also presented.

DIMENSIONAL MODELING · MICROBATCH · DATA WAREHOUSE · SLOWLY CHANGING DIMENSION
HIGH PERFORMANCE · TWINE · TEMPORAL DIMENSION · REAL-TIME ANALYTICS

1. INTRODUCTION

For the last 25 years, two techniques have been dominating when it comes to data warehouse implementations. (Inmon 1992) with his *normalized model* and (Kimball 1996) with his *dimensional modeling*. Initially, few constructs or guidelines for managing data that changes over time were available, but as such requirements became more common, temporal features were retrofitted onto these technologies (Bliujute et al. 1998; Body et al. 2002). This paper focuses on slowly changing dimension types (Kimball 2008; Ross 2013), which were introduced to manage change in dimensional modeling, and how well these fit with current requirements for high performance. Along with the traditional types of dimensions, a simple and new type of dimension is introduced, called a *temporal dimension*. It is based on current research in temporality (Hultgren 2012; Rönnbäck et al. 2010; Golec, Mahnič, and Kovač 2017), and suffers from none of the refresh anomalies associated with slowly changing dimensions (R. J. Santos and Bernardino 2008; Slivinskas et al. 1998). For most types of slowly changing dimensions the fact table and its dimension tables become temporally dependent, leading to update anomalies and performance degradation (Araque 2003). This is not the case when using temporal dimensions, in which the tables are temporally independent, such that existing relationships are preserved regardless of all changes. The authors, we, recommend that this new type of dimension should become the de-facto standard for managing change in dimensional modeling and particularly when considering real-time analytics, a growing requirement within data warehousing (Russom, Stodder, and Halper 2014; Azvine, Cui, and Nauck 2005). While much research has been done into strategies for loading data at near real-time (Jörg and Dessloch 2009; Vassiliadis and Simitsis 2009), little has been done with respect to finding the most suitable modeling techniques.

The temporal dimension and many of the

*Thanks to Up to Change (www.uptochange.com) sponsoring research on ANCHOR and TRADITIONAL MODELING.

1

TEMPORAL JOINS AND TWINES

Lars Rönnbäck

2023-11-28

Let's Twine Again

https://www.anchormodeling.com/♫-lets-twine-again-♫/

https://www.researchgate.net/publication/330798405_Temporal_Dimensional_Modeling

NON-TEMPORAL INNER JOIN

```
select *
from PERSON p
join HAIR_COLOR hc
on hc.Person_Id = p.Person_Id
```

PERSON

Person_Id
1
2
3

(1) to (1)

HAIR_COLOR

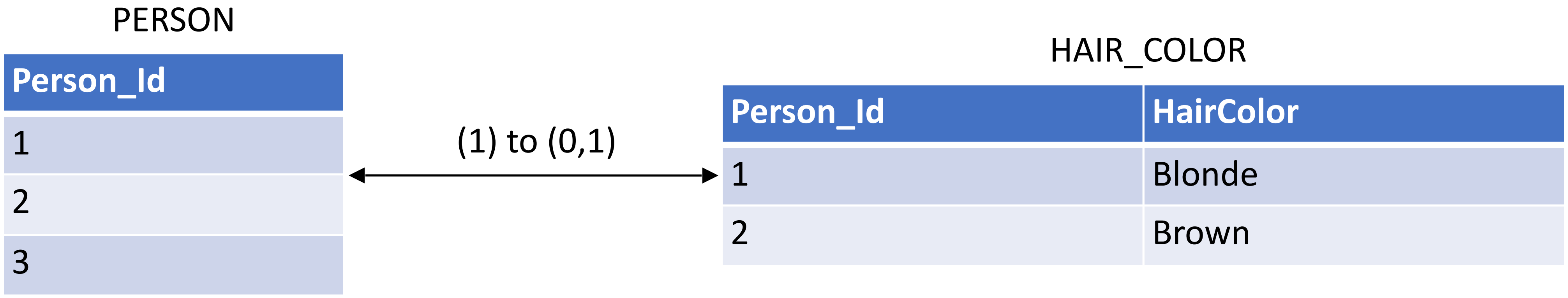
Person_Id	HairColor
1	Blonde
2	Brown
3	Gray

```
drop table if exists PERSON;
create table PERSON (
   Person_Id int not null primary key
);
insert into PERSON values (1), (2), (3);
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
   Person_Id int not null primary key foreign key references PERSON (Person_Id),
   HairColor varchar(42) not null
);
insert into HAIR_COLOR values (1, 'Blonde'), (2, 'Brown'), (3, 'Gray');
```

NON-TEMPORAL OUTER JOIN

```
select p.Person_Id, isnull(hc.HairColor, 'Unknown')
from PERSON p
left join HAIR_COLOR hc
on hc.Person_Id = p.Person_Id
```



PERSON

Person_Id
1
2
3

(1) to (0,1)

HAIR_COLOR

Person_Id	HairColor
1	Blonde
2	Brown

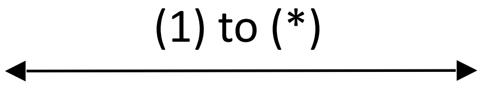
```
drop table if exists PERSON;
create table PERSON (
   Person_Id int not null primary key
);
insert into PERSON values (1), (2), (3);
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
   Person_Id int not null primary key foreign key references PERSON (Person_Id),
   HairColor varchar(42) not null
);
insert into HAIR_COLOR values (1, 'Blonde'), (2, 'Brown');
```

TEMPORALLY INDEPENDENT INNER JOIN

PERSON

Person_Id
1
2
3



(1) to (*)

HAIR_COLOR

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12
3	Gray	2001-01-01

```sql
drop table if exists PERSON;
create table PERSON (
   Person_Id int not null primary key
);
insert into PERSON values (1), (2), (3);
```

```sql
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12'),
(3, 'Gray', '2001-01-01');
```

TEMPORALLY INDEPENDENT INNER JOIN

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
2	Purple	2011-11-11
3	Gray	2001-01-01

What was the hair color of every person on 11/11 of 2011?

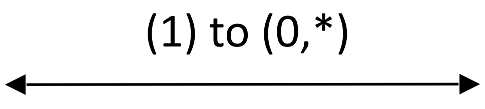
```
select p.Person_Id, hc.HairColor, hc.Valid_From
from PERSON p
join (
  select *
  from HAIR_COLOR hc_sub
  where hc_sub.Valid_From = (
    select top 1 hc_at.Valid_From
    from HAIR_COLOR hc_at
    where hc_at.Person_Id = hc_sub.Person_Id
    and hc_at.Valid_From <= '2011-11-11’
    order by hc_at.Valid_From desc
  )
) hc
on hc.Person_Id = p.Person_Id;
```

A temporally independent join can be reduced to a non-temporal join by first resolving the temporality of the involved tables.

TEMPORALLY INDEPENDENT OUTER JOIN

PERSON

Person_Id
1
2
3


(1) to (0,*)

HAIR_COLOR

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12

```
drop table if exists PERSON;
create table PERSON (
   Person_Id int not null primary key
);
insert into PERSON values (1), (2), (3);
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12');
```

TEMPORALLY INDEPENDENT OUTER JOIN

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
2	Unknown	*NULL*
3	Unknown	*NULL*

What was the hair color of every person on 31/12 of 2001?

```
select p.Person_Id, isnull(hc.HairColor, 'Unknown'), hc.Valid_From
from PERSON p
left join (
  select *
  from HAIR_COLOR hc_sub
  where hc_sub.Valid_From = (
    select top 1 hc_at.Valid_From
    from HAIR_COLOR hc_at
    where hc_at.Person_Id = hc_sub.Person_Id
    and hc_at.Valid_From <= '2001-12-31’
    order by hc_at.Valid_From desc
  )
) hc
on hc.Person_Id = p.Person_Id;
```

A temporally independent join can be reduced to a non-temporal join by first resolving the temporality of the involved tables.

```
select
  p.Person_Id,
  case
    when hc_exist.Person_Id is null then 'Unknown (person)'
    when hc.HairColor is null then 'Unknown (timepoint)'
    else hc.HairColor
  end,
  hc.Valid_From
from PERSON p
left join (
  select *
  from HAIR_COLOR hc_sub
  where hc_sub.Valid_From = (
    select top 1 hc_at.Valid_From
    from HAIR_COLOR hc_at
    where hc_at.Person_Id = hc_sub.Person_Id
    and hc_at.Valid_From <= '2001-12-31’
    order by hc_at.Valid_From desc
  )
) hc
on hc.Person_Id = p.Person_Id
left join (
  select distinct Person_Id
  from HAIR_COLOR
) hc_exist
on hc_exist.Person_Id = p.Person_Id;
```

Additional information is needed in order to resolve the exact reason why a hair color is unknown.

TEMPORALLY DEPENDENT INNER JOIN

PERSON

Person_Id	MemberSince
1	2020-12-24
2	2011-12-13
3	2018-08-18

(1) to (*)

HAIR_COLOR

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12
3	Gray	2001-01-01

```
drop table if exists PERSON;
create table PERSON (
  Person_Id int not null primary key,
  MemberSince date null
);
insert into PERSON values
(1, '2020-12-24'), (2, '2011-12-13'),
(3, '2018-08-18');
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12'),
(3, 'Gray', '2001-01-01');
```

TEMPORALLY DEPENDENT INNER JOIN

Person_Id	HairColor	MemberSince	Valid_From	Valid_To
1	Gray	2020-12-24	2020-12-24	9999-12-31
2	Brown	2011-12-13	2011-11-12	9999-12-31
3	Gray	2018-08-18	2001-01-01	9999-12-31

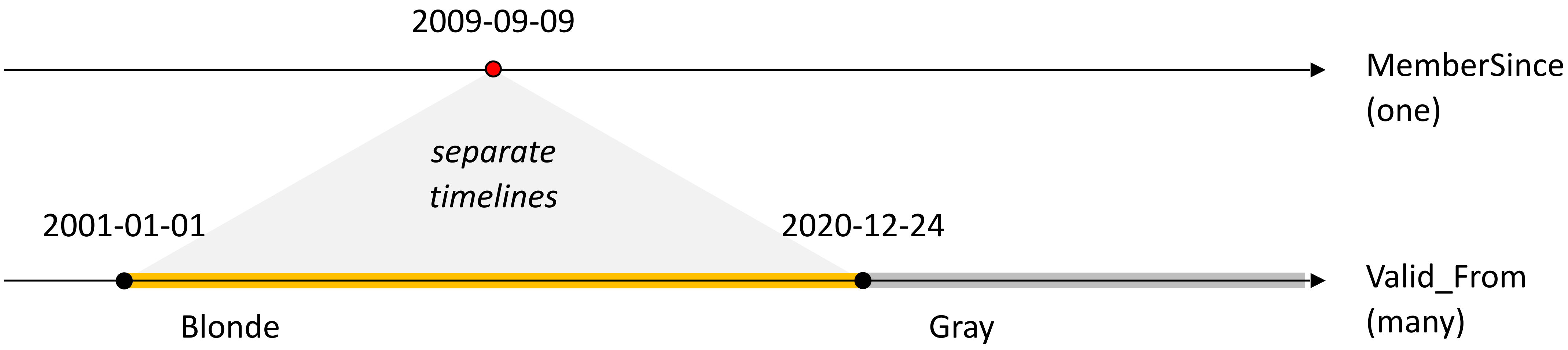
Valid_To might already be materialized depending on the style of modeling

What was the hair color of every person when they became a member?

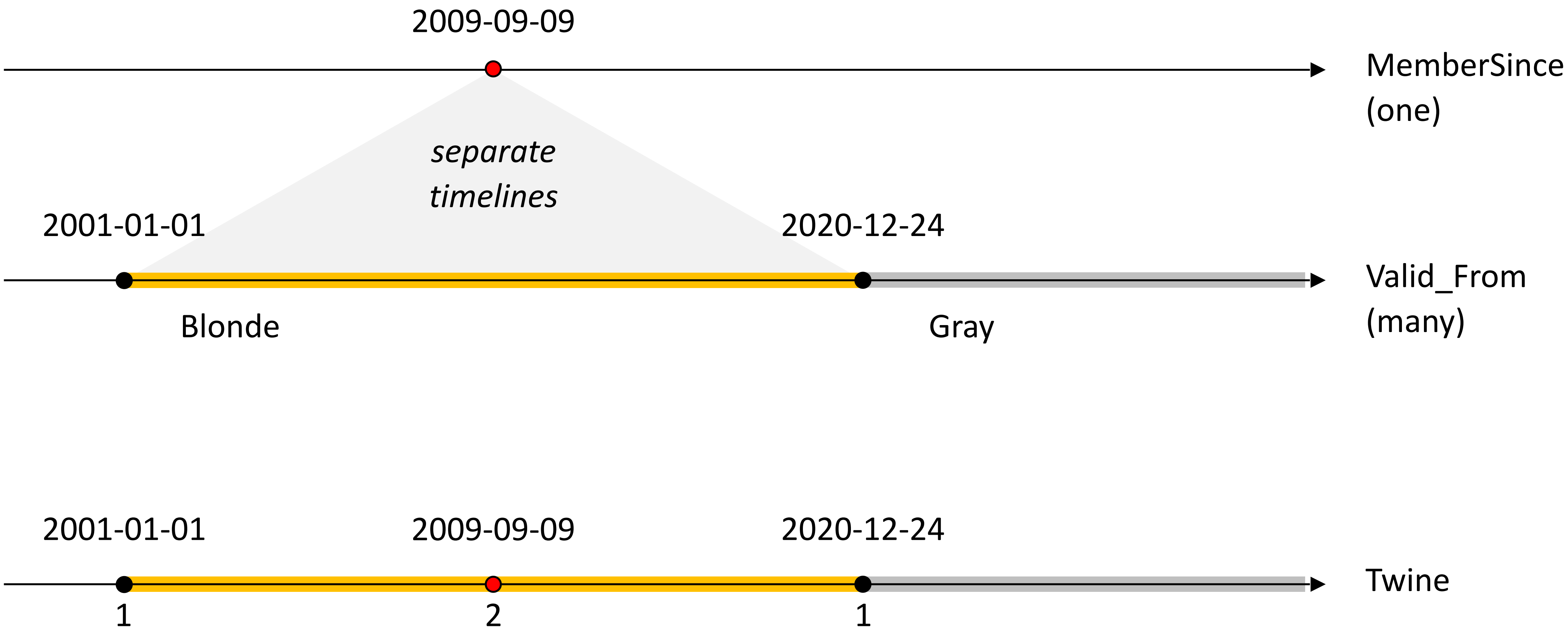
```
select p.Person_Id, hc.HairColor, p.MemberSince, hc.Valid_From, hc.Valid_To
from PERSON p
join (
  select
    Person_Id, HairColor, Valid_From,
    LEAD(Valid_From, 1, '9999-12-31') over (partition by Person_Id order by Valid_From) as Valid_To
  from HAIR_COLOR
) hc
on hc.Person_Id = p.Person_Id
and hc.Valid_From <= p.MemberSince
and hc.Valid_To > p.MemberSince
```

The temporally dependent join **is not** reduced to a non-temporal join and timepoints become a part of the join condition.

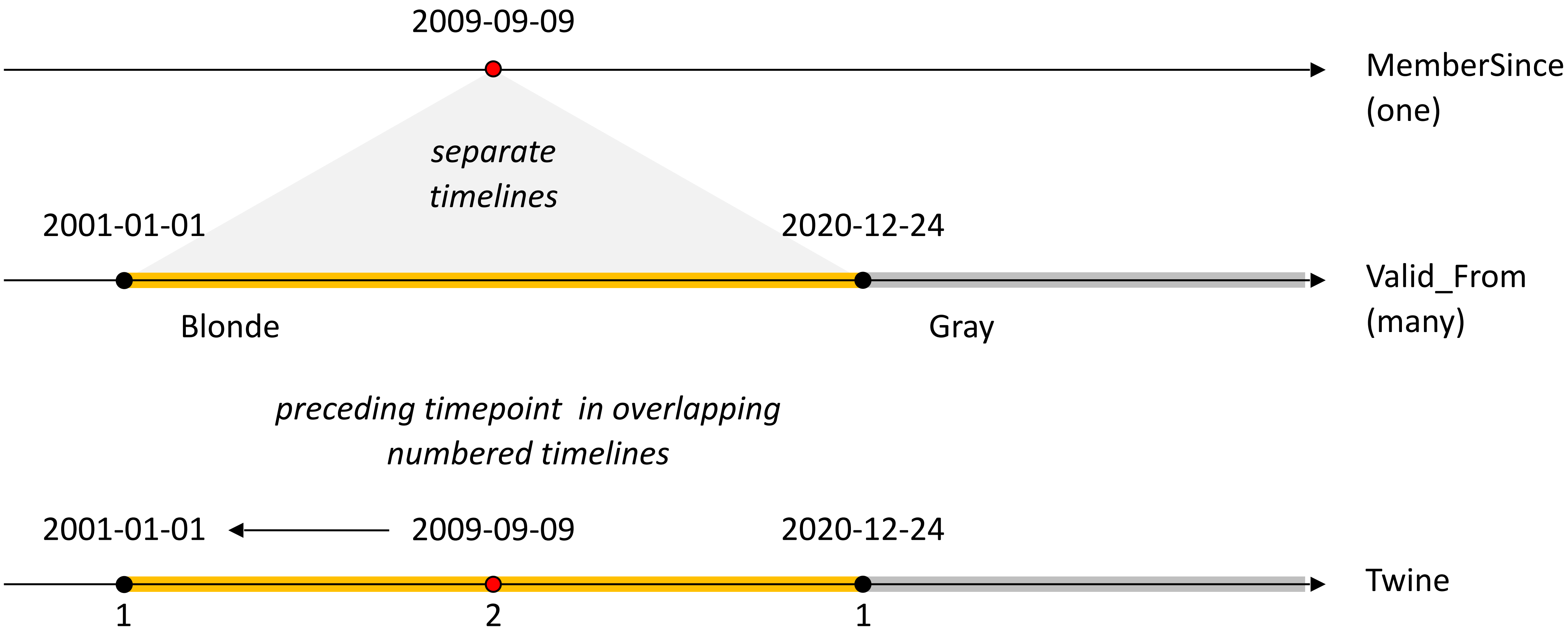
TEMPORALLY DEPENDENT JOIN / TWINE



TEMPORALLY DEPENDENT JOIN / TWINE



TEMPORALLY DEPENDENT JOIN / TWINE



2009-09-09

MemberSince
(one)

separate
timelines

2001-01-01

2020-12-24

Valid_From
(many)

Blonde

Gray

preceding timepoint in overlapping
numbered timelines

2001-01-01 ← 2009-09-09

2020-12-24

Twine

1

2

1

TEMPORALLY DEPENDENT SPECIALIZED TWINE

Person_Id	HairColor	MemberSince	Valid_From
1	Gray	2020-12-24	2020-12-24
2	Brown	2011-12-13	2011-11-12
3	Gray	2018-08-18	2001-01-01

With the twine, no join is necessary. Instead a windowed function is used to find the value in effect.

```
select Person_Id, HairColor, MemberSince, Valid_From
from (
  select
    Person_Id, Timeline, Timepoint as MemberSince,
    LAG(HairColor, 1) over (partition by Person_Id order by Timepoint, Timeline) as HairColor,
    LAG(Timepoint, 1) over (partition by Person_Id order by Timepoint, Timeline) as Valid_From
  from (
    select Person_Id, cast(1 as tinyint) as Timeline, Valid_From as Timepoint, HairColor
    from HAIR_COLOR
    union all
    select Person_Id, cast(2 as tinyint) as Timeline, MemberSince as Timepoint, null
    from PERSON
  ) timelines
) twine
where twine.Timeline = 2
```

Valid_To is never used in a twine!

TEMPORALLY DEPENDENT OUTER JOIN

PERSON

Person_Id	MemberSince
1	1999-12-31
2	*NULL*
3	2018-08-18

(0,1) to (0,*)

HAIR_COLOR

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12

```
drop table if exists PERSON;
create table PERSON (
  Person_Id int not null primary key,
  MemberSince date null
);
insert into PERSON values
(1, '1999-12-31'), (2, null), (3, '2018-08-18');
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12');
```

```sql
select
  p.Person_Id,
  case
    when p.MemberSince is null then 'Unknown (non-member)’
    when hc_exist.Person_Id is null then 'Unknown (person)’
    when hc.HairColor is null then 'Unknown (timepoint)’
    else hc.HairColor
  end,
  p.MemberSince, hc.Valid_From, hc.Valid_To
from PERSON p
left join (
  select
    Person_Id, HairColor, Valid_From,
    LEAD(Valid_From, 1, '9999-12-31')
      over (partition by Person_Id order by Valid_From) as Valid_To
  from HAIR_COLOR
) hc
on hc.Person_Id = p.Person_Id
and hc.Valid_From <= p.MemberSince
and hc.Valid_To > p.MemberSince
left join (
  select distinct Person_Id
  from HAIR_COLOR
) hc_exist
on hc_exist.Person_Id = p.Person_Id;
```

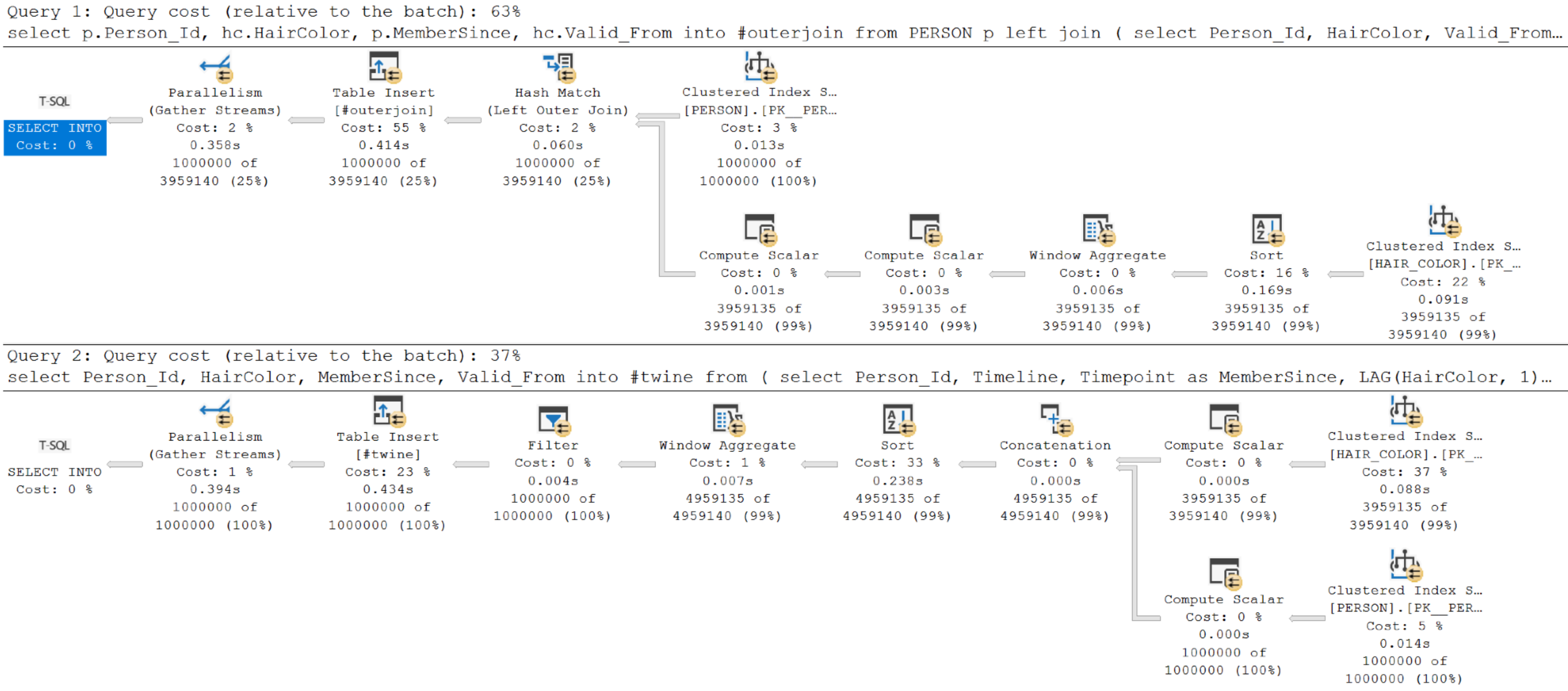
There are now three reasons for why the hair color may be unknown.

```
select
  Person_Id,
  case
    when MemberSince is null then 'Unknown (non-member)’
    when hc_exist = 0 then 'Unknown (person)’
    when HairColor is null then 'Unknown (timepoint)’
    else HairColor
  end,
  MemberSince, Valid_From
from (
  select
    Person_Id, Timeline, Timepoint as MemberSince,
    MAX(case when Timeline = 1 then 1 else 0 end) over (partition by Person_Id) as hc_exist,
    LAG(HairColor, 1) over (partition by Person_Id order by Timepoint, Timeline) as HairColor,
    LAG(Timepoint, 1) over (partition by Person_Id order by Timepoint, Timeline) as Valid_From
  from (
    select Person_Id, 1 as Timeline, Valid_From as Timepoint, HairColor from HAIR_COLOR
    union all
    select Person_Id, 2 as Timeline, MemberSince as Timepoint, null from PERSON
  ) timelines
) twine
where twine.Timeline = 2
```

An additional column is necessary for checking the existence of a Person_Id in the HAIR_COLOR table.

TEMPORALLY DEP. OUTER JOIN VS SPEC. TWINE

1 000 000 ↔ 4 000 000



```
Query 1: Query cost (relative to the batch): 63%
select p.Person_Id, hc.HairColor, p.MemberSince, hc.Valid_From into #outerjoin from PERSON p left join ( select Person_Id, HairColor, Valid_From…
```

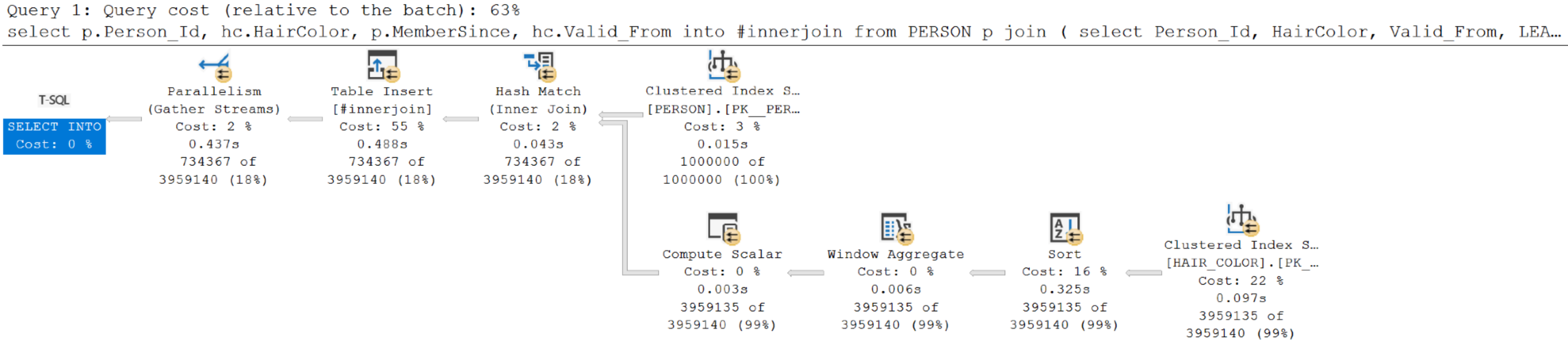
T-SQL
SELECT INTO Cost: 0 %
Parallelism (Gather Streams) Cost: 2 % 0.358s 1000000 of 3959140 (25%)
Table Insert [#outerjoin] Cost: 55 % 0.414s 1000000 of 3959140 (25%)
Hash Match (Left Outer Join) Cost: 2 % 0.060s 1000000 of 3959140 (25%)
Clustered Index S… [PERSON].[PK__PER… Cost: 3 % 0.013s 1000000 of 1000000 (100%)
Compute Scalar Cost: 0 % 0.001s 3959135 of 3959140 (99%)
Compute Scalar Cost: 0 % 0.003s 3959135 of 3959140 (99%)
Window Aggregate Cost: 0 % 0.006s 3959135 of 3959140 (99%)
Sort Cost: 16 % 0.169s 3959135 of 3959140 (99%)
Clustered Index S… [HAIR_COLOR].[PK_… Cost: 22 % 0.091s 3959135 of 3959140 (99%)

```
Query 2: Query cost (relative to the batch): 37%
select Person_Id, HairColor, MemberSince, Valid_From into #twine from ( select Person_Id, Timeline, Timepoint as MemberSince, LAG(HairColor, 1)…
```

T-SQL
SELECT INTO Cost: 0 %
Parallelism (Gather Streams) Cost: 1 % 0.394s 1000000 of 1000000 (100%)
Table Insert [#twine] Cost: 23 % 0.434s 1000000 of 1000000 (100%)
Filter Cost: 0 % 0.004s 1000000 of 1000000 (100%)
Window Aggregate Cost: 1 % 0.007s 4959135 of 4959140 (99%)
Sort Cost: 33 % 0.238s 4959135 of 4959140 (99%)
Concatenation Cost: 0 % 0.000s 4959135 of 4959140 (99%)
Compute Scalar Cost: 0 % 0.000s 3959135 of 3959140 (99%)
Clustered Index S… [HAIR_COLOR].[PK_… Cost: 37 % 0.088s 3959135 of 3959140 (99%)
Compute Scalar Cost: 0 % 0.000s 1000000 of 1000000 (100%)
Clustered Index S… [PERSON].[PK__PER… Cost: 5 % 0.014s 1000000 of 1000000 (100%)

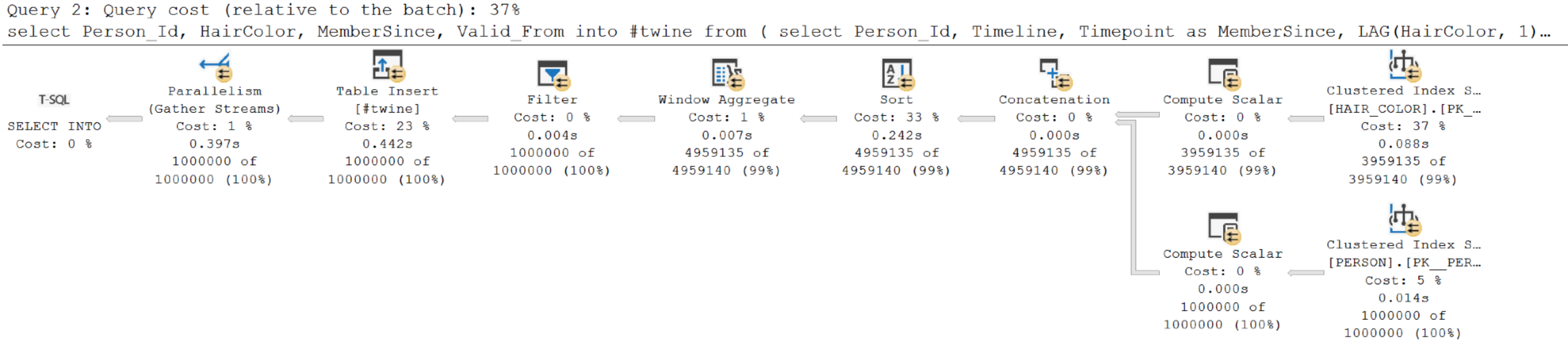
TEMPORALLY DEP. INNER JOIN VS SPEC. TWINE

1 000 000 ⟷ 4 000 000



```
Query 1: Query cost (relative to the batch): 63%
select p.Person_Id, hc.HairColor, p.MemberSince, hc.Valid_From into #innerjoin from PERSON p join ( select Person_Id, HairColor, Valid_From, LEA…
```

```
T-SQL
SELECT INTO
Cost: 0 %

Parallelism
(Gather Streams)
Cost: 2 %
0.437s
734367 of
3959140 (18%)

Table Insert
[#innerjoin]
Cost: 55 %
0.488s
734367 of
3959140 (18%)

Hash Match
(Inner Join)
Cost: 2 %
0.043s
734367 of
3959140 (18%)

Clustered Index S…
[PERSON].[PK__PER…
Cost: 3 %
0.015s
1000000 of
1000000 (100%)

Compute Scalar
Cost: 0 %
0.003s
3959135 of
3959140 (99%)

Window Aggregate
Cost: 0 %
0.006s
3959135 of
3959140 (99%)

Sort
Cost: 16 %
0.325s
3959135 of
3959140 (99%)

Clustered Index S…
[HAIR_COLOR].[PK_…
Cost: 22 %
0.097s
3959135 of
3959140 (99%)
```

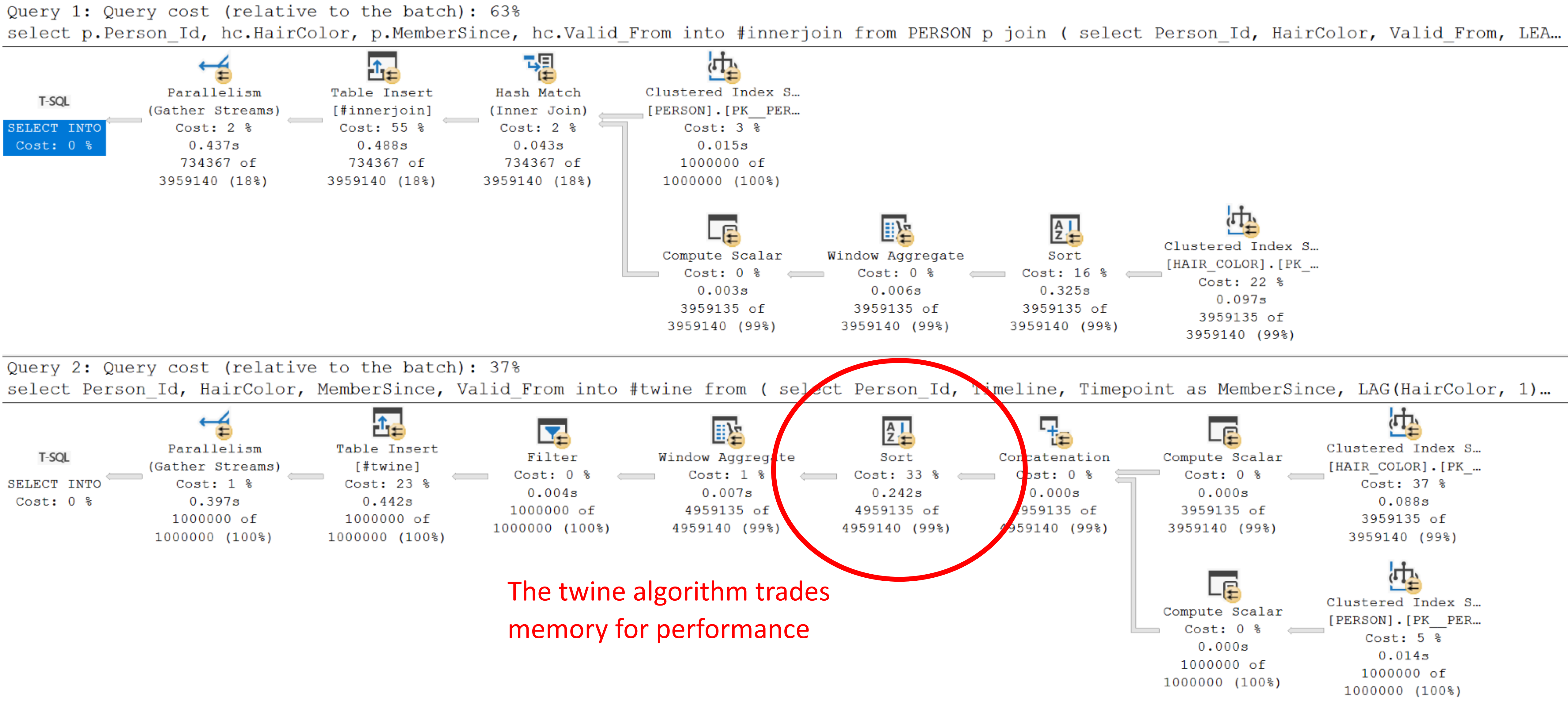


```
Query 2: Query cost (relative to the batch): 37%
select Person_Id, HairColor, MemberSince, Valid_From into #twine from ( select Person_Id, Timeline, Timepoint as MemberSince, LAG(HairColor, 1)…
```

```
T-SQL
SELECT INTO
Cost: 0 %

Parallelism
(Gather Streams)
Cost: 1 %
0.397s
1000000 of
1000000 (100%)

Table Insert
[#twine]
Cost: 23 %
0.442s
1000000 of
1000000 (100%)

Filter
Cost: 0 %
0.004s
1000000 of
1000000 (100%)

Window Aggregate
Cost: 1 %
0.007s
4959135 of
4959140 (99%)

Sort
Cost: 33 %
0.242s
4959135 of
4959140 (99%)

Concatenation
Cost: 0 %
0.000s
4959135 of
4959140 (99%)

Compute Scalar
Cost: 0 %
0.000s
3959135 of
3959140 (99%)

Clustered Index S…
[HAIR_COLOR].[PK_…
Cost: 37 %
0.088s
3959135 of
3959140 (99%)

Compute Scalar
Cost: 0 %
0.000s
1000000 of
1000000 (100%)

Clustered Index S…
[PERSON].[PK__PER…
Cost: 5 %
0.014s
1000000 of
1000000 (100%)
```

THE POSSIBLE DOWNSIDE OF THE TWINE

1 000 000 ⟷ 4 000 000



Query 1: Query cost (relative to the batch): 63%

select p.Person_Id, hc.HairColor, p.MemberSince, hc.Valid_From into #innerjoin from PERSON p join (select Person_Id, HairColor, Valid_From, LEA…

T-SQL
SELECT INTO
Cost: 0 %

Parallelism
(Gather Streams)
Cost: 2 %
0.437s
734367 of
3959140 (18%)

Table Insert
[#innerjoin]
Cost: 55 %
0.488s
734367 of
3959140 (18%)

Hash Match
(Inner Join)
Cost: 2 %
0.043s
734367 of
3959140 (18%)

Clustered Index S…
[PERSON].[PK__PER…
Cost: 3 %
0.015s
1000000 of
1000000 (100%)

Compute Scalar
Cost: 0 %
0.003s
3959135 of
3959140 (99%)

Window Aggregate
Cost: 0 %
0.006s
3959135 of
3959140 (99%)

Sort
Cost: 16 %
0.325s
3959135 of
3959140 (99%)

Clustered Index S…
[HAIR_COLOR].[PK_…
Cost: 22 %
0.097s
3959135 of
3959140 (99%)

Query 2: Query cost (relative to the batch): 37%

select Person_Id, HairColor, MemberSince, Valid_From into #twine from (select Person_Id, Timeline, Timepoint as MemberSince, LAG(HairColor, 1)…

T-SQL
SELECT INTO
Cost: 0 %

Parallelism
(Gather Streams)
Cost: 1 %
0.397s
1000000 of
1000000 (100%)

Table Insert
[#twine]
Cost: 23 %
0.442s
1000000 of
1000000 (100%)

Filter
Cost: 0 %
0.004s
1000000 of
1000000 (100%)

Window Aggregate
Cost: 1 %
0.007s
4959135 of
4959140 (99%)

Sort
Cost: 33 %
0.242s
4959135 of
4959140 (99%)

Concatenation
Cost: 0 %
0.000s
4959135 of
4959140 (99%)

Compute Scalar
Cost: 0 %
0.000s
3959135 of
3959140 (99%)

Clustered Index S…
[HAIR_COLOR].[PK_…
Cost: 37 %
0.088s
3959135 of
3959140 (99%)

Compute Scalar
Cost: 0 %
0.000s
1000000 of
1000000 (100%)

Clustered Index S…
[PERSON].[PK__PER…
Cost: 5 %
0.014s
1000000 of
1000000 (100%)

The twine algorithm trades memory for performance

TEMPORALLY DEPENDENT OUTER JOIN [REVISITED]

PURCHASE

Person_Id	PurchaseDate
1	1999-12-31
1	2001-02-03
1	2004-05-06
2	2011-11-11
2	2023-11-28
3	2018-08-18

(*) to (0,*)

HAIR_COLOR

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12

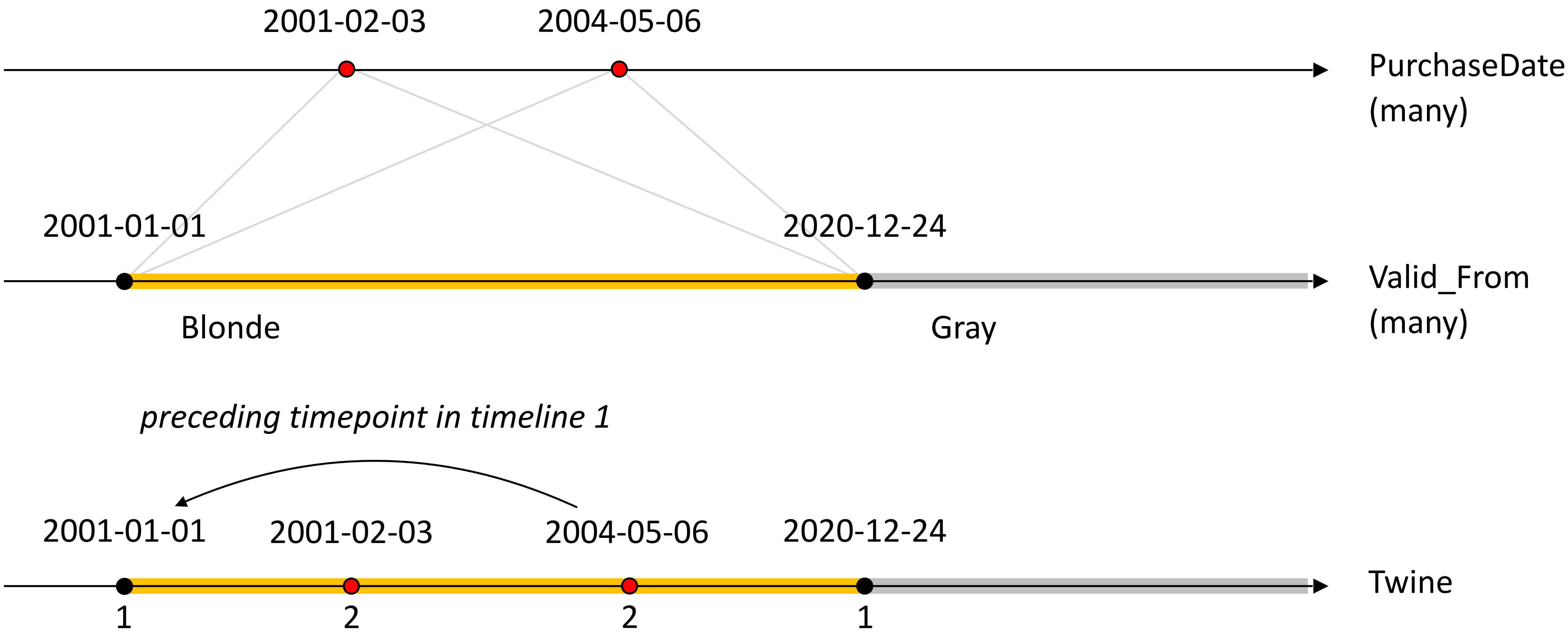
```
drop table if exists PURCHASE;
create table PURCHASE (
  Person_Id int not null,
  PurchaseDate date not null,
  primary key (Person_Id, PurchaseDate)
);
insert into PURCHASE values
(1, '1999-12-31'), (1, '2001-02-03'), (1, '2004-05-06'),
(2, '2011-11-11'), (2, '2023-11-28'), (3, '2018-08-18');
```

```
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12');
```

```
select
  p.Person_Id,
  case
    when hc_exist.Person_Id is null then 'Unknown (person)’
    when hc.HairColor is null then 'Unknown (timepoint)’
    else hc.HairColor
  end as HairColor,
  p.PurchaseDate, hc.Valid_From, hc.Valid_To
from PURCHASE p
left join (
  select
    Person_Id, HairColor, Valid_From,
    LEAD(Valid_From, 1, '9999-12-31')
      over (partition by Person_Id order by Valid_From) as Valid_To
  from HAIR_COLOR
) hc
on hc.Person_Id = p.Person_Id
and hc.Valid_From <= p.PurchaseDate
and hc.Valid_To > p.PurchaseDate
left join (
  select distinct Person_Id
  from HAIR_COLOR
) hc_exist
on hc_exist.Person_Id = p.Person_Id;
```

Person_Id	**HairColor**	**PurchaseDate**	**Valid_From**	**Valid_To**
1	Unknown (timepoint)	1999-12-31	*NULL*	*NULL*
1	Blonde	2001-02-03	2001-01-01	2020-12-24
1	Blonde	2004-05-06	2001-01-01	2020-12-24
2	Purple	2011-11-11	2011-11-11	2011-11-12
2	Brown	2023-11-28	2011-11-12	9999-12-31
3	Unknown (person)	2018-08-18	*NULL*	*NULL*

TEMPORALLY DEPENDENT OUTER JOIN [REVISITED]



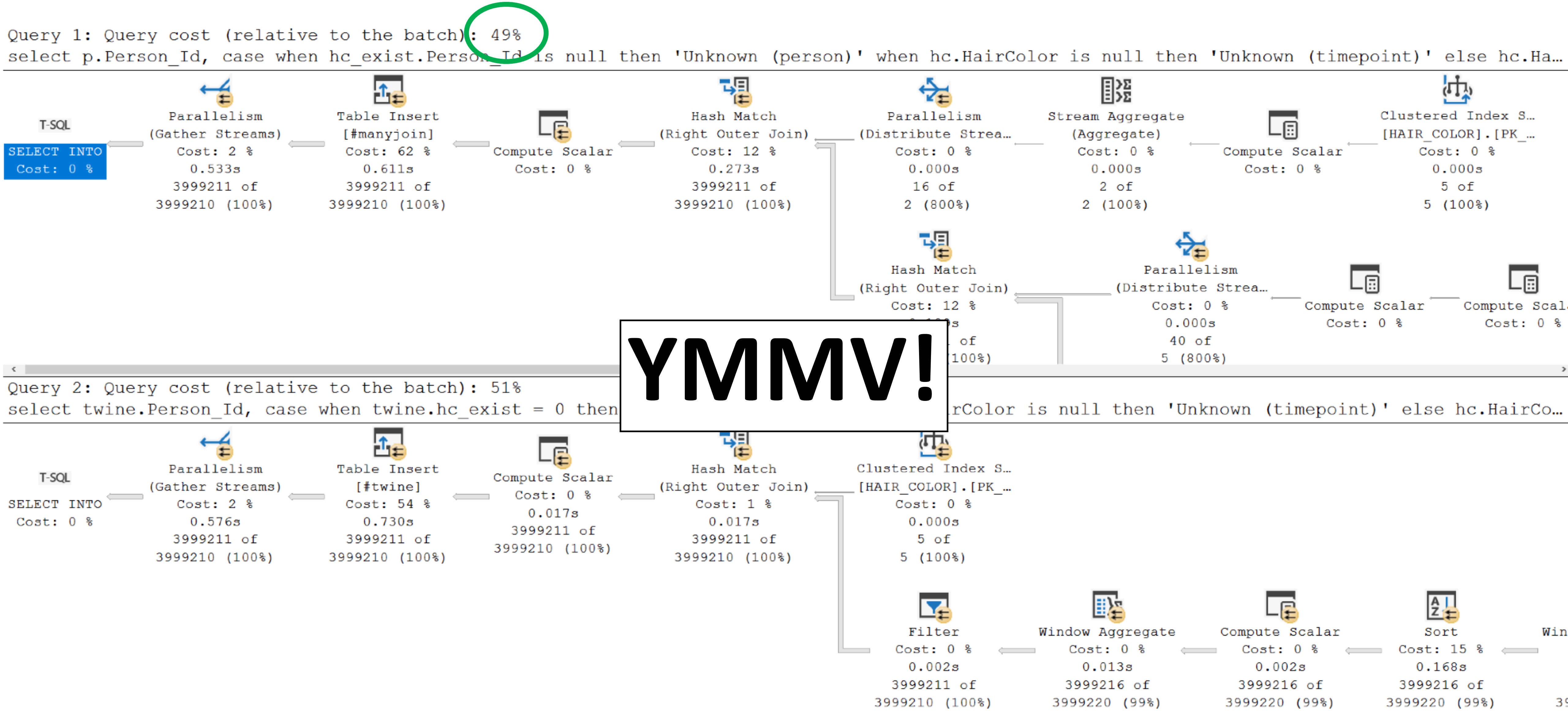
```
select
  twine.Person_Id,
  case
    when twine.hc_exist = 0 then 'Unknown (person)’
    when hc.HairColor is null then 'Unknown (timepoint)’
    else hc.HairColor
  end as HairColor,
  twine.PurchaseDate, hc.Valid_From
from (
  select
    Person_Id, Timeline, Timepoint as PurchaseDate,
    MAX(case when Timeline = 1 then 1 else 0 end)
      over (partition by Person_Id) as hc_exist,
    MAX(case when Timeline = 1 then Timepoint end)
      over (partition by Person_Id order by Timepoint) as Valid_From
  from (
    select Person_Id, 1 as tinyint) as Timeline, Valid_From as Timepoint from HAIR_COLOR
    union all
    select Person_Id, 2 as tinyint) as Timeline, PurchaseDate as Timepoint from PURCHASE
  ) timelines
) twine
left join HAIR_COLOR hc
on hc.Person_Id = twine.Person_Id
and hc.Valid_From = twine.Valid_From
where twine.Timeline = 2
```

Person_Id	HairColor	PurchaseDate	Valid_From
1	Unknown (timepoint)	1999-12-31	*NULL*
1	Blonde	2001-02-03	2001-01-01
1	Blonde	2004-05-06	2001-01-01
2	Purple	2011-11-11	2011-11-11
2	Brown	2023-11-28	2011-11-12
3	Unknown (person)	2018-08-18	*NULL*

An additional join is now necessary after finding the timepoint from timeline 1

TEMPORALLY DEP. JOIN VS GENERALIZED TWINE

4 000 000 ⟷ 4 000 000



```
Query 1: Query cost (relative to the batch): 49%
select p.Person_Id, case when hc_exist.Person_Id is null then 'Unknown (person)' when hc.HairColor is null then 'Unknown (timepoint)' else hc.Ha…
```

```
Query 2: Query cost (relative to the batch): 51%
select twine.Person_Id, case when twine.hc_exist = 0 then … rColor is null then 'Unknown (timepoint)' else hc.HairCo…
```

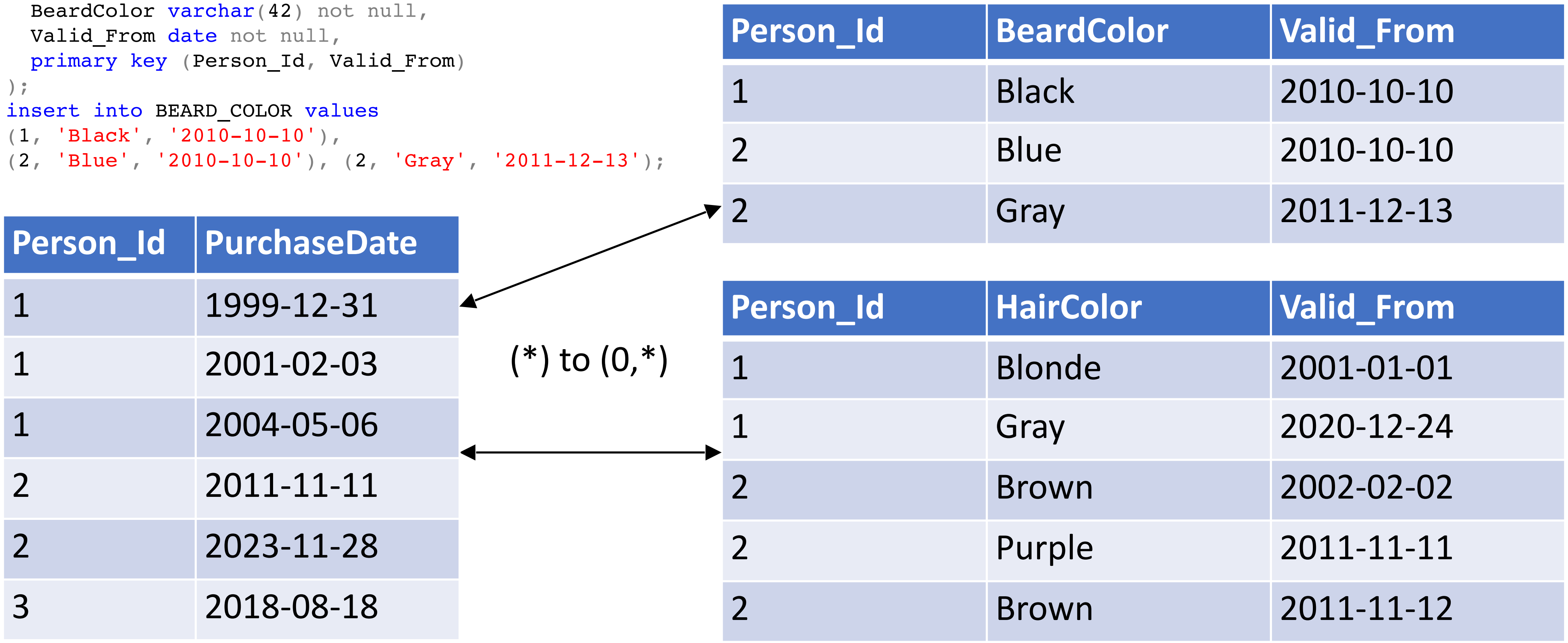
YMMV!

The additional join can actually be avoided, but this query is slightly slower than the one with the join.

```
select
  Person_Id, HairColor, PurchaseDate, Valid_From
from (
  select
    Person_Id, Timeline, PurchaseDate, Valid_From,
    MAX(HairColor) over (partition by Person_Id, Valid_From) as HairColor
  from (
    select
      Person_Id, Timeline, Timepoint as PurchaseDate,
      MAX(case when Timeline = 1 then Timepoint end)
        over (partition by Person_Id order by Timepoint) as Valid_From,
      case
         when Timepoint = MAX(case when Timeline = 1 then Timepoint end)
                            over (partition by Person_Id order by Timepoint)
         then HairColor
      end as HairColor
    from (
      select Person_Id, 1 as Timeline, Valid_From as Timepoint, HairColor from HAIR_COLOR
      union all
      select Person_Id, 2 as Timeline, PurchaseDate as Timepoint, null from PURCHASE
    ) timelines
  ) twine
) t
where t.Timeline = 2
```

MULTIPLE TABLES

```sql
drop table if exists BEARD_COLOR;
create table BEARD_COLOR (
  Person_Id int not null,
  BeardColor varchar(42) not null,
  Valid_From date not null,
  primary key (Person_Id, Valid_From)
);
insert into BEARD_COLOR values
(1, 'Black', '2010-10-10'),
(2, 'Blue', '2010-10-10'), (2, 'Gray', '2011-12-13');
```



Person_Id	BeardColor	Valid_From
1	Black	2010-10-10
2	Blue	2010-10-10
2	Gray	2011-12-13

Person_Id	PurchaseDate
1	1999-12-31
1	2001-02-03
1	2004-05-06
2	2011-11-11
2	2023-11-28
3	2018-08-18

(*) to (0,*)

Person_Id	HairColor	Valid_From
1	Blonde	2001-01-01
1	Gray	2020-12-24
2	Brown	2002-02-02
2	Purple	2011-11-11
2	Brown	2011-11-12

```sql
drop table if exists PURCHASE;
create table PURCHASE (
  Person_Id int not null,
  PurchaseDate date not null,
  primary key (Person_Id, PurchaseDate)
);
insert into PURCHASE values
(1, '1999-12-31'), (1, '2001-02-03'), (1, '2004-05-06'),
(2, '2011-11-11'), (2, '2023-11-28'), (3, '2018-08-18');
```

```sql
drop table if exists HAIR_COLOR;
create table HAIR_COLOR (
  Person_Id int not null foreign key references PERSON (Person_Id),
  HairColor varchar(42) not null,
  Valid_From date not null,
primary key (Person_Id, Valid_From)
);
insert into HAIR_COLOR values
(1, 'Blonde', '2001-01-01'), (1, 'Gray', '2020-12-24'),
(2, 'Brown', '2002-02-02'), (2, 'Purple', '2011-11-11'), (2, 'Brown', '2011-11-12');
```

```
select
  p.Person_Id,
  p.PurchaseDate,
  hc.HairColor, hc.Valid_From as hc_Valid_From,
  bc.BeardColor, bc.Valid_From as bc_Valid_From
from PURCHASE p
left join (
  select Person_Id, HairColor, Valid_From,
         LEAD(Valid_From, 1, '9999-12-31') over
             (partition by Person_Id order by Valid_From) as Valid_To
  from HAIR_COLOR
) hc
on hc.Person_Id = p.Person_Id
and hc.Valid_From <= p.PurchaseDate
and hc.Valid_To > p.PurchaseDate
left join (
  select Person_Id, BeardColor, Valid_From,
         LEAD(Valid_From, 1, '9999-12-31') over
             (partition by Person_Id order by Valid_From) as Valid_To
  from BEARD_COLOR
) bc
on bc.Person_Id = p.Person_Id
and bc.Valid_From <= p.PurchaseDate
and bc.Valid_To > p.PurchaseDate
```

Note that comparison here may yield undesirable results if the granularities of the time types differ.

Multiple timelines can be resolved in a single twine.

```
select
  twine.Person_Id,
  twine.PurchaseDate,
  hc.HairColor, hc.Valid_From as hc_Valid_From,
  bc.BeardColor, bc.Valid_From as bc_Valid_From
from (
  select Person_Id, Timeline, Timepoint as PurchaseDate,
         MAX(case when Timeline = 1 then Timepoint end) over
            (partition by Person_Id order by Timepoint) as hc_Valid_From,
         MAX(case when Timeline = 2 then Timepoint end) over
            (partition by Person_Id order by Timepoint) as bc_Valid_From
  from (
    select Person_Id, 1 as Timeline, Valid_From as Timepoint from HAIR_COLOR
    union all
    select Person_Id, 2 as Timeline, Valid_From as Timepoint from BEARD_COLOR
    union all
    select Person_Id, 0 as Timeline, PurchaseDate as Timepoint from PURCHASE
  ) timelines
) twine
left join HAIR_COLOR hc
on hc.Person_Id = twine.Person_Id
and hc.Valid_From = twine.hc_Valid_From
left join BEARD_COLOR bc
on bc.Person_Id = twine.Person_Id
and bc.Valid_From = twine.bc_Valid_From
where twine.Timeline = 0
```

Note that the union here may fail or implicitly convert time types if the time types differ. Cast to the most granular type if necessary.

CONCLUSIONS

- When you are using twines no end-dating is necessary, as is the case with insert-only data warehouses.
- If you have a temporal one-to-many relationship, a specialized twine is likely to yield the best performance.
- If you have a temporal many-to-many relationship, a generalized twine might yield better performance.
- A single generalized twine can be extended to resolve multiple timelines at once.

- ***Twines are worth testing if performance is an issue!***