Temporal Tables

SQL Server's Built-In Wayback Machine

DevSpace Conference
12 October 2019

Allison Benneth
Allison@sqltran.org
@sqltran
www.sqltran.org

DevSpace would like to thank our sponsors

Welcome to DevSpace

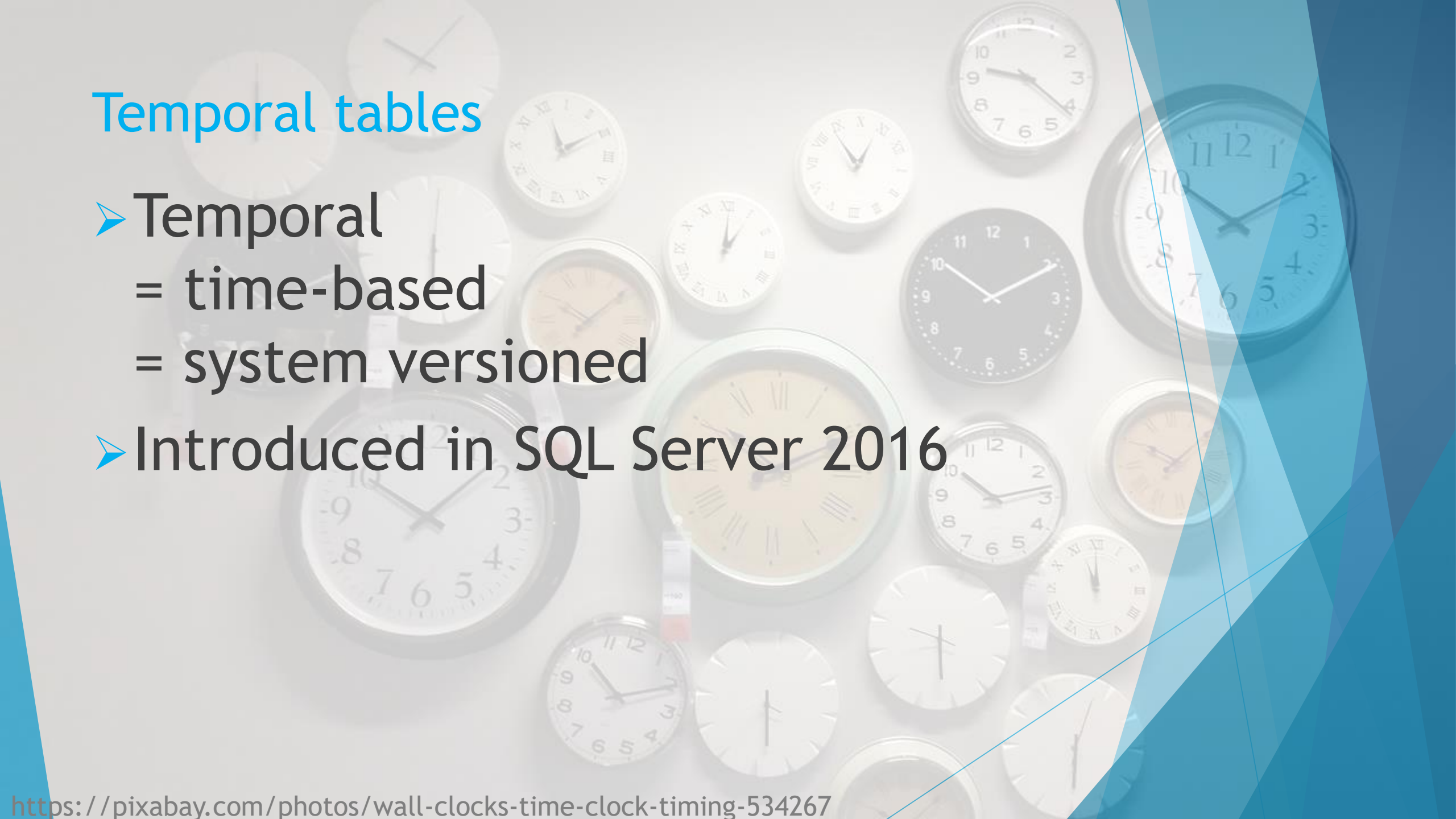
- Enjoy these two days of learning
- Be sure to visit and thank the sponsors
- Be sure to thank the organizer and volunteers
- Take time to NETWORK with others. That's what this is really all about!
- Act professionally and treat others with respect (like this was a work environment)

What we'll cover

- How temporal tables can be used
- Behind the scenes: how they work
- Creating a temporal table
- Querying temporal tables
- Miscellaneous topics

Temporal tables

- Temporal
  = time-based
  = system versioned
- Introduced in SQL Server 2016



https://pixabay.com/photos/wall-clocks-time-clock-timing-534267

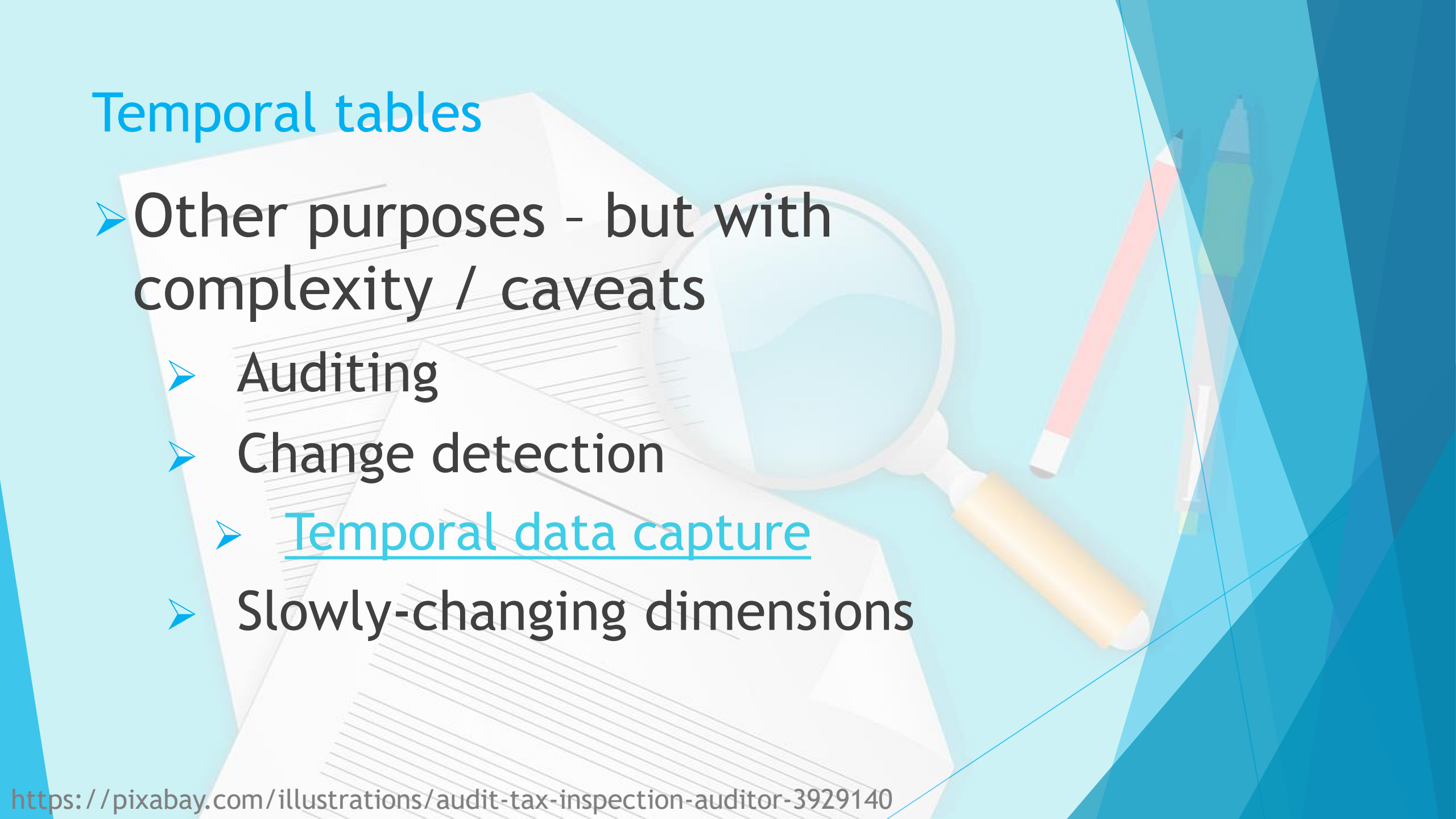
Temporal tables

- Main purposes
  - Logging
  - Reversal of changes
  - Anomaly detection
  - Point-in-time business analytics
  - Trends

https://pixabay.com/illustrations/black-hole-galaxy-science-space-4092609

Temporal tables

- Other purposes – but with complexity / caveats
  - Auditing
  - Change detection
    - Temporal data capture
  - Slowly-changing dimensions



https://pixabay.com/illustrations/audit-tax-inspection-auditor-3929140

How Temporal Tables Work

How it works

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

How it works

ID	Address	City	State	Start	End

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

How it works

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	10/12/19

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

How it works

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	10/12/19

ID	Address	City	State	Start	End
51905	2153 N Marazzani Ln	Crown Point	IN	10/12/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

How it works

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	10/12/19
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	10/12/19

ID	Address	City	State	Start	End
51905	2153 N Marazzani Ln	Crown Point	IN	10/12/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

How it works

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	10/12/19
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	10/12/19

ID	Address	City	State	Start	End
51905	2153 N Marazzani Ln	Crown Point	IN	10/12/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

Instance and database configuration

- Instance-level configuration required for temporal tables

- Database-level configuration required for temporal tables

Creating Temporal Tables

Create temporal table

```
create table dbo.Manufacturer
(
    ManufacturerId int not null identity(1,1),
    ManufacturerName nvarchar(50) not null,



    constraint pk_Manufacturer primary key clustered (ManufacturerId)
);
```

Create temporal table

```
create table dbo.Manufacturer
(
    ManufacturerId int not null identity(1,1),
    ManufacturerName nvarchar(50) not null,
    ValidFrom datetime2 generated always as row start not null,
    ValidTo datetime2 generated always as row end not null,

    constraint pk_Manufacturer primary key clustered (ManufacturerId)
);
```

Create temporal table

```
create table dbo.Manufacturer
(
    ManufacturerId int not null identity(1,1),
    ManufacturerName nvarchar(50) not null,
    ValidFrom datetime2 generated always as row start not null,
    ValidTo datetime2 generated always as row end not null,
    period for system_time (ValidFrom, ValidTo),
    constraint pk_Manufacturer primary key clustered (ManufacturerId)
);
```

Create temporal table

```
create table dbo.Manufacturer
(
    ManufacturerId int not null identity(1,1),
    ManufacturerName nvarchar(50) not null,
    ValidFrom datetime2 generated always as row start not null,
    ValidTo datetime2 generated always as row end not null,
    period for system_time (ValidFrom, ValidTo),
    constraint pk_Manufacturer primary key clustered (ManufacturerId)
)
with (system_versioning = on);
```

Three ways to create history table

- Let SQL Server create and name it
- Let SQL Server create it with a name of your choosing
- Create and name it yourself

History table (let SQL Server name it)

```
create table dbo.Manufacturer (...)
with (system_versioning = on);
```

- SQL Server creates a table with a name like:

```
dbo.MSSQL_TemporalHistoryFor_565577053
```

History table (let SQL Server create it)

```
create table dbo.Manufacturer (...)
with (system_versioning = on
(history_table = history.ManufacturerHistory));
```

History table (DIY)

```
create table history.ManufacturerHistory
(
   ManufacturerId int not null,
   ManufacturerName nvarchar(50) not null,
   ValidFrom datetime2 not null,
   ValidTo datetime2 not null
);

create table dbo.Manufacturer(...)
with (system_versioning = on
(history_table = history.ManufacturerHistory));
```

History table (DIY)

- When creating your own history table
  - Must have same structure, data types, nullability
  - Cannot have constraints (primary key, foreign keys, etc.)
  - None of these:
    - `generated always as row start / end`
    - `period for system_time`
    - `identity`
- So why DIY?

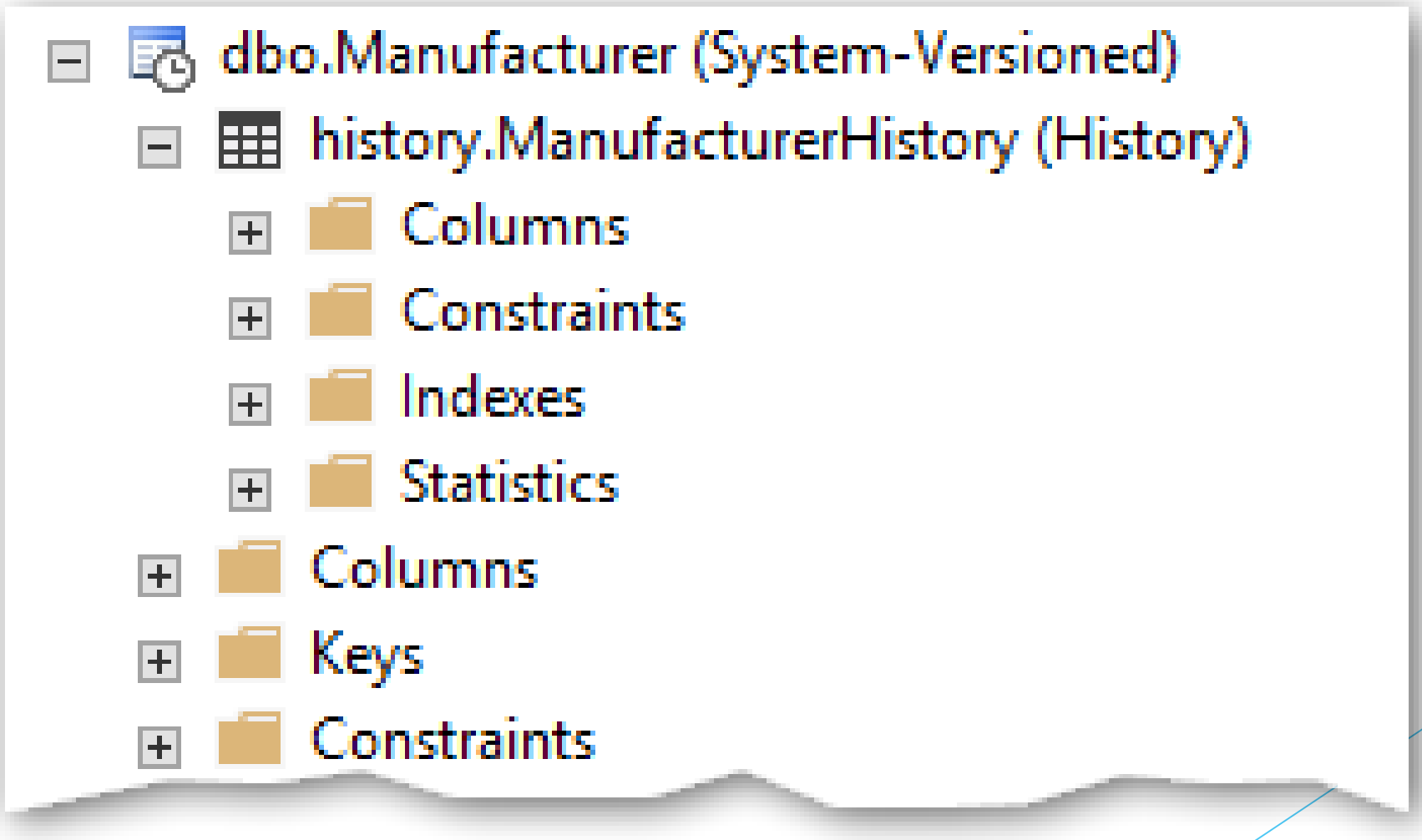
History table defaults

- History tables created by SQL Server are page compressed
  - Consider page compressing DIY history tables as well

```
create table history.CustomerHistory
(...) with (data_compression = page);
```

- Clustered index on history table created by SQL is on (ValidTo, ValidFrom)
  - Consider appropriate clustered index for all history tables, whether SQL-created or DIY

SSMS and temporal tables



datetime2(n)

- Introduced in SQL Server 2008
- Has a date range from 0001-Jan-01 to 9999-Dec-31
- When declaring, give an optional precision from 0 to 7
  - Default is 7
  - Represents number of digits in fractional part of a second
  - Maximum/default precision is thus 100 ns (0.0000001 s)
- All system-generated times in temporal tables are UTC
- 9999-12-31 23:59:59.9999999

Heat death of the universe

Writing to temporal tables

```
insert dbo.Manufacturer (ManufacturerName)
    values ('Acme, Inc.');

select * from dbo.Manufacturer;
select * from history.ManufacturerHistory;
```

ManufacturerId	ManufacturerName	ValidFrom	ValidTo
1	Acme, Inc.	2019-09-04 17:59:54.0484623	9999-12-31 23:59:59.9999999

ManufacturerId	ManufacturerName	ValidFrom	ValidTo

Writing to temporal tables

```
update dbo.Manufacturer
set ManufacturerName = 'XYZ Corp.'
where ManufacturerId = 1;


select * from dbo.Manufacturer;
select * from history.ManufacturerHistory;
```

ManufacturerId	ManufacturerName	ValidFrom	ValidTo
1	XYZ Corp.	2019-09-04 18:23:39.7147725	9999-12-31 23:59:59.9999999

ManufacturerId	ManufacturerName	ValidFrom	ValidTo
1	Acme, Inc.	2019-09-04 17:59:54.0484623	2019-09-04 18:23:39.7147725

Writing to temporal tables

```
delete from dbo.Manufacturer
where ManufacturerId = 1;

select * from dbo.Manufacturer;
select * from history.ManufacturerHistory;
```

ManufacturerId	ManufacturerName	ValidFrom	ValidTo

ManufacturerId	ManufacturerName	ValidFrom	ValidTo
1	Acme, Inc.	2019-09-04 17:59:54.0484623	2019-09-04 18:23:39.7147725
1	XYZ Corp.	2019-09-04 18:23:39.7147725	2019-09-04 18:29:01.1964976

Hidden columns

```
create table dbo.Manufacturer
(
    ManufacturerId int not null identity(1,1),
    ManufacturerName nvarchar(50) not null,
    ValidFrom datetime2 generated always as row start hidden not null,
    ValidTo datetime2 generated always as row end hidden not null,
    period for system_time (ValidFrom, ValidTo),
    constraint pk_Manufacturer primary key clustered (ManufacturerId)
)
with (system_versioning = on);
```

Hidden columns

```
insert dbo.Manufacturer (ManufacturerName)
    values ('Acme, Inc.');


select * from dbo.Manufacturer;
```

ManufacturerId	ManufacturerName
1	Acme, Inc.

```
select ManufacturerId, ManufacturerName, ValidFrom, ValidTo
from dbo.Manufacturer;
```

ManufacturerId	ManufacturerName	ValidFrom	ValidTo
1	Acme, Inc.	2019-09-23 14:01:37.7139591	9999-12-31 23:59:59.9999999

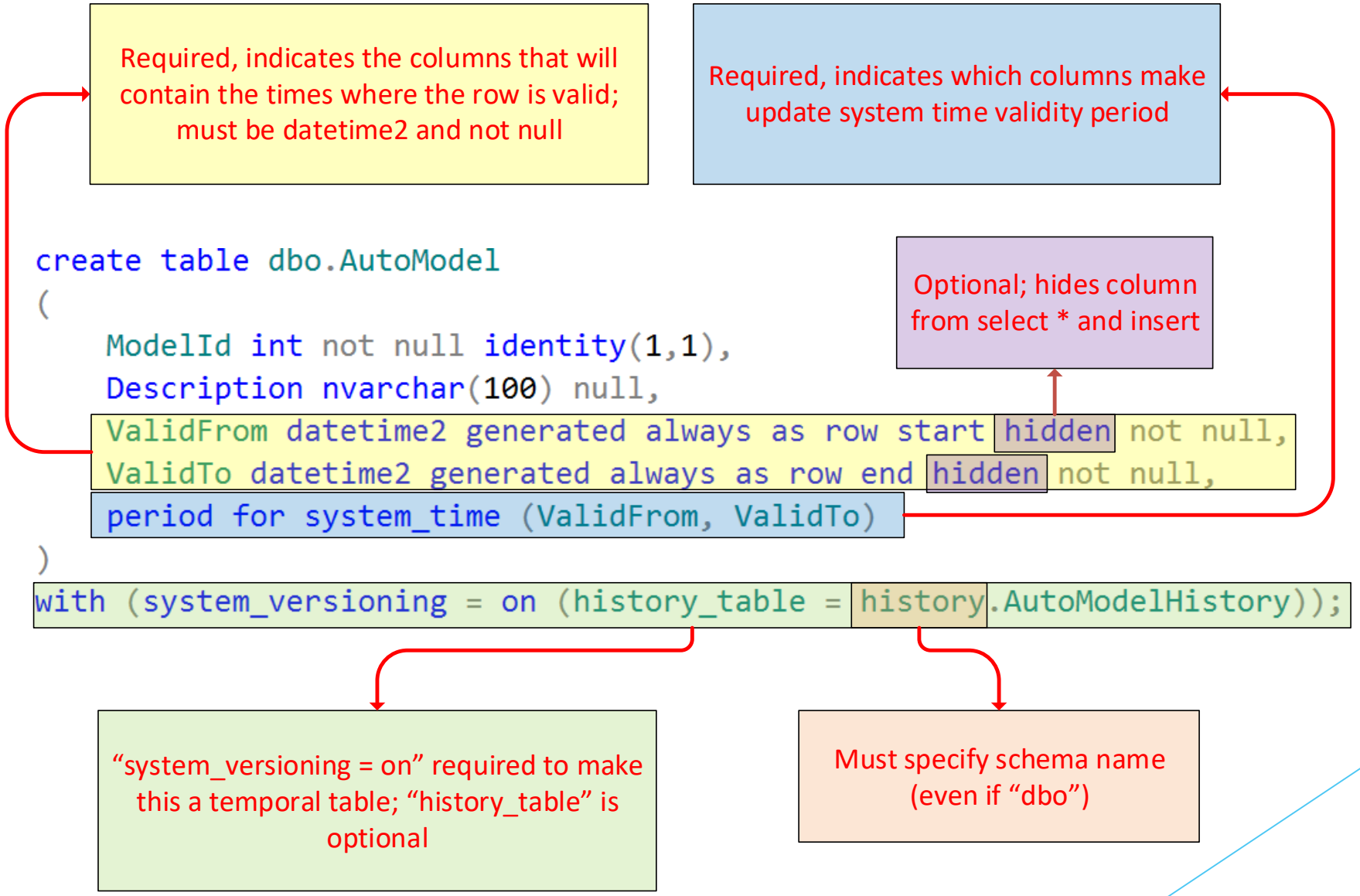
Hidden columns

```
insert dbo.Manufacturer
    values ('XYZ Corp.');
```

- This works OK with “hidden” attribute
- If “hidden” attribute was missing:

```
Msg 213, Level 16, State 1, Line 35
Column name or number of supplied
values does not match table
definition.
```

Summary of creating temporal tables



Querying Temporal Tables

Querying temporal tables

- Temporal table can be queried just like any other table
- History table can be queried just like any other table
- SQL now provides a new clause (used with temporal table)

```
for system_time
```

- Is followed by one of five time period definitions

```
select …
from dbo.Manufacturer
for system_time as of '2019-10-12 16:00';
```

Querying temporal tables

- `for system_time` performs a “logical” concatenation (UNION ALL) on the temporal table plus the history table.

ID	Address	City	State	Start	End
51905	2153 N Marazzani Ln	Crown Point	IN	10/12/19	12/31/99
57400	922 NE Ribstone Ln	Garden Grove	CA	2/15/19	12/31/99
46627	548 S Castlehaven Ln	Shreveport	LA	9/4/19	12/31/99
20052	806 E Aste St	Greenville	NC	10/10/19	12/31/99

ID	Address	City	State	Start	End
51905	13777 NE Grandys St	Omaha	NE	1/1/19	10/12/19
24009	4068 N Tillingbourne St	Miami Gardens	FL	3/31/19	10/12/19

Querying temporal tables

Temporal querying:	`FROM TableName FOR SYSTEM_TIME ____`
Point in time	`AS OF '2019-10-11 08:00:00'`
Full history	`ALL`
Between (ValidFrom <= EndTime AND ValidTo > StartTime)	`BETWEEN '2019-10-11 08:00:00' AND '2019-10-12 17:00:00'`
From (ValidFrom< EndTime AND ValidTo > StartTime)	`FROM '2019-10-11 08:00:00' TO '2019-10-12 17:00:00'`
Contained in (ValidFrom >= StartTime AND ValidTo <= EndTime)	`CONTAINED IN ('2019-10-11 08:00:00', '2019-10-12 17:00:00')`

Querying temporal tables

- Time(s) in `for system_time` clause can be constants or variables:

```
declare @asOfTime datetime2 =
   '2019-10-12 16:00:00';


select …
from dbo.Manufacturer
for system_time as of @asOfTime;
```

Querying multiple temporal tables

- Query against 3 temporal tables:

```
select …
from dbo.Manufacturer
   for system_time as of '2019-10-12 16:00' mfg
join dbo.Product
   for system_time as of '2019-10-12 16:00' prd
   on prd.ManufacturerId = mfg.ManufacturerId
join dbo.OrderDetail
   for system_time as of '2019-10-12 16:00' od
   on od.ProductId = prd.ProductId
where prd.Name = 'Widget';
```

- Can be clunky and error prone

Querying multiple temporal tables

- Solution: Create a view

```
create view vwProductOrder as
select …
from dbo.Manufacturer mfg
join dbo.Product prd
  on prd.ManufacturerId = mfg.ManufacturerId
join dbo.OrderDetail od
  on od.ProductId = prd.ProductId
```

Querying multiple temporal tables

- Then query the view

```
select …
from vwProductOrder
   for system_time
   as of '2019-10-12 16:00' po
where po.Name = 'Widget';
```

Miscellaneous Topics

Miscellaneous topics

- Effect of transactions on insert/update/delete operations
- Propagation of schema changes
- AT TIME ZONE function

```
select sysdatetime()
at time zone 'Central Standard Time’
at time zone 'UTC';

select sysdatetimeoffset()
at time zone 'UTC’;
```

- Performance considerations
- Indexing the history table

Retention and archiving

- Built-in method in SQL 2017+ only
- Retention only, no archiving

```
create table dbo.Customer
(
…
)
with (system_versioning = on
(history_table = history.CustomerHistory,
history_retention_period = 3 months));
```

Retention and archiving (SQL 2016+)

- Stretch database (move data to table in Azure, becomes extension of history table)
- Custom scripts
  - Partitioning on history table (and optional archive table)
  - Trickle copy and/or delete

Miscellaneous topics

- New catalog objects
  - sys.periods (view)
  - sys.tables.temporal_type (column)
  - sys.tables.temporal_type_desc (column)
  - sys.tables.history_table_id (column)
  - sys.columns.generated_always_type (column)
  - sys.columns.generated_always_type_desc (column)

Thank You

- This presentation and supporting materials can be found at www.sqltran.org/temporal.
  - Slide deck
  - Scripts

- allison@sqltran.org • @sqltran