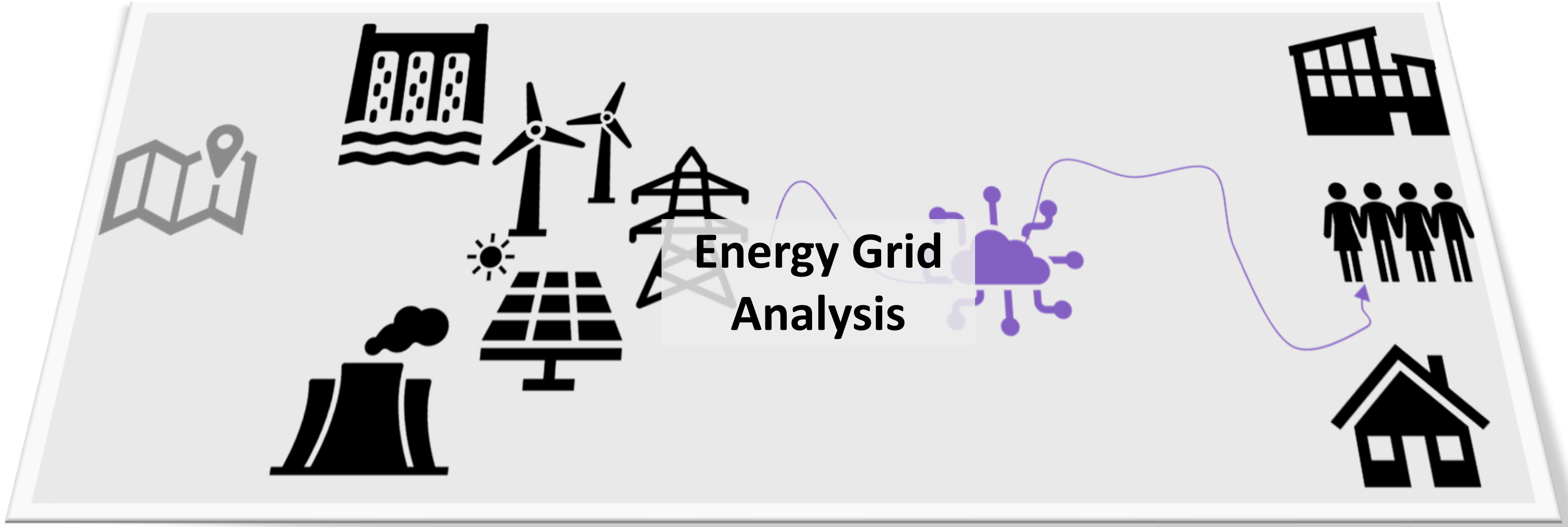
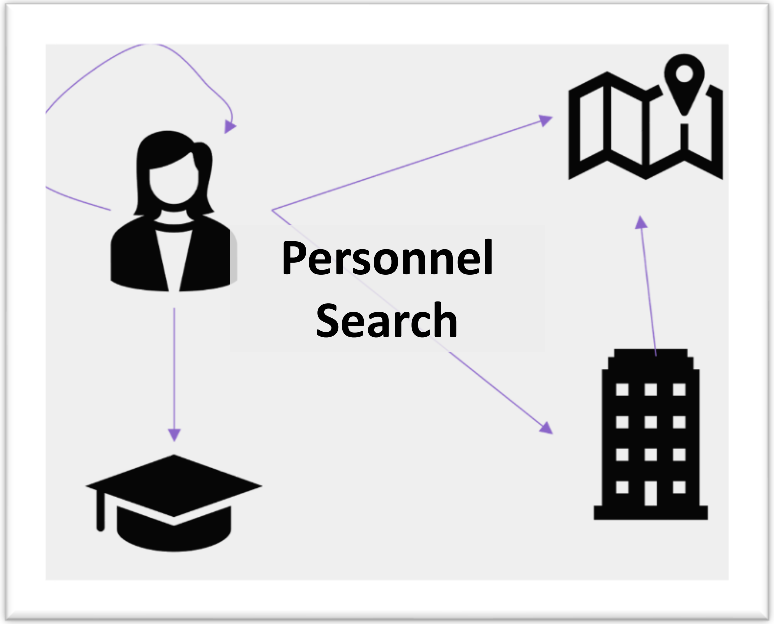
Arvind Shyamsundar

Principal Product Manager | Azure SQL DB (Microsoft)

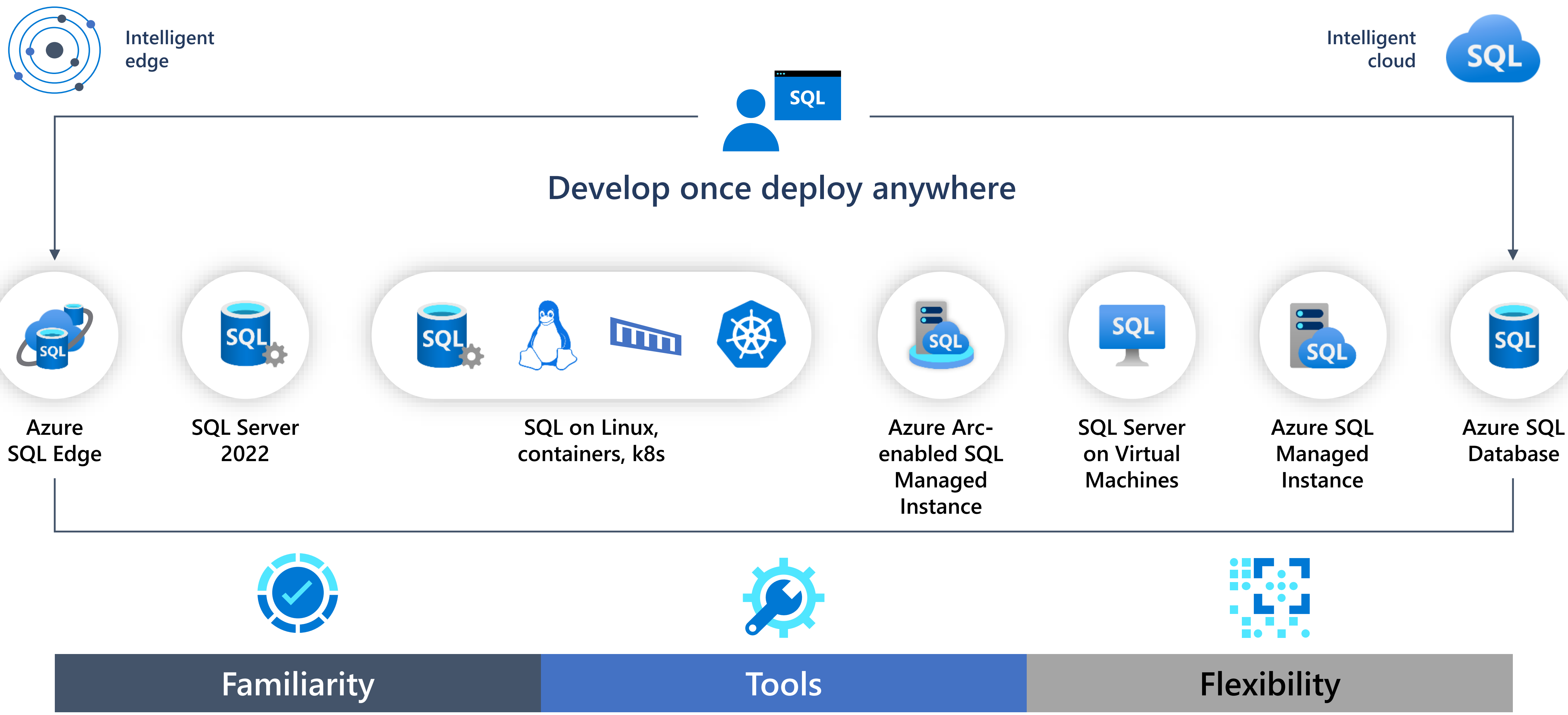
Graph capabilities in Microsoft SQL Server and Azure SQL Database

Typical customer scenarios

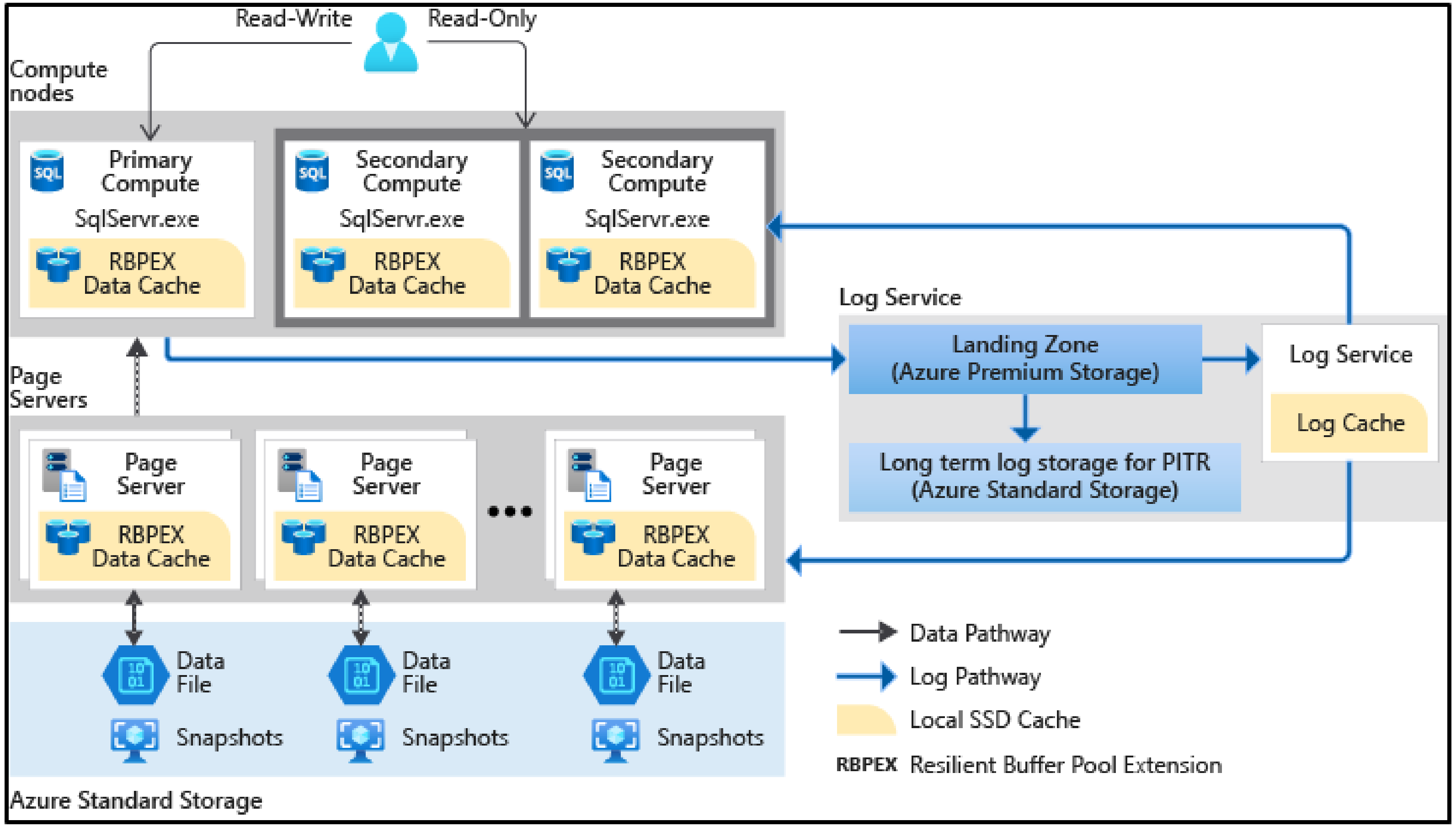




The "MSSQL" family



Azure SQL Database – Hyperscale tier



Salient features:

- 100% PaaS offering
- Scale compute up / down independent of the amount storage
- Resilient SSD-based caches
- Redundant data and log backed by durable Azure Storage
- 0 to 4 secondary HA replicas
- 0-30 named replicas for read scale
- Backup/Restore via storage snapshots

Paper: “Socrates: The New SQL Server in the Cloud”: https://www.microsoft.com/en-us/research/uploads/prod/2019/05/socrates.pdf

(MS)SQL Graph

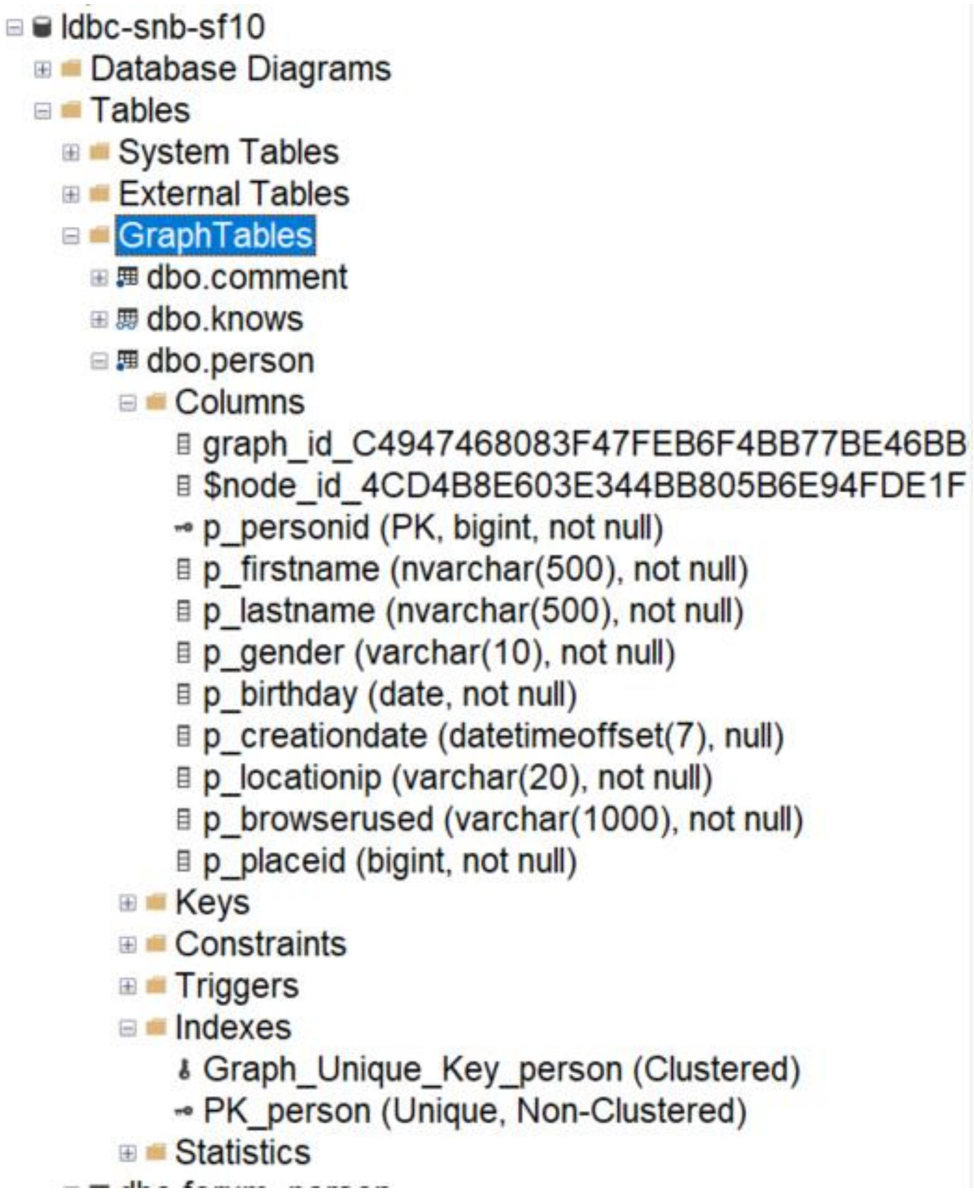
“What” and “how”

CREATE TABLE … AS NODE

- A node table in (MS)SQL Graph looks almost identical to a regular table:

```
CREATE TABLE person (
    p_personid     BIGINT          ,
    p_firstname    NVARCHAR (500) NOT NULL,
    p_lastname     NVARCHAR (500) NOT NULL,
    p_gender       VARCHAR (10)   NOT NULL,
    p_birthday     DATE           NOT NULL,
    p_creationdate DATETIMEOFFSET,
    p_locationip   VARCHAR (20)   NOT NULL,
    p_browserused  VARCHAR (1000) NOT NULL,
    p_placeid      BIGINT         NOT NULL,
    CONSTRAINT PK_person PRIMARY KEY NONCLUSTERED
(p_personid ASC) WITH (DATA_COMPRESSION = PAGE),
    CONSTRAINT Graph_Unique_Key_person UNIQUE CLUSTERED
($node_id) WITH (DATA_COMPRESSION = PAGE),

) AS NODE;
```

- Note: SQL treats the set of node and edge tables within one database as one logical graph.



Inserting data into a node table

- Regular INSERTs work; the graph ID column (and hence the user-visible $node_id pseudo-column) is auto-generated:

```
INSERT  INTO [dbo].[person] ([p_firstname], [p_lastname], [p_gender], [p_birthday], [p_creationdate],
[p_locationip], [p_browserused], [p_placeid])
VALUES (...)
```

- Here's how the data (including the $node_id column) looks:

$node_id_4CD4B8E603E344BB805B6E94FDE1FBE4	p_personid	p_firstname	p_lastname	p_gender	p_birthday	p_creationdate
{"type":"node","schema":"dbo","table":"person","id":65}	65	Marc	Ravalomanana	female	1989-06-15	2010-02-26 23:1
{"type":"node","schema":"dbo","table":"person","id":94}	94	K.	Sen	female	1980-08-16	2010-01-06 18:3
{"type":"node","schema":"dbo","table":"person","id":96}	96	Anson	Chen	female	1981-12-25	2010-01-17 12:1
{"type":"node","schema":"dbo","table":"person","id":102}	102	Philibert	Roindefo	female	1987-05-09	2010-02-19 09:1
{"type":"node","schema":"dbo","table":"person","id":143}	143	Maria	Alkaios	female	1983-01-06	2010-01-06 07:1
{"type":"node","schema":"dbo","table":"person","id":150}	150	Alfonso	Alvarez	female	1983-01-01	2010-01-11 08:5
{"type":"node","schema":"dbo","table":"person","id":238}	238	Burak	Koksal	male	1982-10-22	2010-01-07 11:2
{"type":"node","schema":"dbo","table":"person","id":250}	250	Rahul	Kumar	female	1983-10-08	2010-01-27 18:5
{"type":"node","schema":"dbo","table":"person","id":267}	267	Aburizal	[illegible]	female	1985-11-02	2010-01-18 20:1

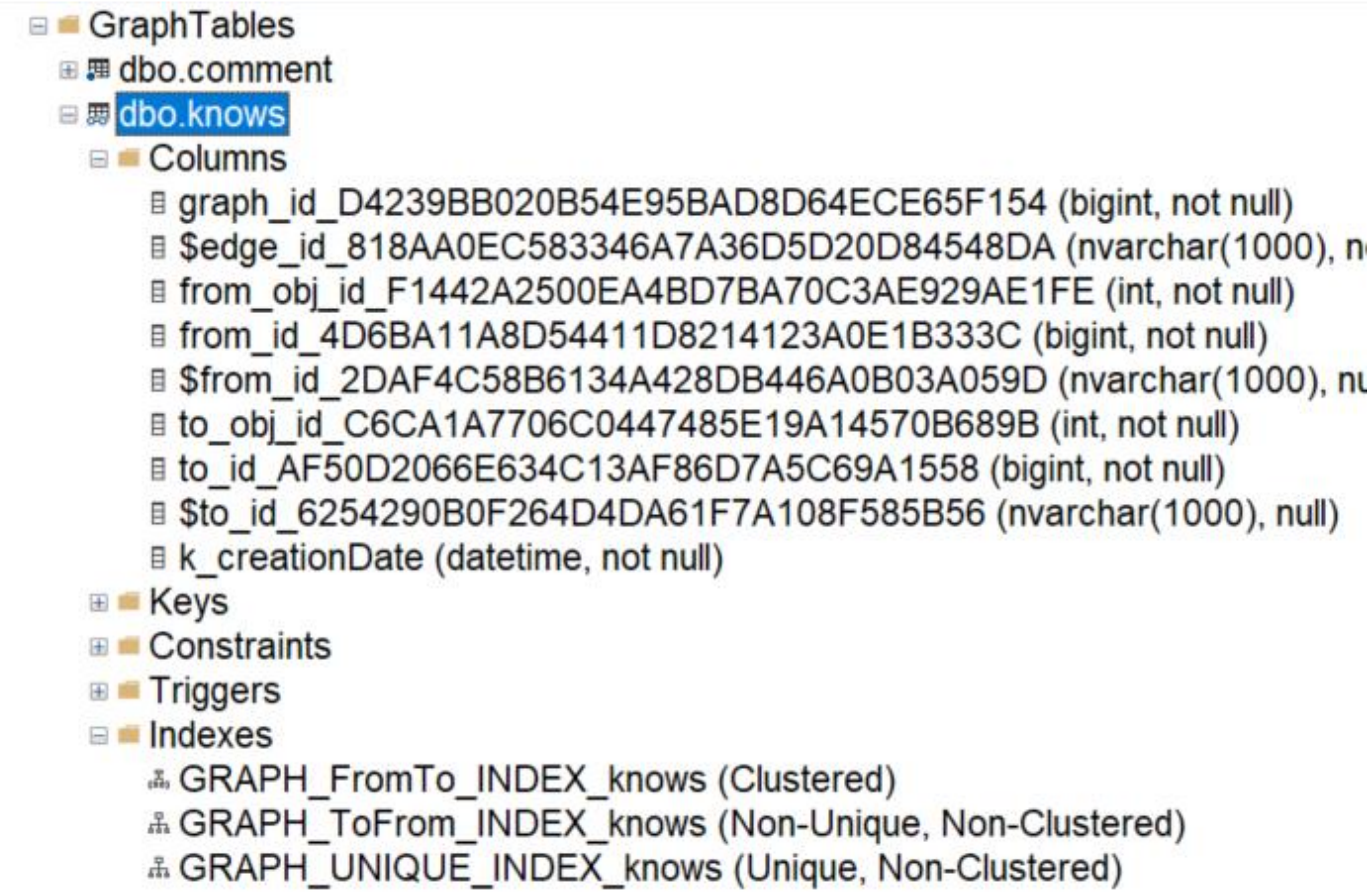
Bulk insert into node table

```sql
INSERT person ($NODE_ID, p_personid, p_firstname, p_lastname, p_gender, p_birthday,
p_creationdate, p_locationip, p_browserused, p_placeid)
SELECT NODE_ID_FROM_PARTS(object_id('person'), id),
       id,
       firstName,
       lastName,
       gender,
       birthday,
       creationDate,
       locationIP,
       browserUsed,
       placeId
FROM   OPENROWSET (BULK 'unsplit/social_network-csv_merge_foreign-
sf10/dynamic/person_0_0.csv', DATA_SOURCE = 'ldbcstorage', FORMATFILE = 'format-
files/person.xml', FORMATFILE_DATA_SOURCE = 'ldbcstorage', FIRSTROW = 2) AS raw;
```

Edge tables

- Edge tables in (MS)SQL Graph can have 0 or more user defined columns (“properties”)
- Edge tables can be used to “connect” any node to any other node in the graph.

```
CREATE TABLE [dbo].[knows] (
k_creationDate DATETIME NOT NULL,
INDEX [GRAPH_UNIQUE_INDEX_knows] UNIQUE NONCLUSTERE
($edge_id) WITH (DATA_COMPRESSION = PAGE),
INDEX [GRAPH_FromTo_INDEX_knows] CLUSTERED
($from_id, $to_id) WITH (DATA_COMPRESSION = PAGE)
, INDEX [GRAPH_ToFrom_INDEX_knows] NONCLUSTERED
($to_id, $from_id) WITH (DATA_COMPRESSION = PAGE),
 CONSTRAINT ec_person_person CONNECTION (person to
person),
) AS EDGE;
```



Inserting data into an edge table

- INSERTs require specifying the $from_id and $to_id pseudo-columns, along with any other required properties (columns). For example:

```
INSERT  knows ($from_id, $to_id, k_creationDate)
VALUES       ((SELECT $NODE_ID FROM   person WHERE  p_personId = 94)
    , (SELECT $NODE_ID FROM   person WHERE  p_personId = 60927)
    , '2010-01-25 08:36:33.053');
```

- Here's how the (edge table) data looks:

Results | Messages

$edge_id_818AA0EC583346A7A36D5D20D84548DA	$from_id_2DAF4C58B6134A428DB446A0B03A059D	$to_id_6254290B0F264D4DA61F7A108F585B56	k_creationDate
{"type":"edge","schema":"dbo","table":"knows","id":1938518}	{"type":"node","schema":"dbo","table":"person","id":94}	{"type":"node","schema":"dbo","table":"person","id":60927}	2010-01-25 08:36:33.053

- An efficient way to reuse the natural keys in the data as the graph IDs is possible, by using the NODE_ID_FROM_PARTS function:

```
INSERT  knows ($from_id, $to_id, k_creationDate)
VALUES       (NODE_ID_FROM_PARTS(object_id('person'), 94), NODE_ID_FROM_PARTS(object_id('person'), 60927), '2010-01-25
08:36:33.053');
```

Bulk insert into edge table

- Using INSERT … SELECT … OPENROWSET … BULK and NODE_ID_FROM_PARTS, it is possible to efficiently insert bulk data into edge tables:

```
INSERT knows ($FROM_ID, $TO_ID, k_creationDate)
SELECT NODE_ID_FROM_PARTS(object_id('person'), from_id) AS from_id,
       NODE_ID_FROM_PARTS(object_id('person'), to_id) AS to_id,
       creationDate
FROM   OPENROWSET (BULK 'unsplit/social_network-csv_merge_foreign-
sf10/dynamic/person_knows_person_0_0.csv', DATA_SOURCE = 'ldbcstorage',
FORMATFILE = 'format-files/person_knows_person.xml', FORMATFILE_DATA_SOURCE =
'ldbcstorage', FIRSTROW = 2) AS raw;
```

Querying the graph

Pattern matching using MATCH

- The MATCH predicate provides multi-hop navigation and join-free pattern matching using ASCII-art syntax to facilitate graph traversal. Here's the full SQL query using MATCH:

```
SELECT TOP 5 p2.p_personid,
             p2.p_firstname,
             p2.p_lastname
FROM   person AS p1, knows, person AS p2
WHERE  MATCH(p1-(knows)->p2)
       AND p1.p_personId = 94;
```

- LDBC SNB query IC2:

```
SELECT   TOP (20) p2.p_personid,
                  p2.p_firstname,
                  p2.p_lastname,
                  m_messageid,
                  COALESCE (m_ps_imagefile, m_content),
                  m_creationdate
FROM     person AS p1, knows, person AS p2, [message]
WHERE    MATCH(p1-(knows)->p2)
         AND p2.p_personid = m_creatorid
         AND m_creationdate <= '2012-12-31'
         AND p1.p_personid = 94
ORDER BY m_creationdate DESC, m_messageid ASC;
```

LDBC SNB IC1 with “naïve” MATCH

```
;WITH      FriendQuery
AS        (SELECT (Person2.p_personid) AS friendId, 1 AS distanceFromPerson,
                  (Person2.p_firstname) AS friendFirstName, (Person2.p_lastname) AS friendLastName,
(Person2.p_birthday) AS friendBirthday,
                  (Person2.p_creationdate) AS friendCreationDate, (Person2.p_gender) AS friendGender,
(Person2.p_browserused) AS friendBrowserUsed,
                  (Person2.p_locationip) AS friendLocationIp, (Person2.p_placeid) AS friendPlaceId
          FROM    person AS Person1, knows AS k, person AS Person2
          WHERE   MATCH(Person1-(k)->Person2)
                  AND Person1.p_personid = 94 AND Person1.p_personid != Person2.p_personid
          UNION
          SELECT (Person3.p_personid) AS friendId, 2 AS distanceFromPerson,
                  (Person3.p_firstname) AS friendFirstName, (Person3.p_lastname) AS friendLastName,
(Person3.p_birthday) AS friendBirthday,
                  (Person3.p_creationdate) AS friendCreationDate, (Person3.p_gender) AS friendGender,
(Person3.p_browserused) AS friendBrowserUsed,
                  (Person3.p_locationip) AS friendLocationIp, (Person3.p_placeid) AS friendPlaceId
          FROM    person AS Person1, knows AS k1, person AS Person2, knows AS k2, person AS Person3
          WHERE   MATCH(Person1-(k1)->Person2 AND Person2-(k2)->Person3)
                  AND Person1.p_personid = 94 AND Person1.p_personid != Person3.p_personid
          UNION
          SELECT (Person4.p_personid) AS friendId, 3 AS distanceFromPerson,
                  (Person4.p_firstname) AS friendFirstName, (Person4.p_lastname) AS friendLastName,
                  (Person4.p_birthday) AS friendBirthday, (Person4.p_creationdate) AS
friendCreationDate,
                  (Person4.p_gender) AS friendGender, (Person4.p_browserused) AS friendBrowserUsed,
                  (Person4.p_locationip) AS friendLocationIp, (Person4.p_placeid) AS friendPlaceId
          FROM    person AS Person1, knows AS k1, person AS Person2, knows AS k2, person AS Person3,
knows AS k3, person AS Person4
          WHERE   MATCH(Person1-(k1)->Person2 AND Person2-(k2)->Person3 AND Person3-(k3)->Person4)
                  AND Person1.p_personid = 94 AND Person1.p_personid != Person4.p_personid)
```

```
SELECT   TOP (20) friendId, friendFirstName, friendLastName,
distanceFromPerson, friendBirthday, friendCreationDate, friendGender,
friendBrowserUsed, friendLocationIp,
                  (SELECT   string_agg(pe_email, ';')
                   FROM     person_email
                   WHERE    pe_personid = friendId
                   GROUP BY pe_personid) AS emails,
                  (SELECT   string_agg(plang_language, ';')
                   FROM     person_language
                   WHERE    plang_personid = friendId
                   GROUP BY plang_personid) AS languages,
                  (SELECT pl_name
                   FROM   place AS p1
                   WHERE  p1.pl_placeid = friendPlaceId) AS pl_name,
                  (SELECT   string_agg(CONCAT(o2.o_name, '|',
pu_classyear, '|', p2.pl_name), ';')
                   FROM     person_university, organisation AS o2,
place AS p2 WHERE   pu_personid = friendId AND pu_organisationid =
o2.o_organisationid AND o2.o_placeid = p2.pl_placeid GROUP BY
pu_personid) AS university,
                  (SELECT   string_agg(CONCAT(o3.o_name, '|',
pc_workfrom, '|', p3.pl_name), ';')
                   FROM     person_company, organisation AS o3, place
AS p3
                   WHERE    pc_personid = friendId AND
pc_organisationid = o3.o_organisationid AND o3.o_placeid =
p3.pl_placeid GROUP BY pc_personid) AS company
FROM     FriendQuery AS Q
WHERE    Q.friendFirstName = 'Peter'
ORDER BY Q.distanceFromPerson ASC, Q.friendLastName ASC, Q.friendId
ASC;
```

LDBC SNB query IC1 with SHORTEST_PATH

```
;WITH     FriendQuery
AS        (SELECT LAST_VALUE(Person2.p_personid) WITHIN GROUP ( GRAPH PATH) AS friendId,
                  COUNT(Person2.p_personid) WITHIN GROUP ( GRAPH PATH) AS distanceFromPerson,
                  LAST_VALUE(Person2.p_firstname) WITHIN GROUP ( GRAPH PATH) AS friendFirstName,
                  LAST_VALUE(Person2.p_lastname) WITHIN GROUP ( GRAPH PATH) AS friendLastName,
                  LAST_VALUE(Person2.p_birthday) WITHIN GROUP ( GRAPH PATH) AS friendBirthday,
                  LAST_VALUE(Person2.p_creationdate) WITHIN GROUP ( GRAPH PATH) AS friendCreationDate,
                  LAST_VALUE(Person2.p_gender) WITHIN GROUP ( GRAPH PATH) AS friendGender,
                  LAST_VALUE(Person2.p_browserused) WITHIN GROUP ( GRAPH PATH) AS friendBrowserUsed,
                  LAST_VALUE(Person2.p_locationip) WITHIN GROUP ( GRAPH PATH) AS friendLocationIp,
                  LAST_VALUE(Person2.p_placeid) WITHIN GROUP ( GRAPH PATH) AS friendPlaceId
           FROM   person AS Person1, knows FOR PATH AS k, person FOR PATH AS Person2
           WHERE  MATCH(SHORTEST_PATH(Person1(-(k)->Person2){1, 3}))
                  AND Person1.p_personid = 94)
SELECT   TOP (20) friendId, friendFirstName, friendLastName, distanceFromPerson, friendBirthday, friendCreationDate, friendGender, friendBrowserUsed,
friendLocationIp,
                  (SELECT   string_agg(pe_email, ';') FROM     person_email WHERE    pe_personid = friendId GROUP BY pe_personid) AS emails,
                  (SELECT   string_agg(plang_language, ';') FROM     person_language WHERE    plang_personid = friendId GROUP BY plang_personid) AS languages,
                  (SELECT pl_name FROM   place AS p1 WHERE  p1.pl_placeid = friendPlaceId) AS pl_name,
                  (SELECT   string_agg(CONCAT(o2.o_name, '|', pu_classyear, '|', p2.pl_name), ';') FROM     person_university, organisation AS o2, place AS p2
                   WHERE    pu_personid = friendId AND pu_organisationid = o2.o_organisationid AND o2.o_placeid = p2.pl_placeid GROUP BY pu_personid) AS
university,
                  (SELECT   string_agg(CONCAT(o3.o_name, '|', pc_workfrom, '|', p3.pl_name), ';') FROM     person_company, organisation AS o3, place AS p3
                   WHERE    pc_personid = friendId AND pc_organisationid = o3.o_organisationid AND o3.o_placeid = p3.pl_placeid
                   GROUP BY pc_personid) AS company
FROM     FriendQuery AS Q
WHERE    Q.friendFirstName = 'Magnus'
ORDER BY Q.distanceFromPerson ASC, Q.friendLastName ASC, Q.friendId ASC;
```

Recursive queries made simpler

SHORTEST_PATH can be easier than writing T-SQL recursive CTEs:

```
-- find path to top-level post
WITH     hierarchy
AS       (SELECT STRING_AGG(mParent.m_messageid, '->') WITHIN GROUP ( GRAPH PATH) AS messageParents,
                COUNT(mParent.m_messageid) WITHIN GROUP ( GRAPH PATH) AS numLevels
          FROM   [message] AS mChild, replyOf FOR PATH, [message] FOR PATH AS mParent
          WHERE  MATCH(SHORTEST_PATH(mChild(-(replyOf)->mParent)+))
                 AND mChild.m_messageid = 7146845053945)
SELECT   TOP 1 CONCAT(7146845053945, '->', messageParents)
FROM     hierarchy
ORDER BY numLevels DESC;

-- for a given post, recursively find all descendent messages
WITH     hierarchy
AS       (SELECT mchild.m_messageId AS messageId,
                LAST_VALUE(mParent.m_messageid) WITHIN GROUP ( GRAPH PATH) AS lastMessageId,
                STRING_AGG(mParent.m_messageid, '->') WITHIN GROUP ( GRAPH PATH) AS messageParents,
                COUNT(mParent.m_messageid) WITHIN GROUP ( GRAPH PATH) AS numLevels
          FROM   [message] AS mChild, replyOf FOR PATH, [message] FOR PATH AS mParent
          WHERE  MATCH(SHORTEST_PATH(mChild(-(replyOf)->mParent)+)))
SELECT   *
FROM     hierarchy
WHERE    lastMessageId = 7146845053933
ORDER BY numLevels DESC;
```

Derived tables and views

- A derived (graph) table which includes the graph pseudo-columns, can used along with MATCH. This is typically used for filtering out nodes / edges.
- Views on top of graph tables can include the (MS)SQL Graph specific pseudo-columns. Such views, can be used with MATCH. For example:

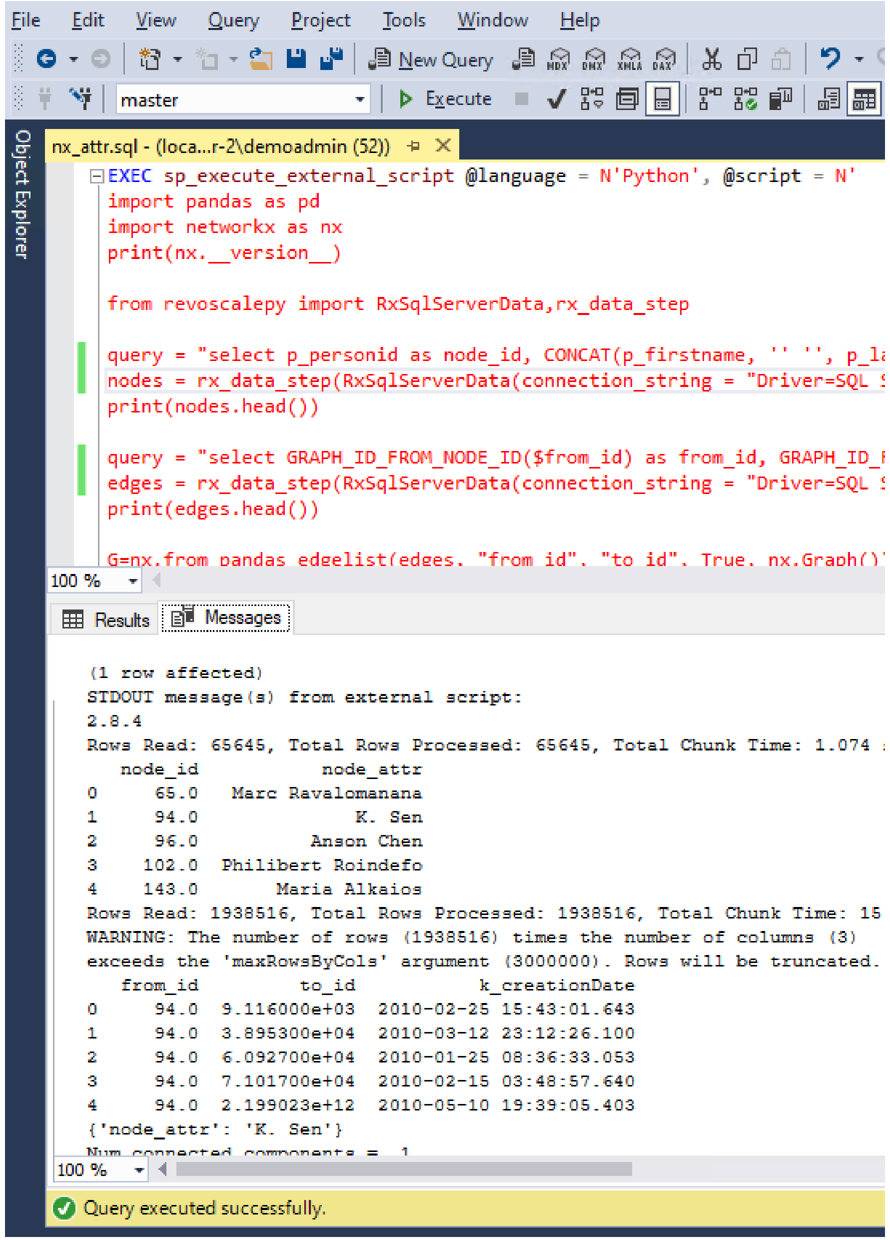
```
CREATE OR ALTER VIEW dbo.[message] AS
SELECT $node_id AS message_node_id, m_messageid, m_ps_imagefile, m_creationdate, m_locationip,
m_browserused, m_ps_language, m_content, m_length, m_creatorid, m_ps_forumid, m_locationid
FROM [dbo].[post]
UNION ALL
SELECT $node_id, m_messageid, NULL AS m_ps_imagefile, m_creationdate, m_locationip, m_browserused,
NULL AS m_ps_language, m_content, m_length, m_creatorid, NULL AS m_ps_forumid, m_locationid
FROM [dbo].[comment];
```

- The view can then be referenced in a MATCH predicate as shown below:

```
SELECT TOP 5 p2.p_personid, p2.p_firstname, p2.p_lastname, m.m_messageid
FROM   person AS p1, knows, person AS p2, likes, [message] AS m
WHERE  MATCH(p1-(knows)->p2
             AND p1-(likes)->m)
       AND m.m_creatorid = p2.p_personId
       AND p1.p_personId = 51934;
```

Extensibility via. sp_execute_external_script [1]

```
EXEC sp_execute_external_script @language = N'Python', @script = N'
import pandas as pd
import networkx as nx
from revoscalepy import RxSqlServerData, rx_data_step

query = "select p_personid as node_id, CONCAT(p_firstname, '' '', p_lastname) as
node_attr from person"
nodes = rx_data_step(RxSqlServerData(connection_string="Driver=SQL
Server;Server=.;Database=ldbc-snb-sf10;Trusted_Connection=Yes;",
        sql_query=query,))

query = "select GRAPH_ID_FROM_NODE_ID($from_id) as from_id,
GRAPH_ID_FROM_NODE_ID($to_id) as to_id, k_creationDate from knows"
edges = rx_data_step(RxSqlServerData(connection_string="Driver=SQL
Server;Server=.;Database=ldbc-snb-sf10;Trusted_Connection=Yes;",
        sql_query=query,))

G = nx.from_pandas_edgelist(edges, "from_id", "to_id", True, nx.Graph())
nx.set_node_attributes(G, nodes.set_index("node_id").to_dict("index"))

# centrality
centrality = nx.eigenvector_centrality(G)
print(sorted((v, f"{c:0.2f}") for v, c in centrality.items()))

# connected components
data = {"col1": [nx.number_connected_components(G)]}
OutputDataSet = pd.DataFrame(data, columns=["col1"])
'
```



[1] Only available for SQL Server / Azure SQL Managed Instance

Keen to know more?

- Documentation
  - https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview
  - https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture
- Samples
  - SQL Graph – samples
  - https://github.com/microsoft/sql-server-samples/tree/master/samples/features/sql-graph/ShortestPath
  - https://github.com/microsoft/sql-server-samples/tree/master/samples/features/sql-graph/DerivedTablesAndViewsInGraphMatch
  - https://github.com/Microsoft/sql-server-samples/tree/master/samples/demos/sql-graph/recommendation-system
  - https://github.com/shkale-msft/GraphRecursiveQueries
  - Million Song Dataset: 1 million nodes, ~ 48 million edges
  - Yelp Dataset: ~ 2 million users (nodes), ~ 19 million edges
  - Open Academic Graph: 2.6 billion nodes, 8.8 billion edges
  - [Work in progress] LDBC SNB Interactive reference implementation with MSSQL - https://github.com/ldbc/ldbc_snb_interactive_impls/pull/264/
- Blogs / case studies
  - https://customers.microsoft.com/en-us/story/825080-bkw-energie-energy-azure
  - https://devblogs.microsoft.com/azure-sql/solving-the-river-crossing-problem-with-sql-graph/
  - https://blogs.msdn.microsoft.com/sqlcat/2017/04/21/build-a-recommendation-system-with-the-support-for-graph-data-in-sql-server-2017-and-azure-sql-db/
  - https://blogs.msdn.microsoft.com/sqlserverstorageengine/2018/11/07/public-preview-of-derived-tables-and-views-on-graph-tables-in-match-queries/
  - https://blogs.msdn.microsoft.com/sqlserverstorageengine/2018/09/28/public-preview-of-graph-edge-constraints-on-sql-server-2019/
  - https://deep.data.blog/2017/11/03/how-we-did-it-pass-2017-summit-session-similarity-using-sql-graph-and-python/
  - https://blogs.technet.microsoft.com/dataplatforminsider/2017/04/20/graph-data-processing-with-sql-server-2017/
  - https://techcommunity.microsoft.com/t5/SQL-Server/Public-Preview-of-Shortest-Path-on-SQL-Server-2019/ba-p/721240
  - http://www.hansolav.net/sql/graphs.html
- Videos
  - Graph Data Models and Query Patterns using #AzureSQL: https://www.youtube.com/watch?v=eYv1z0vfsIQ
  - Generate intelligent insights from your data using Graph features in Azure SQL: https://www.youtube.com/watch?v=w_vzYHcf5L0
  - Exploding Bill of Materials using Graph Shortest Path: https://www.youtube.com/watch?v=9F3Ls0IjPOA
  - A Game of Hierarchies: Graph Processing with SQL Server 2019 - Markus Ehrenmueller-Jensen: https://www.youtube.com/watch?v=EC-4pz2O2Wo
  - SQL Server 2017: Building applications using graph data: https://www.youtube.com/watch?v=s986hslpFtQ

Thank you!

https://www.linkedin.com/in/arvindsh/

https://twitter.com/arvisam