|  |  |
| --- | --- |
| **Title** | A first look at graphs in SQL Server |
| **Author(s)** | Jeff Shurak |
| **Requested**  **Publication Date** | Click or tap to enter a date. |
| **Category (Choose One)** | Customer Engagement  Analytics & Automation  Cloud & Hybrid IT  Customer Success Management |
| **Keywords (Tags)** |  |

**Picture:**

If you have a picture to go with your blog post, paste it here, or add a link to it.

Please ensure there are no copywrite restrictions with the picture

or

Preferred: Anexinet has a subscription to adobestock.com. You can add the link here and we can

**Blog Post:**

# Graph processing

Graph processing is different method of modeling many to many relationships. It was first added in SQL Server 2017 and is different method of storing and querying many to many relationships. In a traditional database, many to many relationships typically make use of a third table that stores a mapping between the two main tables. Graph tables use similar concepts, nodes and edges. Both tables, nodes contain the data and edges contain the relationship between nodes. In this post, I will explain each concept.

## Nodes and Edges

Node tables are just like regular tables, except they contain an automatically generated $node\_id column storing JSON data to identify each record. It is recommended to create a unique constraint or index on this column but it is not required.

Edge tables are slightly different. These tables express the relationship and consist of at least three required columns that store JSON strings. The $edge\_id column uniquely identifies the edge in the database. The $from\_id and $to\_id columns store the $node\_id records from Node tables as their mapping to other node records. In this manner, it’s very similar to traditional many to many mapping. Edges can optionally contain user generated columns, but they are not required.

Creating nodes and edges is also just like creating regular tables, but simply add ‘AS NODE’ or ‘AS EDGE’

Populating nodes is identical to inserting into normal tables. Edges require slightly different inserts as the $node\_id from the source and target nodes will have to be specified during the insert to create the relationships. Below are some sample scripts. For brevity, I am only including examples of each type. If you would like full scripts, they can be downloaded from github: <https://github.com/jshurak/SQL_Graphs>

create table Movies (

MovieID int identity(1,1),

Title Varchar(60)

) as node

Create table Actor (

ActorID int identity(1,1),

FirstName varchar(60),

LastName varchar(60)

) as Node

Create table StarredIn as edge

insert into actor values ('robert','downy')

insert into actor values ('scarlett','johansson')

…..

insert into movies values ('The Avengers')

insert into movies values ('Iron Man')

…

…

insert into StarredIn values ((select $node\_id from actor where lastname = 'downy'),(select $node\_id from Movies where title = 'The Avengers'))

insert into StarredIn values ((select $node\_id from actor where lastname = 'johansson'),(select $node\_id from Movies where title = 'The Avengers'))

## Querying graph databases

There are syntactical differences in querying graph databases. They use the implicit join style structure when specifying tables, with a special MATCH statement in the where clause. Within the MATCH statement is the graph search pattern. This is a uniquely structured pattern that utilizes a hyphen and arrow structure to specify the relationship.

The basic structure is as follows

SELECT *col*

FROM *node, edge*

WHERE MATCH(node-edge->node)

The arrow can be in either direction and multiple graph search patterns can be included with an AND statement. Currently OR and NOT statements are not allowed within the MATCH statement. In some instances, this can lead to simplified queries, in others it can be more complex. Below are some samples of queries with their traditional equivalents. Granted there is more than one way to write a query.