

Arvind Shyamsundar

Principal Program Manager, Microsoft
DataCAT | AzureCAT (part of Cloud + AI)



Large-scale graph querying and analytics with SQL Graph

Please Thank our Sponsors:

 Profisee

 ConduSIV
Formerly Diskeeper

 Quest

 DB-BEST
TECHNOLOGIES

 Microsoft Azure

 ATTUNIX))
a Redapt Company

 COZYROC

 redgate

 Dallas DBAs

 SQLCoop



Credits

Open Academic Society and Microsoft Academic Graph

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In Proceedings of the 24th International Conference on World Wide Web (WWW '15 Companion). ACM, New York, NY, USA, 243-246.

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'2008). pp.990-998.

SQL Graph team (Microsoft)

Shreya Verma, Craig Freedman, Devin Rider, Gjorgji Gjeorgjievski, Daniel Schall, Rajneesh Hegde, Joe Sack, Chandrashekhar Kadiam, Nitish Jindal





<http://aka.ms/arvindsh>
<https://deep.data.blog>

Arvind Shyamsundar

Principal PM, Microsoft AzureCAT



SQL Server

Have worked with SQL Server for almost 20 years. Focused on high-end performance and T-SQL tuning. Part of SQLCAT since 2015; previously with Microsoft Services.

Debug Automation

Having a strong developer background and even stronger troubleshooting and debugging skills; creator of several tools (internal and external) to ease / automate troubleshooting.

Azure Data architecture

Recently focusing on end-to-end data architectures in the cloud. Azure Databricks, Azure Data Lake Gen 2 and related products from the OSS world.



AzureCAT / DataCAT / SQLCAT

Data Customer Advisory Team
Microsoft Azure Engineering

 <http://aka.ms/sqlcat>

 @MSSQLCAT

Our focus

We help customers be successful on Azure

We deliver customer feedback to engineering to improve all data technologies on Azure

We share our knowledge with the community

Our technologies

**SQL Server
(vNext)**

Big Data on Azure (including
Azure Databricks)

Azure SQL DB

Azure DB for MySQL

Azure SQL DW

Azure DB for PostgreSQL

Cosmos DB

Other data technologies on
Azure

A sample of recent engagements

Validation of a complete application workload at over
1,000,000 batch requests/second on a **2-socket** server
using **SQL Server 2019 CTP**

First **production** deployment of **Managed Instance** in
May 2018

Design validation for an analytical workload that will use
10,000 SQL DW databases across **25 Azure regions**

Agenda

Background

Scenario

Dataset

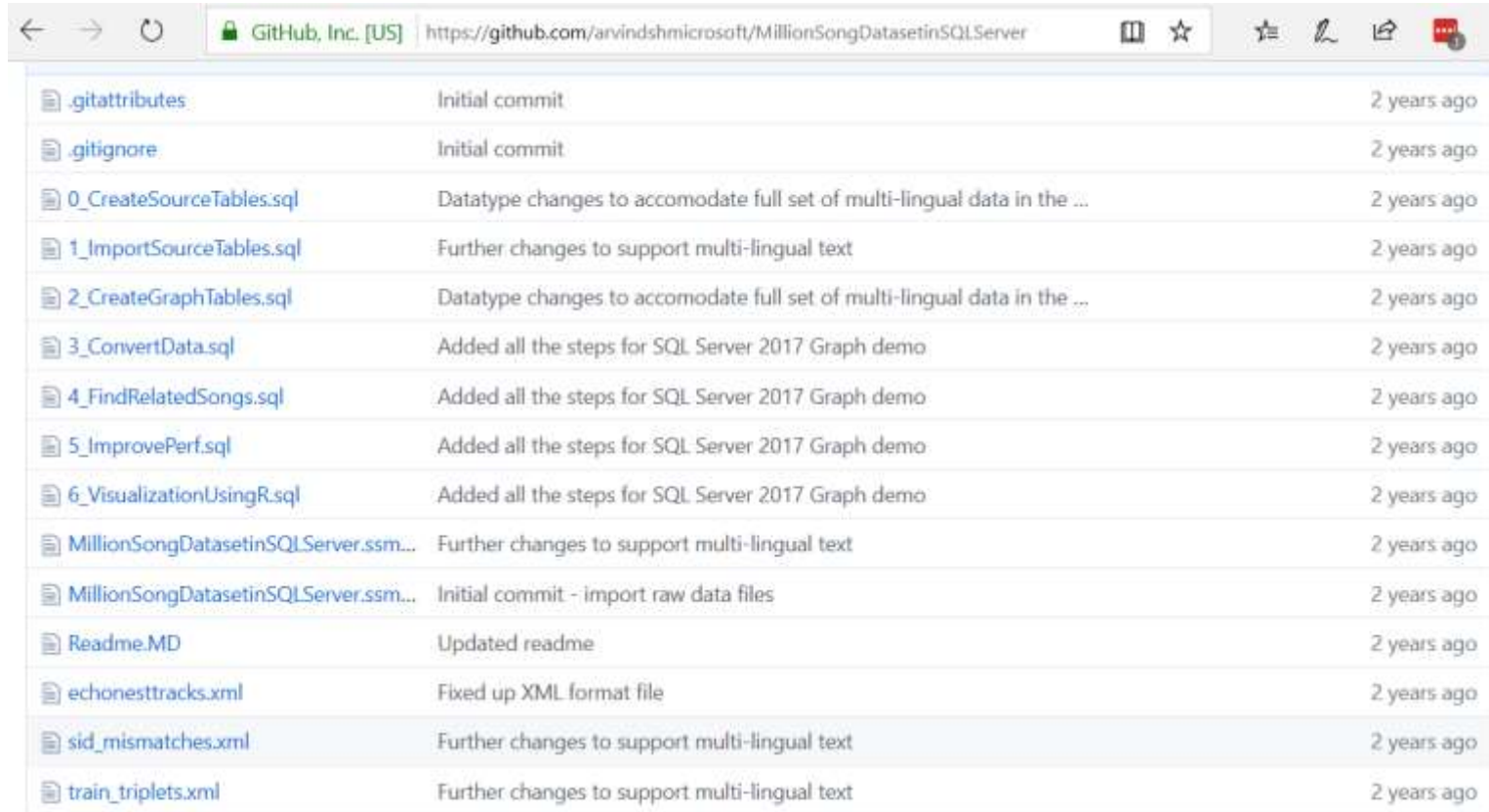
SQL Graph capabilities

Data Loading

Querying the graph



Past Work



.gitattributes	Initial commit	2 years ago
.gitignore	Initial commit	2 years ago
0_CreateSourceTables.sql	Datatype changes to accomodate full set of multi-lingual data in the ...	2 years ago
1_ImportSourceTables.sql	Further changes to support multi-lingual text	2 years ago
2_CreateGraphTables.sql	Datatype changes to accomodate full set of multi-lingual data in the ...	2 years ago
3_ConvertData.sql	Added all the steps for SQL Server 2017 Graph demo	2 years ago
4_FindRelatedSongs.sql	Added all the steps for SQL Server 2017 Graph demo	2 years ago
5_ImprovePerf.sql	Added all the steps for SQL Server 2017 Graph demo	2 years ago
6_VisualizationUsingR.sql	Added all the steps for SQL Server 2017 Graph demo	2 years ago
MillionSongDatasetinSQLServer.ssm...	Further changes to support multi-lingual text	2 years ago
MillionSongDatasetinSQLServer.ssm...	Initial commit - import raw data files	2 years ago
Readme.MD	Updated readme	2 years ago
echonesttracks.xml	Fixed up XML format file	2 years ago
sid_mismatches.xml	Further changes to support multi-lingual text	2 years ago
train_triplets.xml	Further changes to support multi-lingual text	2 years ago

Readme.MD

Million Song Dataset in SQL Server 2017

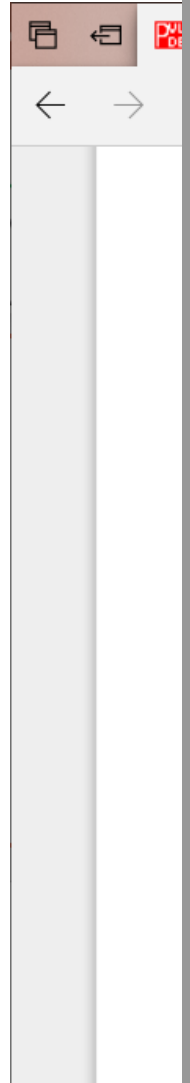
<https://github.com/arvindshmicrosoft/MillionSongDatasetinSQLServer/commit/607a06321cb417> to build a recommendation service for songs.





Scenario

Anatomy of a publication



H.2.4 [Database Management]: Systems – Query Processing.

General Terms

Performance, Design, Experimentation

Keywords

Query optimizer, Cardinality Estimation.

1. INTRODUCTION

Query Optimization remains as relevant a problem as ever before, if not more. Modern day database workloads include On-line Transaction Processing Systems, Enterprise Resource Planning, Customer Relationship Management, On-line Analytical Processing and Data Analysis over Data-warehouses. The queries generated by these workloads are increasingly complex and the databases are larger than ever. Thus, the central role of query optimization that searches the space of different execution strategies and picks a good execution plan remains unquestionable.

It has been thirty years since the publication of the System R paper on Query Optimization [1] that acted as the defining framework for query optimization. As we will review in the next section, significant progress has been made on several aspects of Query Optimization since the early days of relational databases. At the same time, certain fundamental difficulties remain. For example, cardinality estimation remains a difficult problem despite years of research activity, search algorithms in optimizers have significant ad-hoc elements to manage optimization time, and cost estimation is not able to take into account the current state of the server effectively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD '09, June 29–July 2, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-551-2/09/06...\$5.00.

estimation, cost estimation and search) is capable of leveraging application input and past usage information. Such an approach also impacts how we build these components of the optimizer and we will discuss these consequences. The goal of this paper is to motivate the idea of opening up the optimizer rather than to present specifics of the interfaces we require of the optimizer.

The first part of the paper is retrospective in nature. We begin by sketching the relevant history of query optimizers and we summarize the key technical challenges (Section 2). We then provide a broad overview of our proposal for enabling the optimizer to leverage additional input from the application or usage information (Section 3). The next two sections provide further elaboration of this approach for cardinality estimation and search components of the optimizer (Sections 4 and 5). We conclude with an outline of the a few steps that can help push the frontiers of query optimization (Section 6).

2. The State of the Art in Query Optimization

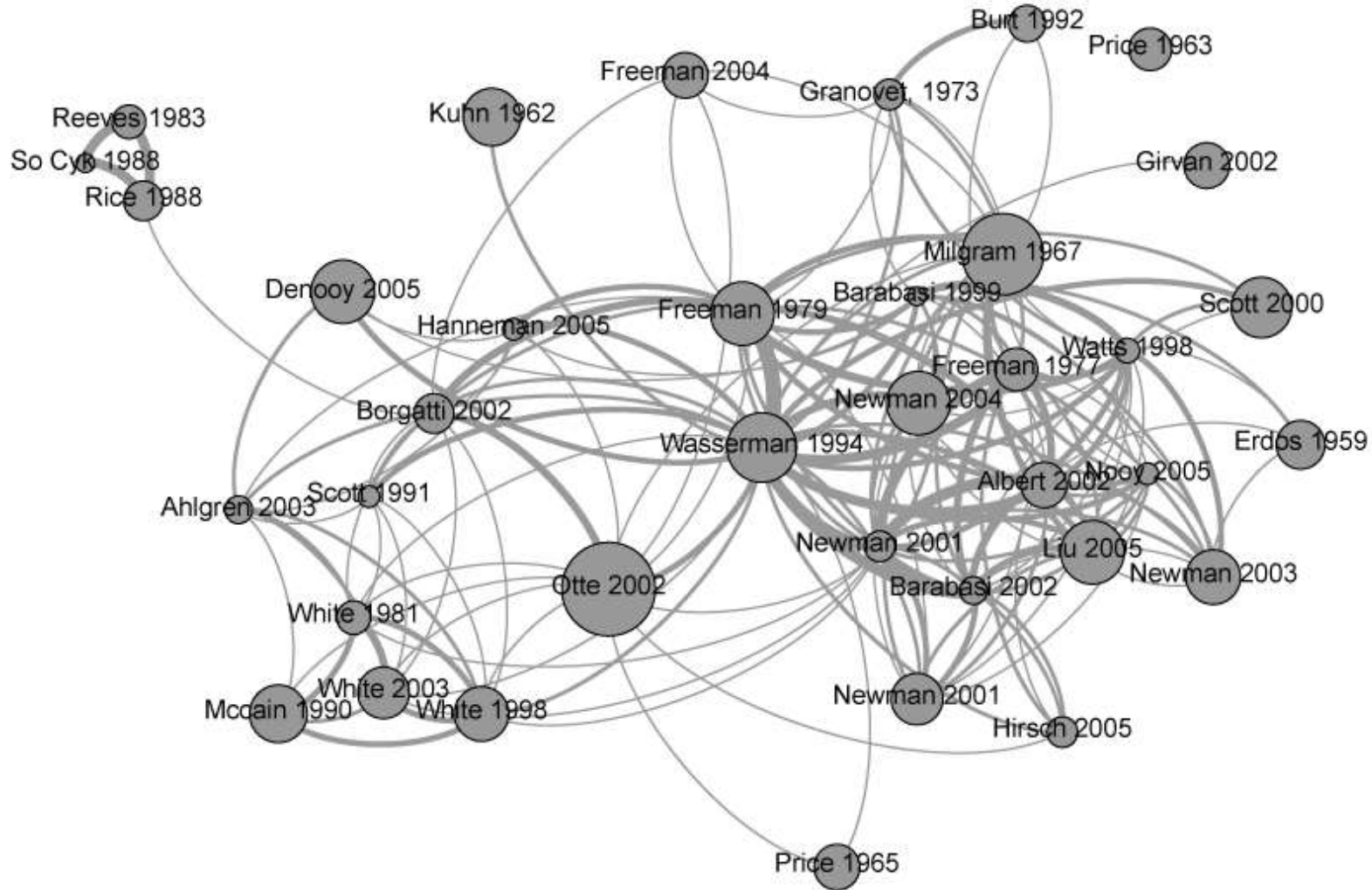
2.1 A Brief History

The seminal System-R paper [1] provided a framework for query optimization that consisted of three pillars: (a) cardinality estimation for SQL expressions, (b) cost estimation for SQL execution plans (or partial plans) and (c) a dynamic programming based algorithm to search the space of execution plans. However, the paper also recognized that an ordering of tuples, even if locally suboptimal, may pay off globally. This was referred to as *interesting orders*. In many ways, this paper solved the query optimization problem quite well for the simple execution engines and relatively simple queries of that era.

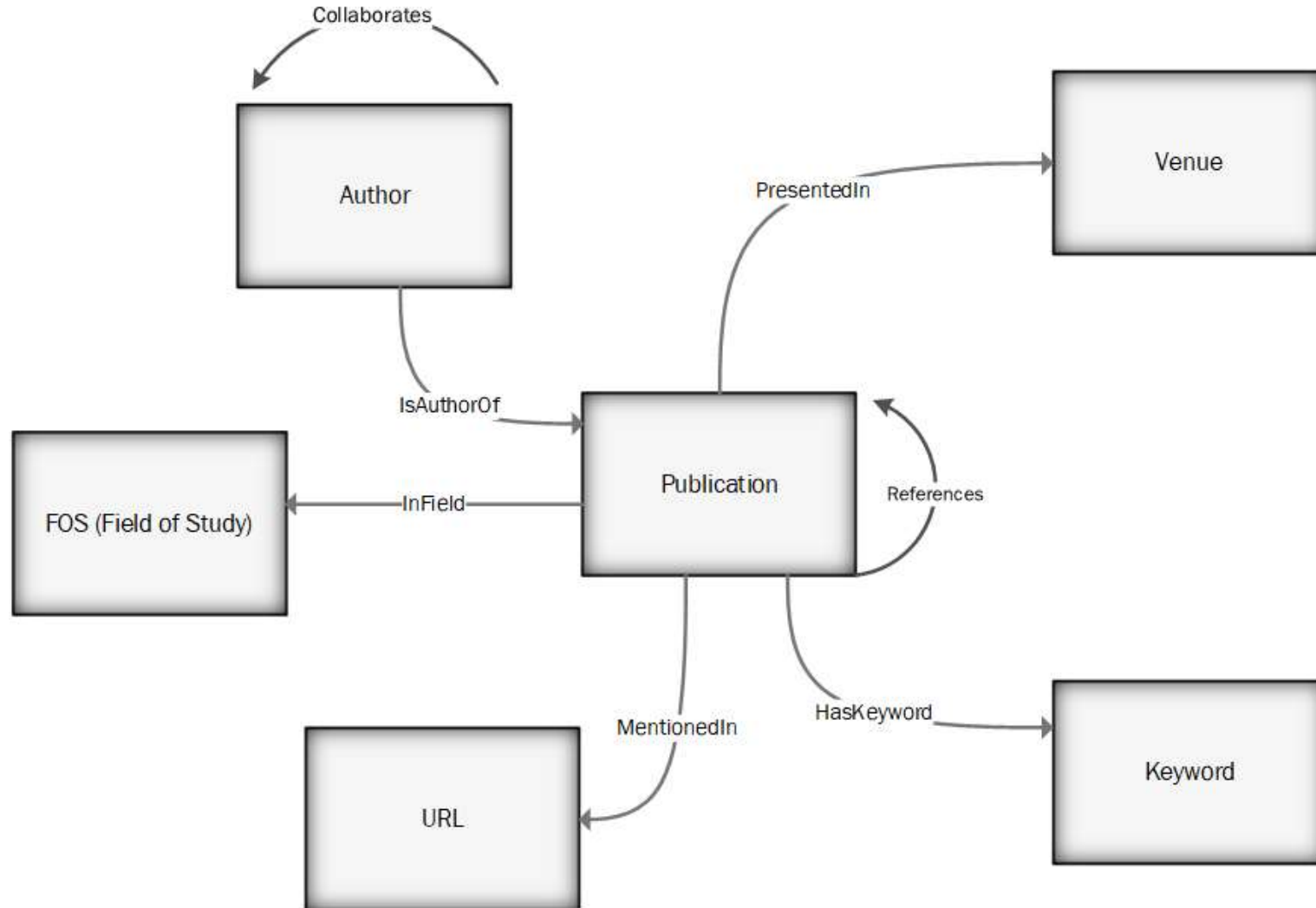
Over the next fifteen years since [1], the query execution engines became far more sophisticated with the addition of new logical and physical operators. Parallel database technology allowed relational systems to handle complex queries over very large data sets. The SQL engines started being used widely during this era for data warehousing and decision support systems. As a result, database systems started experiencing the need to handle more complex queries. Research on query optimization led to explosion of work on query rewrite rules [21]. Some of the rewrite rules, e.g., de-correlation and Magic sets, commuting Group By and join



Publications as a graph



Entities and Relationships



Use Case: Academic Search

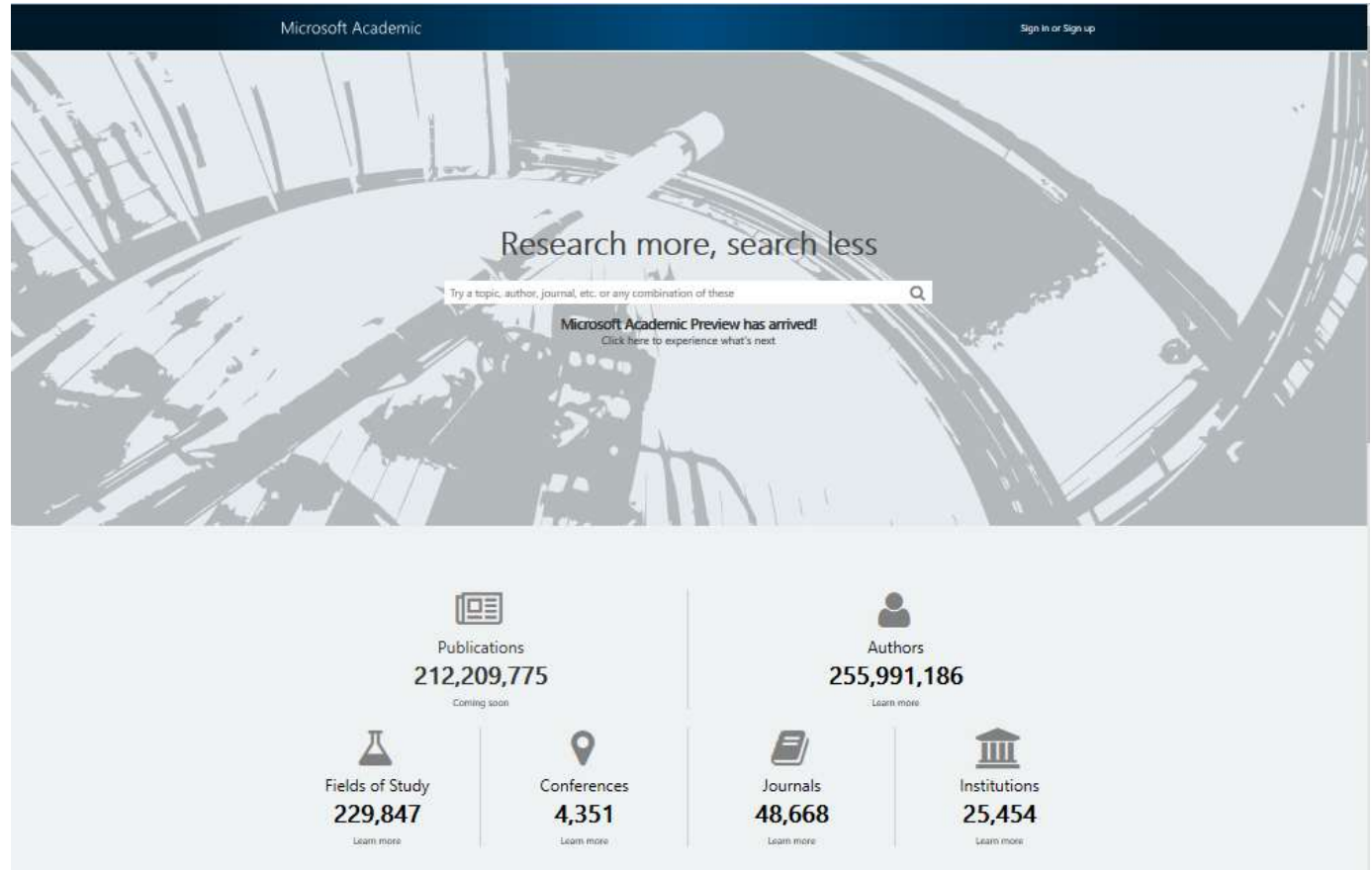
How do we search for publications by keyword?

Are two publications “linked” by a chain of citations?

Have two authors co-authored a paper?

Are authors “linked” by a chain of “collaborations”?

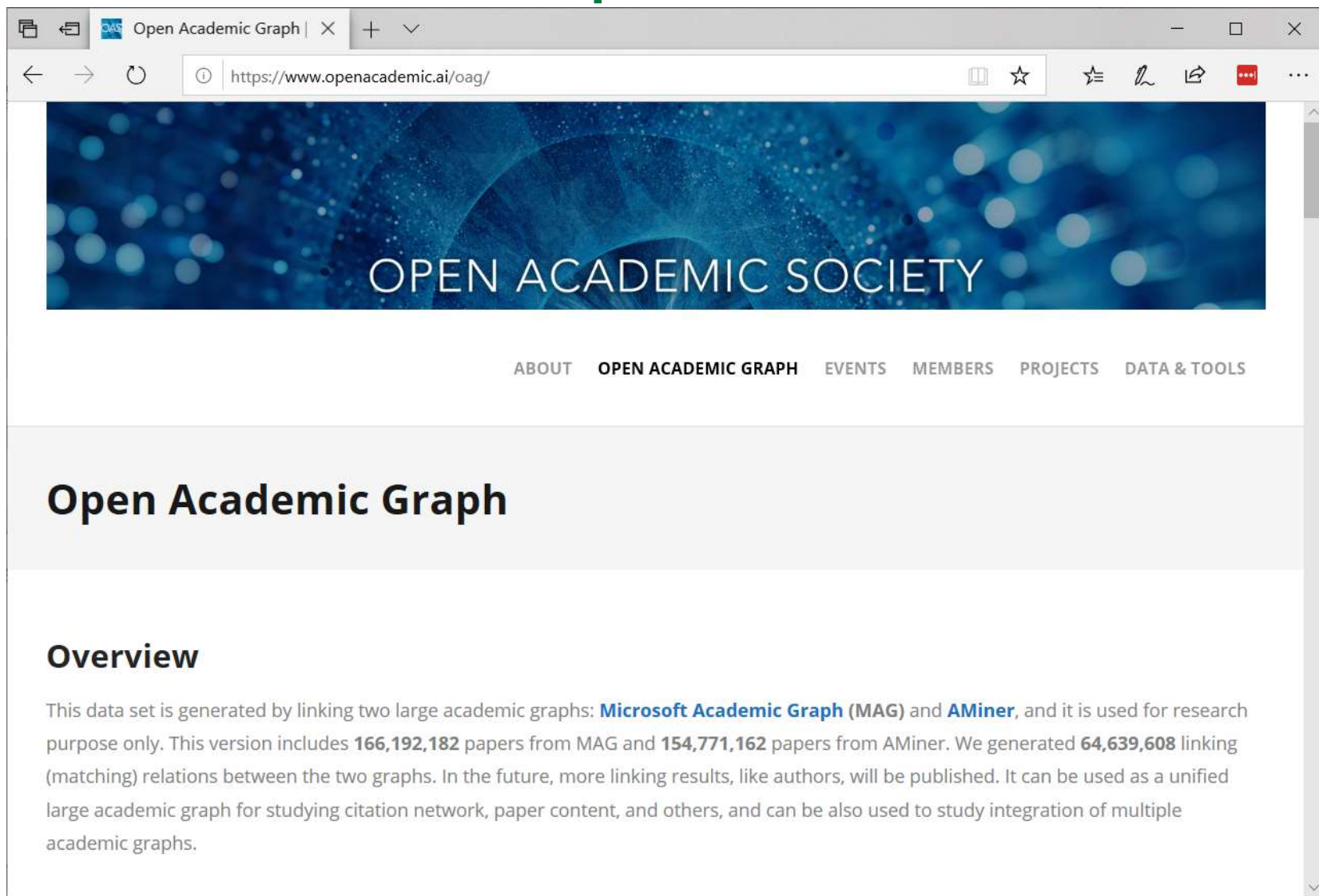
Can we rank papers by their “references” from other papers?



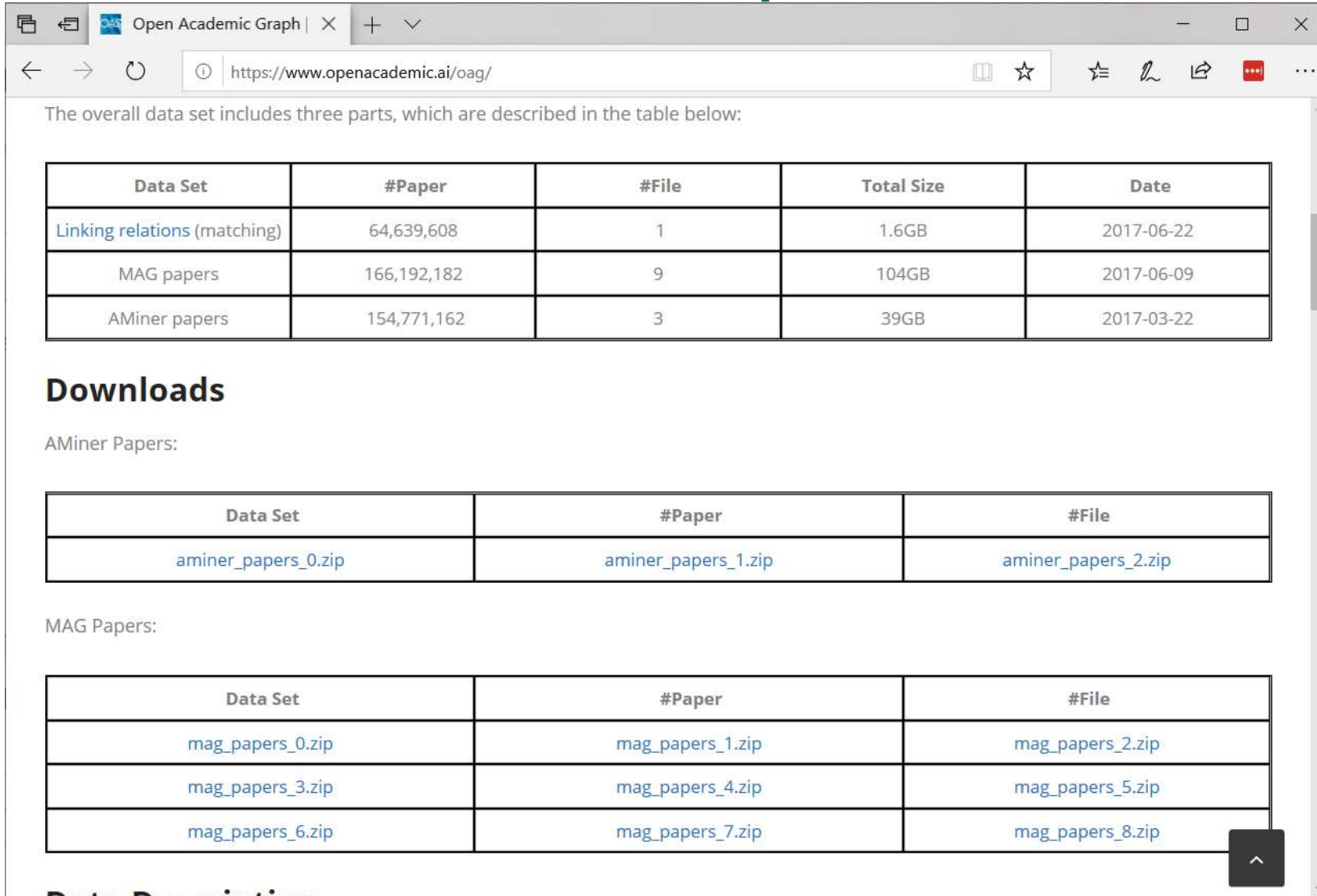
A large, teal-colored abstract graphic on the left side of the slide. It consists of several thick, curved lines that overlap and flow from the top left towards the bottom left, creating a sense of movement and depth.

What data
is available?

Open Academic Graph



Microsoft Academic Graph (MAG)



The overall data set includes three parts, which are described in the table below:

Data Set	#Paper	#File	Total Size	Date
Linking relations (matching)	64,639,608	1	1.6GB	2017-06-22
MAG papers	166,192,182	9	104GB	2017-06-09
AMiner papers	154,771,162	3	39GB	2017-03-22

Downloads

AMiner Papers:

Data Set	#Paper	#File
aminer_papers_0.zip	aminer_papers_1.zip	aminer_papers_2.zip

MAG Papers:

Data Set	#Paper	#File
mag_papers_0.zip	mag_papers_1.zip	mag_papers_2.zip
mag_papers_3.zip	mag_papers_4.zip	mag_papers_5.zip
mag_papers_6.zip	mag_papers_7.zip	mag_papers_8.zip

Data Description



Sample JSON document

```
{
  "id": "53e9ab9eb7602d970354a97e",
  "title": "Data mining: concepts and techniques",
  "authors": [
    {
      "name": "jiawei han",
      "org": "department of computer science university of illinois at urbana champaign"
    },
    {
      "name": "micheline kamer",
      "org": "department of computer science university of illinois at urbana champaign"
    },
    {
      "name": "jian pei",
      "org": "department of computer science university of illinois at urbana champaign"
    }
  ],
  "year": 2000,
  "keywords": [
    "data mining",
    "structured data",
    "world wide web",
    "social network",
    "relational data"
  ],
  "fos": [
    "relational database",
    "data model",
    "social network"
  ],
```

```
    "n_citation": 29790,
    "references": [
      "53e99ef4b7602d97027c2346",
      "53e9aa23b7602d970338fb5e",
      "53e99cf5b7602d97025aac75"
    ],
    "doc_type": "book",
    "lang": "en",
    "publisher": "Elsevier",
    "isbn": "1-55860-489-8",
    "doi": "10.4114/ia.v10i29.873",
    "pdf": "//static.aminer.org/upload/pdf/1254/370/239/53e9ab9eb7602d970354a97e.pdf",
    "url": [
      "http://dx.doi.org/10.4114/ia.v10i29.873",
      "http://polar.lsi.uned.es/revista/index.php/ia/article/view/479"
    ],
    "abstract": "Our ability to generate and collect data has been increasing rapidly. Not only are all of our business, scientific, and government transactions now computerized, but the widespread use of digital cameras, publication tools, and bar codes also generate data. On the collection side, scanned text and image platforms, satellite remote sensing systems, and the World Wide Web have flooded us with a tremendous amount of data. This explosive growth has generated an even more urgent need for new techniques and automated tools that can help us transform this data into useful information and knowledge. Like the first edition, voted the most popular data mining book by KD Nuggets readers, this book explores concepts and techniques for the discovery of patterns hidden in large data sets, focusing on issues relating to their feasibility, usefulness, effectiveness, and scalability. However, since the publication of the first edition, great progress has been made in the development of new data mining methods, systems, and applications. This new edition substantially enhances the first edition, and new chapters have been added to address recent developments on mining complex types of data? including stream data, sequence data, graph structured data, social network data, and multi-relational data."
  ]
}
```



A large, teal-colored abstract graphic on the left side of the slide. It consists of several thick, curved lines that overlap and flow from the top left towards the bottom left, creating a sense of movement and depth.

Graphs in SQL Server

SQL Graph

A graph is a collection of **node** and **edge** tables
Language Extensions

DDL Extensions – create node/edge tables

Query Language Extensions – New built-in: MATCH, to support pattern matching and traversals

Tooling and Eco-system

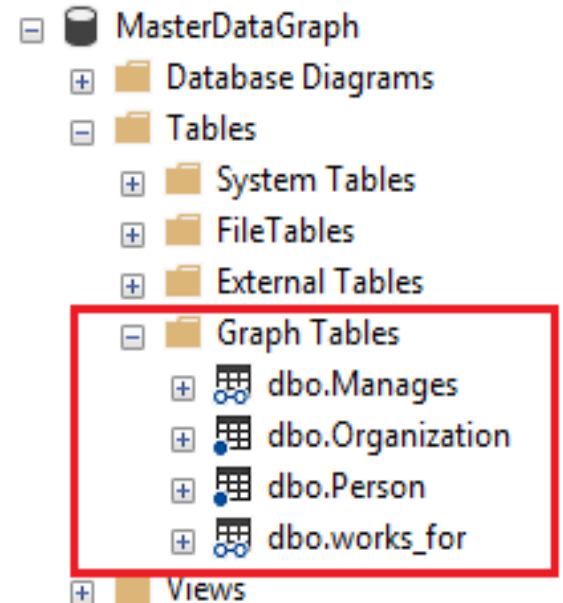
Existing tools all work out of the box, for example SSMS, backup and restore, import / export, etc.

Queries can join existing tables and graph node / edge tables.

Columnstore Indexes, ML (R / Python), HA etc.

Security and Compliance: Dynamic Data Masking, Row Level Security

Available on SQL Server on Linux as well!



DDL Extensions: CREATE NODE

```
CREATE TABLE Customers (  
    [CustomerID] INTEGER NOT NULL,  
    [CustomerName] NVARCHAR(100) NOT NULL,  
    [WebsiteURL] NVARCHAR(256) NOT NULL  
) AS NODE  
GO
```

```
SELECT TOP 5 * FROM Customers;
```

	\$node_id_DAA3CE76E9514CC49EF266C3658F98EA	CustomerID	CustomerName	WebsiteURL
1	{"type":"node","schema":"dbo","table":"Customers","id":0}	1	Tailspin Toys (Head Office)	http://www.tailspintoys.com
2	{"type":"node","schema":"dbo","table":"Customers","id":1}	2	Tailspin Toys (Sylvanite, MT)	http://www.tailspintoys.com/Sylvanite
3	{"type":"node","schema":"dbo","table":"Customers","id":2}	3	Tailspin Toys (Peeples Valley, AZ)	http://www.tailspintoys.com/PeeplesValley
4	{"type":"node","schema":"dbo","table":"Customers","id":3}	4	Tailspin Toys (Medicine Lodge, KS)	http://www.tailspintoys.com/MedicineLodge
5	{"type":"node","schema":"dbo","table":"Customers","id":4}	5	Tailspin Toys (Gasport, NY)	http://www.tailspintoys.com/Gasport



DDL Extensions: CREATE EDGE

```
CREATE TABLE Bought (  
    [PurchasedCount] BIGINT  
) AS EDGE;  
GO
```

```
CREATE TABLE FriendOf AS EDGE  
GO
```

```
SELECT TOP 5 * FROM Bought;
```

	\$edge_id_B9856324C3164E2694E7F1C777...	\$from_id_633265868D26434BBD924BAB4E3FFF50	\$to_id_EA31891D3BAD49F8A71945C8E87C91F6	PurchasedCount
1	{"type":"edge","schema":"dbo","table":"Bough...	{"type":"node","schema":"dbo","table":"Customers","id":36}	{"type":"node","schema":"dbo","table":"StockItems","id":109}	2
2	{"type":"edge","schema":"dbo","table":"Bough...	{"type":"node","schema":"dbo","table":"Customers","id":18...	{"type":"node","schema":"dbo","table":"StockItems","id":109}	1
3	{"type":"edge","schema":"dbo","table":"Bough...	{"type":"node","schema":"dbo","table":"Customers","id":19...	{"type":"node","schema":"dbo","table":"StockItems","id":61}	1
4	{"type":"edge","schema":"dbo","table":"Bough...	{"type":"node","schema":"dbo","table":"Customers","id":38}	{"type":"node","schema":"dbo","table":"StockItems","id":61}	2
5	{"type":"edge","schema":"dbo","table":"Bough...	{"type":"node","schema":"dbo","table":"Customers","id":21}	{"type":"node","schema":"dbo","table":"StockItems","id":61}	1



Inserting data into graph tables

Insert into a NODE table

```
INSERT INTO Customers(CustomerName, ...)  
SELECT CustomerName, ...  
FROM WideWorldImporters.Sales.Customers  
GO
```

Insert into an EDGE table

```
INSERT INTO Bought($from_id, $to_id, PurchasedCount)  
SELECT Customers.$node_id, StockItems.$node_id , @purchasecount  
FROM Customers, StockItems  
WHERE Customers.CustomerID = @customer_id  
AND StockItems.StockItemID = @stockitem_id  
GO
```



Query Language Extensions: MATCH

Multi-hop navigation and join-free pattern matching using MATCH predicate;

“ASCII-art” syntax to facilitate graph traversal

```
SELECT
    CustomerName,
    StockItemName
FROM
    StockItems,
    Customers,
    Bought
WHERE MATCH(Customers-(Bought)->StockItems)
    AND StockItemName = 'All-Purpose Bike Stand'
```



Sample: Product Recommendations using "Relational" approach

```
;WITH Current_Usr AS
(
    SELECT
        PersonID = 11065,
        ProductID = 879, -- 'All-Purpose Bike Stand'
        PurchasedCount = 1
    ) ,
-- Identify the other users who have also purchased the item he/she
is looking for
Other_Usr AS
(
    SELECT
        C.PersonID,
        P.ProductID,
        Purchased_by_others = COUNT(*)
    FROM
        Sales.SalesOrderDetail AS SOD
    JOIN
        Sales.SalesOrderHeader AS SOH ON
SOH.SalesOrderID=SOD.SalesOrderID
    JOIN
        Sales.Customer AS C ON SOH.CustomerID=C.PersonID
    JOIN
        Current_Usr AS P ON P.ProductID=SOD.ProductID
    WHERE
        C.PersonID<>P.PersonID
    GROUP BY
        C.PersonID, P.ProductID
    ) ,
-- Find the other items which those other customers have also
purchased
```

```
Other_Items AS
(
    SELECT
        C.PersonID,
        P.ProductID,
        Other_purchased = COUNT(*)
    FROM
        Sales.SalesOrderDetail AS SOD
    JOIN
        Sales.SalesOrderHeader AS SOH ON SOH.SalesOrderID=SOD.SalesOrderID
    JOIN
        Other_Usr AS C ON SOH.CustomerID=C.PersonID
    JOIN
        Production.Product AS P ON P.ProductID = SOD.ProductID
    WHERE
        P.Name<>'All-Purpose Bike Stand'
    GROUP BY
        C.PersonID, P.ProductID
    )
-- Outer query
-- Recommend to the current user to the top items from those other items,
-- ordered by the number of times they were purchased
SELECT
    top 10 P.Name as ProductName,
    COUNT(Other_purchased)
FROM
    Other_Items
JOIN
    Production.Product AS P ON P.ProductID = Other_Items.ProductID
GROUP BY
    P.Name
ORDER BY
    COUNT(Other_purchased) DESC;
GO
```



Sample: Product Recommendations "graph" / MATCH version

```
SELECT
  TOP 10 RecommendedItem.ProductName,
  COUNT(*)
FROM
  RetailGraph.Products AS Item,
  RetailGraph.Customers AS C,
  RetailGraph.Bought AS BoughtOther,
  RetailGraph.Bought AS BoughtThis,
  RetailGraph.Products AS RecommendedItem
WHERE
  Item.ProductName LIKE 'All-Purpose Bike Stand'
  AND MATCH(RecommendedItem<-(BoughtOther)-C-(BoughtThis)->Item)
  AND (Item.ProductName <> RecommendedItem.ProductName)
  and C.PersonID <> 11065
GROUP BY
  RecommendedItem.ProductName
ORDER BY COUNT(*) DESC;
GO
```



A large, teal-colored abstract graphic on the left side of the slide. It consists of several thick, curved lines that sweep from the top left towards the bottom right, creating a sense of motion or a stylized arrow pointing towards the text.

Loading the MAG into SQL Server

Choices and Decisions

Bulk Insert and JSON

- Line-by-line vs. all-at-once

- OPENROWSET cardinality estimates

Source data location

- ZIP vs. uncompressed

- Local SSD vs. Azure Blob

Heaps or Clustered Columnstore Indexes?

- Logging

- Compression

SQL specifics

- Data File layout

- Azure VM: I/O and network considerations





Walkthrough

JSON and Bulk Insert

High-performance JSON bulk import

Why a custom .NET “Data Loader” application?

Threading and control over memory consumption

Custom logic for Shredding / “Exploding” JSON

Powered by these libraries:

Newtonsoft’s [Json.NET](#)

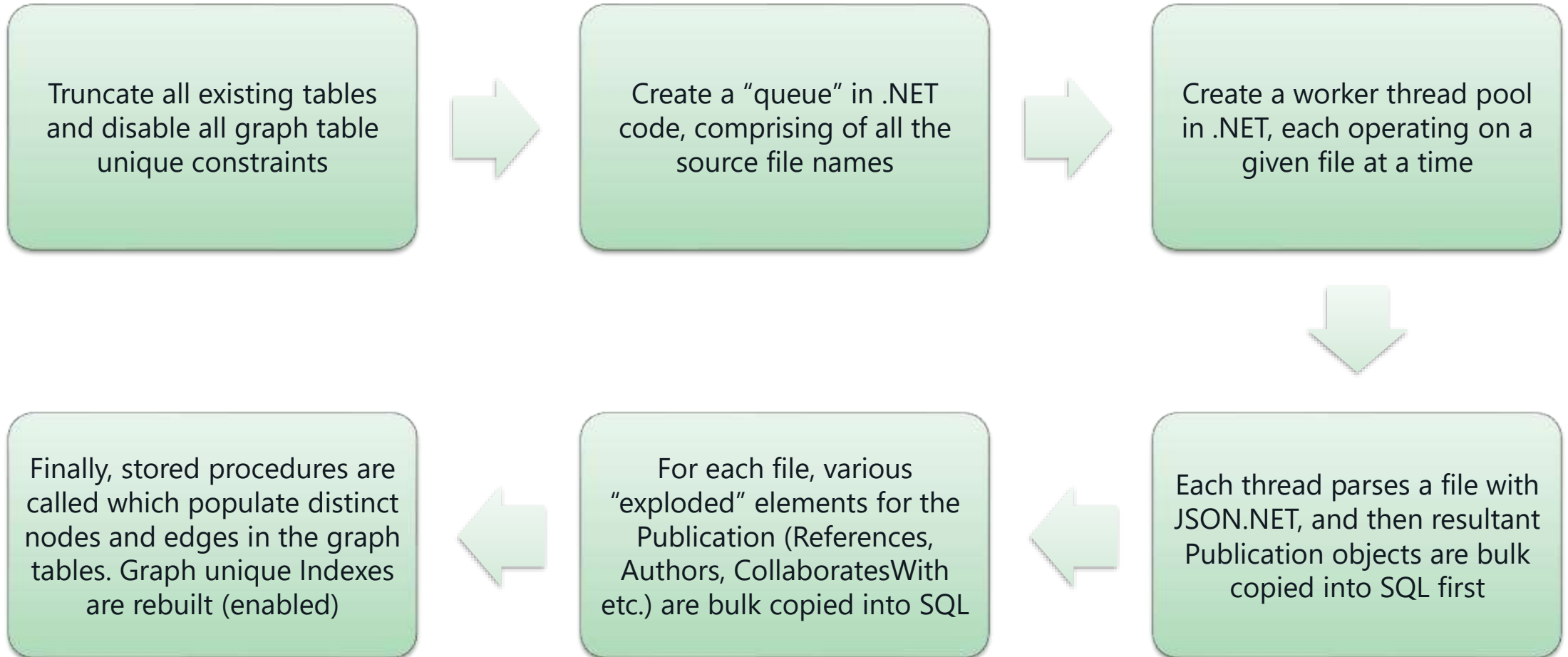
Marc Gravell’s [FastMember](#)

SQLBulkCopy class

[Data.HashFunction](#) (C# implementation of [CityHash](#))



.NET Data Loader overview



.NET perf improvements

gcServer enabled

Server GC is much more scalable on multi-CPU machines

gcAllowVeryLargeObjects enabled

Allows for very large arrays > 2GB in size (x64 only); important when dealing with "Collaborations" data

Optional: Thread UseAllCpuGroups and GCCpuGroup enabled

Only useful if running on servers with > 64 CPUs (i.e. multiple processor groups)

Field note: Watch out for cached (and stale) LINQPad.UserQuery.exe.config files



"Convert" to graph

OAG v1 dataset has non-integer "keys"

Nodes in SQL Graph have their own "IDs"

Therefore a large join is currently necessary to populate the edges

Bulk insert of pre-populated nodes / edges is also possible, but currently suffers from metadata spinlock contention with many concurrent bulk inserts





Walkthrough

Data Loading



Walkthrough

Querying the graph



Recap

In Conclusion

Value proposition of SQL Graph

If your data is already in SQL, use SQL Graph

MATCH makes it easier to express intent of the code, and makes the code easier to read

“Moving” data into a graph

“Converting” to node tables: parallelize as much as possible

“Converting” to edge tables: sizeof(data) operation; use in-built SQL parallelism

Manage indexes carefully: leverage Columnstore if possible and disable / rebuild the inbuilt unique indexes pre/post bulk loads

Query scalability

Analytical algorithms like PageRank: Scale-up: RAM helps more than CPUs

For high-volume, “point” queries: Scale-out using readable secondaries if needed





Q&A

References

- GitHub repo: <https://github.com/arvindshmicrosoft/MicrosoftAcademicGraph>
- Official MAG documentation: <https://docs.microsoft.com/en-us/academic-services/graph/>
- Microsoft Academic blog: <https://www.microsoft.com/en-us/research/project/academic/#!blog>
- OpenAcademic OAG: <https://openacademic.ai/oag/>
- RDF dumps of MAG: <http://ma-graph.org/rdf-dumps/>

