Elastic Query troubleshooting

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

Contents

This article is a copy of the Selfhelp - Common Solutions article which the customer will see when requesting help through the portal.

Resolve Elastic Query issues in Azure SQL Database

Elastic Queries are used to implement cross-database queries in Azure SQL Database. Elastic Query makes data located in an Azure SQL Database available to other Azure SQL Databases. It allows you to run Transact-SQL queries that span two or more databases using a single connection point. Elastic Query is implemented by creating an external table in the local database. The external table connects over an external data source to the remote database for accessing the remote data. No changes are necessary on the remote database.

See the sections below for details and troubleshooting steps that will help resolving your issues.

Example Transact-SQL script and Best practices

Example Transact-SQL script

Here is a simple Transact-SQL script demonstrating the basic steps for creating an external table and executing an Elastic Query:

```
-- execute in remote database
CREATE TABLE [dbo].[Customer] (
   [CustomerID] [int] NOT NULL,
   [CustomerName] [varchar] (50) NULL,
   [Company] [varchar] (50) NULL
   CONSTRAINT [CustID] PRIMARY KEY CLUSTERED ([CustomerID] ASC)
);
-- sample rows
insert into Customer (CustomerID, CustomerName, Company) values (123, 'customer1', 'company1');
insert into Customer (CustomerID, CustomerName, Company) values (321, 'customer2', 'company2');

-- execute in local database
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<master_key_password>';

CREATE DATABASE SCOPED CREDENTIAL ElasticQueryCred
   WITH IDENTITY = '<username>',
   SECRET = '<password>';

CREATE EXTERNAL DATA SOURCE MyElasticQueryDataSrc WITH
   (TYPE = RDBMS,
   LOCATION = '<remote_server_name>.database.windows.net',
   DATABASE_NAME = '<remote database>', -- remote database
   CREDENTIAL = ElasticQueryCred,
);

-- execute in local database
CREATE EXTERNAL TABLE [dbo].[Customer] (
   [CustomerID] [int] NOT NULL,
   [CustomerName] [varchar] (50) NOT NULL,
   [Company] [varchar] (50) NOT NULL)
WITH (DATA_SOURCE = MyElasticQueryDataSrc);

SELECT * FROM [dbo].[Customer] WHERE [CustomerID] = 321;
```

For a step-by-step Transact-SQL example about creating external tables, see the article Get started with cross-database queries (vertical partitioning).

Best practices

Review the following items to avoid issues with connectivity, security, and performance.

- **Required:** Ensure that the Elastic Query local database has access to the remote database. In the remote Azure SQL Database server's firewall configuration, you must enable the option "Allow Azure services and resources to access this server", and you must set the option "Public network access" to "Selected networks" (not to "Disabled"). If either "Allow Azure services" or "Public network access" is disabled, the external table cannot connect to the remote database. See the "Connectivity issues" section below for further details about this requirement.
- **Required:** Ensure that the credential provided in the external data source definition can successfully log into the remote database and has sufficient permissions to access the remote table.
- **Security consideration:** Azure Active Directory (AAD) principals are not supported in Elastic Query, only SQL Authentication will work. Managed Identity is in the scope of AAD principals and is also not supported.
- **Security consideration:** Users with access to the external table automatically gain access to the underlying remote tables under the credential given in the external data source definition. Carefully manage access to the external table in order to avoid undesired elevation of privileges through the credential of the external

data source. Regular SQL permissions can be used to GRANT or REVOKE access to an external table just as it were a regular table.

- **Performance:** Elastic Query works best for queries where most of the computation can be done on the remote databases. You typically get the best query performance with selective filter predicates that can be evaluated on the remote databases, or joins that can be performed completely on the remote database. Other query patterns may need to load large amounts of data from the remote database and may perform poorly.
- **Performance:** Avoid including `NVARCHAR(max)` columns in the external table. Columns of data type `NVARCHAR(max)` will disable the advanced batching techniques that are used in the Elastic Query implementation. If an external table includes `NVARCHAR(max)` columns, it will significantly affect the query performance if a large amount of data needs to be transferred from the remote to the local database.
- **Performance:** The performance of Elastic Queries over large databases will be slow in the lower service tiers, like Standard or the smaller General Purpose tiers. Use at least the Premium tier, a higher-level General Purpose size, or Business Critical to receive reliable performance for large databases.
- **Performance:** Elastic Query is not the preferred solution for ETL (Extract - Transform - Load) operations when there is a large amount of data movement from and especially to a remote database. If your workloads include heavy ETL processing, consider moving to Azure SQL Managed Instance instead of using Elastic Query on Azure SQL Database.

Connection error to remote data source

Symptom

After the external table has been created successfully and you try to query it, you may see one of the following error messages:

> Msg 46832, Level 16, State 3, Line 1
> An error occurred while establishing connection to remote data source: [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Reason: An instance-specific error occurred while establishing a connection to SQL Server.

> A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

This symptom indicates that the local database does not have access to the remote database. The most likely cause is that either firewall option "Allow Azure services" or "Public network access" is disabled at the remote server.

Resolution

The best and preferred solution is to enable the option "Allow Azure services and resources to access this server" and to set the option "Public network access" to "Selected networks" in the remote Azure SQL Database server's firewall configuration.

If "Allow Azure services" is not secure enough for your requirements, you could also try to create IP address firewall rules for each of the local databases. You need to catch the firewall errors and get the public IP address of the database that is attempting to log in. You could then add that IP in a firewall rule to the remote server and retry the query. This approach is relatively complex to implement, as it is a manual process and cannot be automated. It is also possible that the IP address may change in the future. The public IP addresses are available in the Azure IP Ranges and Service Tags – Public Cloud download list for your reference.

If neither of this is a valid option, you may consider moving the Azure SQL Databases to an Azure SQL Managed Instance, which natively allows cross-database queries.

More information

Creating a private endpoint allows a specific virtual network (VNet) to access Azure SQL Database privately without exposing the connection to a public network. The database or server will not become a member of that VNet though and will remain outside the VNet. You may create multiple endpoints for the database server to be accessed over multiple VNets, but the server itself cannot be made a part of these VNets like a physical or virtual machine could be.

If the option "Public network access" is set to disabled, only connections over Private Link endpoints are allowed to access the database server. For a connection to go over Private Link, its TCP connection has to originate inside the VNet that the Private Link has been set up for. The outgoing Elastic Query connection however originates at the local database itself, not in the VNet, because Azure SQL Databases are not members inside the VNet. Therefore, the Elastic Query will fail with error 46832 if the public endpoint has been disabled.

Outbound connections from Azure SQL Database external tables are communication through the Azure Backbone Network, which is outside of any private endpoint address space. Currently, there is no functionality that supports cross-database queries over a private endpoint. Thus Elastic Query is only supported when "Public network access" remains enabled.

Regarding "Allow Azure services", enabling this option allows the local database to pass the firewall of the remote server. If the option is disabled, the Elastic Query connection will be blocked by the remote server firewall. If your security policies demand that "Allow Azure services" remains disabled, you then need to add IP firewall rules to allow the incoming connection from the local database.

Outbound Firewall Rules (Restrict Outbound Networking) enabled

Outbound firewall rules limit network traffic from the Azure SQL Database logical server to a customer-defined list of Azure Storage accounts and Azure SQL Database logical servers. Any attempt to access storage accounts or servers that are not in this list is denied.

If "Outbound Firewall Rules" (portal option: "Restrict Outbound Networking") has been enabled on the remote Azure SQL Database server, the local server cannot retrieve any data from the remote server. Therefore the Elastic Query will fail with the following error:

> Error 46842
> Logical server '<local_server_name>' defined in the elastic query configuration is not in the list of Outbound Firewall Rules on the server. Please add this logical server name to the list of Outbound Firewall Rules on your '<remote_server_name>' server and retry the operation.

To resolve this issue, add the Fully Qualified Domain Name (FQDN) of the local server to the configuration list for "Restrict Outbound Networking" on the remote server. The FQDN name pattern for Azure SQL Database is "servername.database.windows.net". Refer to Set outbound firewall rules in the Azure portal for detailed steps with screenshots and their PowerShell and Azure CLI equivalents.

A local table with the same name already exists

The local database may already contain a table with the same name as the table in the remote database, but with different schema or data. It is then not possible to create a local External Table with that same name. If you need to keep the local table in parallel to the external table, the solution is to use a different schema or table name for the external table.

For this purpose, the `CREATE EXTERNAL TABLE` syntax includes the `SCHEMA_NAME` and `OBJECT_NAME` parameters in its `WITH` clause for providing the details of the remote database object.

Re-using the example from the sample script in the first section above, the external table can be created like this:

```
-- execute in local database
CREATE EXTERNAL TABLE [dbo].[otherDBCustomer] (
   [CustomerID] [int] NOT NULL,
   [CustomerName] [varchar] (50) NOT NULL,
   [Company] [varchar] (50) NOT NULL)
WITH (DATA_SOURCE = MyElasticQueryDataSrc, SCHEMA_NAME = 'dbo', OBJECT_NAME = 'Customer' );

SELECT TOP 100 * FROM dbo.otherDBCustomer ORDER BY CustomerID DESC;
```

Performance issues

The Azure SQL Database query optimizer analyzes the cross-database queries and tries to push as many query restrictions into the remote query execution as possible. It intelligently re-shapes the remote database-related part of the query, re-grouping the Where clause parameters and Join conditions in order to receive the smallest-possible resultset back from the remote database. It relies on the remote database to use its indexes and statistics for optimizing the remote query performance.

Despite these strategies, there are several possible causes for slow query performance. The most common causes are the following:

(1) The local and/or the remote database is reaching its resource limit

This might occur if the current workload is already close to the capacity limits and the Elastic Query has to process a large resultset. Depending on the query itself, Elastic Query might have to process a large amount of data on the remote database, download a large result to the local database, and join the result to local data. This can significantly increase the workload on both local and remote database.

Check Query Performance Insight for both the local and the remote Azure SQL Database to see if the workload is exceeding either database's resource capacity. Query Performance Insight provides intelligent query analysis for single and pooled databases and helps identifying the top resource-consuming and top long-running queries in your workload. See Query Performance Insight for Azure SQL Database for a detailed demonstration and description of the value that Query Performance Insights is giving you.

(2) The performance tier of the databases is too low

The performance of Elastic Queries over large databases will be slow in the lower service tiers like Standard or the smaller General Purpose tiers. Use at least the Premium service level, a higher-level General Purpose size, or Business Critical to receive reliable performance for large databases.

(3) The WHERE or JOIN clause is not sent to the remote query execution

The SQL query optimizer will try to send as many Where clause parameters and Join conditions to the remote database engine as possible. The decision however depends on the perceived local and remote data sizes and the expected size of the resultsets.

There are several possible reasons why the optimizer might consider it more efficient to perform the evaluation of large datasets at the partner database:

- The Where clause is complex, involving sub-selects or referencing other tables
- The external table is joined to several local tables or other external tables
- The external table is part of a LEFT OUTER JOIN or RIGHT OUTER JOIN
- The query optimizer has to take assumptions on the column statistics and number of remote rows, because it is not possible to create indexes or statistics on the local External Table object. The final decision still might be correct, even though it means the transfer of large amounts of data.

In either case, the local or remote table data might be transferred unfiltered over the network, causing latency and slow query executions.

The easiest way to see which part of the Elastic Query is sent to the remote database is through the actual execution plan of the query. To do so, open a query window in SQL Server Management Studio (SSMS) and copy the Elastic Query text into it. On the SSMS menu, select `Query - Include Actual Execution Plan` and then execute the query. Select the `Execution Plan` tab on the results pane and hover the mouse over the `Remote Query` node to see the SQL text that is sent to the remote database. If only a plain Select without Where clause is sent, try to re-arrange or rewrite the Elastic Query to improve the remote execution.

For taking deeper control about what is executed remotely, use the stored procedure sp_execute_remote to receive a single resulset from the remote database. This will force the query execution remotely, which might help significantly if several tables can be joined in the remote database and a selective Where clause can be included with it.

(4) The WHERE or JOIN clause is sent to the remote query execution but executes slowly

This might occur if the remote table does not have helpful statistics or indexes available, or chooses a bad execution plan to run the query. Check the actual execution plan of the Elastic Query (see item (3) above) to retrieve the remote SQL query. Then perform a performance analysis of this query at the remote database, for examply by using Query Performance Insight for Azure SQL Database at the remote database.

(5) Cross-region Elastic Queries might be slow due to latency

If local and remote databases are hosted in different Azure regions, the remote queries and their results will have to be transferred over the network. Although the Azure backbone network is reliable and fast, large resultsets might take more time to receive as compared to queries executed within the same region.

If this impacting your performance, consider moving the databases into the same Azure region to minimize the latency.

(6) The first execution of the Elastic Query is slow, subsequent executions are much faster

This symptom can occur mainly on the Standard service tier. When the external table is accessed the first time, an internal component needs to be loaded before retrieving any remote data.

If the Elastic Query is joining the remote data to local data, it is also possible that the local data needs to be read from storage first. The remote results are not cached at the local database, and repeated executions will always re-read the data from the remote database.

Migration from SQL Server instances to Azure SQL Database

Migrating SQL Server instances, either on-premise or cloud-hosted Virtual Machines, to Azure SQL Database may be blocked because the source databases are using cross-database queries. Stored procedures or views might be using 3-part or 4-part names like "server.database.schema.object", which are not supported on Azure SQL Database. The Data Migration Assistant can help identifying affected objects and databases and will provide additional recommendations to unblock the migration.

The following options can help resolving and avoiding the cross-database queries limitation in Azure SQL Database:

1. Migrate the dependent databases to Azure SQL Database and use the Elastic Query functionality to query across Azure SQL databases. This requires editing the affected stored procedures and views, replacing the cross-database query syntax with external tables.
2. Move the dependent tables from the other databases into the database that is being migrated, preferably before the migration.
3. Migrate to Azure SQL Managed Instance instead of Azure SQL Database. Managed Instance natively supports cross-database queries and Linked Server access, similar to a full SQL Server instance. This is the best option for migrating complex workflows that cannot easily be mapped into external tables.
4. Migrate to SQL Server hosted in an Azure Virtual Machine.
5. Follow a hybrid approach, for example, migrate to SQL Server in Azure VM first and move to Azure SQL Database in a later step.

Resources

- Azure SQL Database elastic query overview
- Get started with cross-database queries (vertical partitioning)
- Cross-database Query in Azure SQL Database
- Query across cloud databases with different schema (vertical partitioning)
- Reporting across scaled-out cloud databases (horizontal partitioning)
- Lesson Learned #191: Performance comparison using Inner vs Left Join in Linked Server/Elastic Query
- Outbound firewall rules for Azure SQL Database

How good have you found this content?