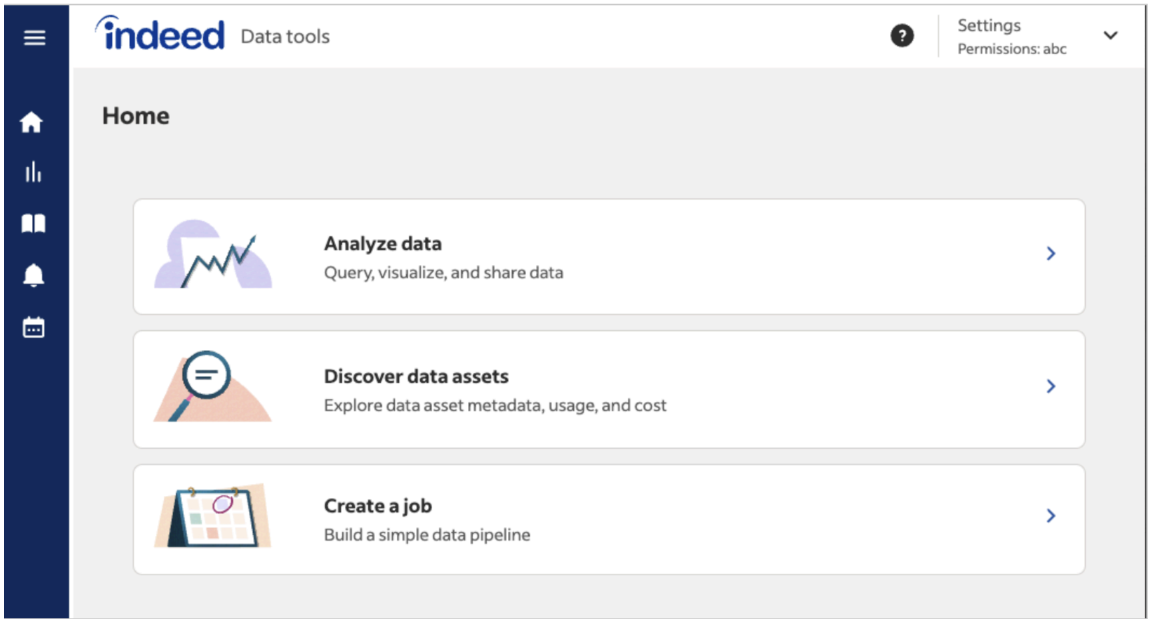
Trino quickstart

Learn to run SQL queries on various data sources including the data lake, Imhotep datasets with required unixtime filters, LogRepo event data, and more.

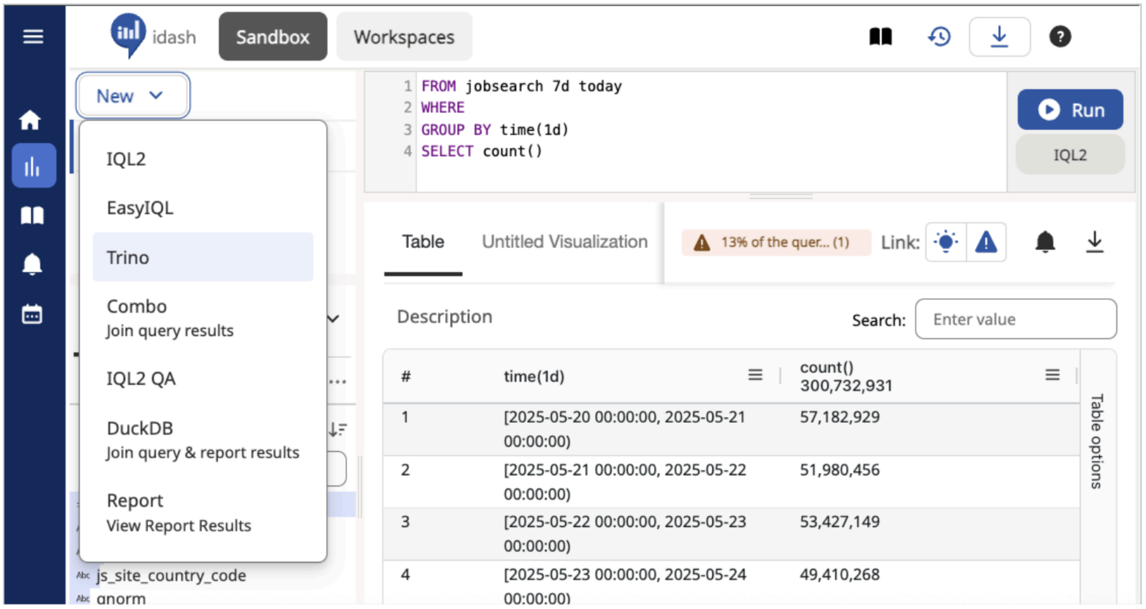
Get started with Trino

Trino is one of the query engines a user can choose in Analyze.

1. Go to the Data tools homepage and select **Analyze data**.


indeed
Data tools
Settings
Permissions: abc
Home
Analyze data
Query, visualize, and share data
Discover data assets
Explore data asset metadata, usage, and cost
Create a job
Build a simple data pipeline

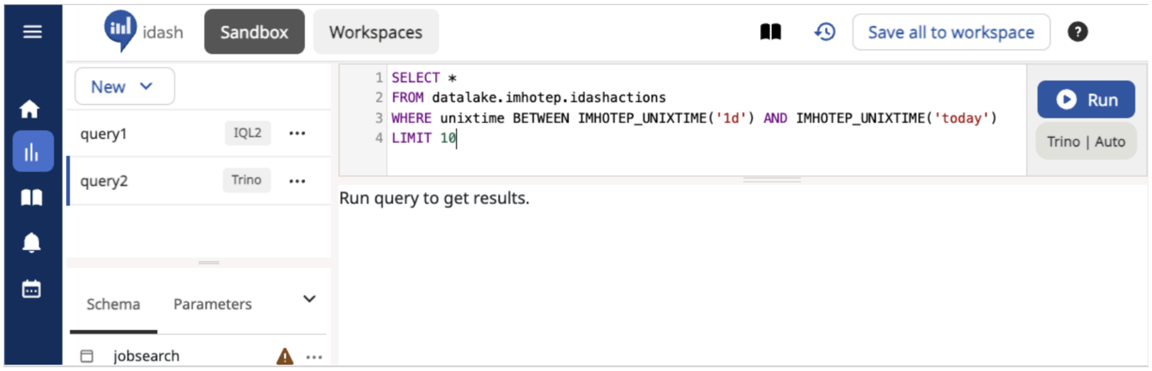
2. Click **New**, then select **Trino** from the dropdown list.



```
FROM jobsearch 7d today
WHERE
GROUP BY time(1d)
SELECT count()
```

#	time(1d)	count() 300,732,931
1	[2025-05-20 00:00:00, 2025-05-21 00:00:00)	57,182,929
2	[2025-05-21 00:00:00, 2025-05-22 00:00:00)	51,980,456
3	[2025-05-22 00:00:00, 2025-05-23 00:00:00)	53,427,149
4	[2025-05-23 00:00:00, 2025-05-24 00:00:00)	49,410,268

The **query2** tab shows a sample Trino query.



```
SELECT *
FROM datalake.imhotep.idashactions
WHERE unixtime BETWEEN IMHOTEP_UNIXTIME('1d') AND IMHOTEP_UNIXTIME('today')
LIMIT 10
```

Run query to get results.

3. To run the query, select **Run**.
   The results window shows the results.

Trino features

Run SQL on datasets of any size

Trino is an ANSI SQL–compliant query engine. A user can run SQL statements based on their access permissions.

- All users can run read-only operations like `SELECT`, `DESCRIBE`, and `SHOW`.
- Admins can run data management operations like `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `MERGE`.

Trino is built for efficient, low-latency analytics. To handle large-scale queries across diverse data sources, Trino uses parallel and distributed query processing.

Query datalake

The data lake is a centralized storage system for analytical datasets. All users can access the data lake in Trino. A user's LDAP account's permissions determine their access to individual tables.

Trino uses a three-part identifier to reference a table: `<catalog>.<schema>.<table_name>`

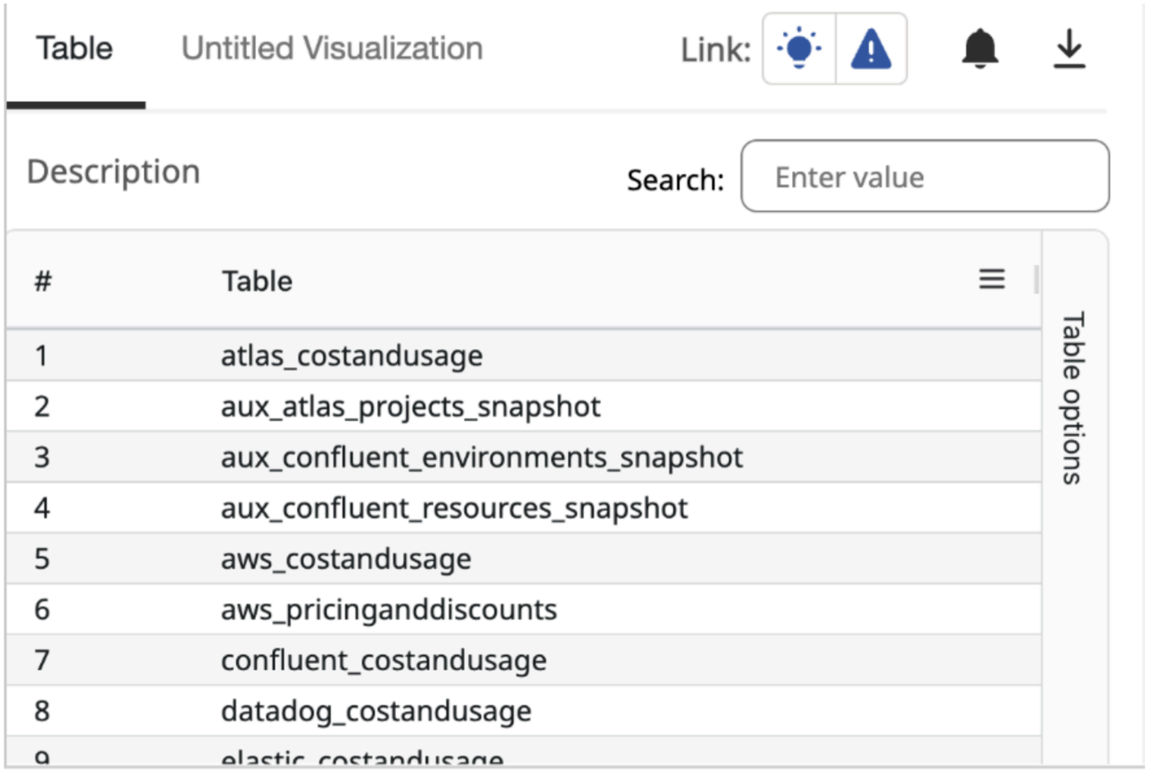
Example: `datalake.imhotep.idashactions`

- `<catalog>` is datalake
- `<schema>` is imhotep
- `<table_name>` is the dataset name
- To list all schemas in the data lake catalog, use this SQL command:

None

```
SHOW SCHEMAS FROM datalake
```

- The command returns these results:



#	Table
1	atlas_costandusage
2	aux_atlas_projects_snapshot
3	aux_confluent_environments_snapshot
4	aux_confluent_resources_snapshot
5	aws_costandusage
6	aws_pricinganddiscounts
7	confluent_costandusage
8	datadog_costandusage
9	elastic_costandusage

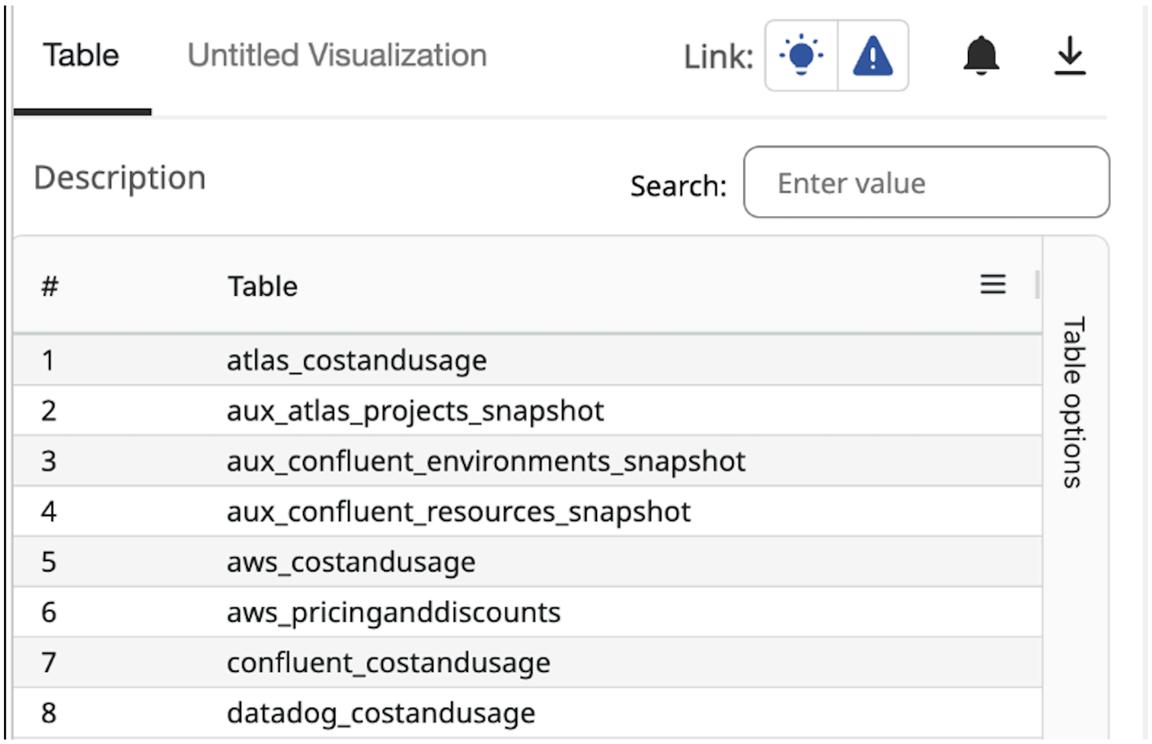
- To list all tables in a schema such as *cloudcost*, use this SQL command:

None

```
SHOW TABLES FROM datalake.cloudcost
```

-

  The command returns these results:



#	Table
1	atlas_costandusage
2	aux_atlas_projects_snapshot
3	aux_confluent_environments_snapshot
4	aux_confluent_resources_snapshot
5	aws_costandusage
6	aws_pricinganddiscounts
7	confluent_costandusage
8	datadog_costandusage

Use SQL to query Imhotep datasets

Imhotep, a high-performance analytics system designed for low-latency queries on large datasets, uses an inverted index. This custom data storage format enables fast filtering and aggregation.

Typically, users call IQL, a specialized query language, to query Imhotep datasets. Users can also call standard SQL to query datasets, enabling `SELECT`, `JOIN`, `UNION`, and other operations.

Table name

To query Imhotep datasets, use these values:

- `<catalog>` is datalake
- `<schema>` is imhotep
- `<table_name>` is the dataset name

Example: To query the jobsearch dataset:

None

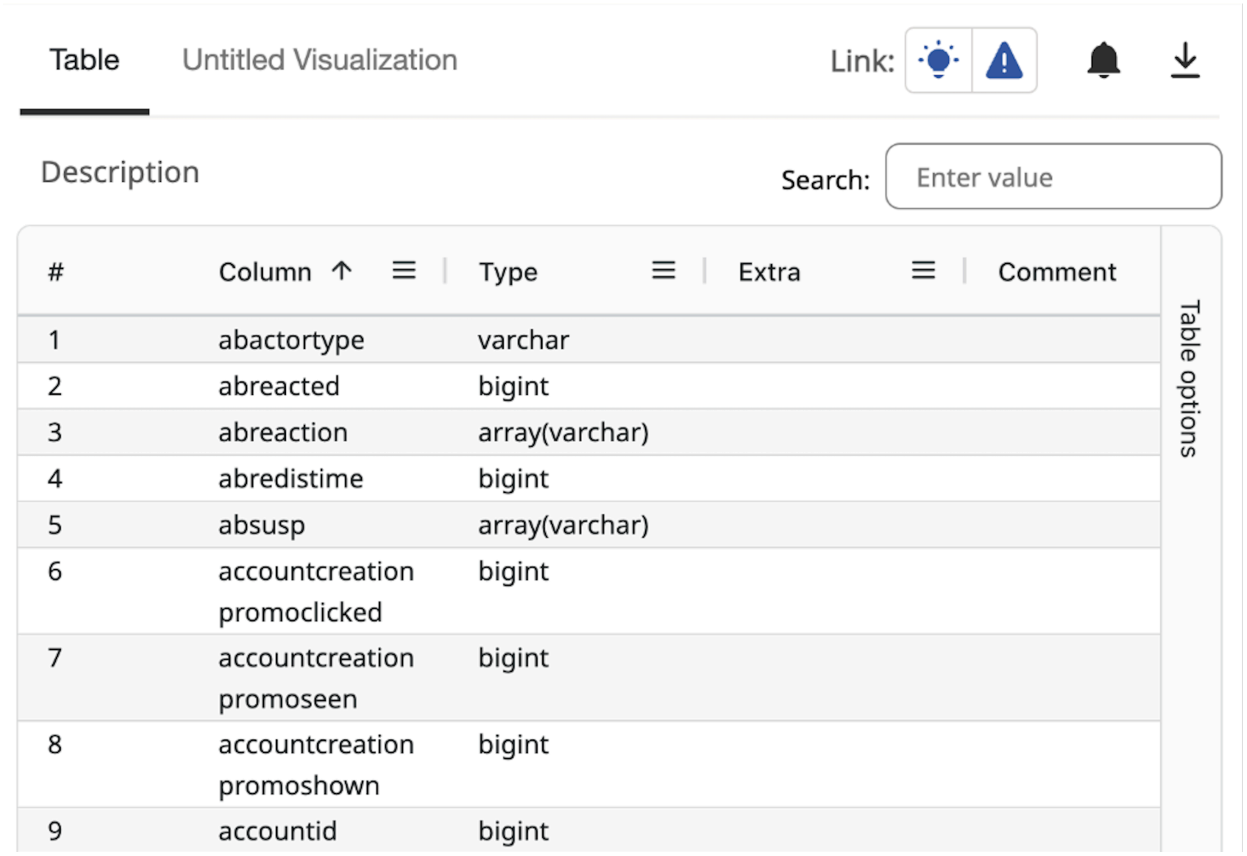
```
SELECT * FROM datalake.imhotep.jobsearch;
```

To view the schema of a dataset:

None

```
DESCRIBE datalake.imhotep.jobsearch;
```

The command returns these results:



#	Column	Type	Extra	Comment
1	abactortype	varchar		
2	abreacted	bigint		
3	abreaction	array(varchar)		
4	abredistime	bigint		
5	absusp	array(varchar)		
6	accountcreation promoclicked	bigint		
7	accountcreation promoseen	bigint		
8	accountcreation promoshown	bigint		
9	accountid	bigint		

Special requirement for SELECT statements

Each Imhotep dataset includes a unixtime field, which represents the time in UNIX timestamp format. Trino requires a unixtime filter in all queries that access Imhotep datasets.

This requirement is similar to the `FROM` clause with a date range in IQL, where users must specify a time window.

To filter dates, use the custom Trino UDF 'imhotep_unixtime()' function, which converts date strings or timestamps into the correct unixtime format.

Example:

None

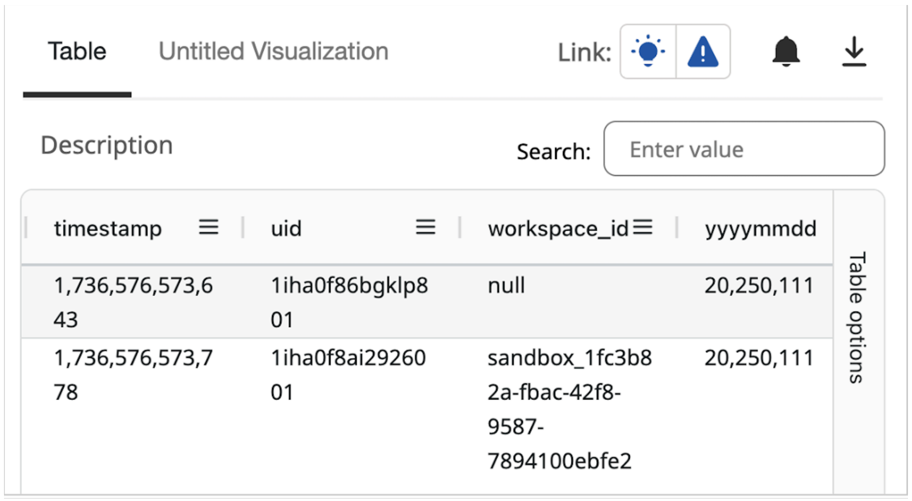
```
SELECT *

FROM datalake.imhotep.jobsearch

WHERE unixtime BETWEEN imhotep_unixtime('2024-01-01')

AND imhotep_unixtime('2024-01-31');
```

The command returns these results:



timestamp	uid	workspace_id	yyyymmdd
1,736,576,573,643	1iha0f86bgklp801	null	20,250,111
1,736,576,573,778	1iha0f8ai2926001	sandbox_1fc3b82a-fbac-42f8-9587-7894100ebfe2	20,250,111

Users can use the `imhotep_unixtime()` UDF with these formats:

Format	Example	Notes

`'yyyy-MM-dd'`	`'2019-01-01'`	
`'yyyy-MM-ddZ'`	`'2019-01-01-06:00'`	`-06:00` is the time zone offset
`'yyyy-MM-dd HH:mm:SS'`	`'2019-01-01 00:00:00'`	
`'yyyy-MM-dd HH:mm:SSZ'`	`'2019-01-01 00:00:00-06:00'`	Includes time zone offset
`'yesterday'`	`'yesterday'`	Relative to current date
`'today'`	`'today'`	
`'tomorrow'`	`'tomorrow'`	
`'5m'`	`'5m'`	5 minutes ago (`m` = minutes)
`'1h'`	`'1h'`	1 hour ago
`'2d'`	`'2d'`	2 days ago
`'3w'`	`'3w'`	3 weeks ago

`'4M'`	`'4M'`	4 months ago (`M` = months)
`'5y'`	`'5y'`	5 years ago

Query team-owned data

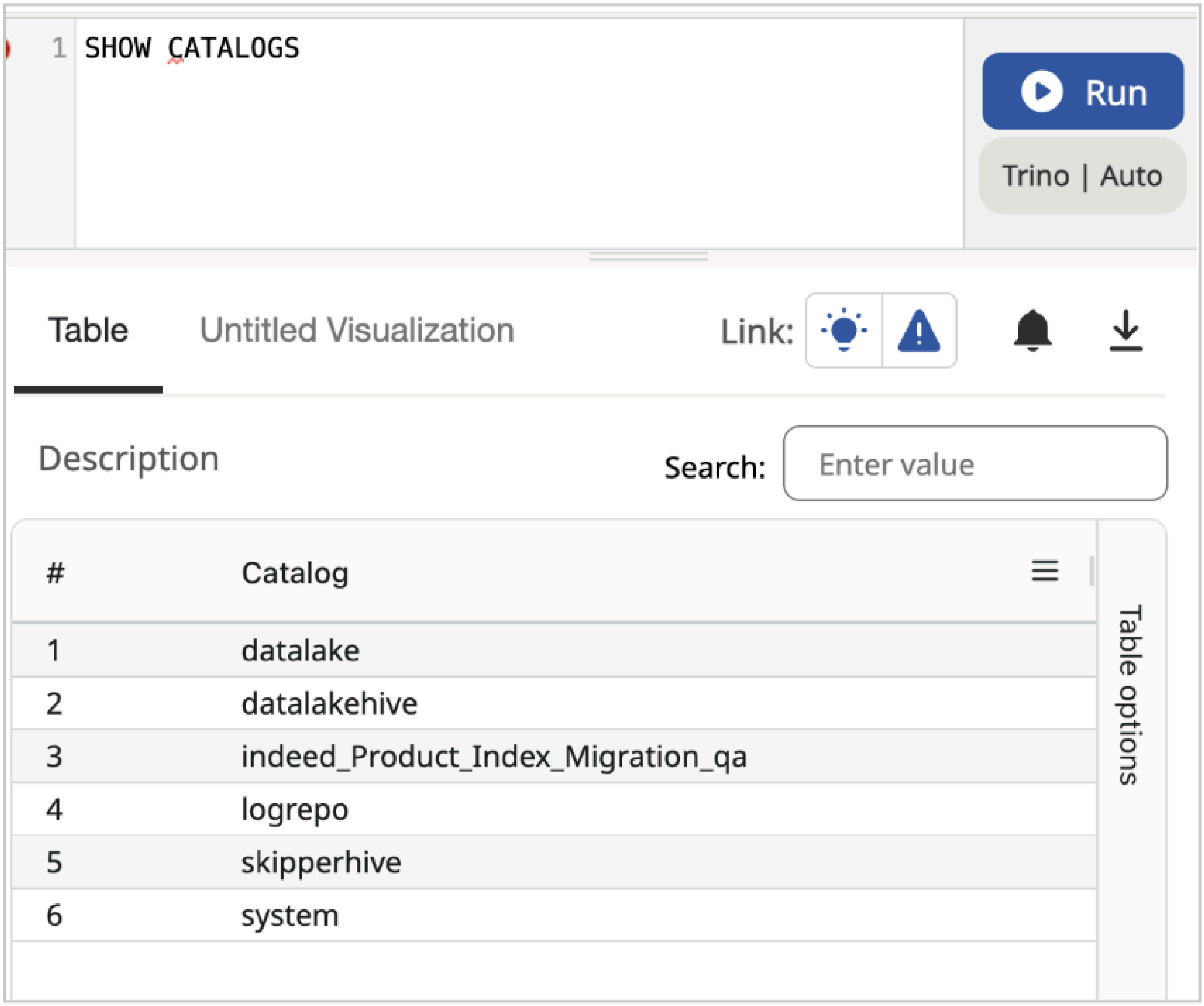
Some teams store their datasets in their own AWS accounts. If the user's account admin has configured the integration and user has access, the user can use Trino to query this data.

In Trino, team-owned datasets appear as separate catalogs. To see the available catalogs:

None

```
SHOW CATALOGS;
```

The command returns these results:



```
SHOW CATALOGS
```

#	Catalog
1	datalake
2	datalakehive
3	indeed_Product_Index_Migration_qa
4	logrepo
5	skipperhive
6	system

If user's team catalog is listed, the user can use it like any other Trino catalog to run queries.

Use cases

Join datasets across sources

Trino can join multiple datasets to perform cross-source analytics. Although IQL does not support SQL-style joins, Trino uses standard SQL syntax to support full join operations.

For better performance, aggregate each dataset separately before joining them, rather than directly joining large raw datasets. This action reduces overhead to speed up processing.

Avoid this pattern (not recommended):

None

```
SELECT js.country,

      js.ipcountry,

      COUNT(js.country) AS js_counts,

      COUNT(ms.country) AS ms_counts

FROM jobsearch AS js

JOIN mobsearch AS ms

 ON js.country = ms.country

AND js.ipcountry = ms.ipcountry

WHERE js.unixtime BETWEEN imhotep_unixtime('2025-01-01
00:00:00')

                     AND imhotep_unixtime('2025-01-02
00:00:00')
```

```
 AND ms.unixtime BETWEEN imhotep_unixtime('2025-01-01
00:00:00')

                     AND imhotep_unixtime('2025-01-02
00:00:00')

GROUP BY 1, 2;
```

Use this pattern (recommended):

None

```

WITH js AS (

   SELECT country,

          ipcountry,

          COUNT(1) AS js_counts

   FROM jobsearch

   WHERE unixtime BETWEEN imhotep_unixtime('2025-01-01
00:00:00')

                     AND imhotep_unixtime('2025-01-02
00:00:00')

   GROUP BY 1, 2

),

ms AS (
   SELECT country,

          ipcountry,

          COUNT(1) AS ms_counts

   FROM mobsearch

   WHERE unixtime BETWEEN imhotep_unixtime('2025-01-01
00:00:00')

                     AND imhotep_unixtime('2025-01-02
00:00:00')

   GROUP BY 1, 2

)

SELECT js.country,

      js.ipcountry,

      js_counts,

      ms_counts

FROM js

JOIN ms

 ON js.country = ms.country

AND js.ipcountry = ms.ipcountry;
```

Use features not supported in IQL

Trino provides advanced SQL capabilities that IQL does not provide:

Regular expressions (Regex)

IQL offers limited regex support. For example:

- User cannot perform case-insensitive matches.
- User cannot extract substrings or use grouping patterns.

However, Trino uses using standard SQL functions like `regexp_like()`, `regexp_extract()`, and `regexp_replace()` to support full regex capabilities, including pattern matching, extraction, and replacements.

Window functions

IQL minimally supports window functions, such as `RUNNING`, `DISTINCT_WINDOW`.

Trino fully supports SQL window functions, including but not limited to:

- `ROW_NUMBER()`
- `RANK()`
- `DENSE_RANK()`
- `LEAD()` / `LAG()`
- `SUM() OVER (...)`
- `PARTITION BY` and `ORDER BY` clauses

These functions enable complex analytics over partitions of data without subqueries or joins.

FAQs

What timezone does Trino use?

Trino and other Indeed reporting systems use Ramses time, or UTC−6. Imhotep originated, and IQL uses, Ramses time. To ensure consistency, any data that joins with Imhotep data must also use Ramses time.

What is the character limit for Trino queries?

Trino queries support up to 1 million characters.

Can Trino access datasets in regions outside us-east-2?

No. Trino currently only supports datasets in the us-east-2 region.

Reasons for this limitation:

- Cross-region network costs
- Cross-region latency
- Legal constraints related to data transfer

How do I resolve the PageTooLargeException: Remote page is too large error?

When this error occurs, a field in the results is too large for a Trino node to process it.

Try this:

- Reduce the size of queried fields.

How do I resolve the =Glue table 'X' column 'yyyymmdd' has invalid data type: long error?

This error suggests user's querying LogRepo data through the wrong catalog.

Solution:

- Use `logrepo.log.X` instead of `datalake.log.X`.

How do I resolve the line A:B: Column 'X' cannot be resolved error?

This error can occur for a few reasons:

- The referenced column does not exist in the table.
- The user is using **double quotes** ("string") for a string literal. Use **single quotes** ('string') for strings.

Tip:

- Single quotes are for Strings.
- Double quotes are for database identifiers.

Why do I get different results when I run the same query multiple times?

User's query might be non-deterministic. It can produce different results each time.

Common causes:

- Functions like `shuffle()`, `random()`, or `now()` generate new values on each run.
- `LIMIT` returns the first results available, which can vary by execution.
- `ORDER BY` doesn't guarantee consistent results unless the sorted fields are unique.
- `row_number()` can return different results for the same reason.
- User-Defined Functions (UDF) can behave differently across runs.

Solutions:

- Use deterministic alternatives like hashing functions.
- Ensure sorted fields are unique for consistent ordering.

Glossary

Term	Description
Cluster	Several Trino nodes: one coordinator and zero or more workers. Users connect to the coordinator with their SQL query tool. The coordinator manages workers and accesses data sources via catalogs.
Coordinator	Parses statements, plans queries, and manages worker nodes. It is the central control node and the point of connection for clients submitting queries.

Glue Catalog	Central metadata repository used within the AWS ecosystem. It supports data discovery, governance, and integration, and is required to define a Trino catalog.
Node	Any Trino server in a specific Trino cluster is considered a node of the cluster.
Schema	Organizes tables within a catalog. Together, a catalog and schema define the scope of tables and queryable objects.
Table	Set of unordered rows organized into named columns with types, similar to tables in traditional relational databases.
Trino Catalog	Set of configuration properties to access a data source. Includes the connector type, credentials, and connection details.
Worker	Runs tasks and processes data. It fetches data from connectors and exchanges intermediate data with other worker nodes.

Summary and next steps

🎉 **Congratulations!** You've completed the Trino quickstart. You can now:

- ✅ Access Trino through the Analyze platform
- ✅ Query data lake tables using standard SQL
- ✅ Work with Imhotep datasets using SQL (with required time filters)
- ✅ Explore schemas and tables to discover available data

- ✅ Use advanced SQL features like joins and aggregations

What you've learned

Data Lake Querying:

- Three-part identifier format: `catalog.schema.table`
- Schema discovery with `SHOW SCHEMAS` and `SHOW TABLES`
- Standard SQL operations on analytical datasets

Imhotep Integration:

- SQL access to high-performance analytics data
- Required `unixtime` filters for optimal performance
- Schema inspection for understanding data structure

What's next?

Explore advanced features:

- Learn about Trino's advanced SQL functions
- Practice with joins across different data sources
- Explore user-defined functions (UDFs) for custom analytics

Performance optimization:

- Use appropriate time filters for large datasets
- Leverage Trino's parallel processing for complex queries
- Monitor query performance in the Analyze interface

Integration workflows:

- Save frequently used queries in Analyze workspaces

- Schedule regular data processing with Orchestrate
- Create dashboards and visualizations from your Trino queries

Need help?

Common issues:

- **Permission errors**: Verify your LDAP permissions for specific tables
- **Slow queries**: Always use time filters on large Imhotep datasets
- **Connection timeouts**: Contact platform support for cluster issues
- **Data not found**: Check schema and table names with `SHOW` commands

Getting support:

- Reference the Trino FAQ section above for specific error solutions
- Check Indeed's data documentation for data source details
- Contact your data team for dataset-specific questions
- Reach out to platform support for technical Trino issues

Best practices:

- Start with small datasets and `LIMIT` clauses
- Use `DESCRIBE` to understand data before querying
- Include time filters for optimal performance
- Save and organize queries in Analyze workspaces