Faker connector

The Faker connector generates random data matching a defined structure. It uses the Datafaker [https://www.datafaker.net/] library to make the generated data more realistic.

Use the connector to test and learn SQL queries without the need for a fixed, imported dataset, or to populate another data source with large and realistic test data. This allows testing the performance of applications processing data, including Trino itself, and application user interfaces accessing the data.

Configuration

Create a catalog properties file that specifies the Faker connector by setting the `connector.name` to `faker`.

For example, to generate data in the `generator` catalog, create the file `etc/catalog/generator.properties`.

```
connector.name=faker
faker.null-probability=0.1
faker.default-limit=1000
faker.locale=pl
```

Create tables in the `default` schema, or create different schemas first. Tables in the catalog only exist as definition and do not hold actual data. Any query reading from tables returns random, but deterministic data. As a result, repeated invocation of a query returns identical data. See Usage [#faker-usage] for more examples.

Schemas, tables, and views in a catalog are not persisted, and are stored in the memory of the coordinator only. They need to be recreated every time after restarting the coordinator.

The following table details all general configuration properties:

Faker configuration properties

Property name	Description
`faker.null-probability`	Default probability of a value created as `null` for any column in any table that allows them. Defaults to `0.5`.
`faker.default-limit`	Default number of rows in a table. Defaults to `1000`.
`faker.locale`	Default locale for generating character-based data, specified as a IETF BCP 47 language tag string. Defaults to `en`.
`faker.sequence-detection-enabled`	If true, when creating a table using existing data, columns with the number of distinct values close to the number of rows are treated as sequences. Defaults to `true`.
`faker.dictionary-detection-enabled`	If true, when creating a table using existing data, columns with a low number of distinct values are treated as dictionaries, and get the `allowed_values` column property populated with random values. Defaults to `true`.

The following table details all supported schema properties. If they’re not set, values from corresponding configuration properties are used.

Faker schema properties

Property name	Description
`null_probability`	Default probability of a value created as `null` in any column that allows them, in any table of this schema.
`default_limit`	Default number of rows in a table.
`sequence_detection_enabled`	If true, when creating a table using existing data, columns with the number of distinct values close to the number of rows are treated as sequences. Defaults to `true`.
`dictionary_detection_enabled`	If true, when creating a table using existing data, columns with a low number of distinct values are treated as dictionaries, and get the `allowed_values` column property populated with random values. Defaults to `true`.

The following table details all supported table properties. If they’re not set, values from corresponding schema properties are used.

Faker table properties

Property name	Description
`null_probability`	Default probability of a value created as `null` in any column that allows `null` in the table.
`default_limit`	Default number of rows in the table.
`sequence_detection_enabled`	If true, when creating a table using existing data, columns with the number of distinct values close to the number of rows are treated as sequences. Defaults to `true`.
`dictionary_detection_enabled`	If true, when creating a table using existing data, columns with a low number of distinct values are treated as dictionaries, and get the `allowed_values` column property populated with random values. Defaults to `true`.

The following table details all supported column properties.

Faker column properties

Property name	Description
`null_probability`	Default probability of a value created as `null` in the column. Defaults table or schema property, if set, or the `faker.null-probability` conf
`generator`	Name of the Faker library generator used to generate data for the colu of a character-based type. Defaults to a 3 to 40 word sentence from th [https://javadoc.io/doc/net.datafaker/datafaker/latest/net/datafaker/ provider.
`min`	Minimum generated value (inclusive). Cannot be set for character-bas
`max`	Maximum generated value (inclusive). Cannot be set for character-bas
`allowed_values`	List of allowed values. Cannot be set together with the `min`, or `max` pr
`step`	If set, generate sequential values with this step. For date and time colu Cannot be set for character-based type columns.

Character types

Faker supports the following character types:

- `CHAR`
- `VARCHAR`
- `VARBINARY`

Columns of those types use a generator producing the Lorem ipsum [https://en.wikipedia.org/wiki/Lorem_ipsum] placeholder text. Unbounded columns return a random sentence with 3 to 40 words.

To have more control over the format of the generated data, use the `generator` column property. Some examples of valid generator expressions:

- `#{regexify '(a|b){2,3}'}`
- `#{regexify '\\.\\*\\?\\+'}`
- `#{bothify '????','false'}`
- `#{Name.first_name} #{Name.first_name} #{Name.last_name}`
- `#{number.number_between '1','10'}`

See the Datafaker's documentation for more information about the expression [https://www.datafaker.net/documentation/expressions/] syntax and available providers [https://www.datafaker.net/documentation/providers/].

random_string(*expression_string***)** → string

> Create a random output `string` with the provided input `expression_string`. The expression must use the syntax from Datafaker [https://www.datafaker.net/documentation/expressions/].
>
> Use the `random_string` function from the `default` schema of the `generator` catalog to test a generator expression:
>
> ```
> SELECT generator.default.random_string('#{Name.first_name}');
> ```

Non-character types

Faker supports the following non-character types:

- `BIGINT`
- `INTEGER` or `INT`

- `SMALLINT`
- `TINYINT`
- `BOOLEAN`
- `DATE`
- `DECIMAL`
- `REAL`
- `DOUBLE`
- `INTERVAL DAY TO SECOND`
- `INTERVAL YEAR TO MONTH`
- `TIMESTAMP` and `TIMESTAMP(P)`
- `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP(P) WITH TIME ZONE`
- `TIME` and `TIME(P)`
- `TIME WITH TIME ZONE` and `TIME(P) WITH TIME ZONE`
- `IPADDRESS`
- `UUID`

You can not use generator expressions for non-character-based columns. To limit their data range, set the `min` and `max` column properties - see Usage [#faker-usage].

Unsupported types

Faker does not support the following data types:

- Structural types `ARRAY`, `MAP`, and `ROW`
- `JSON`
- Geometry
- HyperLogLog and all digest types

To generate data using these complex types, data from column of primitive types can be combined, like in the following example:

```
CREATE TABLE faker.default.prices (
  currency VARCHAR NOT NULL WITH (generator = '#{Currency.code}'),
  price DECIMAL(8,2) NOT NULL WITH (min = '0')
);

SELECT JSON_OBJECT(KEY currency VALUE price) AS complex
FROM faker.default.prices
LIMIT 3;
```

Running the queries returns data similar to the following result:

```
      complex
--------------------
 {"TTD":924657.82}
 {"MRO":968292.49}
 {"LTL":357773.63}
(3 rows)
```

Number of generated rows

By default, the connector generates 1000 rows for every table. To control how many rows are generated for a table, use the `LIMIT` clause in the query. A default limit can be set using the `default_limit` table, or schema property or in the connector configuration file, using the `faker.default-limit` property. Use a limit value higher than the configured default to return more rows.

Null values

For columns without a `NOT NULL` constraint, `null` values are generated using the default probability of 50%. It can be modified using the `null_probability` property set for a column, table, or schema. The default value of 0.5 can be also modified in the catalog configuration file, by using the `faker.null-probability` property.

Type mapping

The Faker connector generates data itself, so no mapping is required.

SQL support

The connector provides globally available [../language/sql-support.html#sql-globally-available] and read operation [../language/sql-support.html#sql-read-operations] statements to generate data.

To define the schema for generating data, it supports the following features:

- CREATE TABLE [../sql/create-table.html]
- CREATE TABLE AS [../sql/create-table-as.html], see also Using existing data statistics [#faker-statistics]
- DROP TABLE [../sql/drop-table.html]
- CREATE SCHEMA [../sql/create-schema.html]
- DROP SCHEMA [../sql/drop-schema.html]
- View management [../language/sql-support.html#sql-view-management]

Usage

Faker generates data when reading from a table created in a catalog using this connector. This makes it easy to fill an existing schema with random data, by copying only the schema into a Faker catalog, and inserting the data back into the original tables.

Using the catalog definition from Configuration you can proceed with the following steps.

Create a table with the same columns as in the table to populate with random data. Exclude all properties, because the Faker connector doesn't support the same table

properties as other connectors.

```
CREATE TABLE generator.default.customer (LIKE
production.public.customer EXCLUDING PROPERTIES);
```

Insert random data into the original table, by selecting it from the `generator` catalog. Data generated by the Faker connector for columns of non-character types cover the whole range of that data type. Set the `min` and `max` column properties, to adjust the generated data as desired. The following example ensures that date of birth and age in years are related and realistic values.

Start with getting the complete definition of a table:

```
SHOW CREATE TABLE production.public.customers;
```

Modify the output of the previous query and add some column properties.

```
CREATE TABLE generator.default.customer (
  id UUID NOT NULL,
  name VARCHAR NOT NULL,
  address VARCHAR NOT NULL,
  born_at DATE WITH (min = '1900-01-01', max = '2025-01-01'),
  age_years INTEGER WITH (min = '0', max = '150'),
  group_id INTEGER WITH (allowed_values = ARRAY['10', '32', '81'])
);
```

```
INSERT INTO production.public.customers
SELECT *
FROM generator.default.customers
LIMIT 100;
```

To generate even more realistic data, choose specific generators by setting the `generator` property on columns.

```
CREATE TABLE generator.default.customer (
  id UUID NOT NULL,
  name VARCHAR NOT NULL WITH (generator = '#{Name.first_name} #
{Name.last_name}'),
```

```
  address VARCHAR NOT NULL WITH (generator = '#
{Address.fullAddress}'),
  born_at DATE WITH (min = '1900-01-01', max = '2025-01-01'),
  age_years INTEGER WITH (min = '0', max = '150'),
  group_id INTEGER WITH (allowed_values = ARRAY['10', '32', '81'])
);
```

Using existing data statistics

The Faker connector automatically sets the `default_limit` table property, and the `min`, `max`, and `null_probability` column properties, based on statistics collected by scanning existing data read by Trino from the data source. The connector uses these statistics to be able to generate data that is more similar to the original data set, without using any of that data:

```
CREATE TABLE generator.default.customer AS
SELECT *
FROM production.public.customer
WHERE created_at > CURRENT_DATE - INTERVAL '1' YEAR;
```

Instead of using range, or other predicates, tables can be sampled, see TABLESAMPLE [../sql/select.html#tablesample].

When the `SELECT` statement doesn't contain a `WHERE` clause, a shorter notation can be used:

```
CREATE TABLE generator.default.customer AS TABLE
production.public.customer;
```

The Faker connector detects sequence columns, which are integer column with the number of distinct values almost equal to the number of rows in the table. For such columns, Faker sets the `step` column property to 1.

Sequence detection can be turned off using the `sequence_detection_enabled` table, or schema property or in the connector configuration file, using the `faker.sequence-detection-enabled` property.

The Faker connector detects dictionary columns, which are columns of non-character types with the number of distinct values lower or equal to 1000. For such columns, Faker generates a list of random values to choose from, and saves it in the `allowed_values` column property.

Dictionary detection can be turned off using the `dictionary_detection_enabled` table, or schema property or in the connector configuration file, using the `faker.dictionary-detection-enabled` property.

For example, copy the `orders` table from the TPC-H connector with statistics, using the following query:

```
CREATE TABLE generator.default.orders AS TABLE tpch.tiny.orders;
```

Inspect the schema of the table created by the Faker connector:

```
SHOW CREATE TABLE generator.default.orders;
```

The table schema should contain additional column and table properties.

```
CREATE TABLE generator.default.orders (
   orderkey bigint WITH (max = '60000', min = '1', null_probability =
0E0, step = '1'),
   custkey bigint WITH (allowed_values =
ARRAY['153','662','1453','63','784', ..., '1493','657'],
null_probability = 0E0),
   orderstatus varchar(1),
   totalprice double WITH (max = '466001.28', min = '874.89',
null_probability = 0E0),
   orderdate date WITH (max = '1998-08-02', min = '1992-01-01',
null_probability = 0E0),
   orderpriority varchar(15),
   clerk varchar(15),
   shippriority integer WITH (allowed_values = ARRAY['0'],
null_probability = 0E0),
   comment varchar(79)
)
WITH (
```

```
    default_limit = 15000
)
```