Session Stream Data

Once you have access to your Session Stream files, you need an automated way to consume these files. If you're unfamiliar with managing scripts and database operations, consult the members of your team who are responsible for handling data for additional guidance.

Reading Session Stream Files

After you download session files, you must parse the data into in-memory objects. You can use the following sample Python script as a basic starting point:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from argparse import ArgumentParser
import gzip
import json
import os

def main():
    parser = ArgumentParser(
            description="Parse Monetate Session Stream files recursively under the specified path.")
    parser.add_argument('input_path', help="The root directory under which your files live")
    args = parser.parse_args()

    for root, subfolders, files in os.walk(args.input_path):
        for filename in files:
            with gzip.open(os.path.join(root, filename), "r") as fileobject:
                for line in fileobject:
                    session = json.loads(line)
                    # ADD CODE TO INTERACT WITH THE SESSION DATA HERE

if __name__ == '__main__':
    main()
```

Now you can transform and load the data into a database, use it to drive application code, join Monetate data with your own data for further analysis, and more. Consult the Session Data Description for the detailed structure of the data object.

Using Session Stream Data with a BI Platform

If your BI platform accepts JSON files as a data source, you can load Session Stream files as is. You may need to decompress the JSON files first. Consult your platform's documentation to determine if it supports JSON files and to learn how to load data.

Due to the volume of data involved, Monetate highly recommends loading Session Stream data into a database

first. You can then configure the database as a data source in your BI platform. Most BI platforms better support connecting to databases rather than loading JSON files.

The session JSON structure contains nested data. For this reason you should split the data into several tables if you plan to use a relational database. Consult your database's documentation, your database administrator, and the Session Data Description for guidance on designing the table structure.

You can use the Monetate Metadata API to get interpretable name strings for values such as offers and page events. You should place them in tables that you can access along with your Session Stream data. See Get Interpretable Values with the Metadata API for more information.

Creating the Database Schema

The schema below provides an example of a MySQL database without metadata:

```
CREATE TABLE `session_sessions` (
  `session_id` varchar(255) NOT NULL,
  `account_id` int(11) NOT NULL,
  `browser` varchar(255) NOT NULL,
  `browser_version` varchar(255) NOT NULL,
  `city` varchar(255) NOT NULL,
  `country_code` varchar(2) NOT NULL,
  `customer_id` varchar(255) DEFAULT NULL,
  `customer_link` varchar(255) DEFAULT NULL,
  `device_type` varchar(255) NOT NULL,
  `end_time` bigint(13) NOT NULL,
  `guid` varchar(255) NOT NULL,
  `has_cart` tinyint(1) NOT NULL,
  `has_new_customer` tinyint(1) NOT NULL,
  `has_product_view` tinyint(1) NOT NULL,
  `has_purchase` tinyint(1) NOT NULL,
  `has_stealth` tinyint(1) NOT NULL,
  `is_bounce` tinyint(1) NOT NULL,
  `is_closed` tinyint(1) NOT NULL,
  `os` varchar(255) NOT NULL,
  `os_version` varchar(255) NOT NULL,
  `page_views` int(11) NOT NULL,
  `page_views_ss` int(11) NOT NULL,
  `product_view_count` int(11) NOT NULL,
  `purchase_count` int(11) NOT NULL,
  `purchase_value_ss` decimal(16,4) NOT NULL,
  `region` varchar(255) NOT NULL,
  `screen_height` int(11) NOT NULL,
  `screen_width` int(11) NOT NULL,
  `session_count` int(11) NOT NULL,
  `session_value` decimal(16,4) NOT NULL,
  `session_value_ss` decimal(16,4) NOT NULL,
  `start_time` bigint(13) NOT NULL
```

```
  `start_time` bigint(13) NOT NULL,
  `time_on_site` int(11) NOT NULL,
  `time_on_site_ss` int(11) NOT NULL,
  PRIMARY KEY (`session_id`)
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_custom_targets` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `custom_target_id` bigint(13) NOT NULL,
  `custom_target_value` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_idx` (`session_id`),
  CONSTRAINT `session_id_custom_targets` FOREIGN KEY (`session_id`) REFERENCES `session_ses
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_offers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `offer_id` varchar(255) NOT NULL,
  `timestamp` bigint(13) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_offers_idx` (`session_id`),
  CONSTRAINT `session_id_offers` FOREIGN KEY (`session_id`) REFERENCES `session_sessions` (`s
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_page_event_ids` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `event_id` int(11) NOT NULL,
  `timestamp` bigint(13) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_page_event_ids_idx` (`session_id`),
  CONSTRAINT `session_id_page_event_ids` FOREIGN KEY (`session_id`) REFERENCES `session_ses
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_purchases` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `purchase_id` varchar(255) NOT NULL,
  `purchase_value` decimal(16,4) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_purchases_idx` (`session_id`),
  CONSTRAINT `session_id_purchases` FOREIGN KEY (`session_id`) REFERENCES `session_sessions`
```

```
  CONSTRAINT `session_id_purchases` FOREIGN KEY (`session_id`) REFERENCES `session_sessions`
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_cart_lines` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `product_id` varchar(255) NOT NULL,
  `sku` varchar(255) NOT NULL,
  `quantity` int(11) NOT NULL,
  `time` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_cart_lines_idx` (`session_id`),
  CONSTRAINT `session_id_cart_lines` FOREIGN KEY (`session_id`) REFERENCES `session_sessions`
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_purchase_lines` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `purchase_id` varchar(255) NOT NULL,
  `time` datetime NOT NULL,
  `total` decimal(16,4) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_purchase_lines_idx` (`session_id`),
  KEY `purchase_id_purchase_lines_idx` (`purchase_id`),
  CONSTRAINT `session_id_purchase_lines` FOREIGN KEY (`session_id`) REFERENCES `session_sess
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_purchase_line_items` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `purchase_id` varchar(255) NOT NULL,
  `product_id` varchar(255) NOT NULL,
  `sku` varchar(255) NOT NULL,
  `quantity` int(11) NOT NULL,
  `unit_price` decimal(16,4) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `purchase_id_purchase_line_items_idx` (`purchase_id`),
  CONSTRAINT `purchase_id_purchase_line_items` FOREIGN KEY (`purchase_id`) REFERENCES `ses
) DEFAULT CHARSET=utf8;

CREATE TABLE `session_view_lines` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `session_id` varchar(255) NOT NULL,
  `product_id` varchar(255) NOT NULL,
  `time` datetime NOT NULL,
  PRIMARY KEY (`id`),
```

```
  UNIQUE KEY `id_UNIQUE` (`id`),
  KEY `session_id_view_lines_idx` (`session_id`),
  CONSTRAINT `session_id_view_lines` FOREIGN KEY (`session_id`) REFERENCES `session_sessions`
) DEFAULT CHARSET=utf8;
```

Loading the Data

The following sample script demonstrates how to load data into the tables created by the schema above.

While it's not a requirement for using Session Stream, you need MySQL Connector/Python to run this sample script.

```
#!/usr/bin/env python
from argparse import ArgumentParser
import gzip
import json
import os

import mysql.connector

def main():
    parser = ArgumentParser(description="Insert data from Monetate Session Stream files organize
    parser.add_argument('input_path', help="The root directory where your files live")
    parser.add_argument('-u', '--username', required=True, help='Database connection username')
    parser.add_argument('-p', '--password', required=True, help='Database connection password')
    parser.add_argument('-d', '--database', required=True, help='Name of the schema where sessio
    parser.add_argument('-a', '--address', default='127.0.0.1', help='Database address')
    parser.add_argument('-v', '--verbose', action='count', help='v for filenames, vv for sessions, vvv
    args = parser.parse_args()

    sessions_sql = """
        INSERT INTO session_sessions (
            session_id,
            account_id,
            browser,
            browser_version,
            city,
            country_code,
            customer_id,
            customer_link,
            device_type,
            end_time,
            guid,
            has_cart,
            has_new_customer,
```

```
        has_product_view,
        has_purchase,
        has_stealth,
        is_bounce,
        is_closed,
        os,
        os_version,
        page_views,
        page_views_ss,
        product_view_count,
        purchase_count,
        purchase_value_ss,
        region,
        screen_height,
        screen_width,
        session_count,
        session_value,
        session_value_ss,
        start_time,
        time_on_site,
        time_on_site_ss
    )
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %
    %s, %s, %s, %s, %s, %s, %s, %s)
"""
custom_targets_sql = """
    INSERT INTO session_custom_targets (
        session_id,
        custom_target_id,
        custom_target_value
    )
    VALUES (%s, %s, %s)
"""
offers_sql = """
    INSERT INTO session_offers (
        session_id,
        offer_id,
        timestamp
    )
    VALUES (%s, %s, %s)
"""
page_event_ids_sql = """
    INSERT INTO session_page_event_ids (
        session_id,
        event_id,
        timestamp
    )
```

```
    )
    VALUES (%s, %s, %s)
"""
purchases_sql = """
    INSERT INTO session_purchases (
        session_id,
        purchase_id,
        purchase_value
    )
    VALUES (%s, %s, %s)
"""
cart_lines_sql = """
    INSERT INTO session_cart_lines (
        session_id,
        product_id,
        sku,
        quantity,
        time
    )
    VALUES (%s, %s, %s, %s, %s)
"""
view_lines_sql = """
    INSERT INTO session_view_lines (
        session_id,
        product_id,
        time
    )
    VALUES (%s, %s, %s)
"""
purchase_lines_sql = """
    INSERT INTO session_purchase_lines (
        session_id,
        purchase_id,
        time,
        total
    )
    VALUES (%s, %s, %s, %s)
"""
purchase_line_items_sql = """
    INSERT INTO session_purchase_line_items (
        purchase_id,
        product_id,
        sku,
        quantity,
        unit_price
    )
    VALUES (%s, %s, %s, %s, %s)
"""
```

```
conn = mysql.connector.connect(user=args.username, password=args.password, host=args.ad

for root, subfolders, files in os.walk(args.input_path):
    for filename in files:
        with gzip.open(os.path.join(root, filename), "r") as infile:
            if args.verbose >= 1:
                print(">>> " + filename)
            raw_data_iter = (json.loads(line) for line in infile)
            cursor = conn.cursor()
            for raw_datum in raw_data_iter:
                sessions_data = (
                        raw_datum['session_id'],
                        raw_datum['account_id'],
                        raw_datum['browser'],
                        raw_datum['browser_version'],
                        raw_datum['city'],
                        raw_datum['country_code'],
                        raw_datum['customer_id'],
                        raw_datum['customer_link'],
                        raw_datum['device_type'],
                        raw_datum['end_time'],
                        raw_datum['guid'],
                        1 if raw_datum['has_cart'] == 't' else 0,
                        1 if raw_datum['has_new_customer'] == 't' else 0,
                        1 if raw_datum['has_product_view'] == 't' else 0,
                        1 if raw_datum['has_purchase'] == 't' else 0,
                        1 if raw_datum['has_stealth'] == 't' else 0,
                        1 if raw_datum['is_bounce'] == 't' else 0,
                        1 if raw_datum['is_closed'] == 't' else 0,
                        raw_datum['os'],
                        raw_datum['os_version'],
                        raw_datum['page_views'],
                        raw_datum['page_views_ss'],
                        raw_datum['product_view_count'],
                        raw_datum['purchase_count'],
                        raw_datum['purchase_value_ss'],
                        raw_datum['region'],
                        raw_datum['screen_height'],
                        raw_datum['screen_width'],
                        raw_datum['session_count'],
                        raw_datum['session_value'],
                        raw_datum['session_value_ss'],
                        raw_datum['start_time'],
                        raw_datum['time_on_site'],
                        raw_datum['time_on_site_ss']
                        )
```

```
try:
    cursor.execute(sessions_sql, sessions_data)
except mysql.connector.IntegrityError as err:
    # Ignore already-inserted sessions.
    print ("**** Error: {}".format(err))
    continue
if args.verbose >= 2:
    print('---> sessions: {}'.format(sessions_data))

for custom_target_id in raw_datum['custom_targets']:
    custom_target = (
            raw_datum['session_id'],
            custom_target_id,
            raw_datum['custom_targets'][custom_target_id]
            )
    cursor.execute(custom_targets_sql, custom_target)
    if args.verbose >= 3:
        print('-> custom_targets: {}'.format(custom_target))

for offer_id in raw_datum['offers']:
    offer = (raw_datum['session_id'],
             offer_id,
             raw_datum['offers'][offer_id])
    cursor.execute(offers_sql, offer)
    if args.verbose >= 3:
        print('-> offers: {}'.format(offer))

for event_id in raw_datum['page_event_ids']:
    page_event_id = (raw_datum['session_id'],
                     event_id,
                     raw_datum['page_event_ids'][event_id])
    cursor.execute(page_event_ids_sql, page_event_id)
    if args.verbose >= 3:
        print('-> page_event_ids: {}'.format(page_event_id))

for purchase_id in raw_datum['purchases']:
    purchase = (raw_datum['session_id'],
                purchase_id,
                raw_datum['purchases'][purchase_id])
    cursor.execute(purchases_sql, purchase)
    if args.verbose >= 3:
        print('-> purchases: {}'.format(purchase))

for cart_line in raw_datum['cart_lines']:
    line = (raw_datum['session_id'],
            cart_line['product_id'],
```

```
                    cart_line['sku'],
                    cart_line['quantity'],
                    cart_line['time'])
                cursor.execute(cart_lines_sql, line)
                if args.verbose >= 3:
                    print('-> cart_lines: {}'.format(line))

            for view_line in raw_datum['view_lines']:
                # TODO mguo: mysql datetime format?
                view = (raw_datum['session_id'],
                        view_line['product_id'],
                        view_line['time'])
                cursor.execute(view_lines_sql, view)
                if args.verbose >= 3:
                    print('-> view_lines: {}'.format(view))

            for purchase_id in raw_datum['purchase_lines']:
                line = (raw_datum['session_id'],
                        purchase_id,
                        raw_datum['purchase_lines'][purchase_id]['time'],
                        raw_datum['purchase_lines'][purchase_id]['total'])
                cursor.execute(purchase_lines_sql, line)
                if args.verbose >= 3:
                    print('-> purchase_lines: {}'.format(line))

                for purchase_line_item in raw_datum['purchase_lines'][purchase_id]['items']:
                    line_item = (purchase_id,
                                 purchase_line_item['product_id'],
                                 purchase_line_item['sku'],
                                 purchase_line_item['quantity'],
                                 purchase_line_item['unit_price'])
                    cursor.execute(purchase_line_items_sql, line_item)
                    if args.verbose >= 3:
                        print('-> purchase_line_items: {}'.format(line_item))

        if args.verbose >= 2:
            print("------END FILE------\n")
        conn.commit()
        cursor.close()

    conn.close()

if __name__ == '__main__':
    main()
```

If your site receives a large amount of traffic, your dataset may be too large for this type of script to

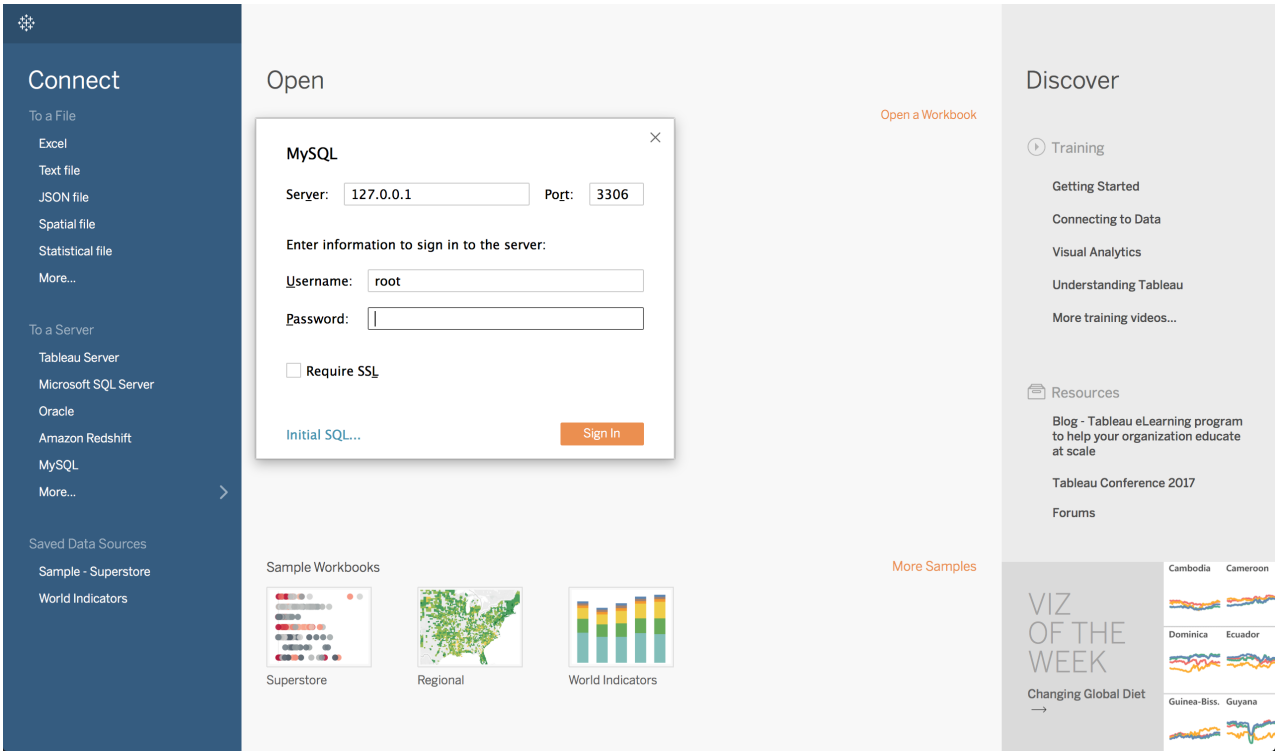
> process in a reasonable amount of time. In that case, use specialized Extract, Transform, Load (ETL) tools for better throughput.

Connecting with a BI Platform

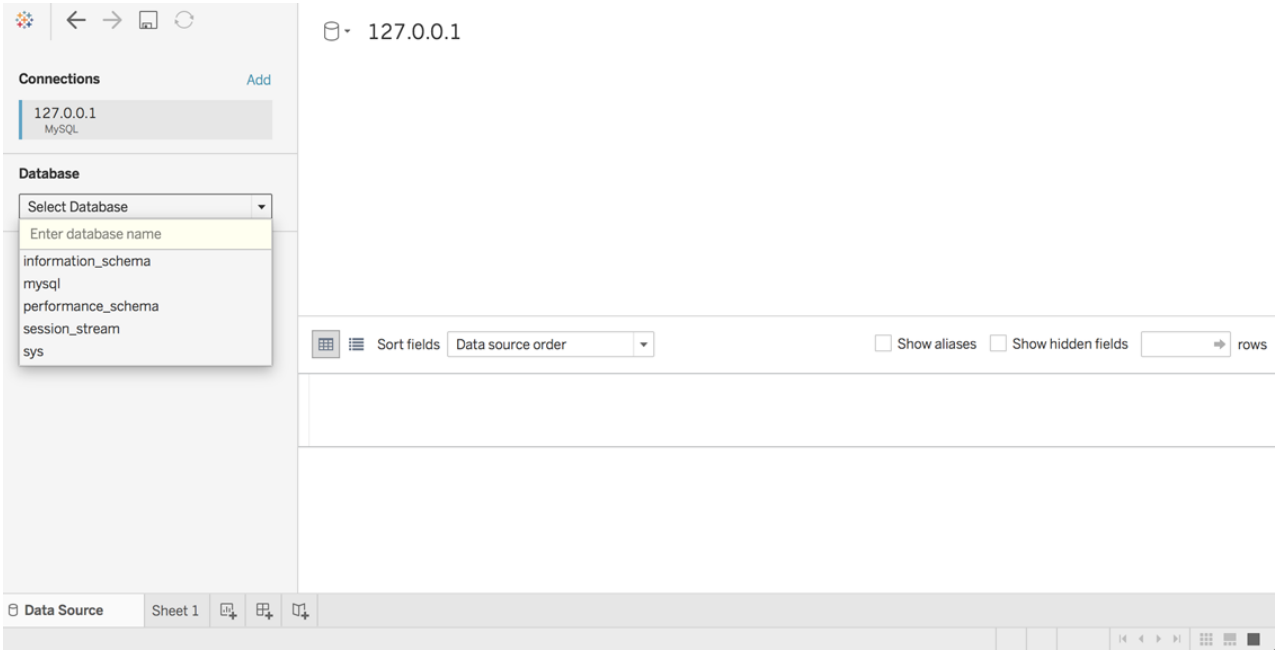
Once you store the data in a database, you can configure it as a data source in your platform. Consult the platform's documentation and support resources for how to do this.

The following example demonstrates how to use Session Stream data for analysis in Tableau Desktop.

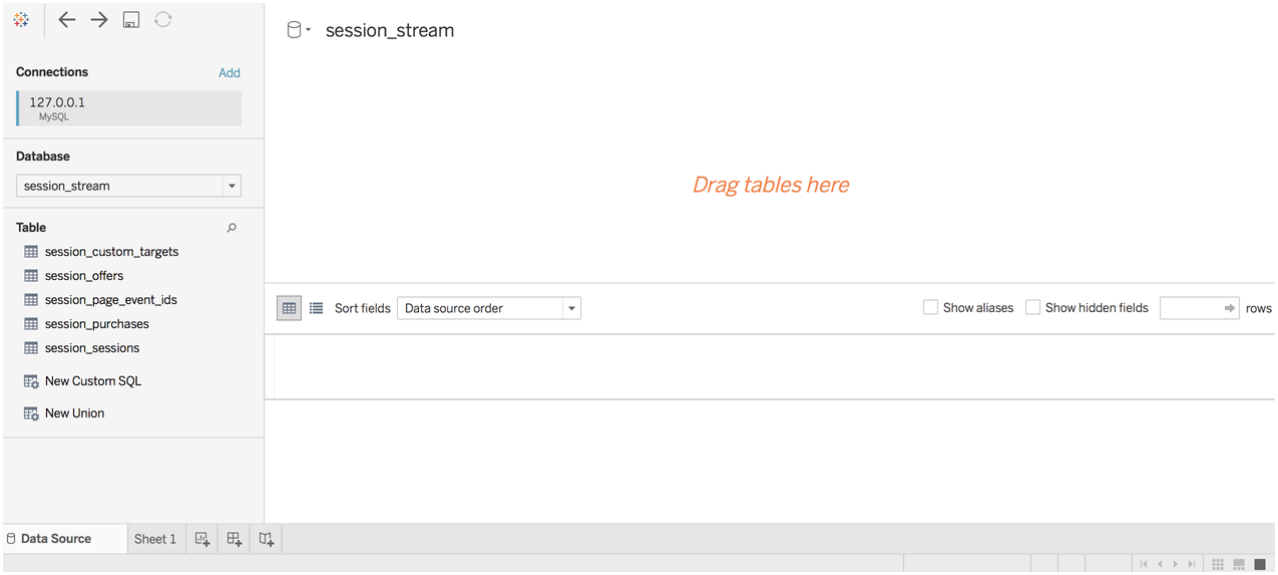
First, create a new workbook in Tableau. Select the MySQL data source, and then fill in the connection details.



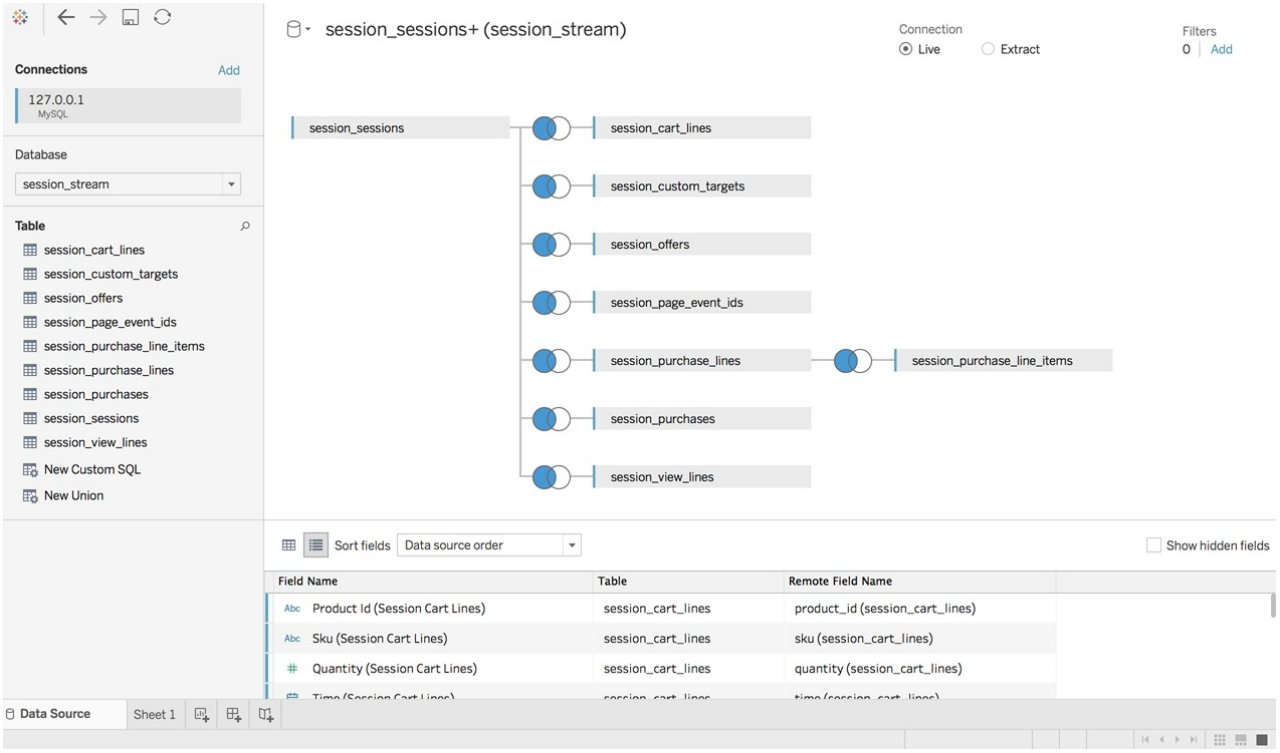
In the Database section on the left sidebar of the Data Source panel, select the database (synonymous with "schema" in MySQL) that contains your Session Stream data tables.



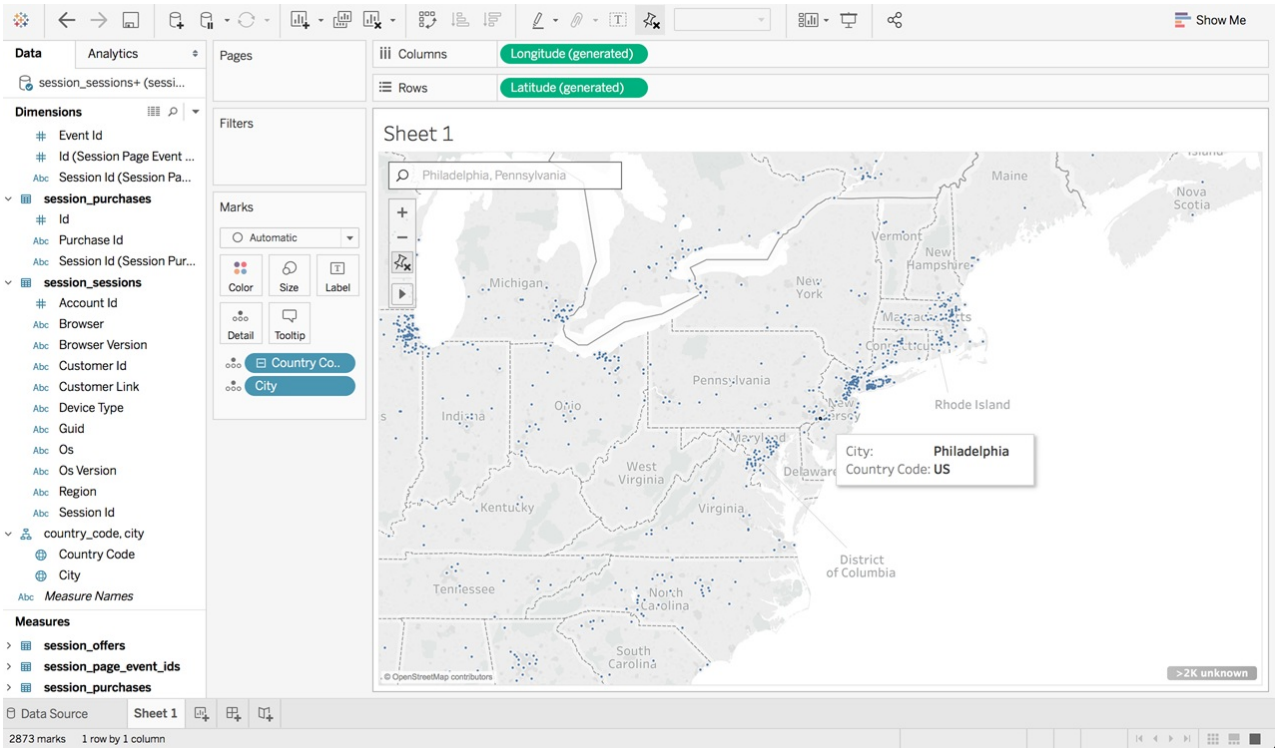
Drag the tables from the left sidebar to the main panel.



Set up a left-join between the main Sessions table and the four auxiliary tables containing nested data.



Build queries and analytics with your data.



You may need to convert some Measures to Dimensions and vice versa in the Data pane in the left sidebar to more accurately characterize the fields.