TigerGraph

TigerGraph Hands-On Training

(Taming the Tiger ☺.)

Unleashing the power of Interconnected Data
for Deeper Insights and Better Outcomes

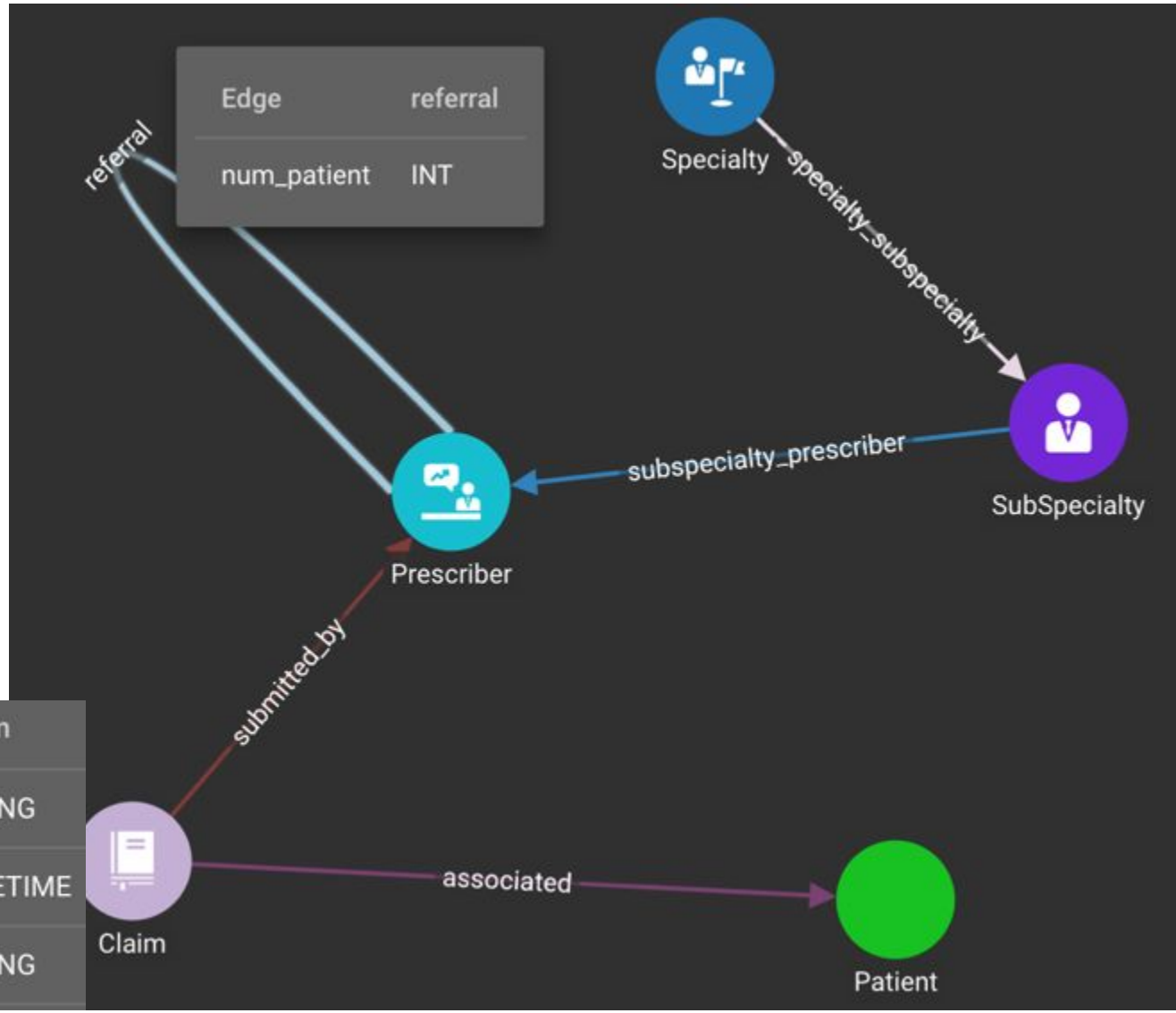
Huiting Su & Gaurav Deshpande (VP of Marketing)

Agenda

1. TigerGraph Introduction
2. Preparation
3. Schema Design and Data Ingestion
4. Simple Query Practice
5. Advanced Query Practice
6. K-hop Query and PageRank Query Practice
7. Graph Algorithms Overview

Understanding Property Graphs

- Vertex = Entities (e.g. claim, patient, phone call, payment, account..)
- Edge = Relationship among entities (e.g. claim to patient, phone call to customer, payment to account)
- Fields or attributes on vertices and edges



Vertex	Claim
(PRIMARY ID) Claim_id	STRING
rx_fill_date	DATETIME
ICD10Code	STRING

TigerGraph

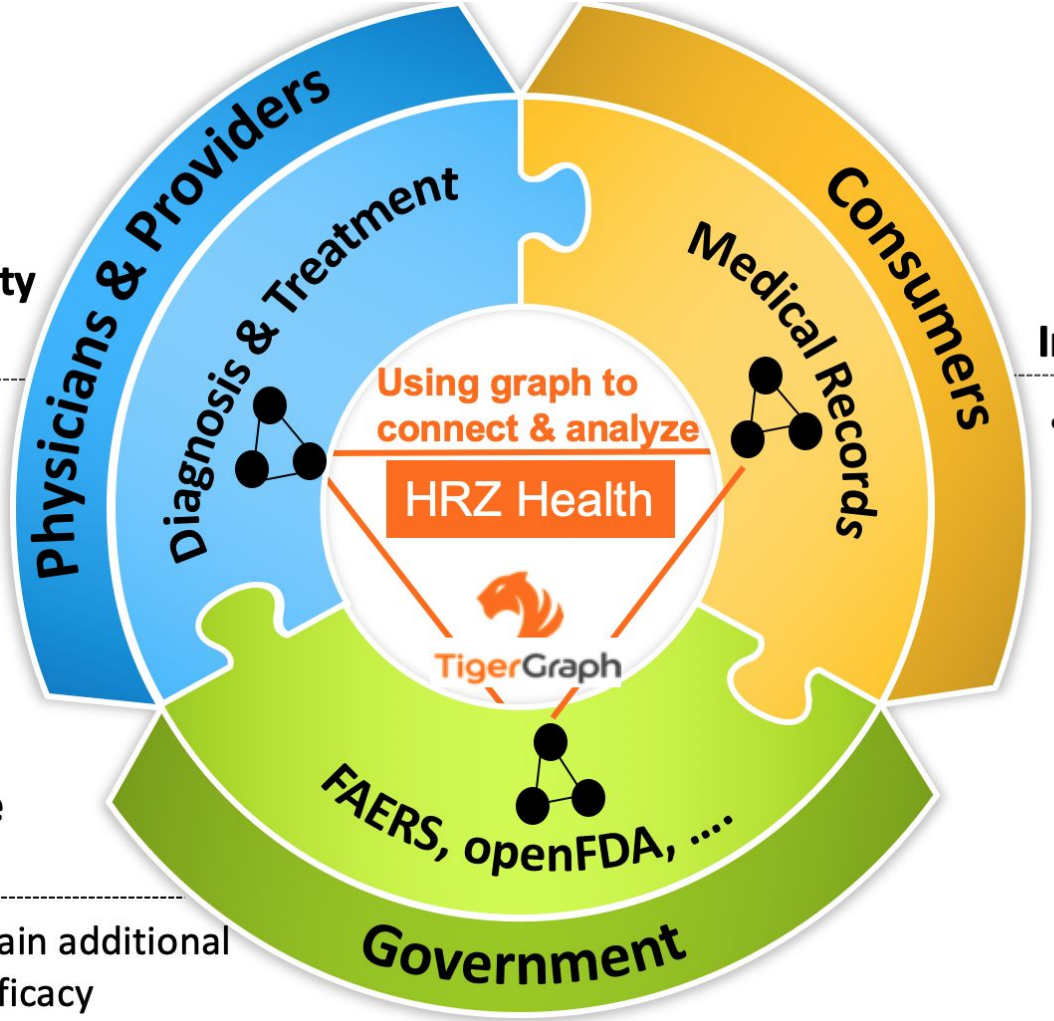
Training Dataset: HRZ Health

- HRZ Health is a diversified healthcare company operating in multiple markets including United States
- Business team at HRZ Healthcare Is interested in:
  - Improve net promoter score (NPS) among members (end consumers using HRZ healthcare services)
  - Deliver consistently high quality of care while controlling costs
  - Leverage public data (such as FAERS) to improve health outcomes for members

Connecting the dots across internal & external data with TigerGraph

Deliver consistently high quality of care while controlling costs

- Detect and prevent waste and abuse
- Understand and optimize referral networks to lower cost of care



Improve Net Promoter Score

- Make better use of the Member 360 data to deliver superior care

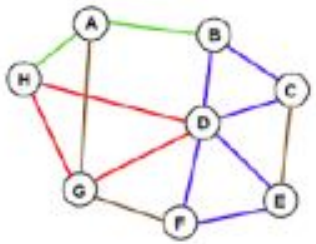
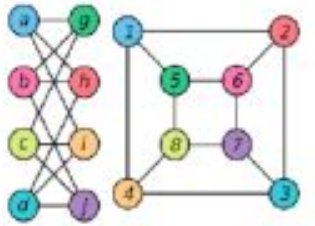
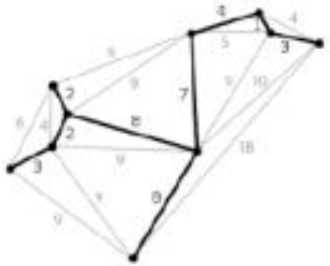
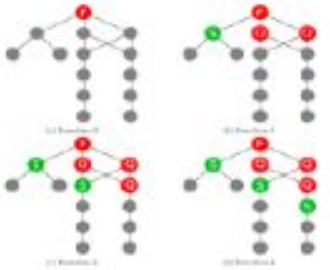
Leverage public data to improve healthcare outcomes

- Link with internal claims data to gain additional insights on drug and treatment efficacy

TigerGraph

Using Graph Analytics Patterns to Power Use Cases

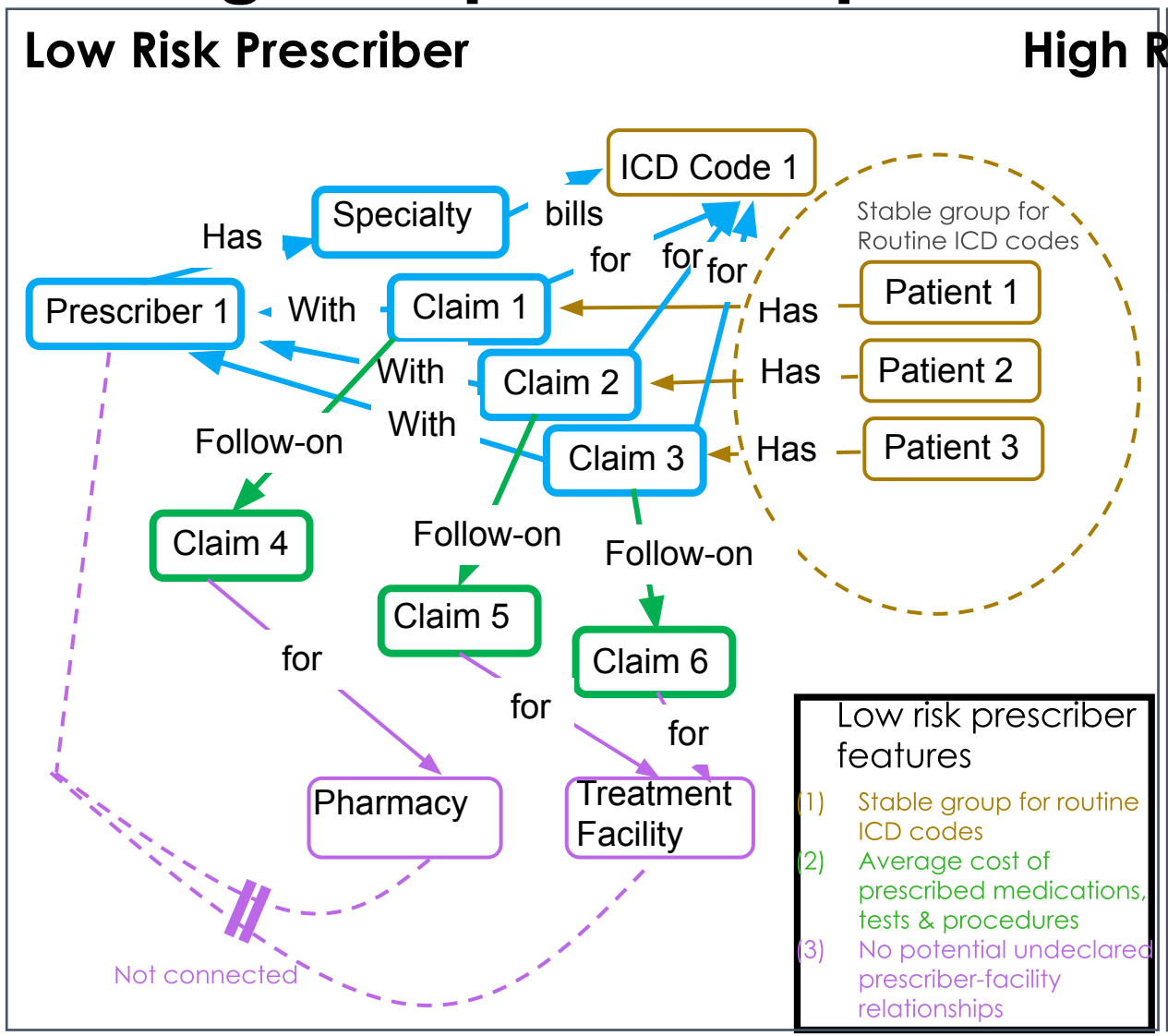
Graph Analytics Patterns

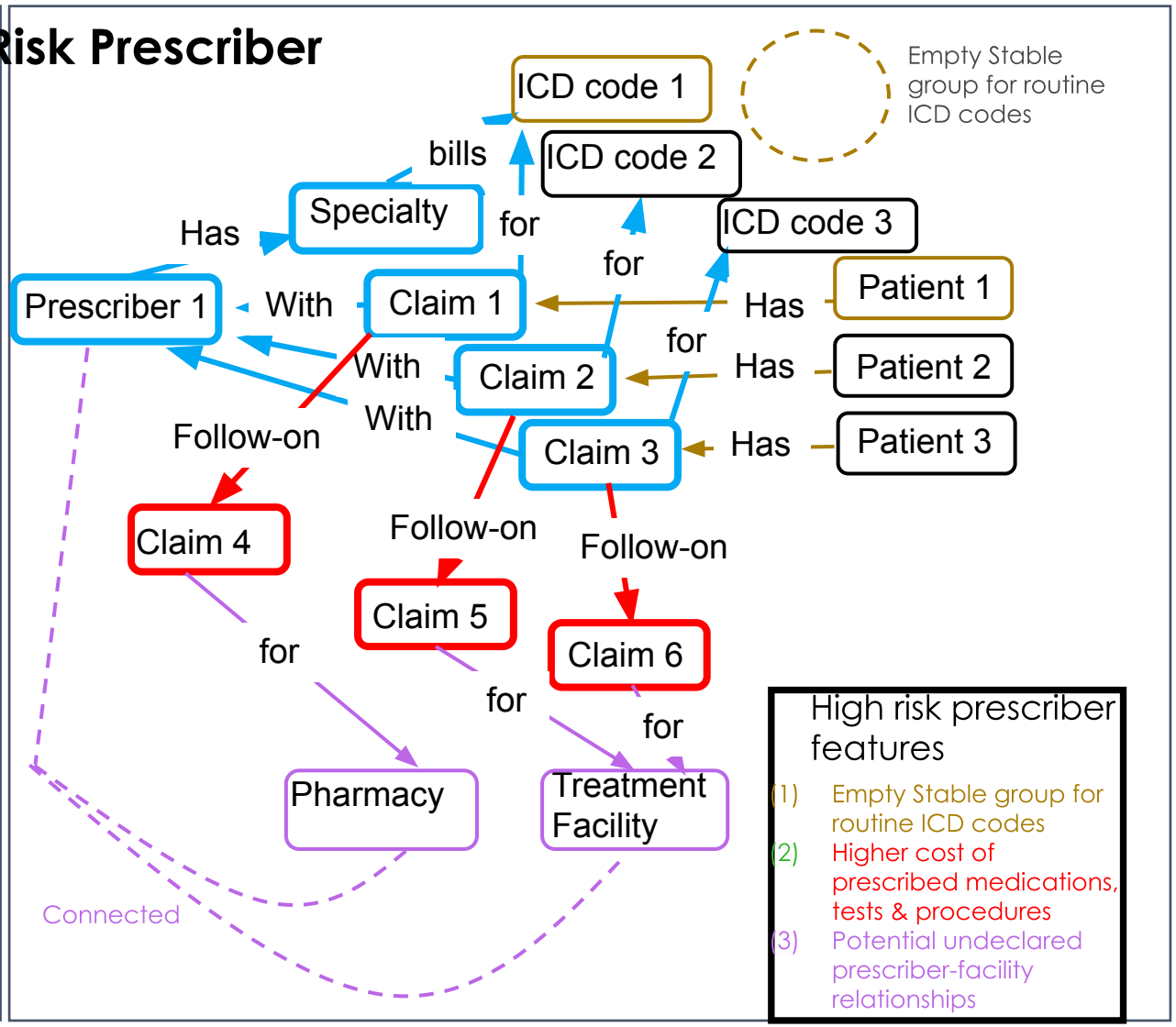
Use Case	Link analysis	Graph matching	Frequent subgraph discovery	Community detection
	Given an entity, show its connections 	Given a graph, find similar graph patterns 	Given a graph, find its relevant subgraphs 	Describe the evolution of a graph 
Detect and prevent waste and abuse	Uncover Prescriber-Facility Collusion	Discover potential fraudulent patterns	Investigate specific prescriber-patient-facility networks	Identify potential fraud networks
Understand and optimize referral networks to lower cost of care	Understand how prescribers are connected	Prescriber & Member Likeness & Benchmarking	Find and visualize Prescriber Referral Relationships	Identify influential prescribers & linked communities
Link with internal claims data to gain insights on drug & treatment efficacy	✓	✓		✓
Make better use of the Member 360 data to deliver superior care	✓	✓	✓	✓

Focus for today's hands-on training

TigerGraph

Using Computed Graph Features To Find Bad Actors





TigerGraph

Use Case - Detect and prevent waste and abuse

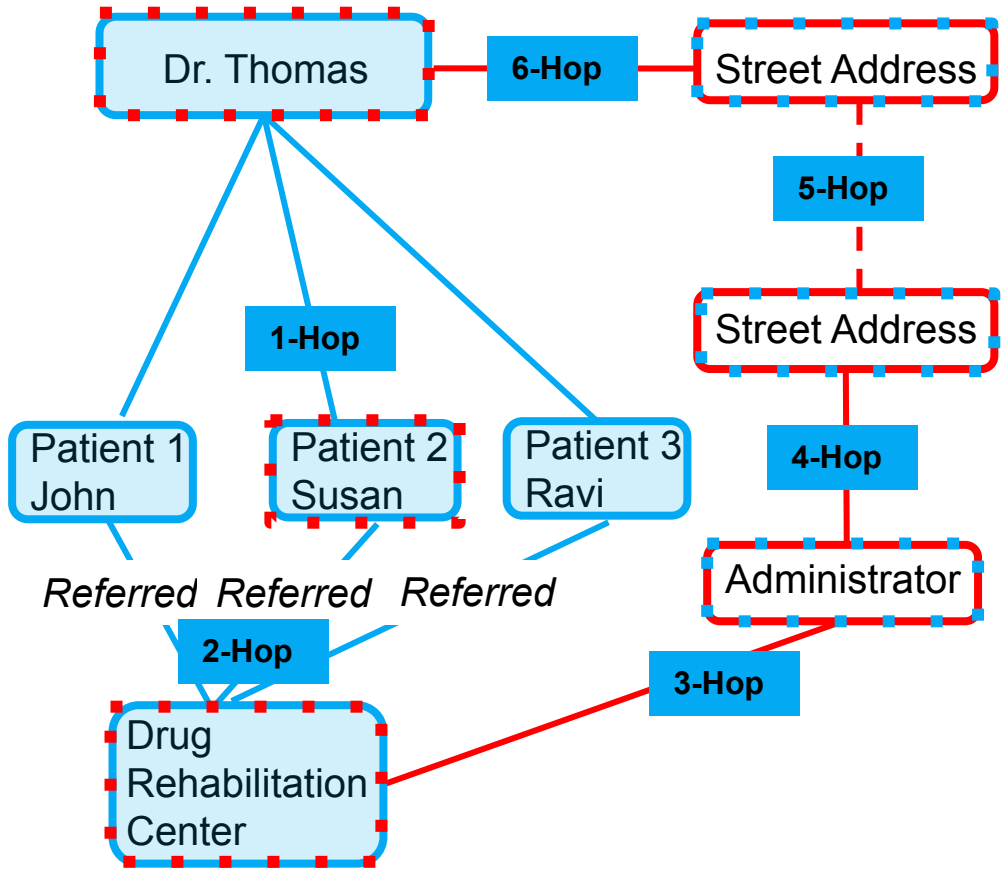
Business Need: Bad-actor providers refer patients to facilities for unprofessional reasons.

6-Hop Query for graph analytics pattern:
Uncover Prescriber–Facility Collusion

Solution:

- **Look for alternative connections** between prescriber/physician & referred facilities
- **Assign risk rating** based on amount of referral & strength of alternative connections
- **Compare billing across facilities** to identify "upcoding" / overbilling
- **Improve efficiency for case investigation** for potential violations with intuitive user interface

Business Value: Reduce improper referrals, improve quality of care for members



TigerGraph

Use Case – Understand & optimize referral networks to lower cost of care

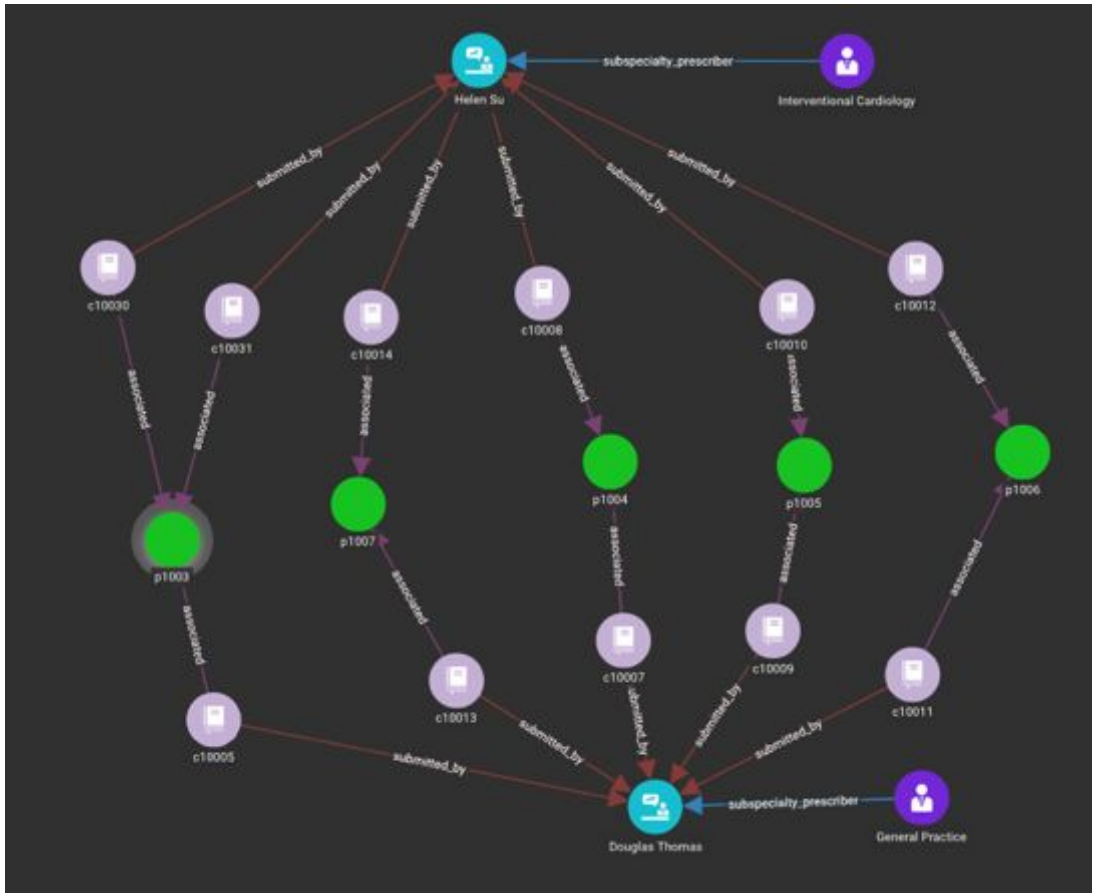
Business Need: Understand and analyze prescriber relationships that support member's healthcare needs

Solution:

- **Understand how prescribers are connected** in terms of linked claims, common patients and treatment facilities
- **Find and visualize Prescriber Referral Relationships** for specific area and/or healthcare condition
- **Find similar prescribers and patients/members** and benchmark cost of care across end-to-end patient care journey
- **Identify influential prescribers & linked communities** to optimize care deliver

Business Value: Lower cost of care while maintaining quality of care

4-Hop Query to find graph analytics pattern:
Understand how prescribers are connected



Focus for today's hands-on training

TigerGraph

Use Case – Understand & optimize referral networks to lower cost of care

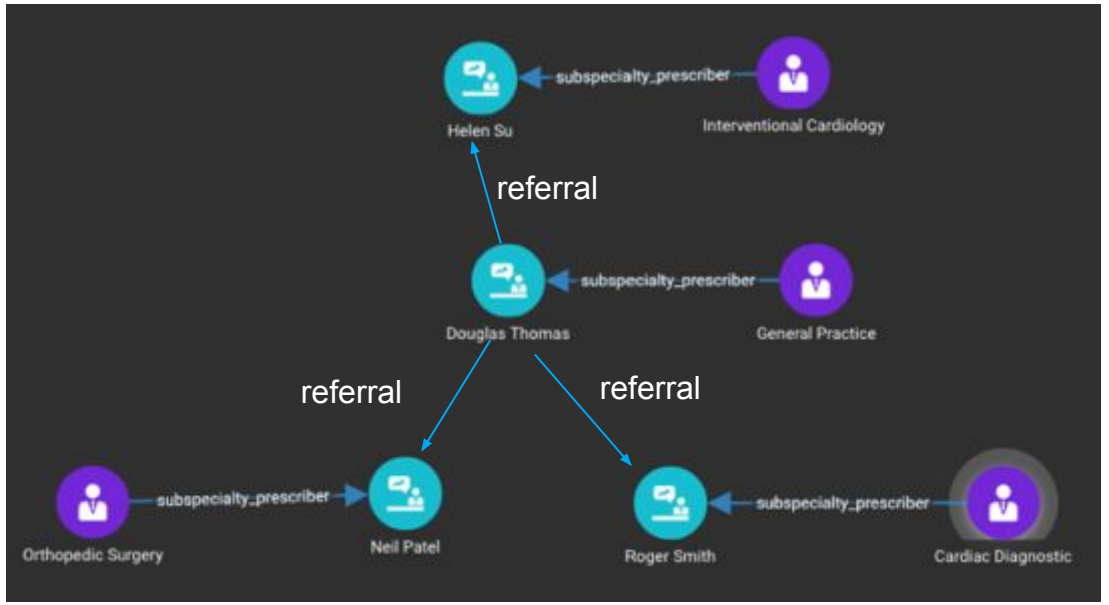
Business Need: Understand and analyze prescriber relationships that support member's healthcare needs

Solution:

- **Understand how prescribers are connected** in terms of linked claims, common patients and treatment facilities
- **Find and visualize Prescriber Referral Relationships** for specific area and/or healthcare condition
- **Find similar prescribers and patients/members** and benchmark cost of care across end-to-end patient care journey
- **Identify influential prescribers & linked communities** to optimize care deliver

Business Value: Lower cost of care while maintaining quality of care

4-Hop Query to find graph analytics pattern:
Find and visualize prescriber referral relationships



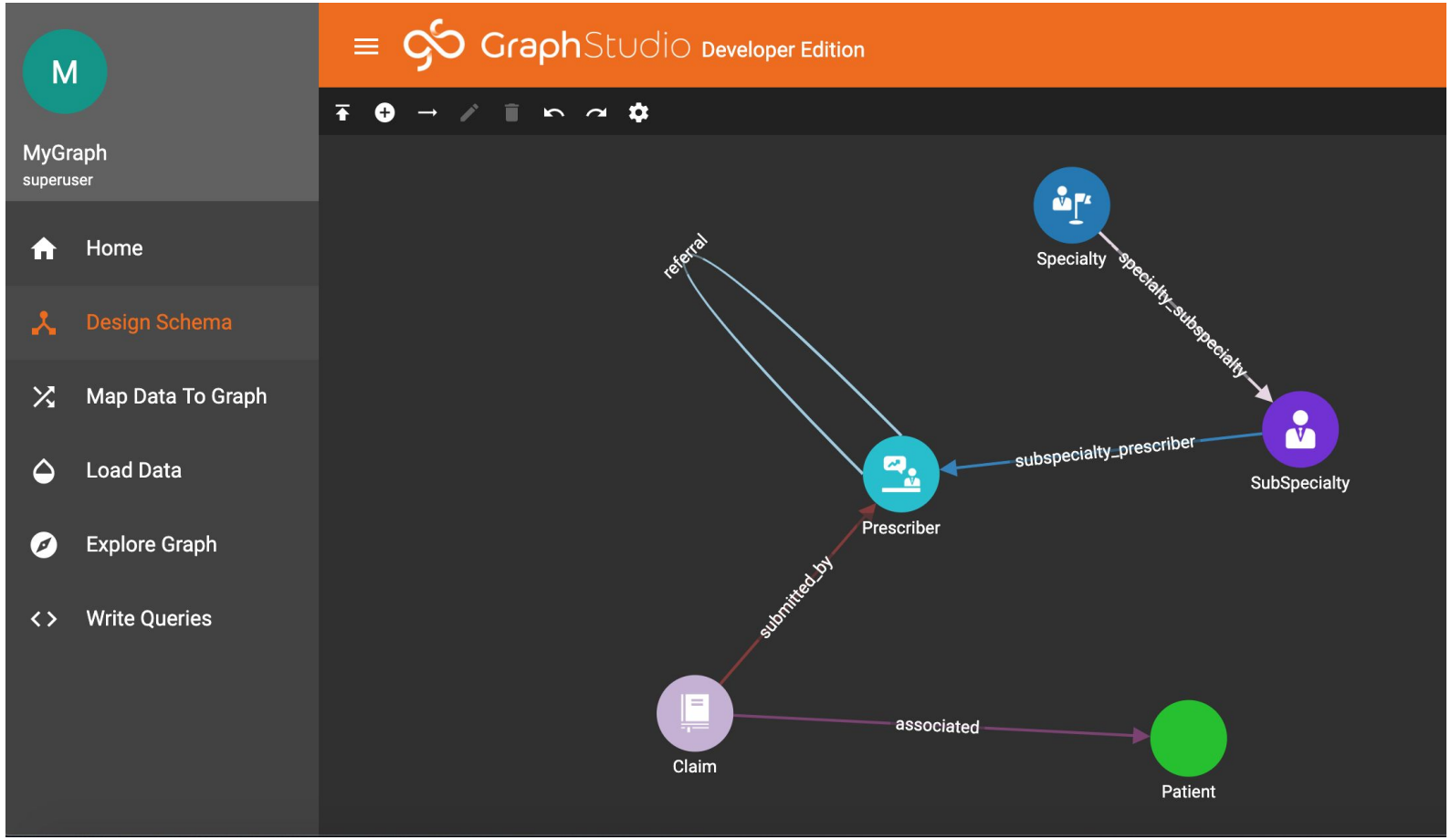
Focus for today's hands-on training

TigerGraph

Hands-on Training Exercises

- Exercise 1 – Create data model, map data and load the data for HRZ Health Graph
- Exercise 2 – Execute graph analytics pattern **"Understand how prescribers are connected"** for use case "Understand & optimize referral networks to lower cost of care" in Graph Studio and with GSQL query
- Exercise 3 – Execute graph analytics pattern **"Find and visualize prescriber referral relationships"** for use case "Understand & optimize referral networks to lower cost of care"
  - Simple referral edge creation based on common patients
- For over achievers –
  - Limit the referral edge creation based on 2 week time window (create referral relationship only if the time between claims is less than 2 weeks)
  - Tailor referral edge to remember the number of times a patient is referred by one Doctor to another

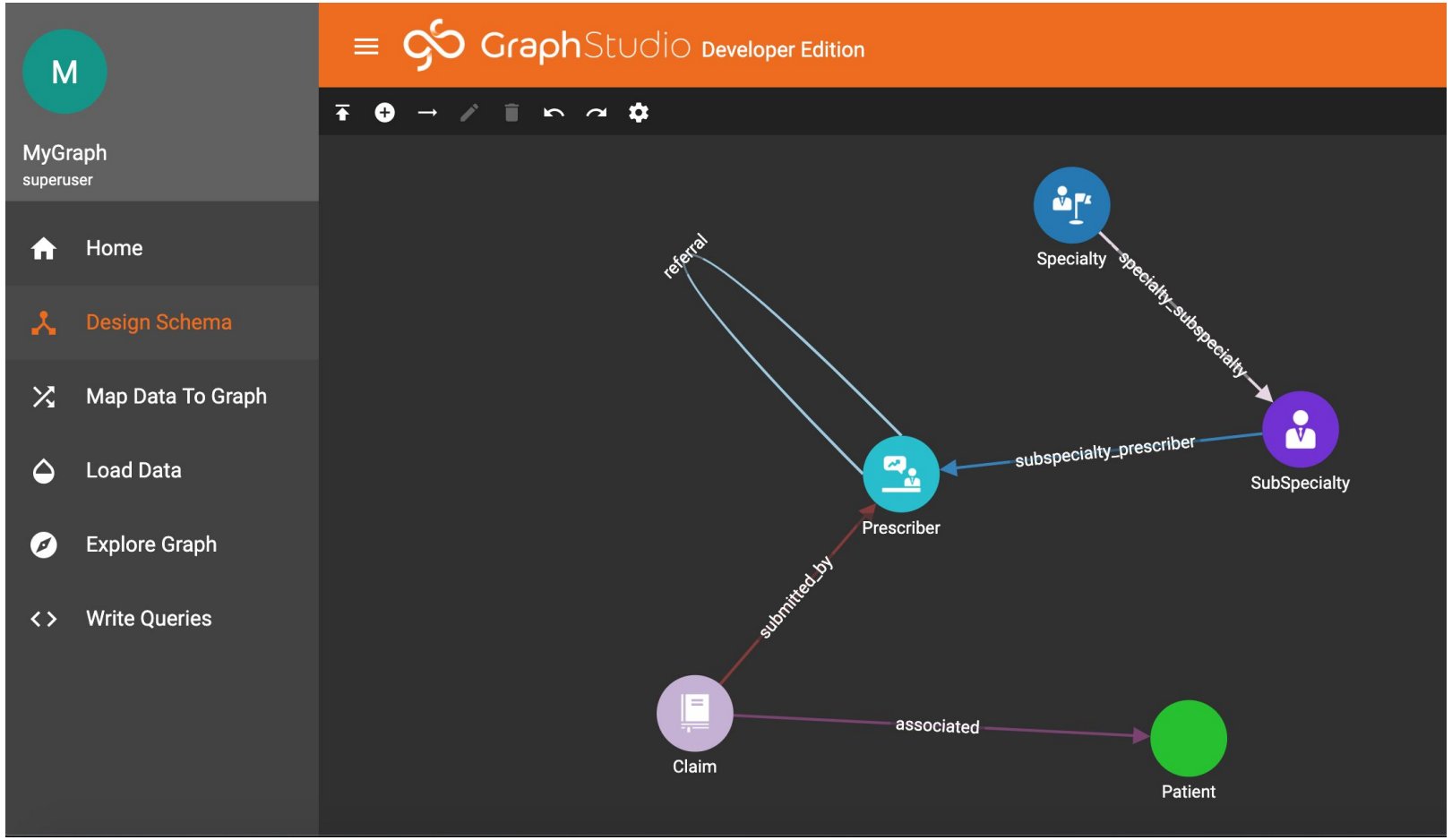
Exercise 1 – Building and Loading HRZ Health Graph in TigerGraph GraphStudio



TigerGraph GraphStudio overview videos located at - https://www.youtube.com/watch?v=29PCZEhyx8M&list=PLq4I3NnrSRp7RfZqrtsjevDjpSV8IHhe-

TigerGraph

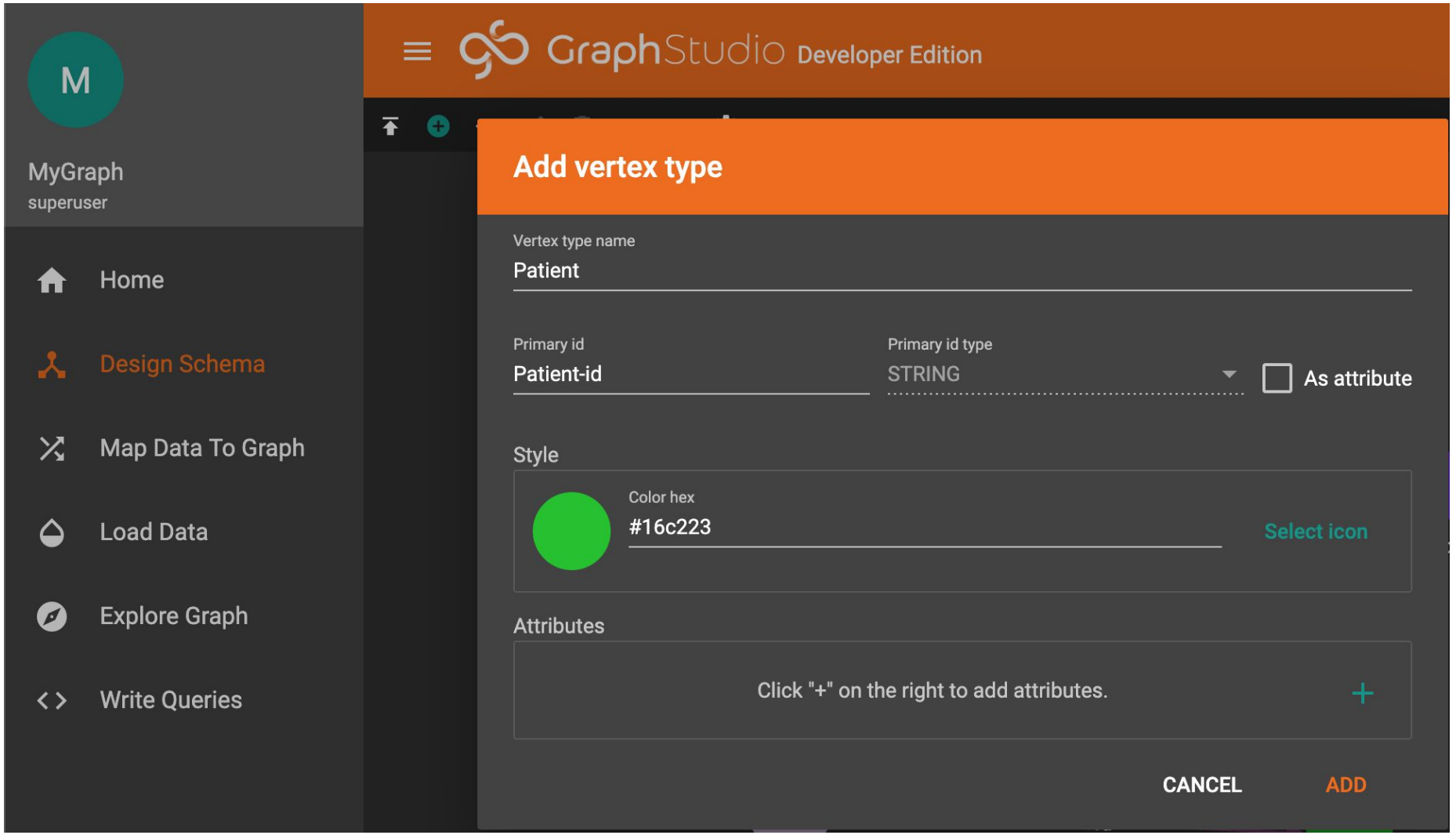
Exercise 1 – Step 1: Create schema for the HRZ Health Graph



Creating Schema video on TigerGraph channel - https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

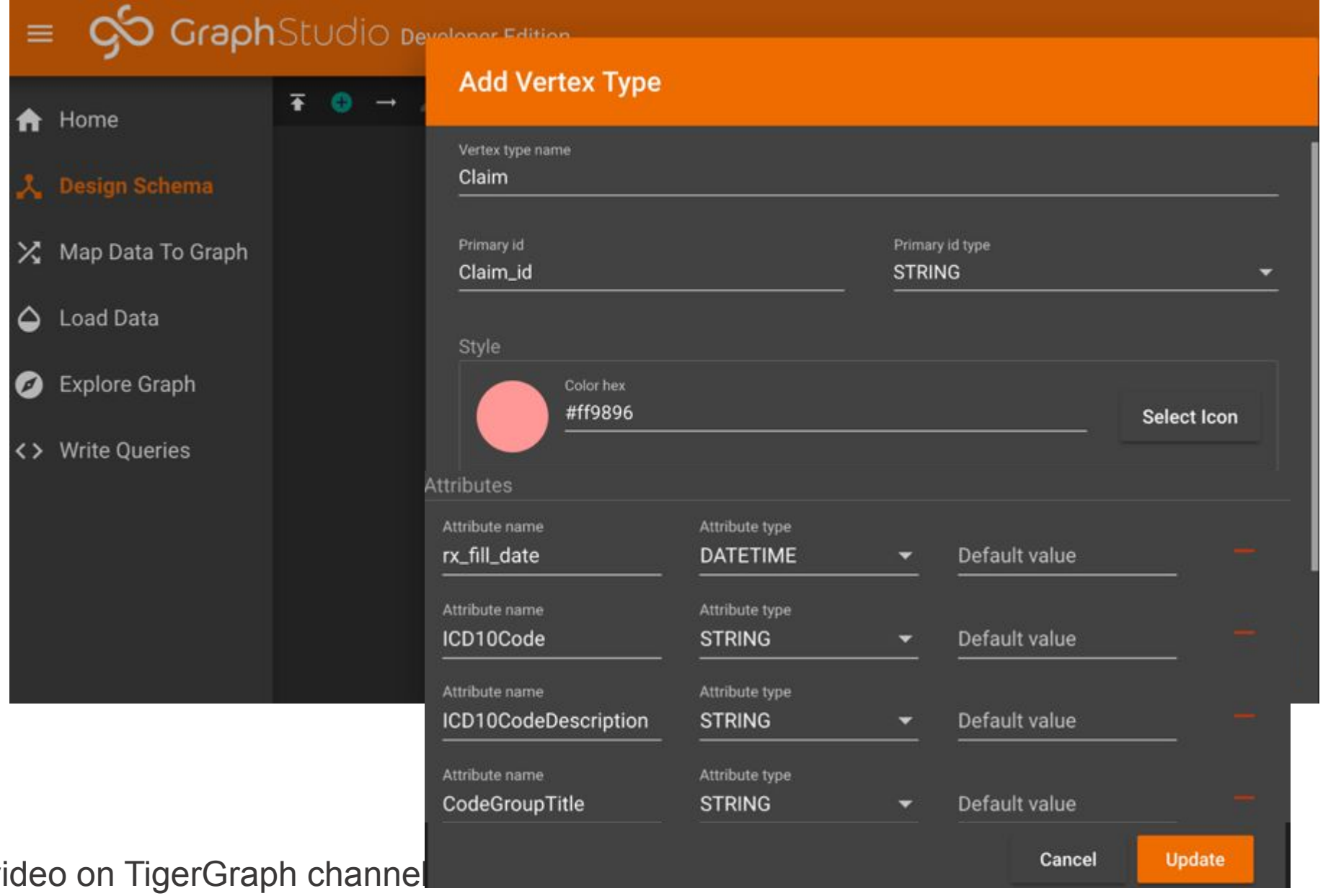
Exercise 1 – Step 1A: Create Patient Vertex



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1B: Create Claim Vertex

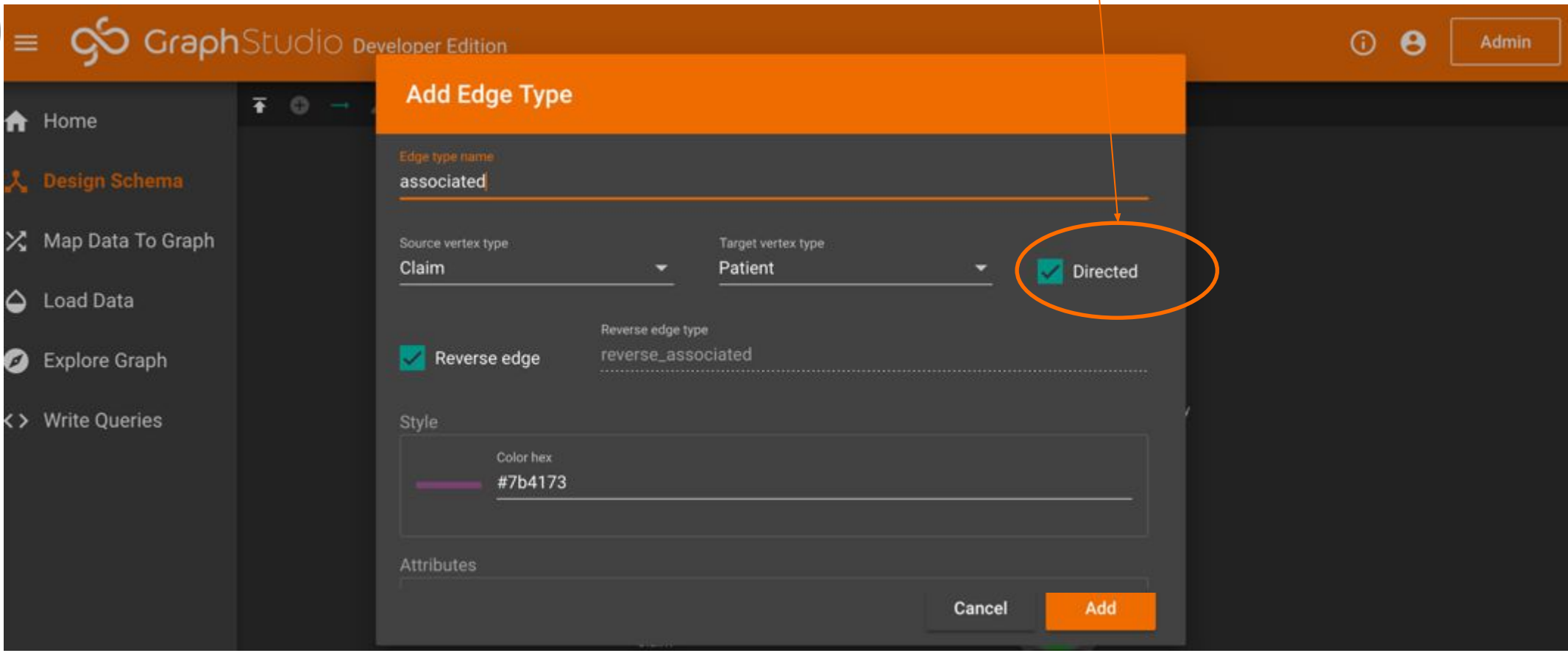


Creating Schema video on TigerGraph channel
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1C: Create Claim to Patient associated edge

Be sure to select this as it will allow you to traverse the graph from patient back to claim

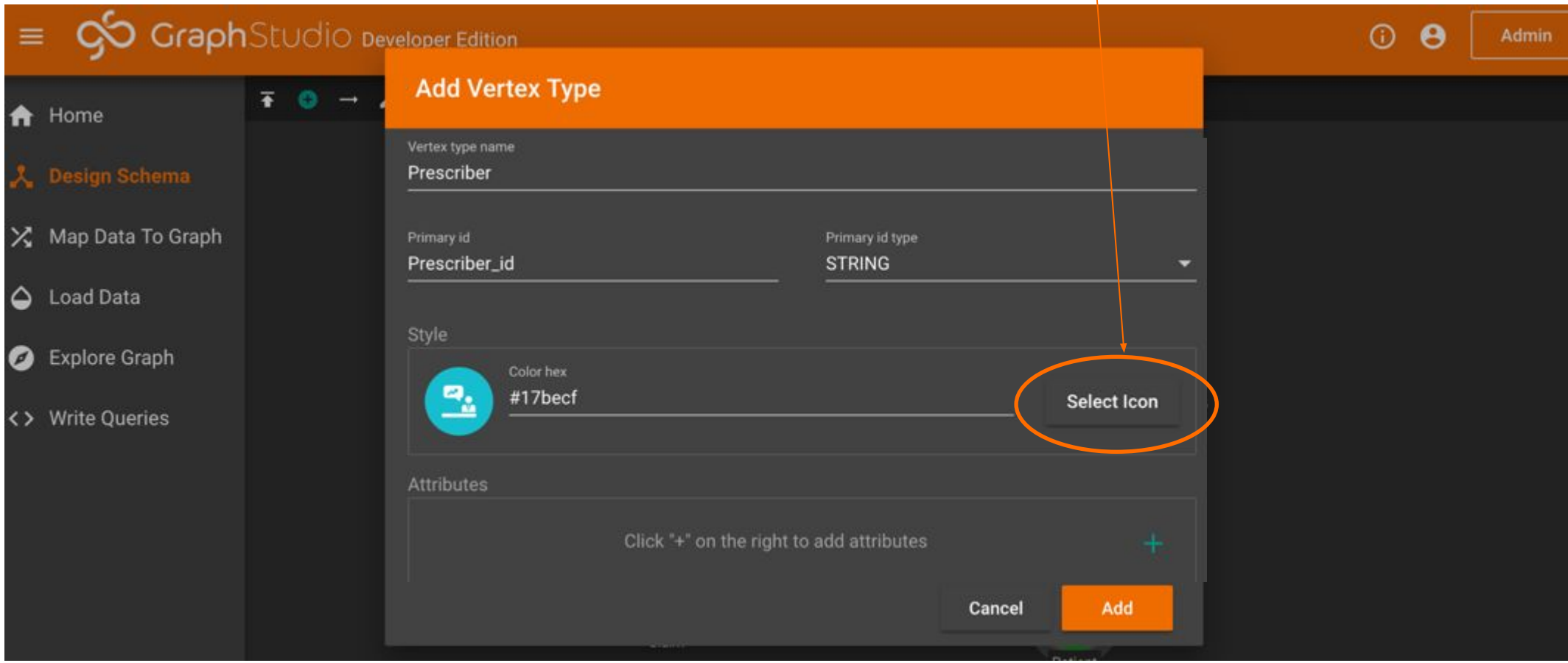


Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1D: Create Prescriber Vertex

Use Select Icon to pick an icon for the node

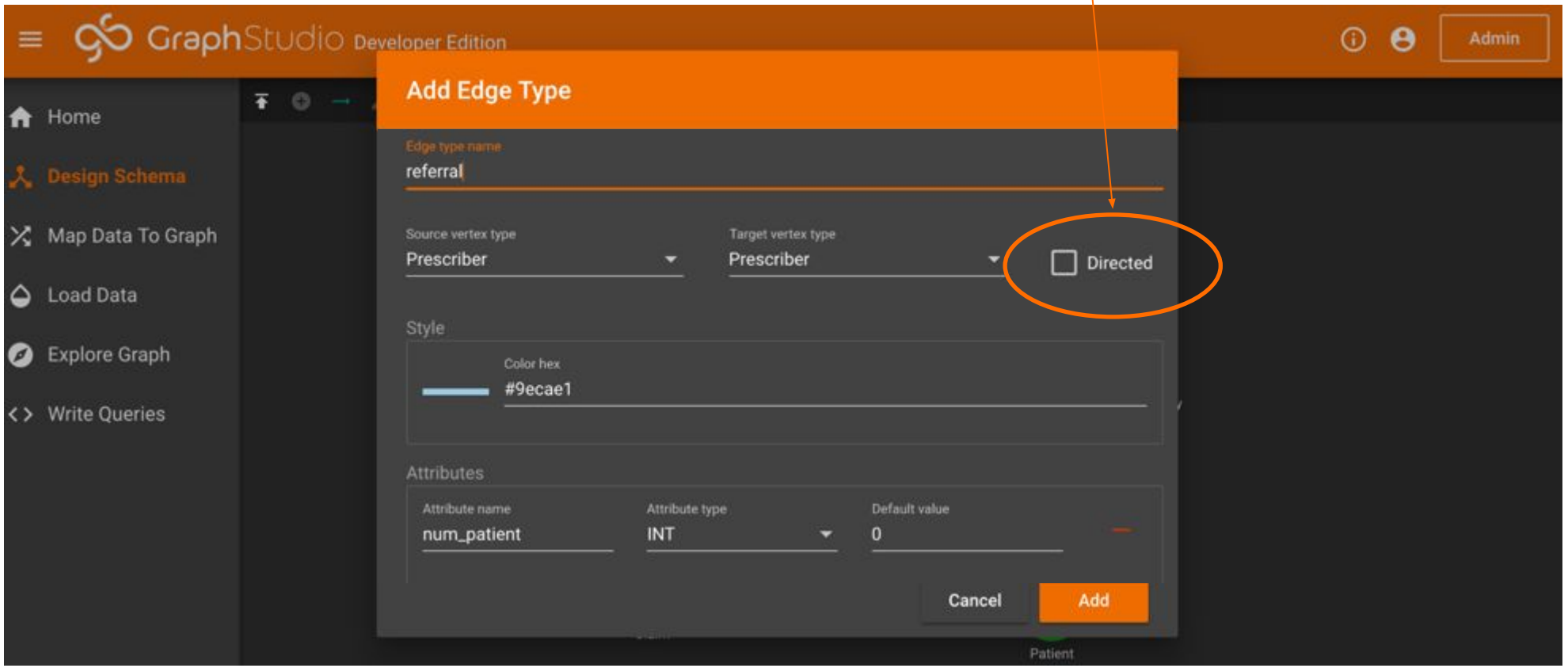


Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1E: Create Referral edge

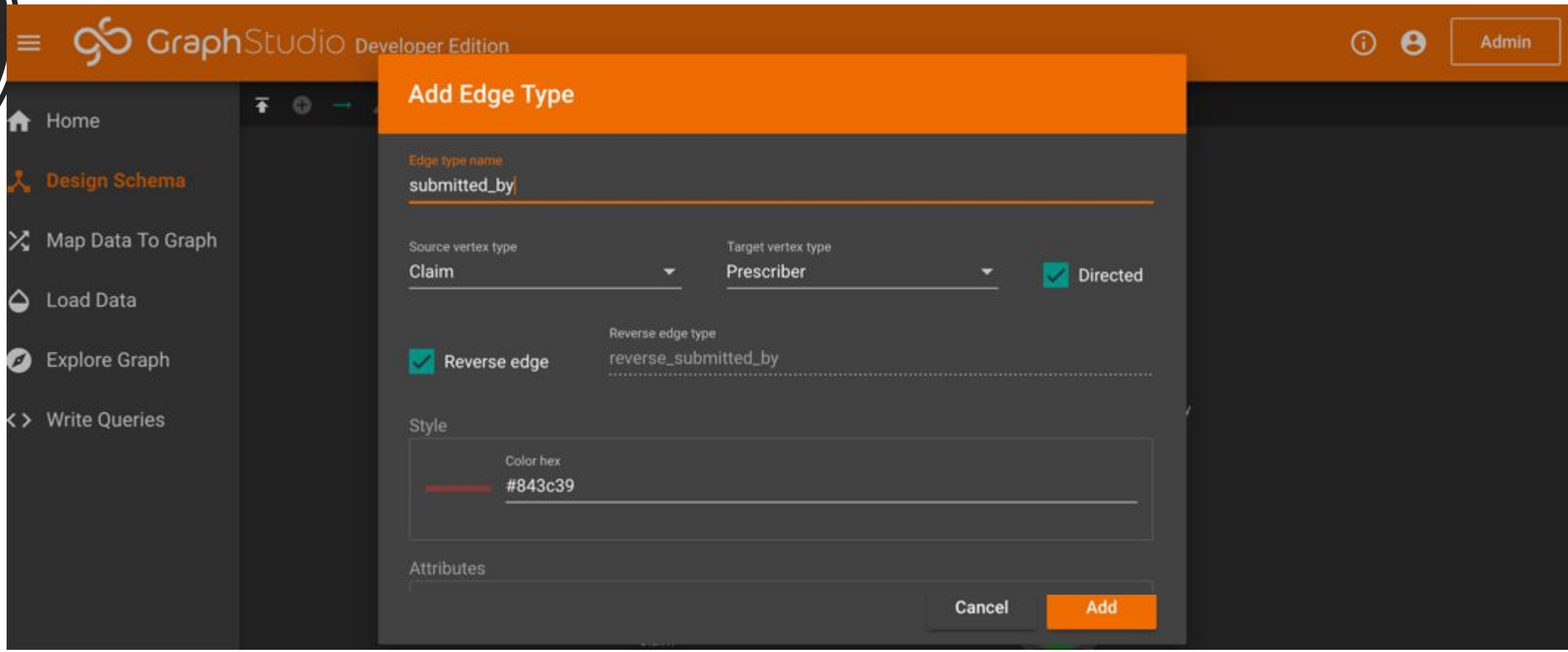
We are keeping this edge “Undirected” for the sake of simplicity – It will be directed in real customer deployments



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

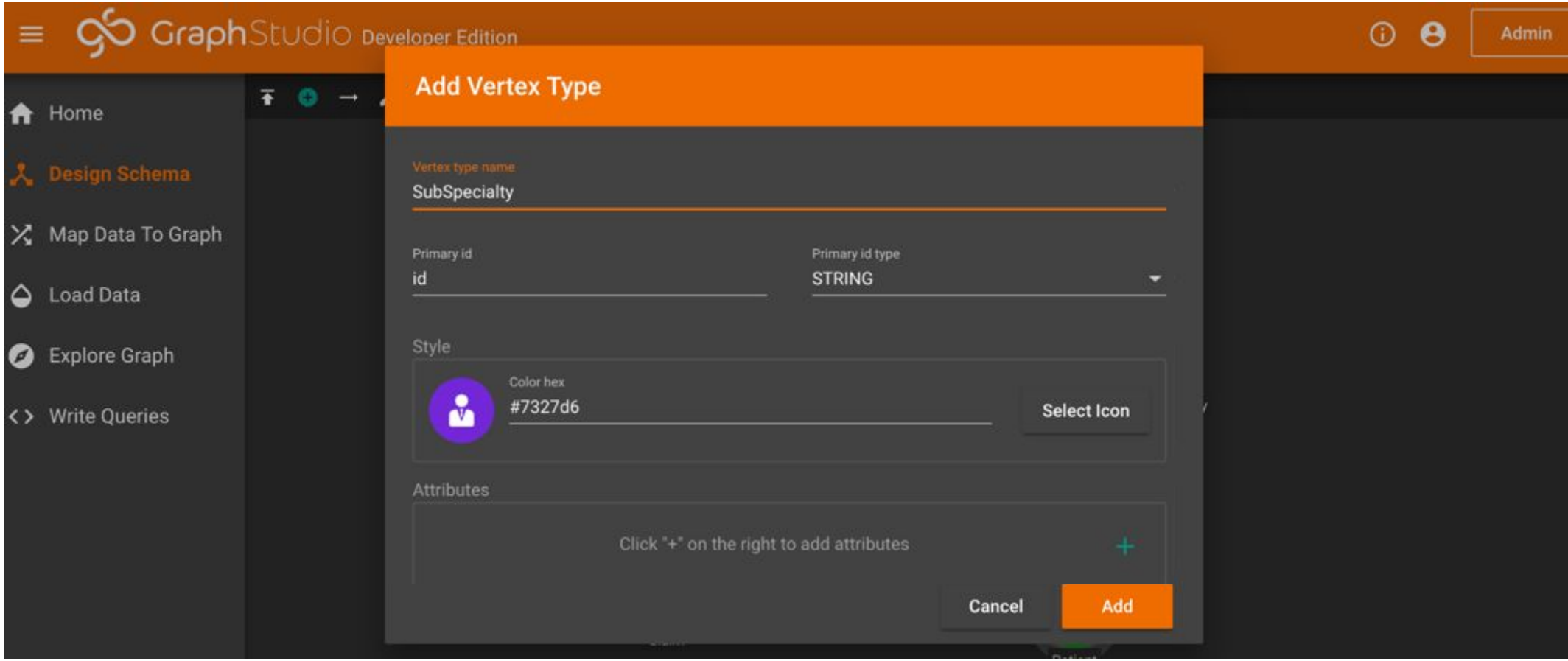
Exercise 1 – Step 1F: Create Claim to Prescriber submitted_by edge



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

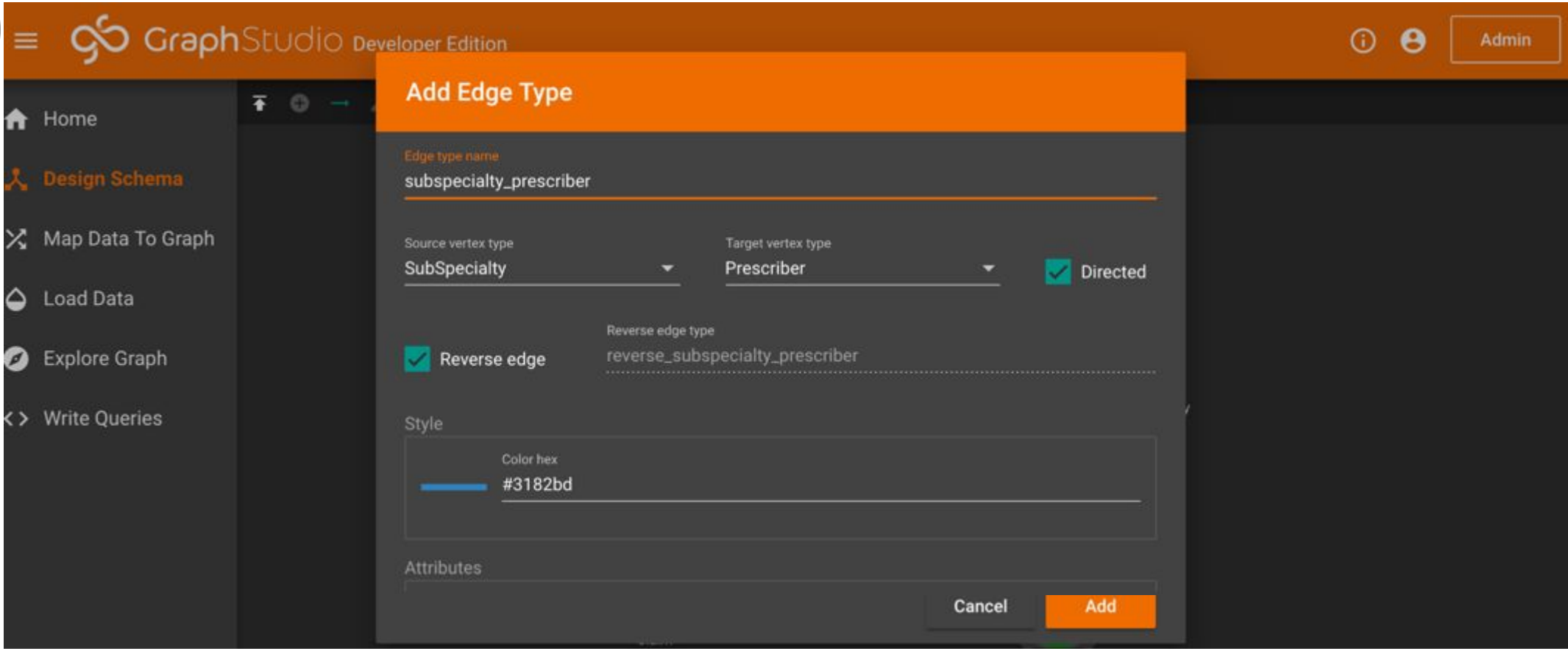
Exercise 1 – Step 1G: Create Subspecialty Vertex



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

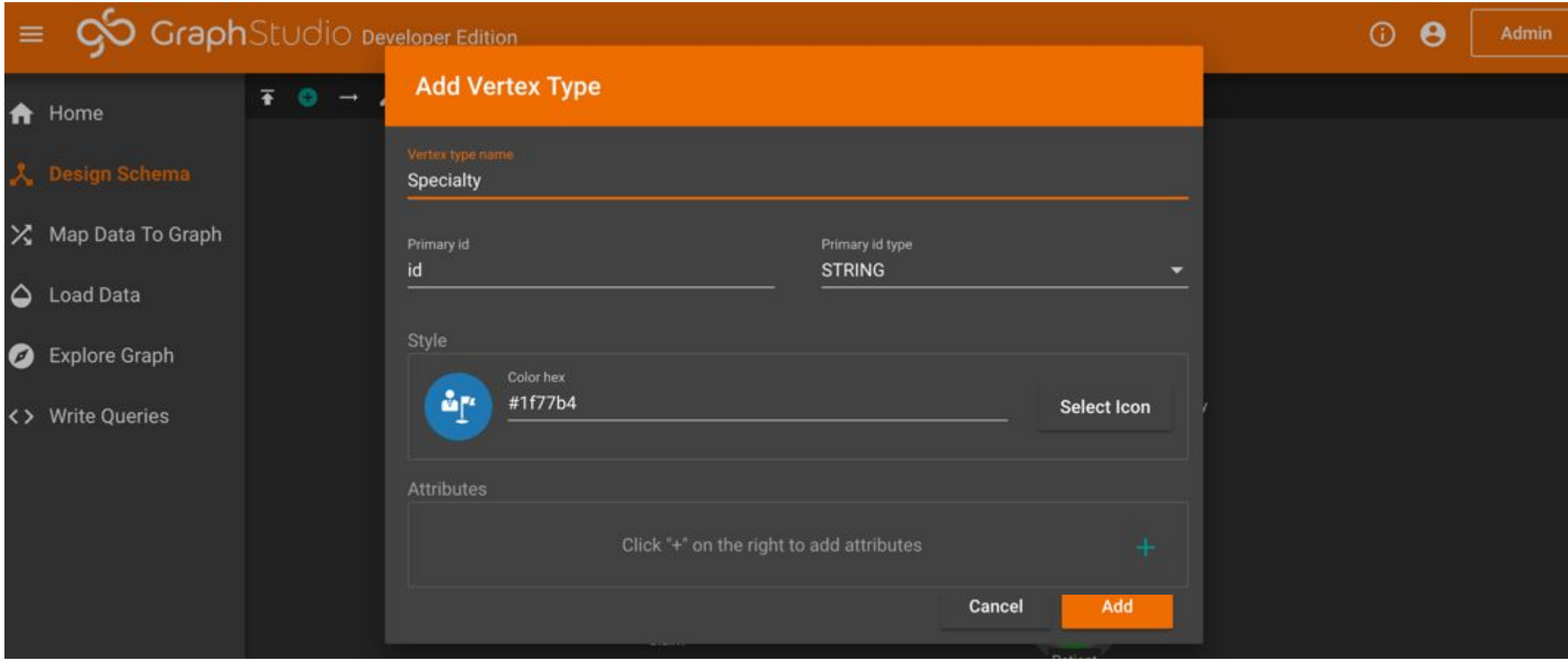
Exercise 1 – Step 1H: Create subspecialty to prescriber edge



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

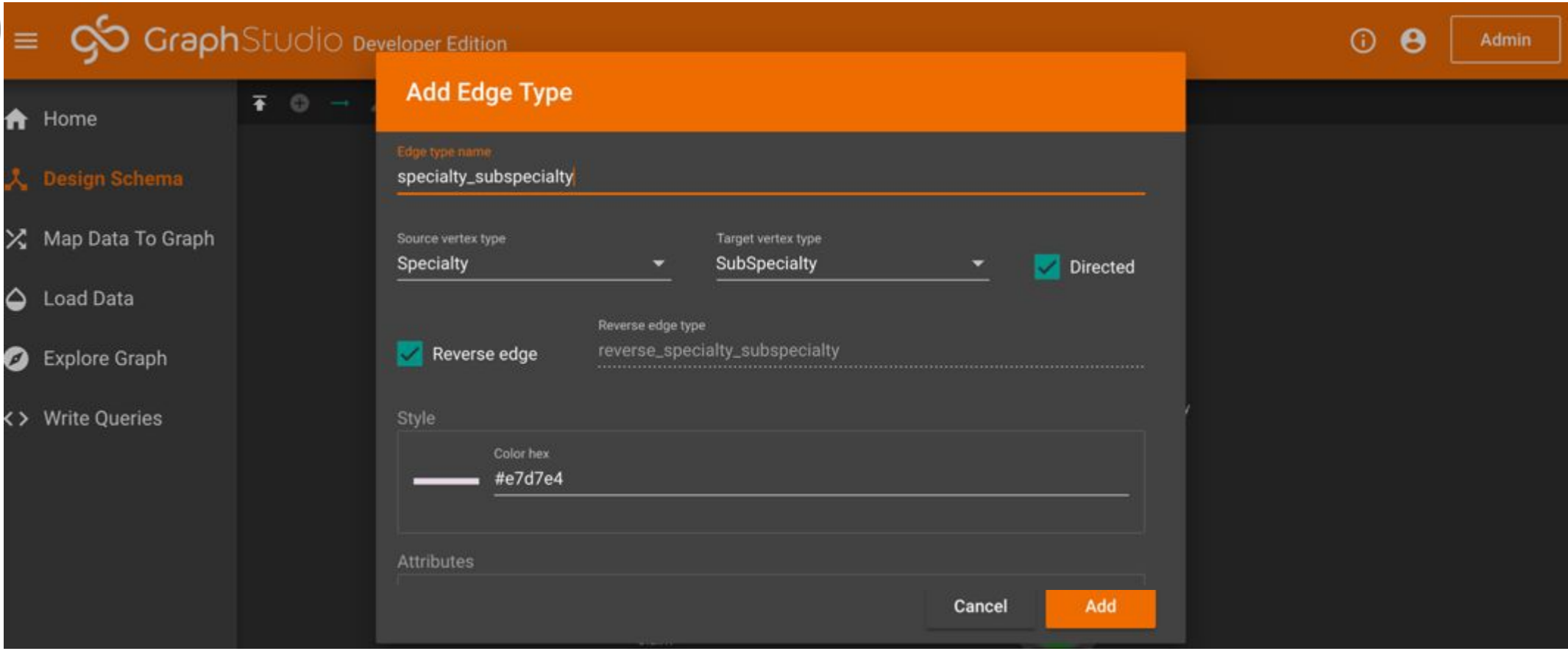
Exercise 1 – Step 1 I: Create Specialty Vertex



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1J: Create Specialty to Subspecialty edge

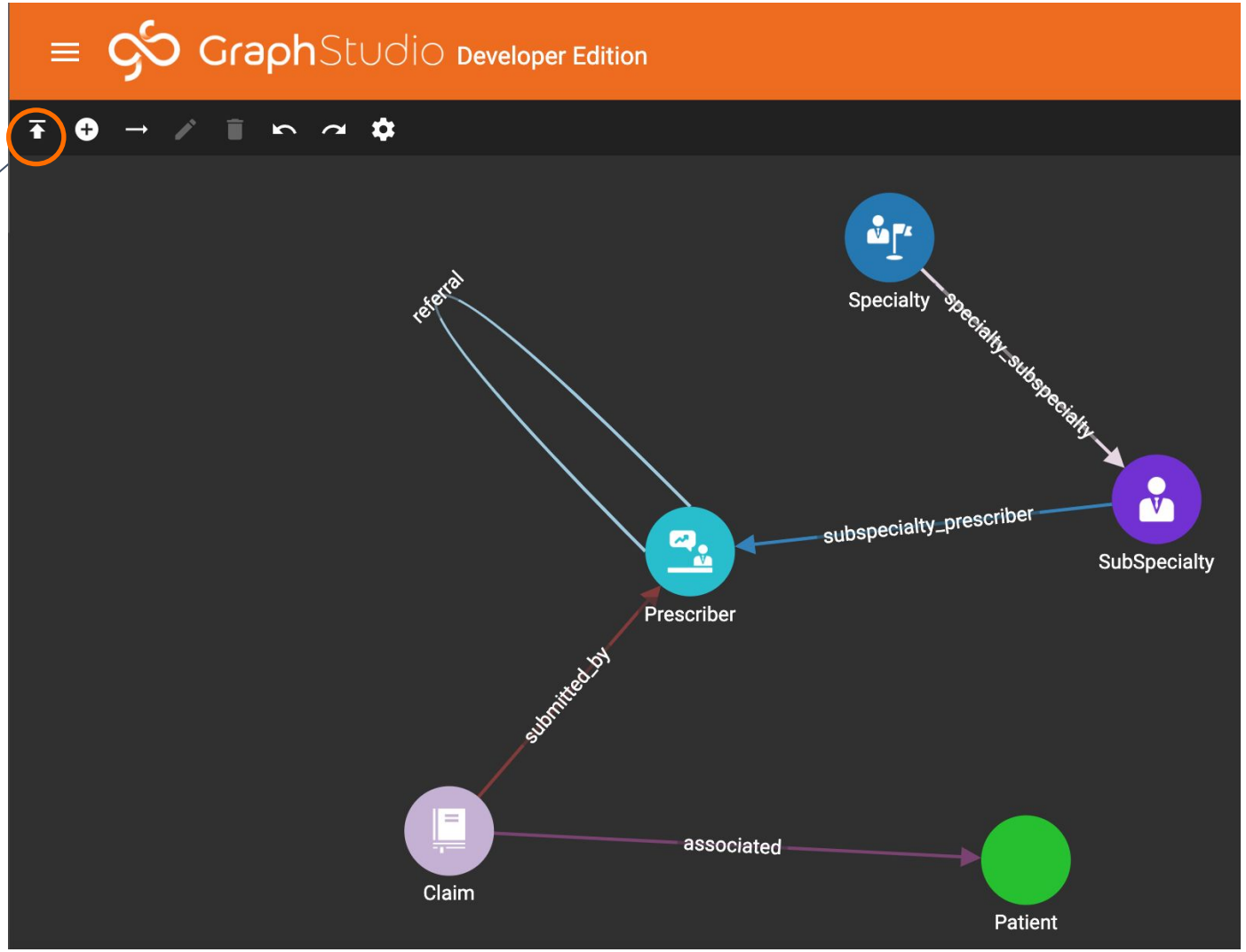


Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

Exercise 1 – Step 1K: Save and Publish Schema

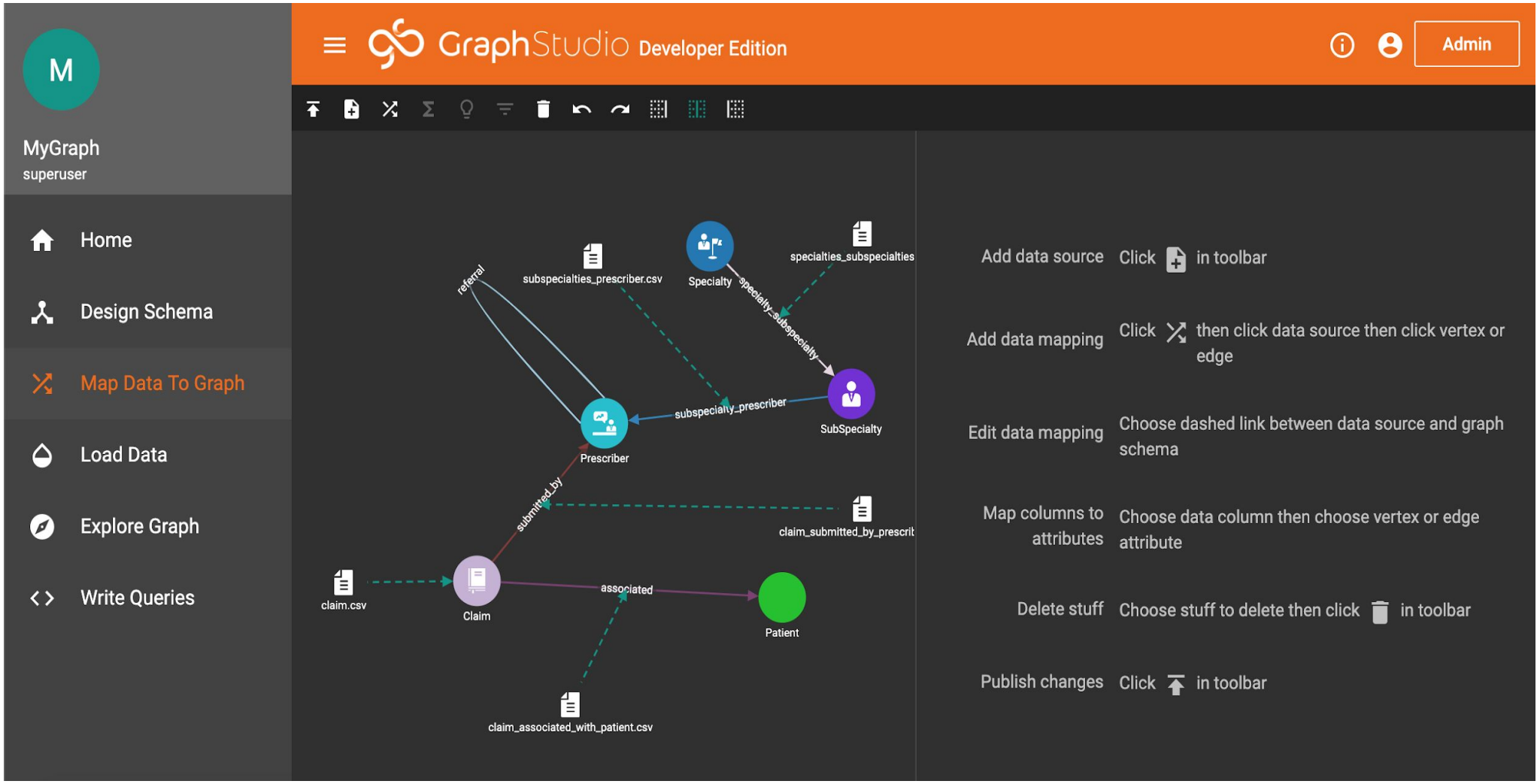
Publish the design schema



Creating Schema video on TigerGraph channel:
https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

TigerGraph

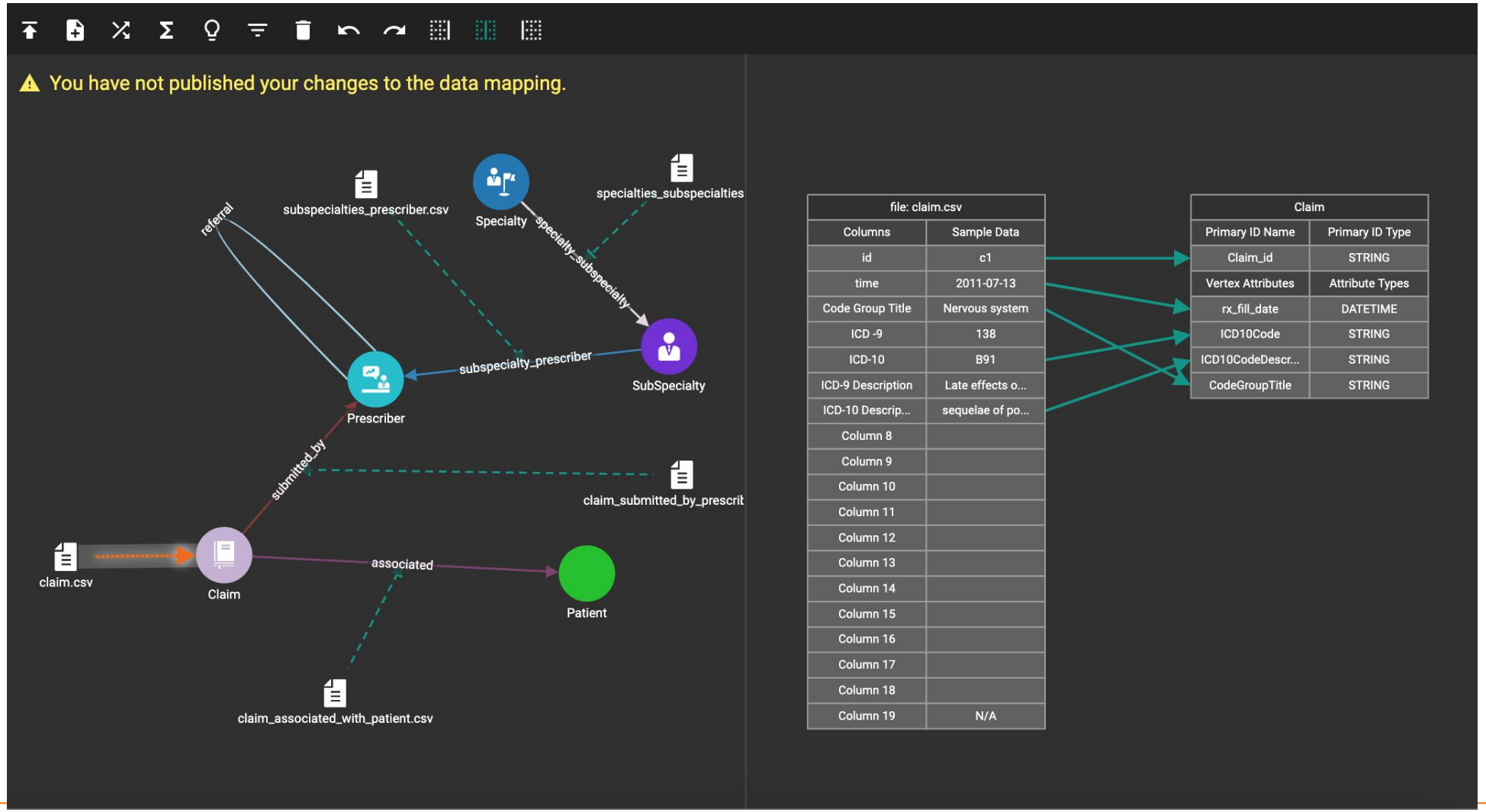
Exercise 1 – Step 2: Map Data to Graph



Creating Schema video on TigerGraph channel - https://www.youtube.com/watch?v=Q0JUkiU0lbs&index=9&list=PLq4l3NnrSRp7RfZqrtsievDjpSV8lHhe-

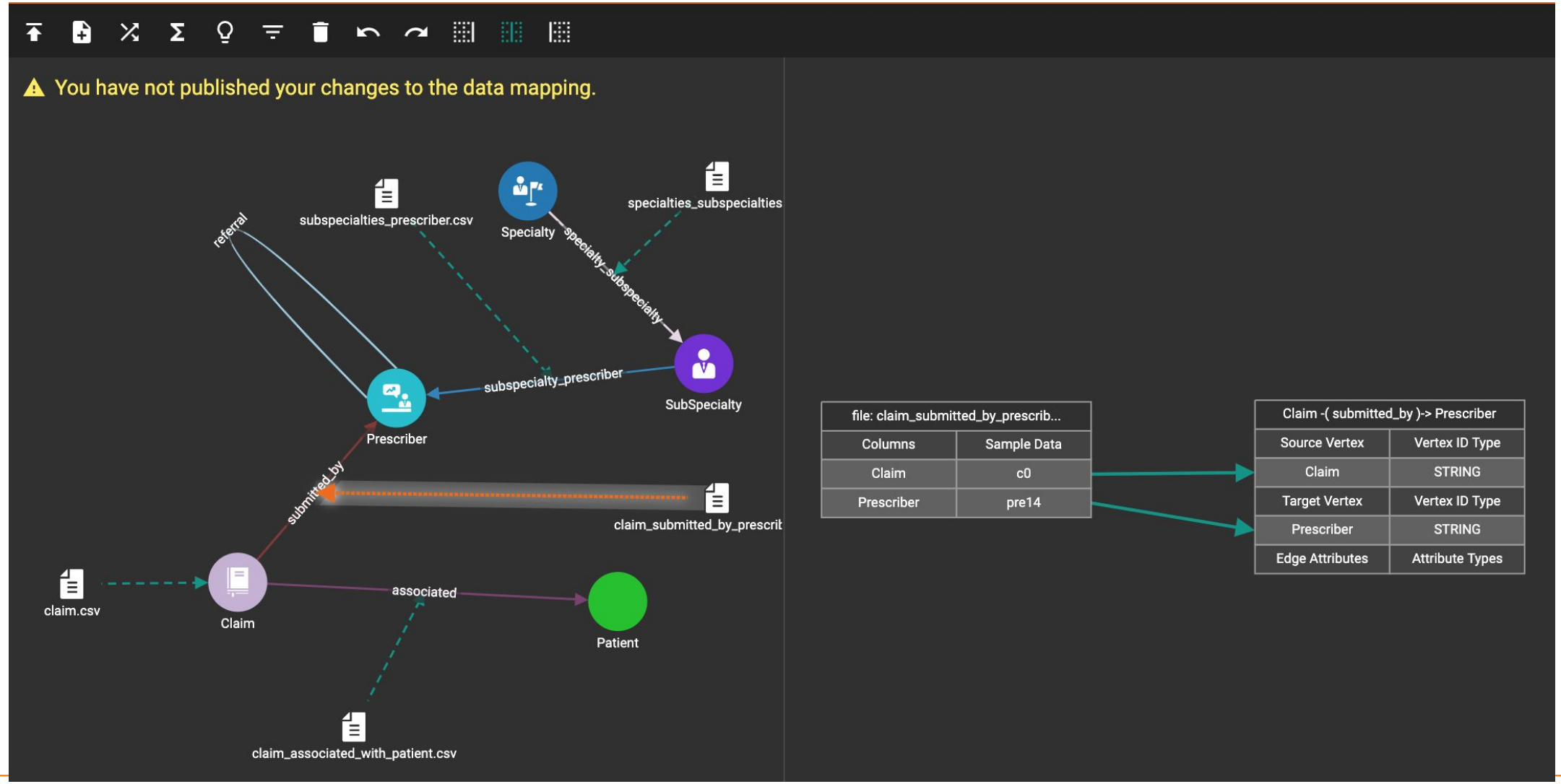
TigerGraph

Data Files and Mappings – claim.CSV □ Claim node



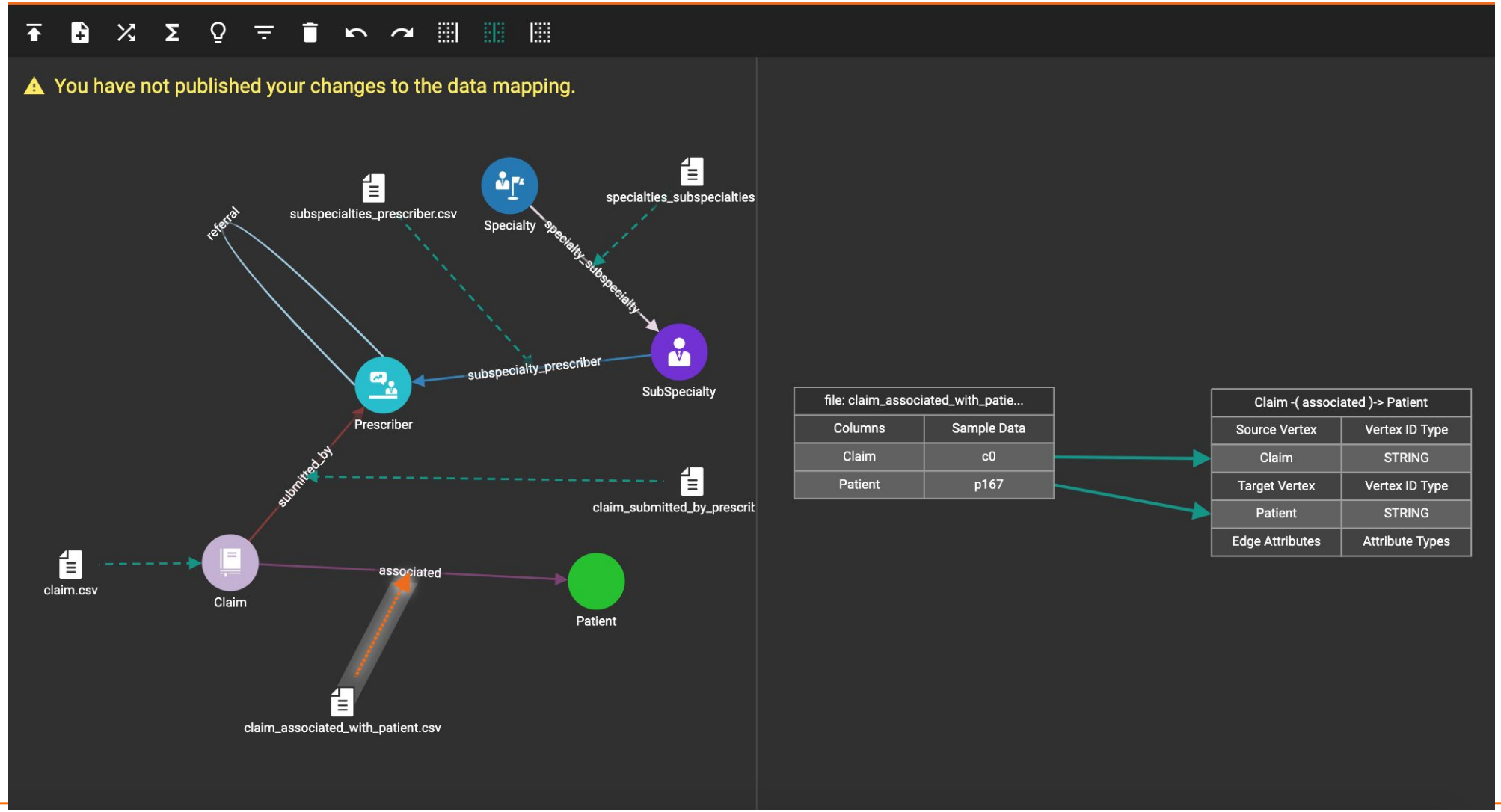
TigerGraph

Data Files and Mappings – claim submitted_by prescriber.CSV □ submitted_by edge



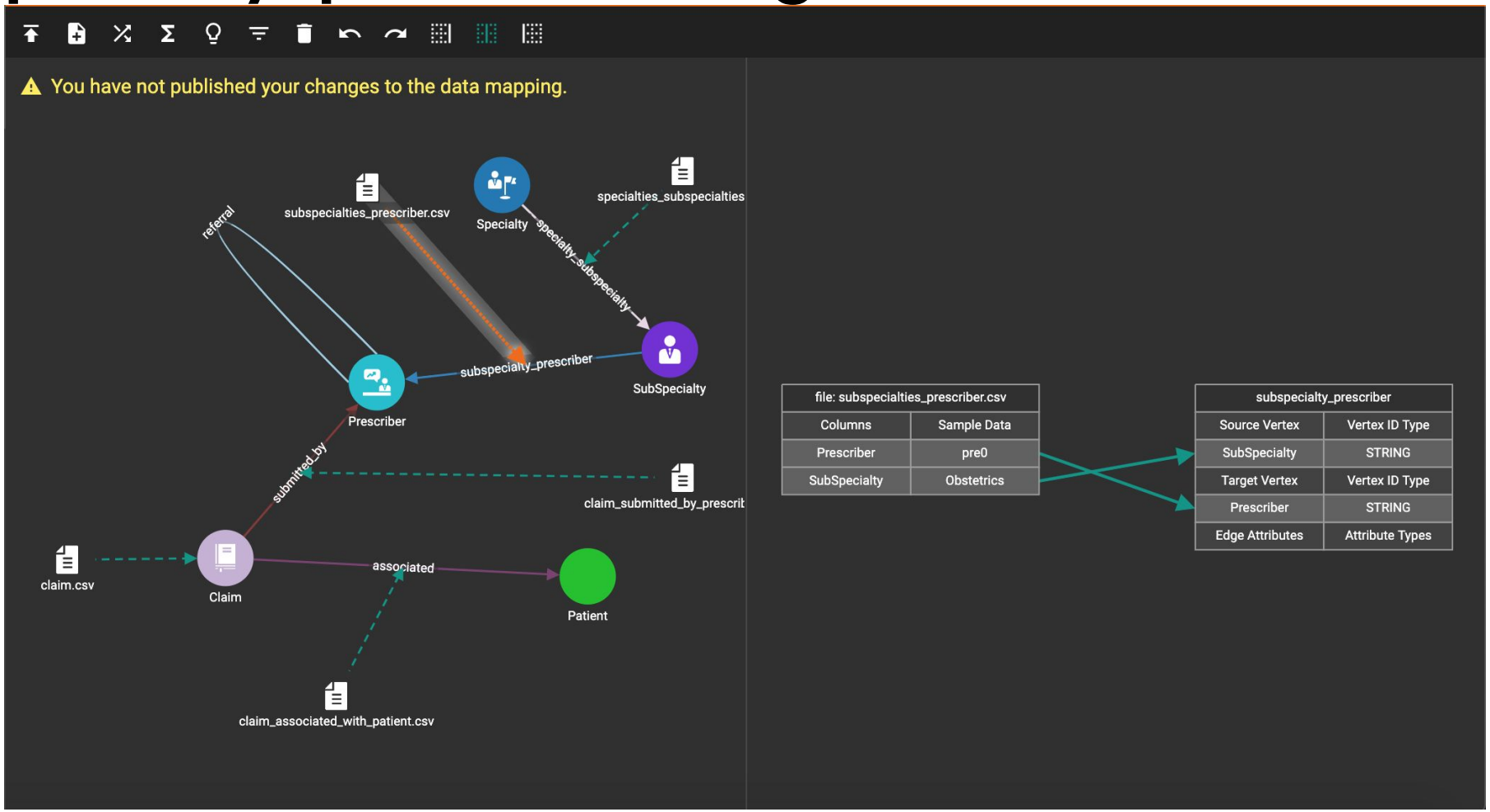
TigerGraph

Data Files and Mappings – claim associated with patient.CSV associated_with edge



TigerGraph

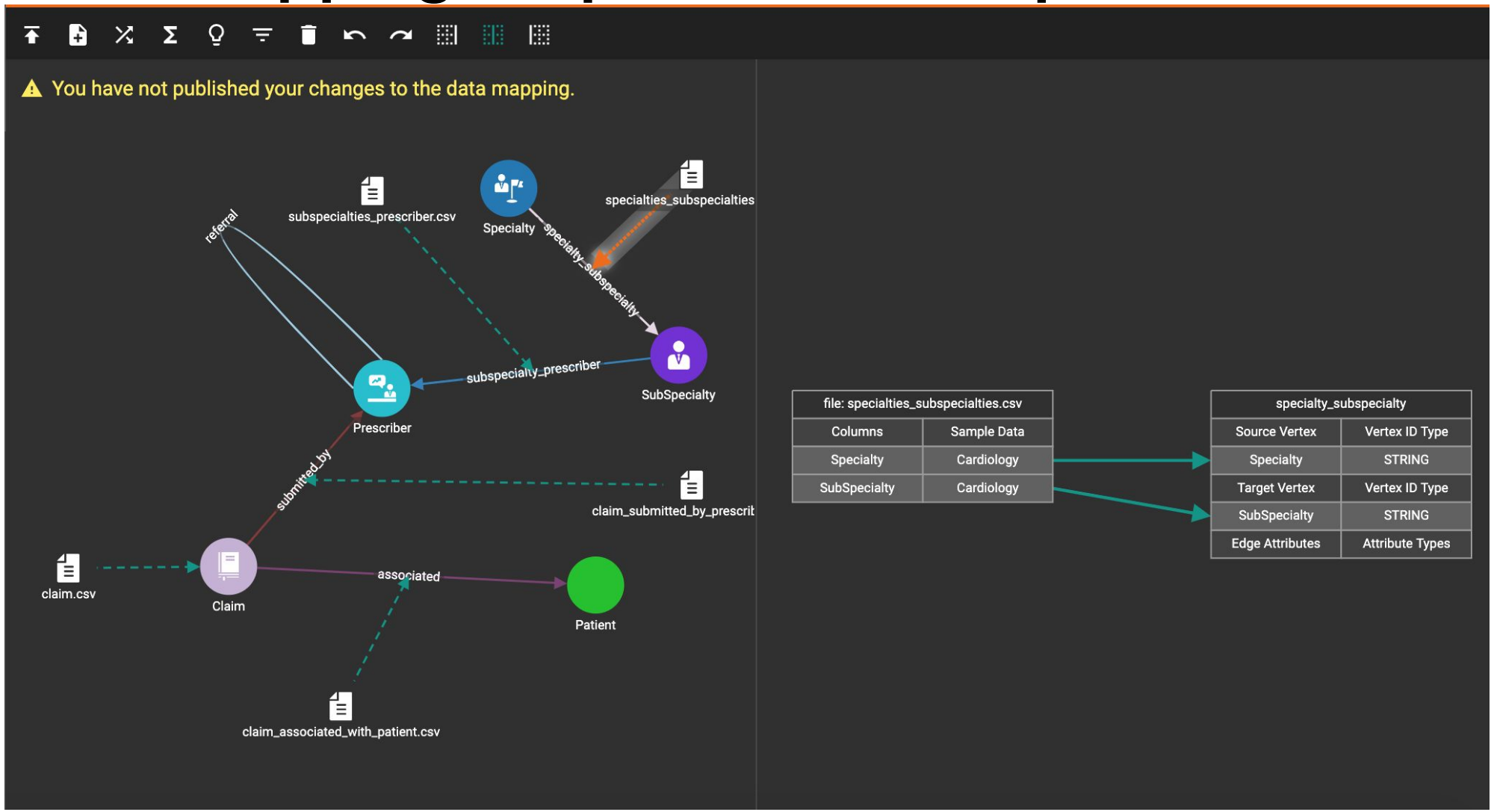
Data Files and Mappings – subspecialties prescriber.CSV subspecialty_prescriber edge



Please note that there is no need to map subspecialties prescriber.csv to Prescriber node – it will be populated automatically based on the edge mapping

TigerGraph

Data Files and Mappings – specialties subspecialties.CSV
□ edge

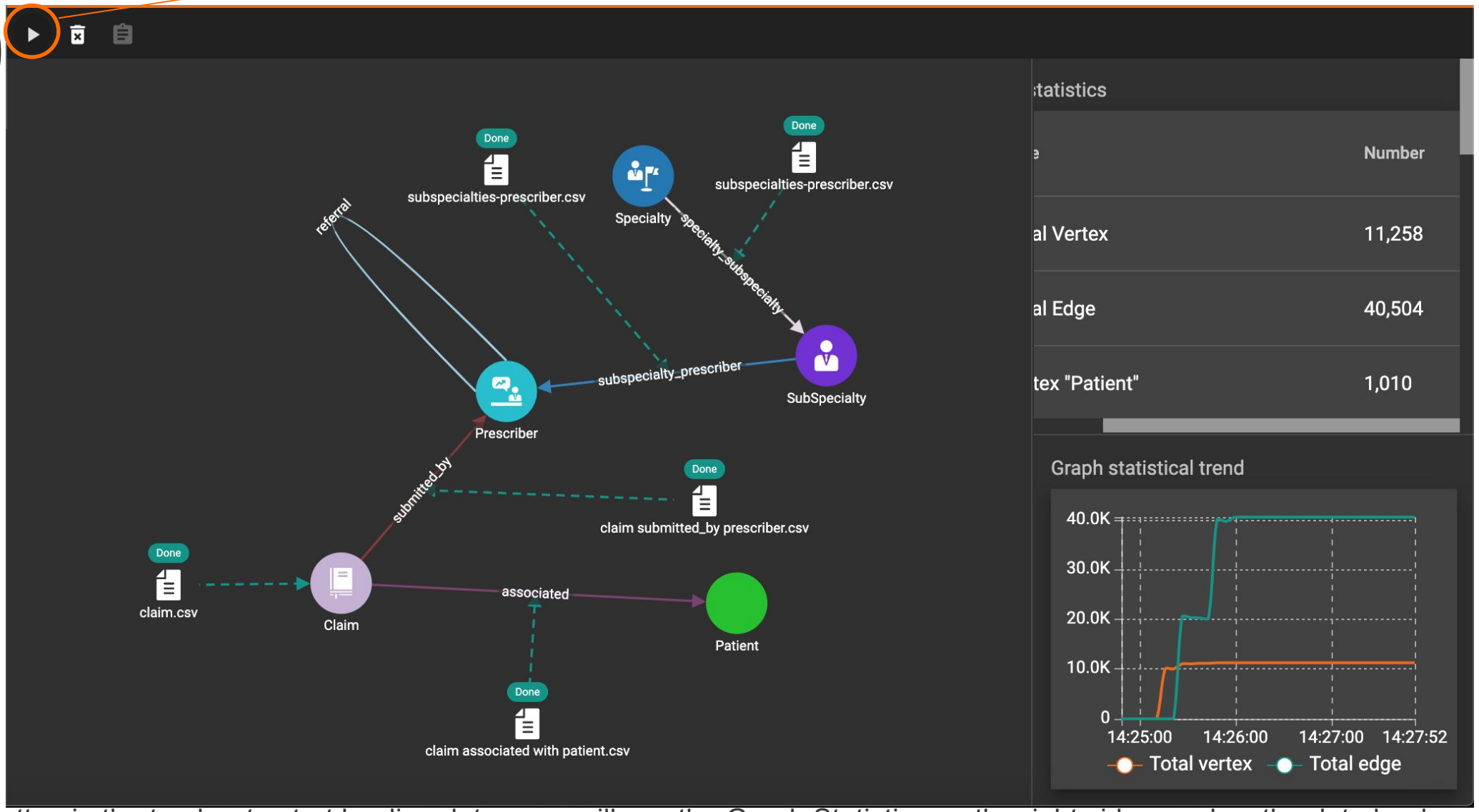


Please note that there is no need to map to specialty or subspecialty nodes – those will be populated automatically

TigerGraph

Exercise 1 – Step 3: Load Data

Click on the Play button to load the data

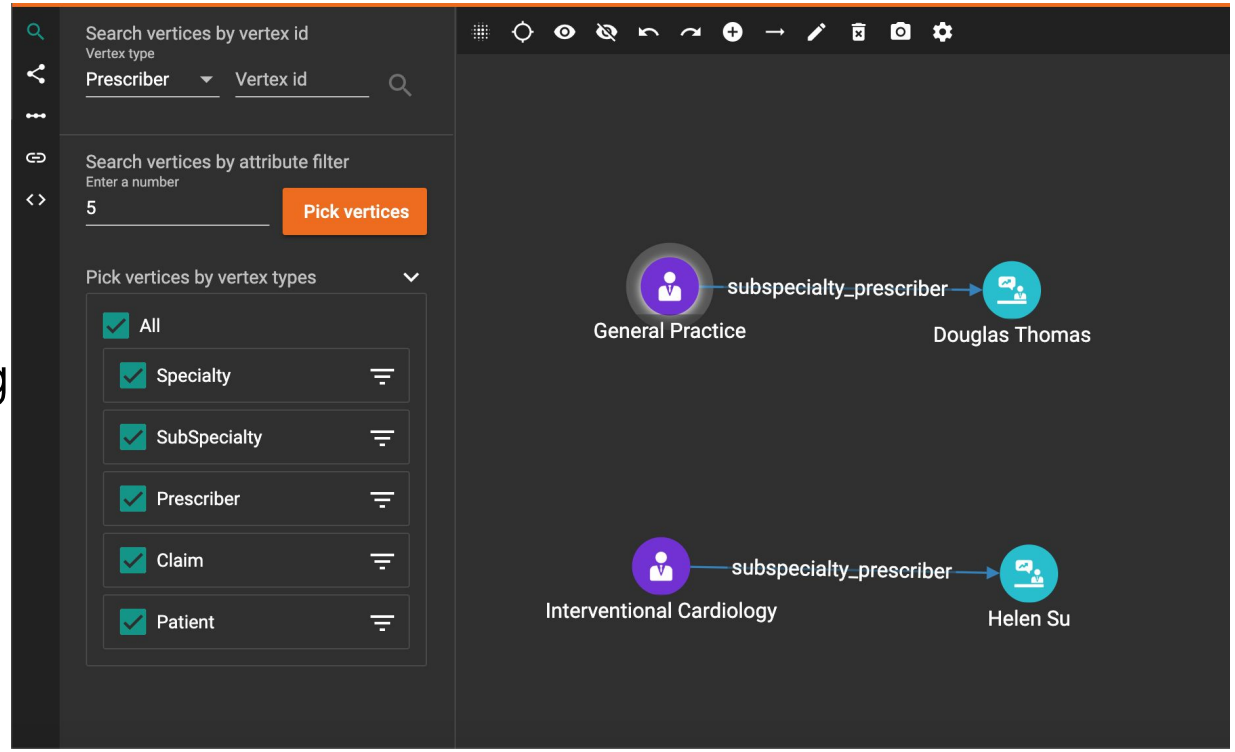


Click on the “Play” button in the top bar to start loading data - you will see the Graph Statistics on the right side panel as the data loads up

TigerGraph

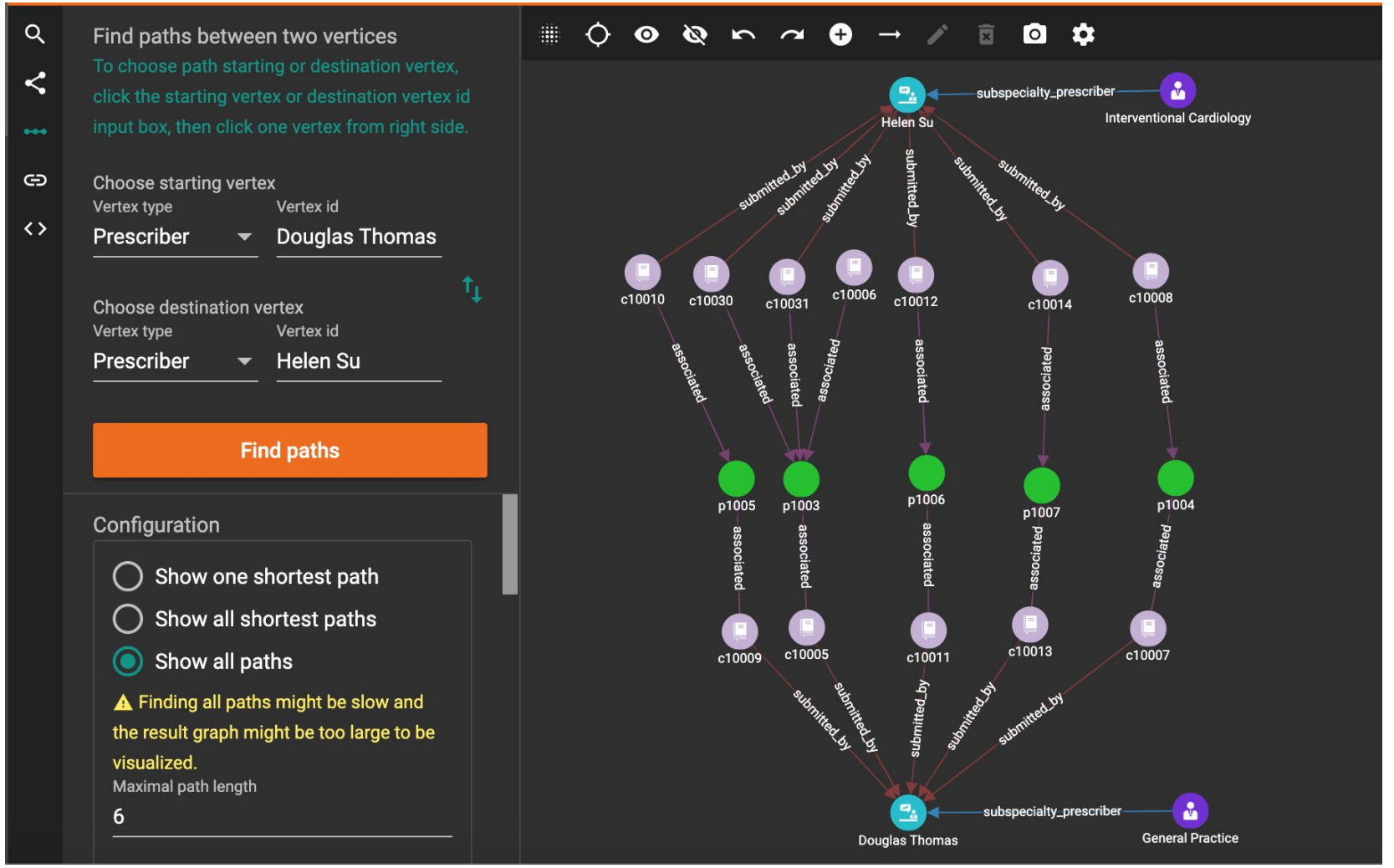
Exercise 2 - Understand how prescribers are connected

- Part 1 - Use GraphStudio to find relationships among following prescribers
  - ”Douglas Thomas” & “Helen Su”
  - **Hint** – Select “Explore Graph” in the Left Navigation and find all ways to navigating from Douglas Thomas to Helen Su or vice versa
  - **Stretch Goal** – Explore Douglas Thomas to expand and find all patients served by Dr. Thomas and find all linked prescribers who are seeing those patients



TigerGraph

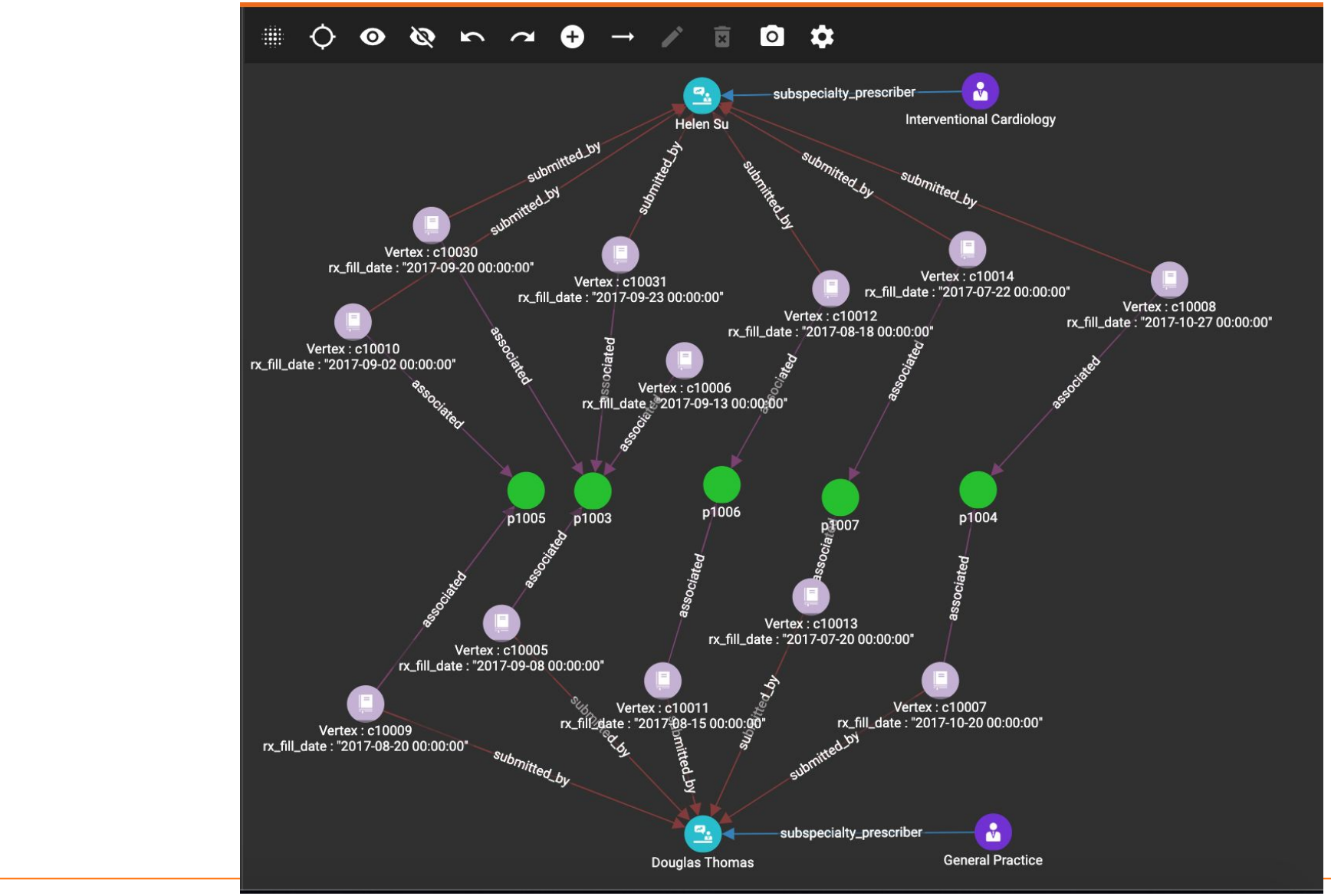
Exercise 2 - Part 1 Answer



Bonus points – Show the date of claim and the ICD 10 code description
Hint: Click on “Settings” Icon to choose the additional attributes to display

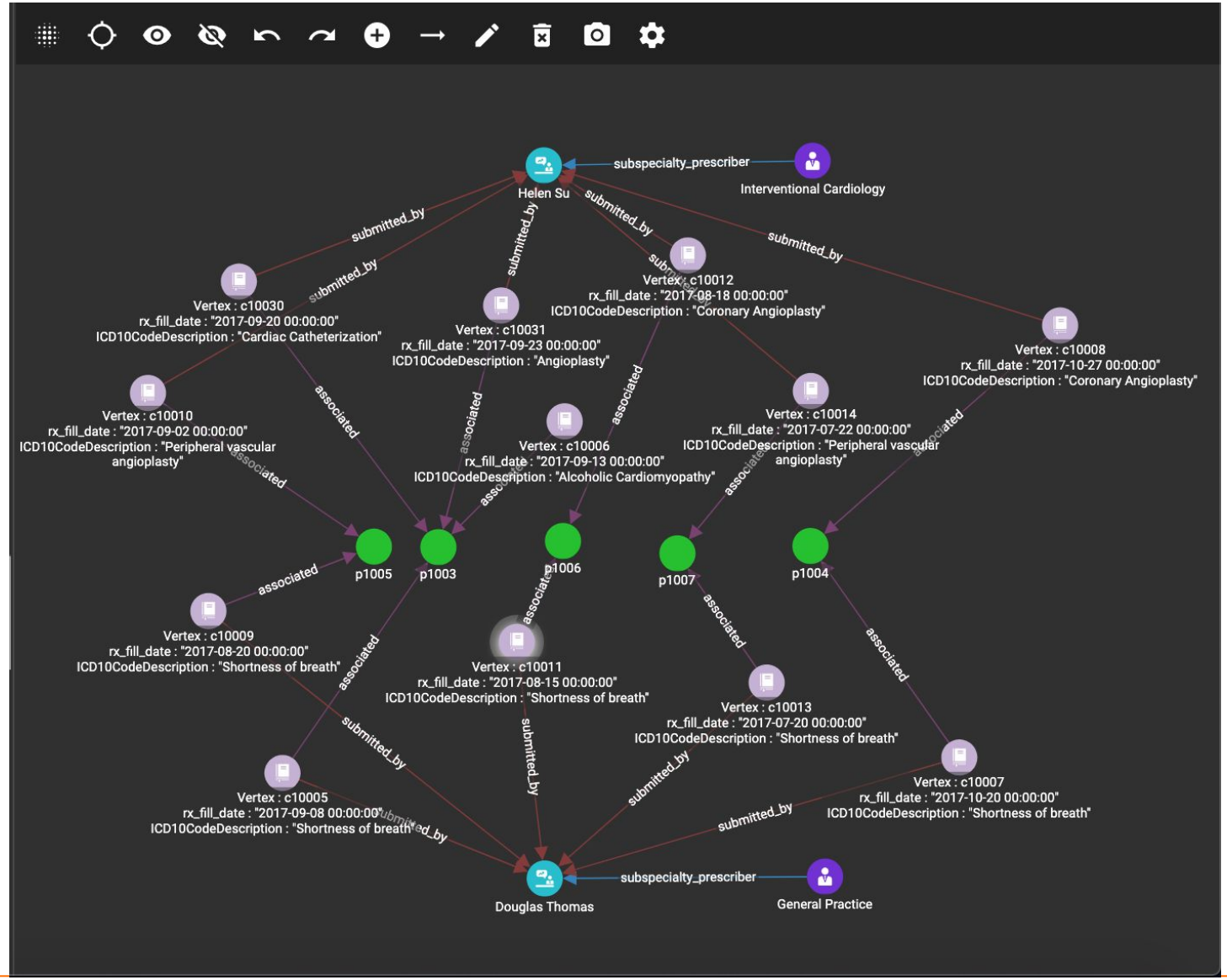
TigerGraph

Exercise 2 - Part 1 Answer with claim dates added in



TigerGraph

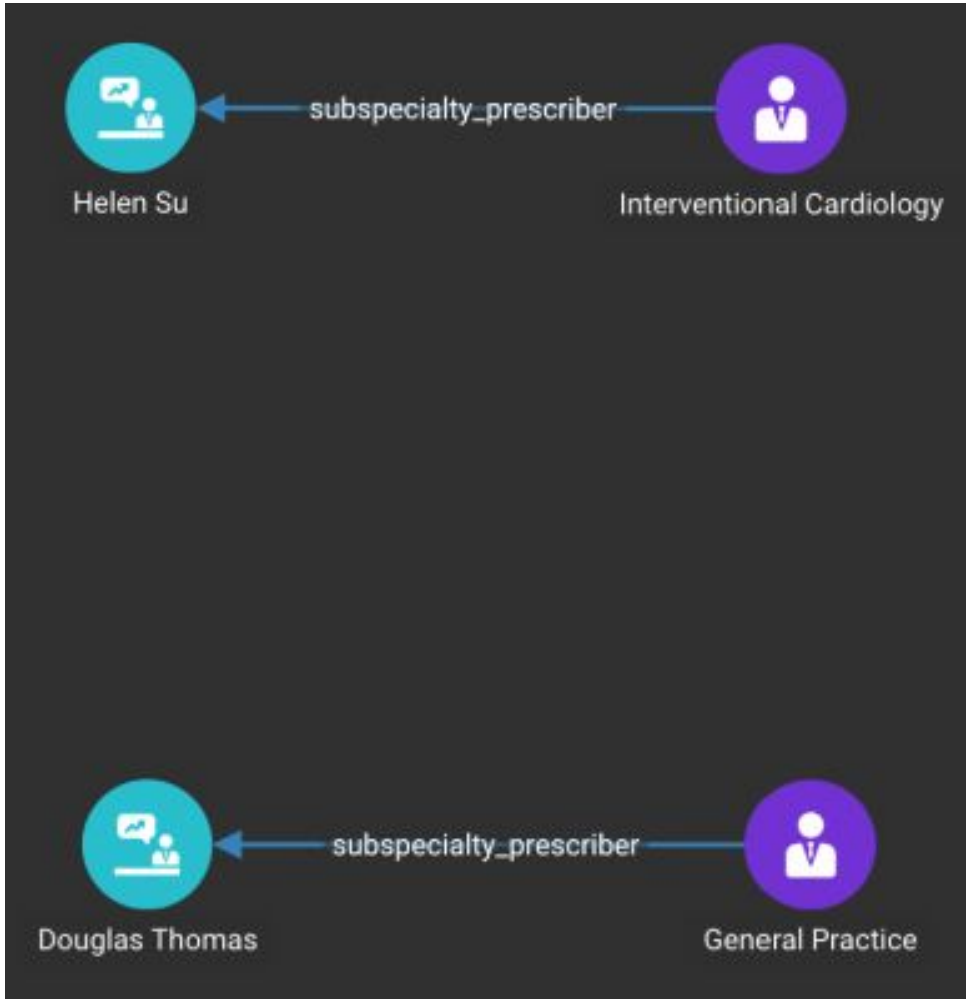
Exercise 2 - Part 1 Answer with claim dates & ICD Code 10 description added in



TigerGraph

Exercise 2 - Understand how prescribers are connected

- **Part 2 – Write your first GSQL Query: Common Patients**
  - Input: two prescribers
  - Output: Print/Return a list of common patients between the two prescribers
  - **Step 1** – Start graph Traversal from first prescriber to find all associated claims
  - **Step 2** – For those claims, find all the linked patients
  - **Step 3** – find the second prescriber connected via common patients
  - **Step 4** – Find common patients by starting from claims in Step 3
  - **Step 5** – From common patients find all claims that have been visited in earlier steps. Collect the edges so they can be printed
  - **Step 6** – From claims find associated prescribers. Collect and print edges (claims – prescribers) and prescribers.
  - **Hint**
    - Use **OrAccum** to traverse and remember which claims and patients were visited



TigerGraph

GSQL Introduction – Local and Global Variables

- local and global variables are supported in GSQL
- Local variables are use with single @
  - **Usage in the exercise -** @visited (Set visited flag on node (e.g. claim, patient, prescriber) to true to remember which nodes have been visited)
- Global variables are used with @@
  - Usage in the exercise - @@edgeSet (Use edgeSet as a global variable to collect edges for printing those later)

For more details - visit docs.tigergraph.com and search on local and global variables in search window

GSQL Introduction – Select & Where Clause

- **Select statement** – Use this to traverse the graph and go from a node (e.g. Prescriber) via an edge (e.g. reverse_submitted) to another node (e.g. Claim).

Syntax for edge traversal - source_type -(edge_type)-> target_type

- **Usage in the exercise –**

```
claims1 = select t
        from Pre1:s -(reverse_submitted:e)-> Claim:t
```

- Use select query in GSQL to go from Prescriber1 / Pre1 via “reverse_submitted” edge to claim.
- Use alias t for claim, so you can refer to it later in the code. Alias e is used for the edge, while alias s is used for the source prescriber Pre1.

- Use **where** clause along with select for conditional traversal. Usage in the exercise -

```
common_patients = select t
                from claims2:s -(associated:e)-> Patient:t
                where t.@visited == true;
```

- Use select query to go from claims2 via “associated” edge to find common_patients.
- Use the where clause to check visited flag on the patient node (alias t used for patient node).

For more details - https://docs.tigergraph.com/intro/gsql-101/built-in-select-queries

TigerGraph

GSQL Introduction – Accumulators

- **AndAccum/OrAccum –** The AndAccum and OrAccum types calculate and store the cumulative result of a series of boolean operations. The output of an AndAccum or an OrAccum is a single boolean value (True or False). AndAccum and OrAccum variables operate on boolean values only.
  - **Usage in the exercise – OrAccum @visited;**
- **SetAccum –** The SetAccum type maintains a collection of unique elements. The output of a SetAccum is a list of elements in arbitrary order. A SetAccum instance can contain values of one type. The element type can be any base type, tuple, or STRING COMPRESS.
- **Usage in the exercise – SetAccum<edge> @@edgeSet;**

For more details - https://docs.tigergraph.com/dev/gsql-ref/querying/accumulators

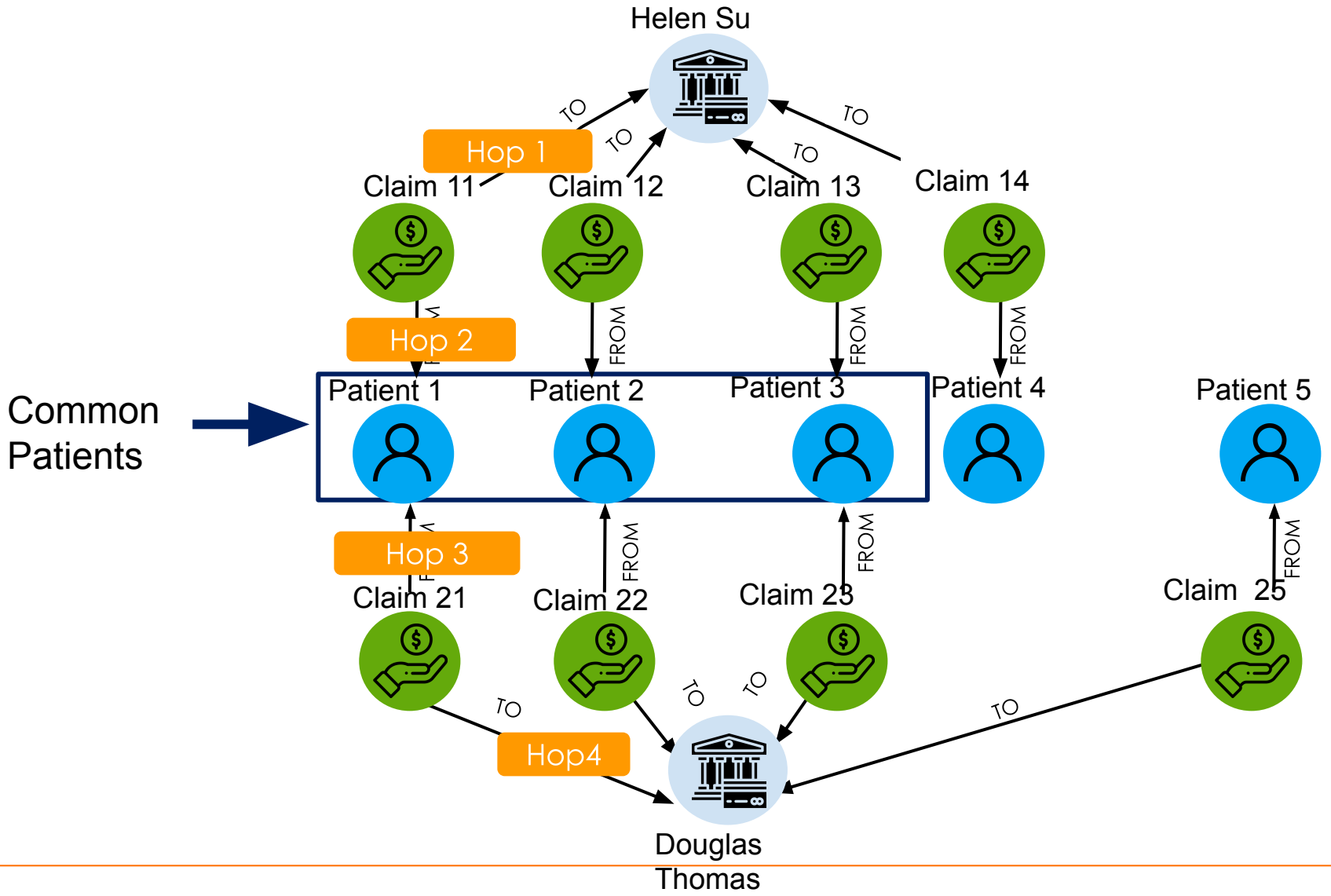
GSQL Introduction – Print command

- **Print –** The PRINT statement specifies output data. Each execution of a PRINT statement adds a JSON object to the results array which will be part of the query output. A PRINT statement can appear anywhere that query-body statements are permitted.
  - **Usage in the exercise – print claims;**

For more details - https://docs.tigergraph.com/dev/gsql-ref/querying/output-statements-and-file-objects#print-statement-api-v-2

SubGraph or Relationship Discovery & Computation Example

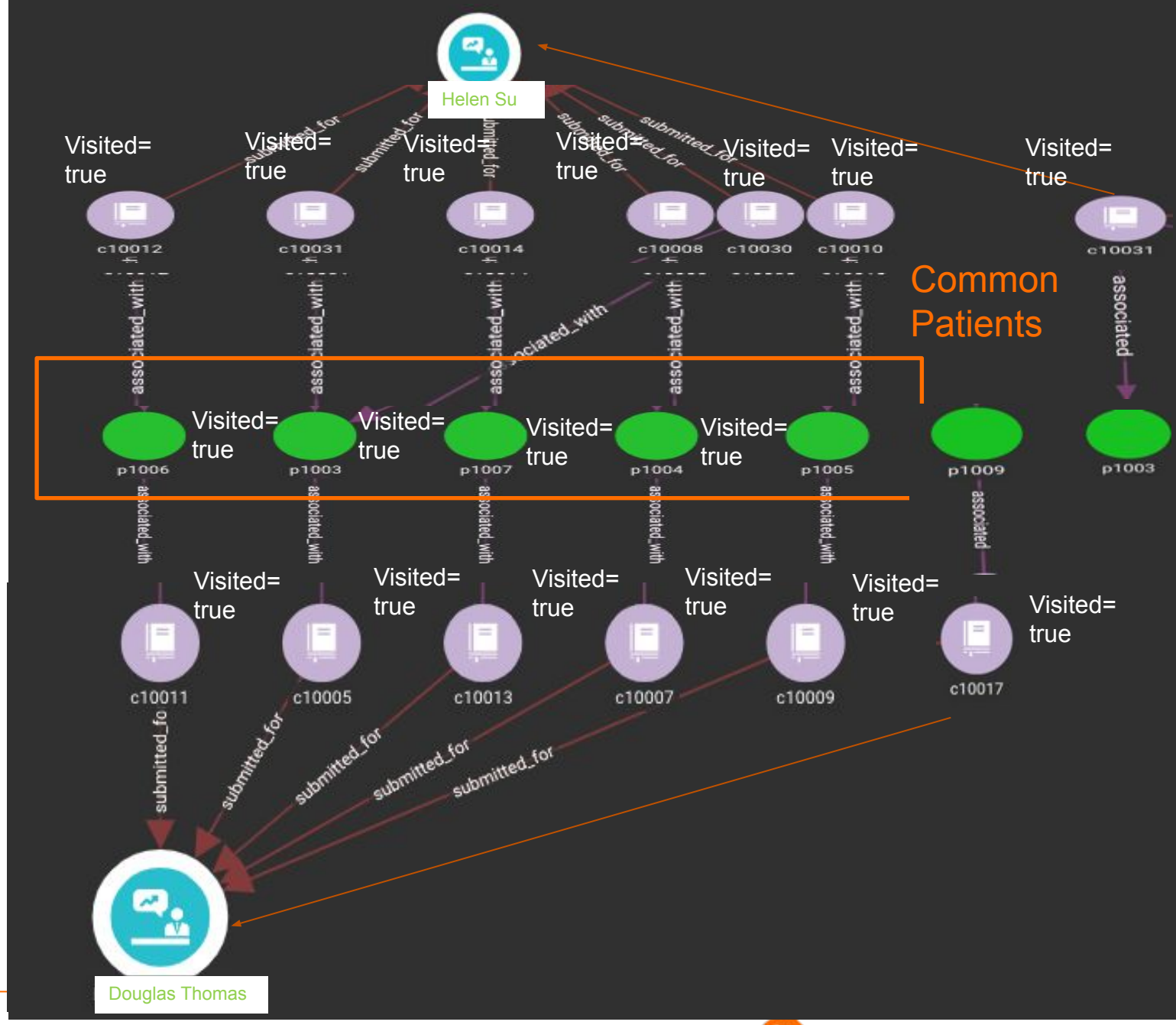
Merchant Relationship Data Analysis in Financial Services



TigerGraph

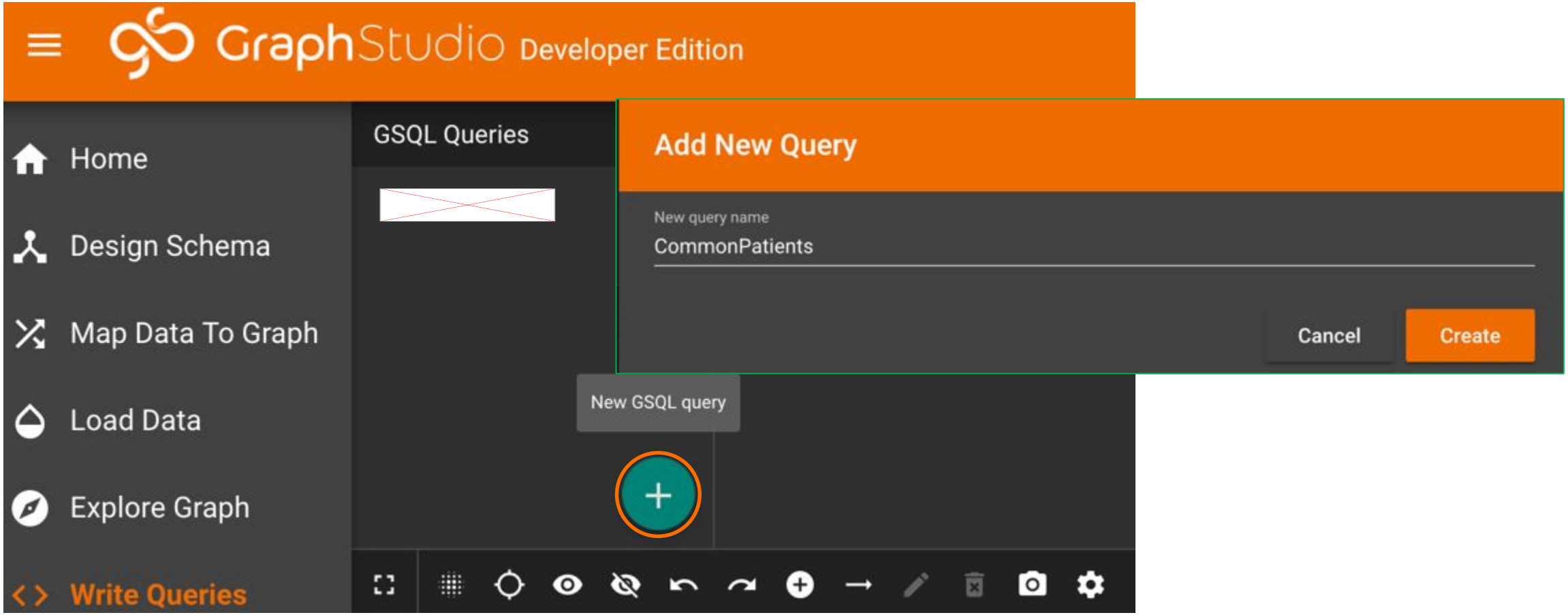
Putting It All Together Step-by-Step for CommonPatients Query

- Input: two prescribers: Douglas Thomas, Helen Su
- Output: Print/Return a list of common patients between the two prescribers
- **Step 1** – Start graph Traversal from first prescriber to find all associated claims
- **Step 2** – For those claims, find all the linked patients
- **Step 3** – Start graph traversal from second prescriber to find all associated claims
- **Step 4** – Find common patients by starting from claims in Step 3
- **Step 5** – From common patients find all claims that have been visited in earlier steps. Collect the edges so they can be printed
- **Step 6** – From claims find associated prescribers. Collect and print edges (claims – prescribers) and prescribers.



TigerGraph

Exercise 2 - Part2: Create new query



TigerGraph

Exercise 2 – Part 2 : Step 1

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

 SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;
```

Query CommonPatients takes two inputs Prescriber 1 and Prescriber 2 of vertex type <Prescriber>. Developer edition is limited to only one graph – MyGraph.

Boolean variable used to remember which nodes or edges have been visited (@ indicates local variable)

Global variable of type <edge> used to collect the edges so that they can be printed in the last step (@@ indicates a global variable)

- Use select query in GSQL to go from Prescriber1 / Pre1 via “reverse_submitted” edge to claim.
- Use alias t for claim, so you can set the variable “visited” on the claim via accum

- Want to learn all the ways you can use built-in SELECT queries? - https://docs.tigergraph.com/intro/gsql-101/built-in-select-queries & https://docs.tigergraph.com/dev/gsql-ref/querying/select-statement
- Curious what more accumulators can do for you? - https://docs.tigergraph.com/dev/gsql-ref/querying/accumulators
- Want to explore more? – Go to https://docs.tigergraph.com/intro/gsql-101 for a quick refresher and use search box to look specific keywords

TigerGraph

Exercise 2 – Part 2 : Step 1, Step 2

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

   SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;

    /* Step 2 – For those claims, find all the linked patients. */

   patients1 = select t

           from claims1:s -(associated:e)-> Patient:t

           accum t.@visited += true;
```

- Use select query in GSQL to go from each claim from claims 1 via “associated” edge to claim.
- Use alias t for patient, so you can set the variable “visited” on the patient node via accum in the last statement

TigerGraph

Exercise 2 – Part 2 : Step 1, Step 2, Step 3

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

   SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;

    /* Step 2 – For those claims, find all the linked patients. */

   patients1 = select t

            from claims1:s -(associated:e)-> Patient:t

            accum t.@visited += true;

/* Step 3 Start graph traversal from second prescriber to find all claims */

   claims2 = select t

          from Pre2:s -(reverse_submitted_by:e)-> Claim:t

          accum t.@visited += true;
```

- Use select query in GSQL to go from second prescriber via “reverse_submitted_by” edge to claims.
- Use alias t for claim, so you can set the variable “visited” on the claim node via accum in the last statement

TigerGraph

Exercise 2 – Part 2 – Step 1, 2, 3, 4

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

   SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;

    /* Step 2 – For those claims, find all the linked patients. */

   patients1 = select t

            from claims1:s -(associated:e)-> Patient:t

            accum t.@visited += true;

/* Step 3 Start graph traversal from second prescriber to find all claims */

   claims2 = select t

          from Pre2:s -(reverse_submitted_by:e)-> Claim:t

          accum t.@visited += true;
```

```
/* Step 4 – Find common patients  by starting from claims in
Step 3 */

common_patients = select t

                from claims2:s -(associated:e)-> Patient:t

                where t.@visited == true;

  print common_patients;
```

- Use select query to go from claims2 from step 3 via “associated” edge to find common_patients.
- Use the where clause to check visited flag on the patient node (alias t used for patient node).

Exercise 2 – Part 2 – Step 1, 2, 3, 4 and 5

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

   SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;

    /* Step 2 – For those claims, find all the linked patients. */

   patients1 = select t

            from claims1:s -(associated:e)-> Patient:t

            accum t.@visited += true;

/* Step 3 Start graph traversal from second prescriber to find all claims */

   claims2 = select t

          from Pre2:s -(reverse_submitted_by:e)-> Claim:t

          accum t.@visited += true;
```

```
/* Step 4 – Find common patients  by starting from claims in
Step 3 */

common_patients = select t

          from claims2:s -(associated:e)-> Patient:t

          where t.@visited == true;

   print common_patients;

/* Step 5 – From common patients find all claims that have been visited
in earlier steps. Collect the edges so they can be printed.*/

claims = select t

          from common_patients:s -(reverse_associated:e)-> Claim:t

          where t.@visited == true

          accum @@edgeSet += e;

   print claims;
```

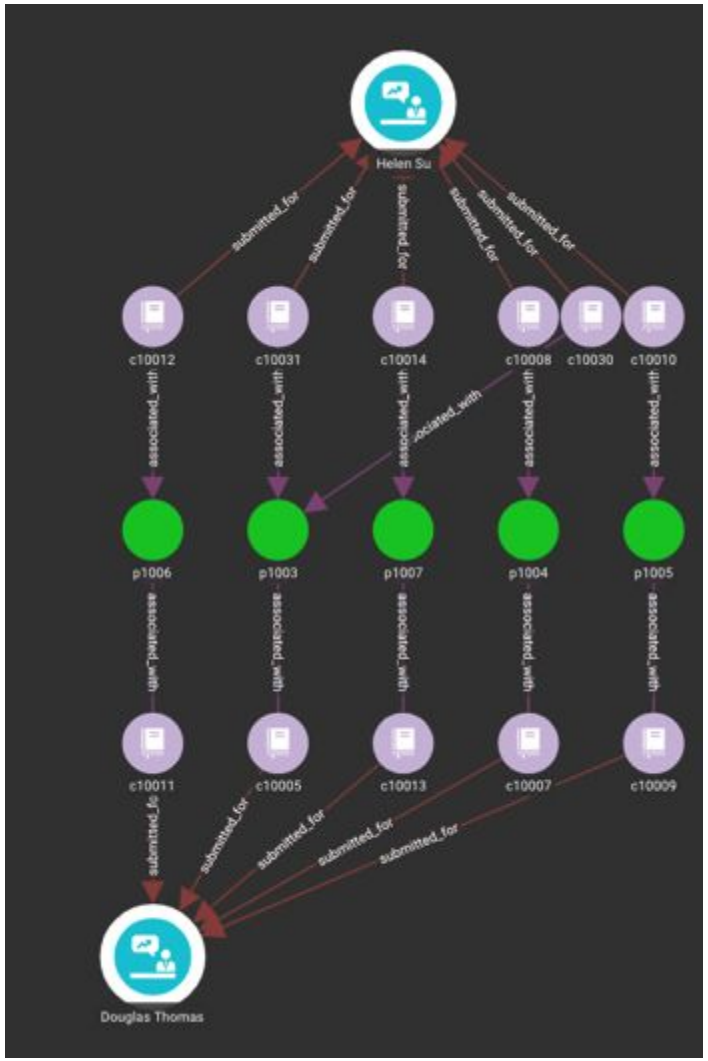
- Use select query to go from common_patients from step 4 via “reverse_associated” edge to find claims (check the visited flag).
- Use accum clause to collect the edges (alias e was assigned to reversed_associated edge in from clause, so that it can be added to edgeSet global variable via accum)

TigerGraph

Exercise 2 – Part 2 - Common Patients Query (all steps)

```
CREATE QUERY CommonPatients(vertex<Prescriber> Prescriber1,
vertex<Prescriber> Prescriber2) FOR GRAPH MyGraph {

 OrAccum @visited;

   SetAccum<edge> @@edgeSet;

   Pre1 =  {Prescriber1};

   Pre2 =  {Prescriber2};

   /* Step 1 – Start graph Traversal from first prescriber to find all associated
claims. Use visited flag to remember claims visited. */

   claims1 = select t

          from Pre1:s -(reverse_submitted:e)-> Claim:t

          accum t.@visited += true;

    /* Step 2 – For those claims, find all the linked patients. */

   patients1 = select t

           from claims1:s -(associated:e)-> Patient:t

           accum t.@visited += true;

/* Step 3 Start graph traversal from second prescriber to find all claims */

   claims2 = select t

          from Pre2:s -(reverse_submitted_by:e)-> Claim:t

          accum t.@visited += true;

/* Step 4 – Find common patients  by starting from claims in
Step 3 */

common_patients = select t

                from claims2:s -(associated:e)-> Patient:t

                where t.@visited == true;

   print common_patients;

/* Step 5 – From common patients find all claims that have been visited
in earlier steps. Collect the edges so they can be printed.*/

 claims = select t

          from common_patients:s -(reverse_associated:e)-> Claim:t

          where t.@visited == true

          accum @@edgeSet += e;

   print claims;

/* Step 6 – From claims find associated prescribers. Collect
and print edges (claims – prescribers) and prescribers. */

   pres = select t

        from claims:s -(submitted_by:e)-> Prescriber:t

        accum @@edgeSet += e;

   print pres;

   print @@edgeSet;

}
```

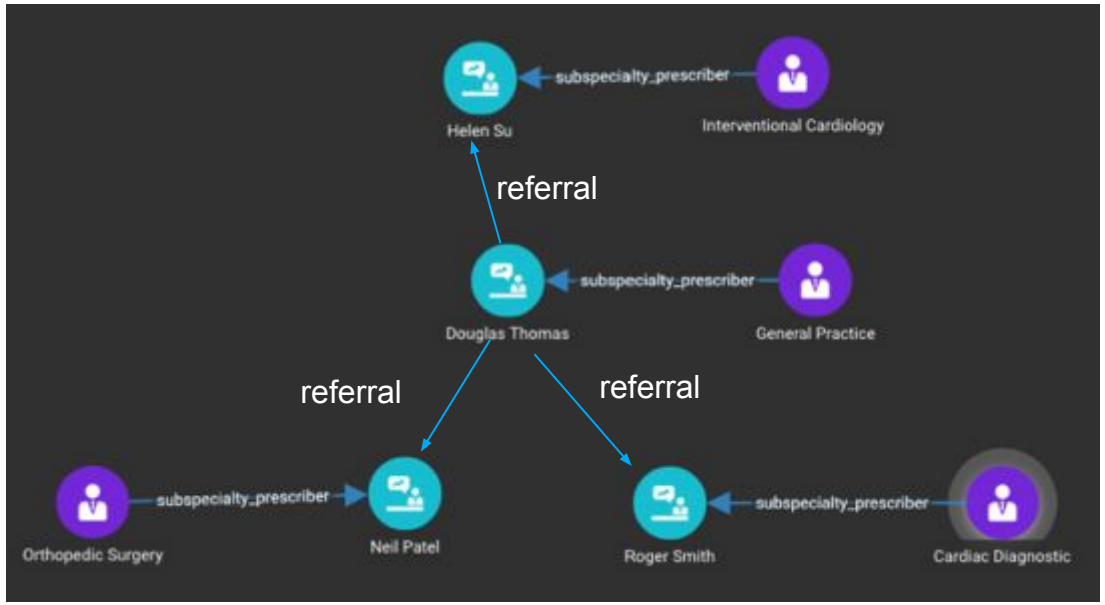
Exercise 2 - **Output** (Common Patients)



TigerGraph

Exercise 3 - Find and visualize prescriber referral relationships

- Write your second GSQL Query: CreateReferralEdge_Simple
  - **Input:** a prescriber (Douglas Thomas)
  - **Output:** Print/Return all doctors that have a referral relationship with that prescriber
  - **Step 1** – Start graph traversal from the input prescriber, find all claims for that prescriber (**be sure use OrAccum to remember which claims were visited**)
  - **Step 2** – For each claim, traverse graph to the patient and find all other claims for that patient (**remember to use output from OrAccum in step 1 to skip claims that have been already visited**)
  - **Step 3** – From all other claims, find related prescriber (**be sure to skip collecting the prescriber if it is same as the input prescriber**)
  - **Step 4** - Print related/likely referred prescribers and the input prescriber
  - **Step 5 (Stretch goal)** – Create the referral edges between input and related prescriber and print



TigerGraph

Exercise 3 – CreateReferralEdge_Simple Query: Step 1

```
CREATE QUERY CreateReferralEdge_Simple(VERTEX<Prescriber>
input) FOR GRAPH MyGraph {

 /* Step 1 - Start graph traversal from the input prescriber, find all
claims for that prescriber.

 */

 OrAccum<bool> @visited;

given_prescriber (ANY) = {input};


claims = select t

          from given_prescriber:s-(:e)-> Claim:t

        accum t.@visited += true;
```

- You will notice that the edge name is missing. It's ok not to provide the edge name if there's only 1 edge that goes between the nodes in the direction of graph traversal.
- In general, it's a good practice to explicitly include the name of clarity

Exercise 3 – CreateReferralEdge_Simple Query: Step 1, 2

```
CREATE QUERY CreateReferralEdge_Simple(VERTEX<Prescriber>
input) FOR GRAPH MyGraph {

 /* Step 1 - Start graph traversal from the input prescriber, find all
claims for that prescriber.

 */

 OrAccum<bool> @visited;

given_prescriber (ANY) = {input};


claims = select t

            from given_prescriber:s-(:e)-> Claim:t

         accum t.@visited += true;

/* Step 2 - For each claim, traverse graph to the patient and find
all other claims for that patient.  Use visited flag set earlier to
avoid claims that are already traversed.

*/

patients = select t

             from claims:s-(:e)-> Patient:t;

other_claims = select t

                 from patients:s-(:e)-> Claim:t

              where t.@visited == false;
```

Exercise 3 – CreateReferralEdge_Simple Query: Step 1, 2, 3

Part A

```
CREATE QUERY CreateReferralEdge_Simple(VERTEX<Prescriber> input) FOR GRAPH MyGraph {

 /* Step 1 - Start graph traversal from the input prescriber, find all claims for that prescriber.

 */

 OrAccum<bool> @visited;

given_prescriber (ANY) = {input};


claims = select t

           from given_prescriber:s-(:e)-> Claim:t

         accum t.@visited += true;

/* Step 2 - For each claim, traverse graph to the patient and find all other claims for that patient.  Use visited flag set earlier to avoid claims that are already traversed.

*/

patients = select t

            from claims:s-(:e)-> Patient:t;
```

Part B

```
other_claims = select t

                from patients:s-(:e)-> Claim:t

               where t.@visited == false;

 /* Step 3 - From all other claims, find related prescriber. be sure to skip collecting the prescriber if it is same as the input prescriber.

 */

 related_prescribers = select t

                       from other_claims:s-(:e)-> Prescriber:t

                      where t != input;
```

Step 5 Hint - Use insert command to populate referral edges

Exercise 3 – CreateReferralEdge_Simple Query

Part A

```
CREATE QUERY CreateReferralEdge_Simple(VERTEX<Prescriber> input) FOR GRAPH MyGraph {

 /* Step 1 - Start graph traversal from the input prescriber, find all claims for that prescriber.

 */

 OrAccum<bool> @visited;

given_prescriber (ANY) = {input};


claims = select t

          from given_prescriber:s-(:e)-> Claim:t

        accum t.@visited += true;

/* Step 2 - For each claim, traverse graph to the patient and find all other claims for that patient.  Use visited flag set earlier to avoid claims that are already traversed.

*/

patients = select t

            from claims:s-(:e)-> Patient:t;
```

Part B

```
other_claims = select t

                from patients:s-(:e)-> Claim:t

              where t.@visited == false;

 /* Step 3 - From all other claims, find related prescriber. be sure to skip collecting the prescriber if it is same as the input prescriber.

 */

 related_prescribers = select t

                       from other_claims:s-(:e)-> Prescriber:t

                     where t != input;

/* Step 4 - Print related/likely referred prescribers and the input prescriber

 */

    print related_prescribers;

    print given_prescriber;

}
```

Step 5 Hint - Use insert command to populate referral edges

TigerGraph

Exercise 3 – CreateReferralEdge_Simple Query

Part A

```
CREATE QUERY CreateReferralEdge_Simple(VERTEX<Prescriber> input) FOR GRAPH MyGraph {

 /* Step 1 - Start graph traversal from the input prescriber, find all claims for that prescriber.

 */

 OrAccum<bool> @visited;

given_prescriber (ANY) = {input};


claims = select t

          from given_prescriber:s-(:e)-> Claim:t

        accum t.@visited += true;

/* Step 2 - For each claim, traverse graph to the patient and find all other claims for that patient.  Use visited flag set earlier to avoid claims that are already traversed.

*/

patients = select t

            from claims:s-(:e)-> Patient:t;
```

Part B

```
other_claims = select t

                from patients:s-(:e)-> Claim:t

              where t.@visited == false;

 /* Step 3 - From all other claims, find related prescriber. be sure to skip collecting the prescriber if it is same as the input prescriber.

 */

 related_prescribers = select t

                       from other_claims:s-(:e)-> Prescriber:t

                     where t != input;

/* Step 4 - Print related/likely referred prescribers and the input prescriber

 */

    print related_prescribers;

    print given_prescriber;

}
```

Step 5 Hint - Use insert command to populate referral edges

TigerGraph

Exercise 3 - Output (Input: Douglas Thomas)

Stretch goal – print referral edges by using insert command to populate referral edges & print them at the end

TigerGraph

Additional Resources

Download the Developer Edition or Enterprise Free Trial

https://www.tigergraph.com/download/

Guru Scripts

https://github.com/tigergraph/ecosys/tree/master/guru_scripts

Join TigerGraph Developer Forum

https://groups.google.com/a/opengsql.org/forum/#!forum/gsql-users

Read the eBook - Native Parallel Graphs

https://www.tigergraph.com/ebook

@TigerGraphDB

youtube.com/tigergraph

facebook.com/TigerGraphDB

linkedin.com/company/TigerGraph

TigerGraph