Octostar Documentation

Welcome to Octostar Documentation

Octostar is a powerful platform for building data-driven applications. This documentation will guide you through:

- **Getting Started** - Learn the basics and set up your first workspace
- **Administration** - Install and manage Octostar deployments
- **Implementation** - Build custom apps and integrations
- **User Guide** - Master the platform features and workflows

Quick Links

- Getting Started with Octostar
- Installing Octostar
- User Guide
- Implementer Guide

Fusion - Semantic Graph Platform

Model, visualize, and query your data as a connected semantic knowledge graph.

- Platform Overview
- SQL Reference
- REST API

Need Help?

Browse through the documentation sections in the sidebar to find detailed information about specific topics.

Introduction to Octostar

This section provides a foundational introduction to Octostar. It explains the purpose and core principles of **the platform**, summarizes its **key concepts**, and presents an overview of the system **architecture**. In addition, it offers guidance on how to use this **documentation** effectively and defines key terminology in the **glossary**.

What is Octostar?

Octostar is an Investigative and Decision Intelligence platform. You use it to run investigations and make decisions using data from many sources, including evidence files (documents, media, exports) and structured records (people, organizations, events, places, objects). Octostar brings search, analysis, and collaboration into one place so you can move from raw evidence to auditable outputs without switching between separate tools. Octostar combines:

- **Workspaces** to organize investigations and control access.
- **Records** to capture structured facts about entities.
- **Relationships** to connect records and support network analysis.
- **Search** across unstructured evidence and structured records.
- **Apps** to run task-specific workflows such as extraction, transcription, imports, and reporting.
- A consistent data model, the **Ontology**, so records, search filters, and visualizations behave consistently.

How Octostar works

Most workflows follow the same pattern:

- **Bring data into Octostar**
- Upload files to a **Workspace**, import a CSV, or connect external data sources (if enabled).
- **Explore and structure the data**
- Use **Search** to find relevant files and entities. Create or review **Records** and **Relationships** as you confirm facts.
- **Analyze and present findings**
- Use a **Link Chart** to explore connections. Use apps and dashboards to run specific analyses and generate outputs.
- **Share results with your team**
- Save artifacts to the workspace and share access with users or roles, based on permissions.

Key Concepts at a Glance

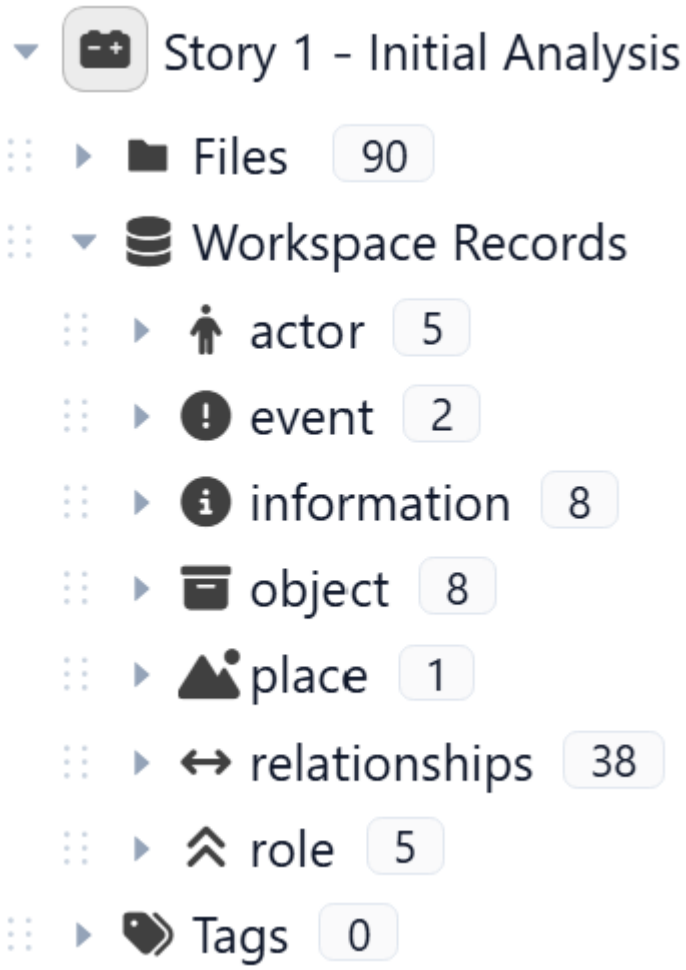
This section introduces the **core concepts** that underpin the system's design. It provides a high-level overview of the primary components — **workspaces**, **records and concepts**, the **ontology**, **apps**, and **LinkChart** — and how they collectively shape the Octostar's structure and behavior. Detailed explanations follow in the subsequent sections.

Workspaces

Workspaces are data containing spaces which are secured for **Read**, **Read/Write** or **Admin** operations - restricted to specific groups or individuals.

In this section, you learn how workspaces are used to separate investigations, how to open and organize them, and how access levels control what you can view, edit, and manage inside a workspace.

Records and Concepts



Records are structured items used to capture and review information in Octostar. Each record is an instance of a *concept*. A concept is a type defined in the **Ontology**—the platform's data model. Examples of concepts include **Person**, **Organization**, **Event**, **Place**, or **Object**.

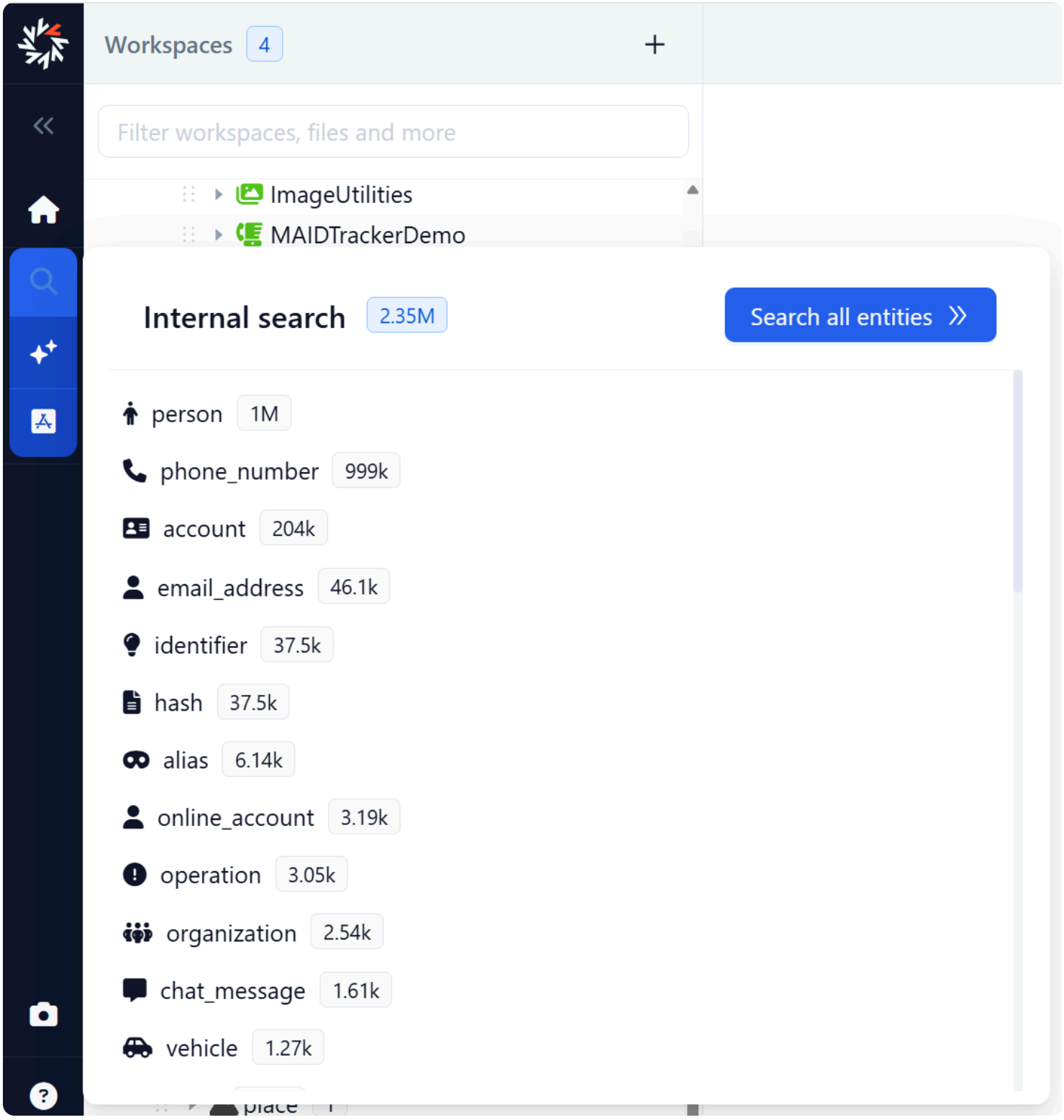
Ontology and Its Role

The Ontology determines:

- **Fields**: Properties a record can have, such as names, identifiers, dates, or locations.

- **Connections**: Relationships a record can establish with other records, like employed_by, communicates_with, or located_at.

Consistency Across Platforms



Because records adhere to the Ontology, entity details and links remain consistent across various platforms, including Search, the Record Viewer, dashboards, and Link Charts.

The Ontology

The **Ontology** serves as Octostar's data model. It plays a crucial role in ensuring consistency and coherence within the system. Here's how it functions.

Key Components

- **Concepts**: These are the elements you explore, such as:
  - **Person**
  - **Organization**
  - **Event**
  - **Place**
  - **Object**
- **Properties**: Each concept can store various attributes, including:
  - Names
  - Identifiers
  - Dates
  - Locations
- **Relationships**: These define how records are interconnected, for example:
  - Employed_by
  - Located_at
  - Communicates_with

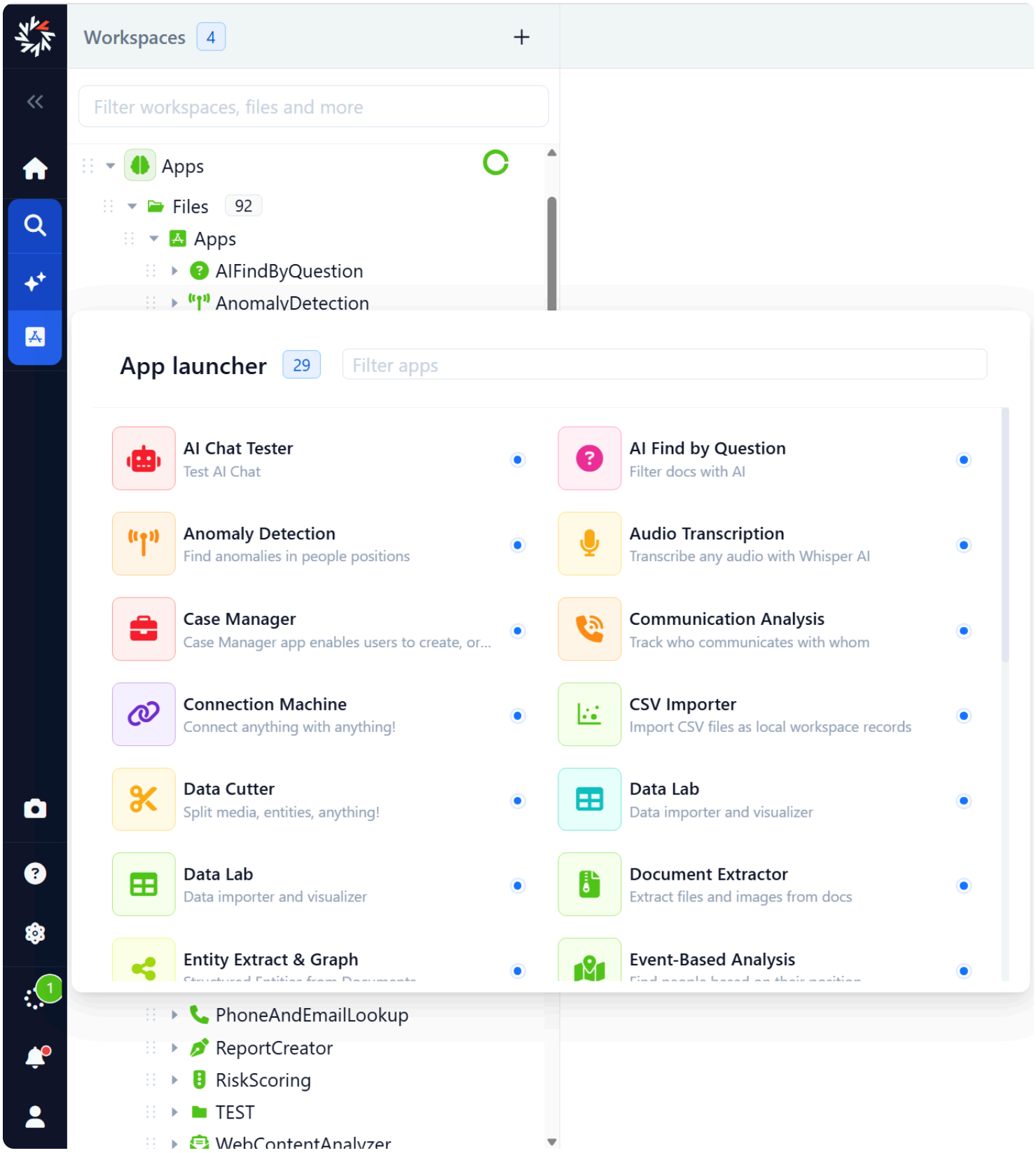
Importance of the Ontology

The Ontology ensures that Octostar remains consistent by:

- Determining how records are labeled and displayed
- Specifying which filters appear in Search
- Guiding how Link Charts draw connections

- Influencing how apps and dashboards interpret your data For more information about the Ontology, visit Implementer Guide (Implementer Guide).

Apps

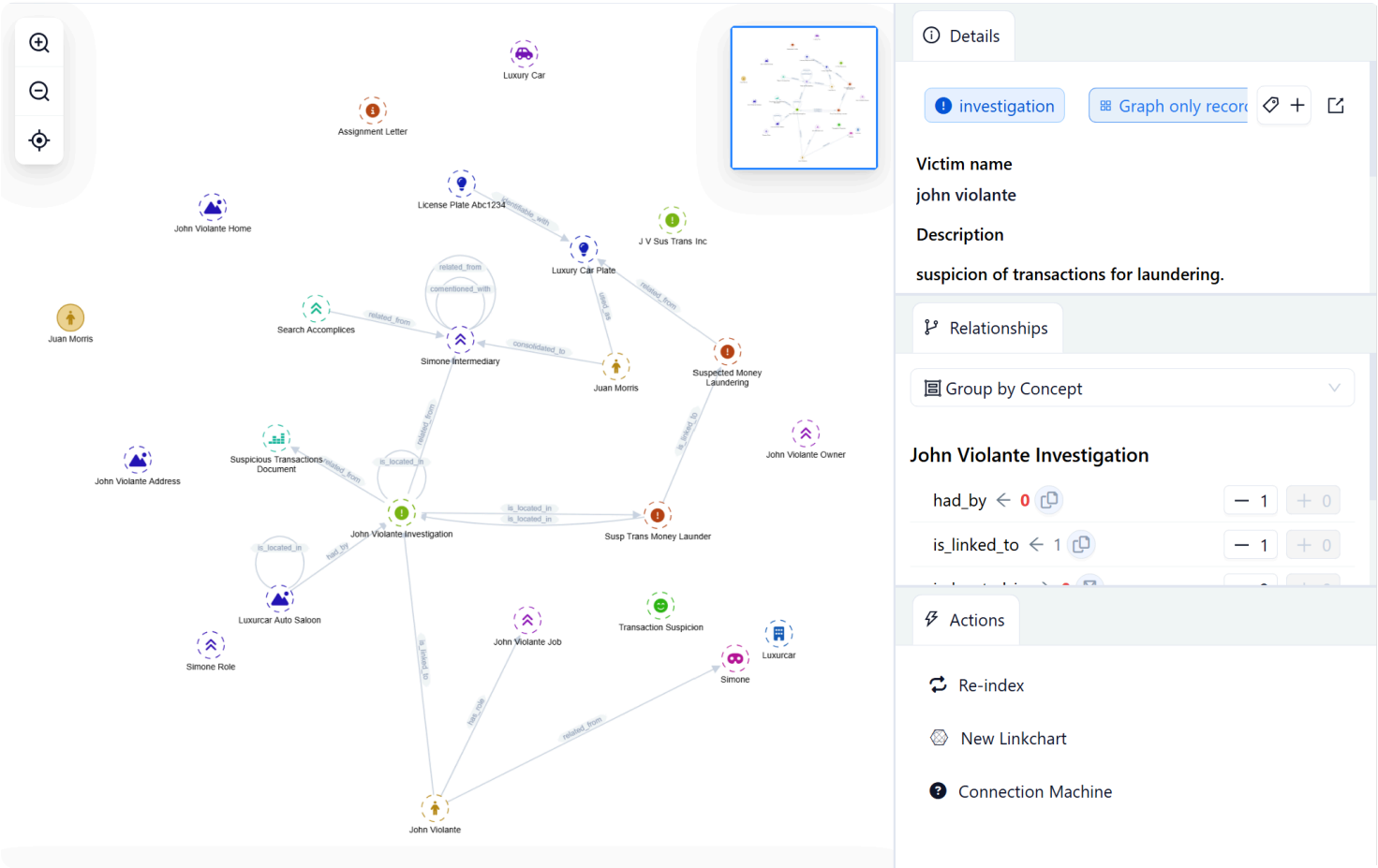


Apps are specialized tools within Octostar designed to assist with specific investigative tasks. They enable you to:

- Process evidence, such as extracting entities from documents, transcribing audio, importing CSV files, or generating reports.
- Conduct specialized analyses on files, records, or datasets. Apps open in the main area and can be launched via the **App Launcher** or the **Open with…** option on a workspace item. They adhere to your workspace context and permissions. The output from an app is saved back to your workspace as files, records, or Link Charts, ensuring results remain linked to your case.

The Linkchart

A **Link Chart** is an interactive network view that shows **records** as nodes and **relationships** as links, so you can explore how people, organizations, places, events, and objects connect. You can start a Link Chart from a record, a search result set, or an app output, then expand neighbors, filter by concept or relationship type, and adjust layout to focus on what matters. Selecting a node shows key details and lets you pivot quickly into deeper work—open the full record, ask AI Chat about the selected context, or create a focused sub-chart. Link Charts can be saved to the workspace, shared with teammates, and exported, while preserving provenance back to the underlying records and evidence.



Platform Architecture Overview

Octostar is organized around a few core services that work together. **Workspaces** provide secured storage for evidence files and case artifacts, while **records** and **relationships** represent structured data defined by the **Ontology**.

When you upload or connect data, Octostar indexes it so you can search across both unstructured evidence and structured records; in many deployments this includes enrichment steps (for example OCR, transcription, and entity extraction) and both full-text and semantic indexing for fast retrieval.

Connected databases and external systems can be exposed through the **Fusion Center** using mappings that align source data to the Ontology, creating a unified (often virtual) knowledge graph without forcing you to move data.

On top of this foundation, **apps**, **Search**, **Link Charts**, dashboards, and **AI Chat** provide the user-facing workflows, and all access is governed by permissions with audit-friendly provenance back to source files and records.

Guide to the Documentation

Use this guide to find the right section of the documentation based on your role and tasks.

For Investigators and Analysts

Start with the **User Guide**. It covers:

- Day-to-day workflows
- Using workspaces
- Searching
- Working with records
- Building Link Charts
- Using AI Chat with context

For Platform Configurators

Use the **Implementer Guide** if you:

- Set up the Ontology
- Create templates
- Design dashboards
- Handle CSV imports
- Establish workspace conventions

For Developers

Refer to the **Developer Guide** for:

- Architecture
- SDKs
- Coding patterns

For Administrators

Consult the **Administration Guide** if you manage:

- Installation

- Infrastructure
- Authentication
- System health

Universal Resources

The **Glossary** and **Ontology Reference** are useful for everyone. They provide quick definitions and confirm how concepts, properties, or relationships are used across Octostar.

Glossary

Access levels (Read / Write / Admin) — Workspace permissions that control what you can do. **Read** lets you view and search. **Write** adds uploading and editing files, creating and editing records, and managing tags/comments. **Admin** adds workspace configuration and member/role management.

Access Manager — A Fusion Center page used to review users, roles, and effective permissions, and to onboard or update access.

Administrative workspace — A workspace used to store configuration assets (for example apps, templates, dashboards) rather than day-to-day evidence. It is often configured to **Auto open** and **Hide by default** for standard users.

AI Chat — A chat interface that answers questions using a defined set of sources. When you add context (files, folders, records, sets, or workspaces), responses are grounded in that evidence and limited by your permissions.

App — A task-focused tool that runs inside Octostar (for example Entity Extract & Graph, CSV Importer, Transcriber, Map, Report Creator). Apps can take files, records, or sets as input and write results back to the workspace.

App Editor — The in-platform workspace used to view and edit an app's project files, manage secrets, preview or run the app, and review logs (where enabled).

App Launcher — The menu that lists apps available to you. You can open apps from here or from **Open with…** on a file or record.

Attachment — A file linked to a record (for example a document, image, transcript, or export) to support provenance and collaboration.

Audit log — A record of user actions and system events used for traceability (availability depends on deployment and permissions).

Cards view — A Search or dataset view that shows results as card tiles with key attributes and quick actions.

Concept — A type of entity defined in the Ontology (for example **Person**, **Organization**, **Event**, **Place**, **Object**). Concepts determine which fields (properties) and links (relationships) are available.

Concept mapping — A mapping that turns datasource rows into entities of a concept, binding columns to ontology properties and defining keys/labels.

Context (AI Chat) — The specific sources AI Chat is allowed to use for an answer (for example selected files, folders, records, sets, or workspaces).

CSV Import — A tool that imports a CSV file into Octostar. Depending on configuration, it can create tables, records, or structured entities.

Datasource — An external system or database connected in Fusion Center (for example MySQL, PostgreSQL, ClickHouse, Redshift, Vertica, SAP HANA). Datasources can be exposed through mappings without moving the data.

Dataset view — A visualization of a result set (for example table, cards, or detailed view). A dataset typically comes from Search or a saved set.

Detailed view — A Search or dataset view that shows richer, expanded attributes per result than Cards or Table.

Embeddings (vector search) — Numeric representations of text or media used for semantic search (for example finding “luxury vehicle” images even without those words).

Entity — A specific instance of a concept (for example a particular person or organization). Entities are represented as records.

Entity Extract & Graph — An app that extracts entities and relationships from a document, highlights them in context, and can normalize them into structured records and a Link Chart.

Filtered concept — A concept plus filters (for example “People with age between 30–40”). Filtered concepts drive Search results and dataset views.

Force re-index — A workspace action that refreshes indexing and enrichment outputs for files or folders so they appear correctly in Search (availability depends on configuration).

Fusion Center — The administrative hub for modeling the Ontology, connecting datasources, creating mappings, managing jobs and access, and validating data (for example via SQL Lab).

Hierarchy level — A concept’s position in the ontology inheritance tree. It helps explain which properties are inherited from parent concepts.

Knowledge graph — A graph representation of entities and relationships defined by an Ontology. In Octostar it is often virtual, with data queried in place.

Label pattern — The rule that defines how an entity is displayed in the UI (for example combining name + date of birth).

Link Chart — An interactive network view that shows entities as nodes and relationships as links. You can expand, filter, layout, annotate, save, share, and export charts.

Many-to-many mapping (relationship mapping) — A mapping that produces relationships (edges) between entities, typically using join logic across tables/views.

Member Access — The workspace dialog where you grant access to users/roles and configure options such as **Auto open** and **Hide by default** (depending on permissions).

Metadata (file) — System and extracted attributes about a file (for example type, size, timestamps, EXIF/geo, tags) used for filtering and provenance.

Multi-value mapping — A mapping that expands lists/arrays into repeatable properties or one-to-many values.

Ontology — Octostar’s data model. It defines concepts, properties, and relationships and ensures consistent behavior across Search, records, Link Charts, apps, and dashboards.

Ontology Explorer — A Fusion Center tool for browsing concepts, properties, and relationships, including inheritance and definition details.

Ontological CSV Importer — A CSV import workflow that maps columns directly to ontology concepts/properties/relationships to create structured entities.

Open with… — A menu that opens a selected file or item in a specific app (for example “Open with → Entity Extract & Graph”).

Permissions — Rules that determine what a user can see or do. Permissions apply across workspaces, files, records, and actions, and are usually managed via roles and access levels.

Primary key — The identifier used to uniquely represent an entity in a mapping or concept definition (critical for deduplication and updates).

Provenance — Traceability back to source evidence. It links structured facts (records/relationships) to the underlying documents, files, or datasource rows.

RAG (retrieval-augmented generation) — An AI pattern where the system retrieves relevant context from your selected evidence before answering, improving grounding and reducing guesswork.

Record — A structured item representing an entity (an instance of a concept). Records store properties and relationships and open in the Record Viewer.

Record template — A configured layout for displaying or editing records of a concept (for example a profile view or timeline), where supported.

Record Viewer — The UI used to view and edit a record’s fields, relationships, attachments, comments, and tags (based on permission).

Relationship — A typed connection between two entities (for example `works_at`, `located_at`, `communicates_with`). Relationships may include attributes (for example dates, roles, or confidence) depending on the model.

Saved search — A stored Search definition you can reuse or share (where enabled). Saved searches often produce a set that can be visualized or sent to apps.

Schema Editor — An admin tool used to control how ontology properties appear in forms and Search filters (for example field order, validation, visibility).

Search (Global) — Concept-aware search across unstructured evidence and structured entities. Supports filtering, multiple result views, and saving queries.

Semantic search — Search that uses embeddings to retrieve results by meaning rather than exact terms, across text and media.

Set — A collection of items (often records, sometimes files) produced by Search or selections. Sets can be visualized, saved, and used as inputs to apps.

SQL Lab — A Fusion Center tool for ad-hoc SQL queries against datasources (where enabled) for validation and exploration.

Table view — A Search or dataset view that shows results in rows and columns for sorting and scanning.

Tags / Comments — Lightweight annotations applied to files and records for triage and collaboration (subject to permissions).

Template (dataset / record) — A configured visualization bound to a concept or dataset to standardize how results are presented.

Virtual knowledge graph (VKG) — A setup where data remains in original systems but is exposed through ontology mappings as a unified graph for Search and analysis.

Workspace — A secured container for investigations that behaves like a file system and can store evidence files, native Octostar items (Link Charts, saved searches), and workspace records.

Workspace records — Structured records visible within a workspace context, alongside unstructured evidence, to support unified investigation workflows.

User Guide

This guide is designed to help you get started and make the most of the platform’s features. You will learn how to work with **workspaces**, manage **records**, use the **search engine**, explore relationships in **LinkChart**, interact with the **AI chatbot**, and navigate available **applications**. Whether you are new to Octostar or looking to deepen your understanding, this guide provides step-by-step guidance for everyday use.

Getting Started with Octostar

Welcome to Octostar. You use Octostar to run investigations and make decisions using data from many sources, including documents, media, and structured systems. Octostar supports workflows in law enforcement, cybersecurity, corporate investigations, and financial crime analysis.

In this section, you learn where to start and what each part of Octostar is used for. It introduces the areas you use most often — workspaces and files, records and relationships, search and visual analysis — so you can understand what to open, what to create, and where results are saved.

Access the Sign-In Page

Use your organization's Octostar URL to open the sign-in page.

Enter Credentials

Enter your username (or email) and password, then select **Sign in**.

Complete Initial Setup

On your first sign-in, Octostar may prompt you to:

- Accept required notices
- Set a new password
- Configure multi-factor authentication (if enabled)

Access Workspaces

After signing in, Octostar opens the main interface and loads the workspaces you can access. If no workspaces open automatically:

- Select **\+ Open workspace**
- Choose a workspace from the list

Troubleshooting

If You Can’t Find the Expected Workspace

You may not have access. Contact a workspace administrator to request **Read**, **Write**, or **Admin** access, depending on your needs.

Navigating the Interface

Octostar uses a multi-panel layout so you can keep evidence, records, and analysis open at the same time.

Use the **left sidebar** to move between core modules such as **AI Chat**, **Search**, and the **App Launcher**.

Use the **workspace area** to open and browse workspaces, navigate folders, and find files, records, and saved artifacts. Items you open appear in the **main area** as tabs; you can switch between tabs, open multiple items, and dock views side-by-side by dragging a tab to the edge of the window. Use the utility controls (for example notifications and user settings) to manage your session and preferences while you work.

Understanding the Homepage and the App Launcher

The homepage is your starting point after you sign in. It gives you quick access to the work you already have open and the actions you use most often.

From here, you can review your recently opened workspaces and items, reopen saved artifacts (such as Link Charts or saved searches), and jump back into ongoing investigations without browsing folders.

Use the **App Launcher** to find and run the apps available to you. Apps are grouped by purpose (for example import, extraction, analysis, and reporting) and may vary depending on your role and workspace permissions.

When you open an app, it loads in the main area and works against your current context — such as the active workspace, a selected file, or a selected set. You can also launch many apps directly from the **context menu** item using **Open with…**, which starts the app with that item already selected as input.

User Settings & Preferences

Use **User settings** to control how Octostar behaves for your account. From the main interface, open the user menu in the utility area and select **Settings**. Most settings are personal (they apply only to you), but some options may be visible only to administrators.

Typical user preferences include how certain views open and what content is visible. For example, you can choose whether **AI Chat** opens in the current tabset or in a new tabset to the right, and you may be able to show or hide specific content such as **hidden workspaces** or **unlabeled records** (records whose concepts do not have label keys).

Administrators may also see platform-level configuration under **System settings**. These controls affect the whole deployment and can include **Timezone**, whether **AI features** are enabled, map configuration (for example the **Tile map server URL**), geocoding settings (for example the **Photon server URL** and request limits), AI provider configuration (API keys, endpoints, and default models), and file upload restrictions such as global size thresholds and restricted file types. Use these settings with care, because changes can impact all users and workflows.

What is a Workspace?

A **Workspace** is the main place where you store and work on an investigation in Octostar. Think of it as a secured case container that combines a file system with structured investigative data. Inside a workspace, you can create folders, upload evidence files (PDFs, documents, images, audio, video, exports), and save Octostar items such as **Link Charts**, saved searches, and app outputs. A workspace can also contain structured content—**Records** created from the **Ontology** (for example **Person**, **Organization**, **Event**, **Vehicle**, **Place**) and the **Relationships** between them—so you can keep raw evidence and confirmed facts in the same working area.

Workspaces facilitate standard investigation workflows. You can begin by uploading raw evidence, and then use applications to extract entities, import CSV files, or generate reports.

As you review findings, you create or refine records and relationships, attach supporting files to preserve provenance, and save artifacts (charts, datasets, notes) back into the workspace for later reuse.

When you open a workspace, you see both the folder structure for files and the workspace records, which lets you move from a document to the related entities, then into a Link Chart, without changing tools or losing context.

Access to each workspace is managed through permissions assigned to users and roles. Octostar offers three access levels: **Read**, **Write**, and **Admin**.

- **Read** access allows you to open and search content.
- **Write** access enables you to upload and manage files, create and edit records, and contribute tags and comments.
- **Admin** access provides control over workspace membership, sharing settings, and other administrative actions.

If a user lacks access to a workspace, it will not appear in the workspace selection dialog. This permission model effectively separates investigations, supports team collaboration where appropriate, and ensures that all actions align with your organization’s access policies.

Access Levels & Permissions

Workspace access levels are assigned to **users** or **roles**. An access level applies to the workspace and all resources inside it. If a user has no access level for a workspace, that workspace does not appear in the left sidebar.

Octostar provides three access levels:

- **Read**
- **Write**
- **Admin**

A workspace user is any user who has an access level assigned for that workspace. The access level determines which actions the user can perform on workspace resources. Workspaces contain different resource types. Each access level enables a different set of actions.

Special roles

Some roles grant an implicit access level across all workspaces:

- `read_all_workspaces`
- `write_all_workspaces`
- `admin_all_workspaces`

Special usernames

The username `admin` automatically has **Admin** access to all workspaces, even without explicit role or permission assignments.

Managing Users and Roles

You manage user and role membership in the **Fusion Center**.

1. Sign in as a privileged user.
2. Open **Fusion Center**.
3. Navigate to **Manage** → **Access Manager**.
4. Add, modify, or delete users and roles, and update role assignments.

> *Note: “permissions” in this view refer to platform-level permissions (for example access to ontologies and data sources). Workspace access levels are configured on the workspace itself.*

Managing Access Levels for a Workspace

As a workspace admin, you manage access levels from the workspace context menu.

1. Sign in to Octostar.
2. Create or open a workspace where you have **Admin** access.
3. Right-click the workspace and select **Member Access.**.
4. Assign one or more users or roles to an access level (**Read**, **Write**, **Admin**).
5. Save changes.

Uploading Files and Media

Upload evidence to a workspace to make it available for review, search, and analysis. In the workspace tree, open the target folder, then use one of these methods:

- **Upload from the menu:** Right-click the folder and select **Upload file**, then choose one or more files.
- **Drag and drop:** Drag files from your desktop and drop them into the workspace folder.

After you upload a file, Octostar initiates a processing pipeline to make the content usable across the platform. This processing generally involves extracting metadata, such as file type, timestamps, and EXIF data when available. It also includes generating searchable text, using OCR for scanned documents and transcription for audio or video files when enabled. Additionally, AI-based enrichment is applied, including entity extraction and semantic indexing. The output is used to power full-text search, semantic search, AI Chat with context, and apps.

Each file shows a small status indicator in the workspace:

- **Green** — processing completed successfully. The file is ready to search and use in apps.
- **Blue** — processing is in progress. Large files may take longer to finish.
- **Red** — processing failed. The file may not be searchable or usable for AI features. If this persists, contact your Octostar administrator.

> **Tip:** If a file is stuck in **Blue** or remains missing from Search after it turns **Green**, refresh the folder using **Force re-index** (available from the Context Menu) or ask an administrator to check the processing pipeline for errors.

Organizing and Managing Files

Use folders and consistent naming to keep evidence easy to find and review. In a workspace, you can create folders for common case structures (for example **Intake**, **Evidence**, **Exports**, **Reports**, **Media**, **Entities**, **Link Charts**) and move items as your investigation progresses. Keep file names descriptive and stable, especially for items you plan to cite in reports or share with teammates.

To manage items, use the workspace tree and the item context menu (right-click):

- **Create folders** to group evidence by source, date, or theme.
- **Move and rename** files to reflect their role in the case (for example “bank_statements_2025_Q3.pdf”).
- **Tag and comment** on files to support triage (for example *Reviewed*, *Needs translation*, *High priority*).
- **Copy link** to share a deep link to a file with teammates (they still need workspace access to open it).
- Use **Open with…** to run specialist apps on a file (for example extraction, transcription, or mapping), keeping outputs saved back into the workspace.

Because Octostar processes files through an indexing and enrichment pipeline, avoid repeatedly uploading duplicates when possible. If you need to replace a file, keep the investigation workflow in mind: update links, tags, and any records that reference the earlier version, and re-run processing if required (for example via **Force re-index**).

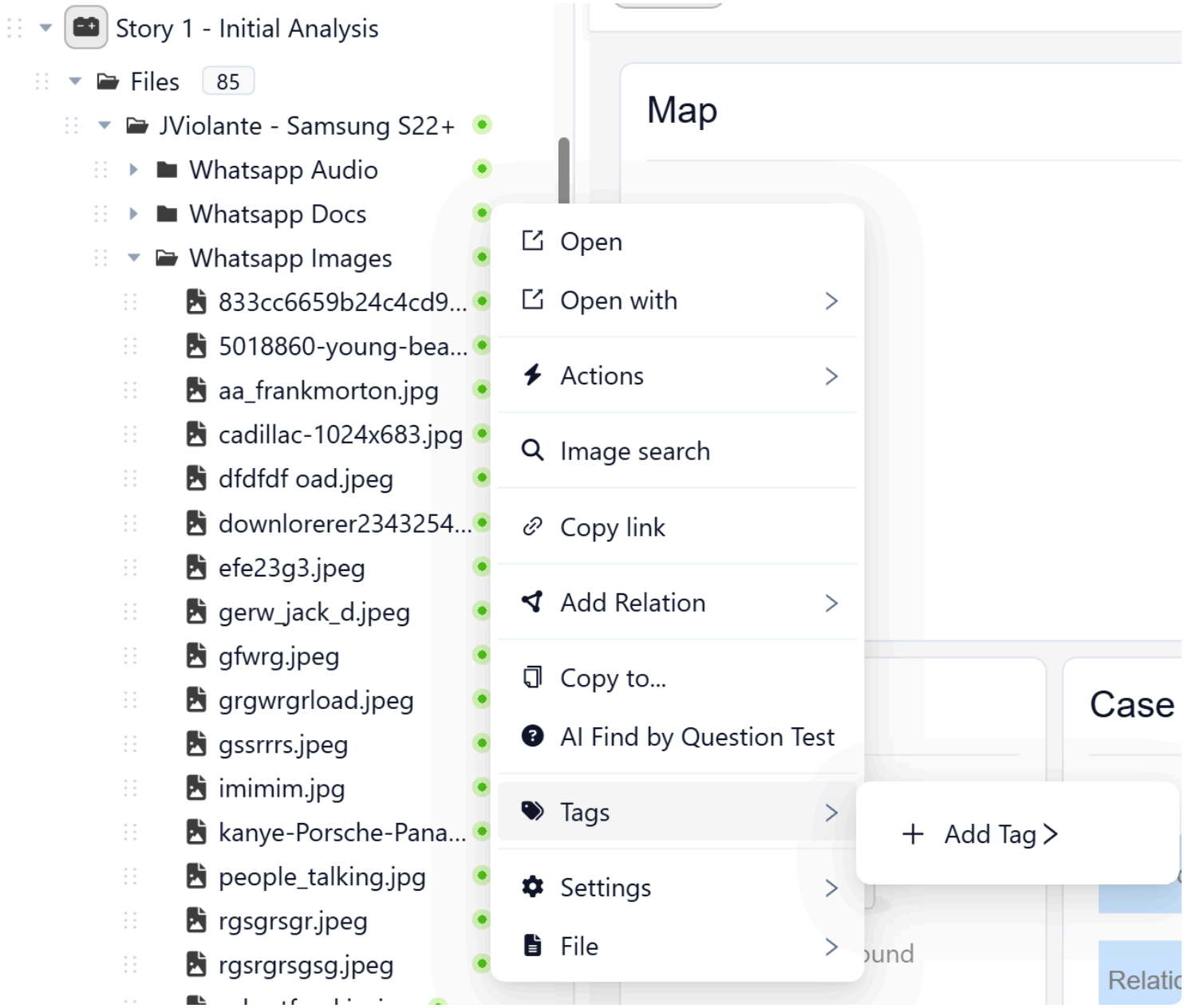
Tagging and Comments

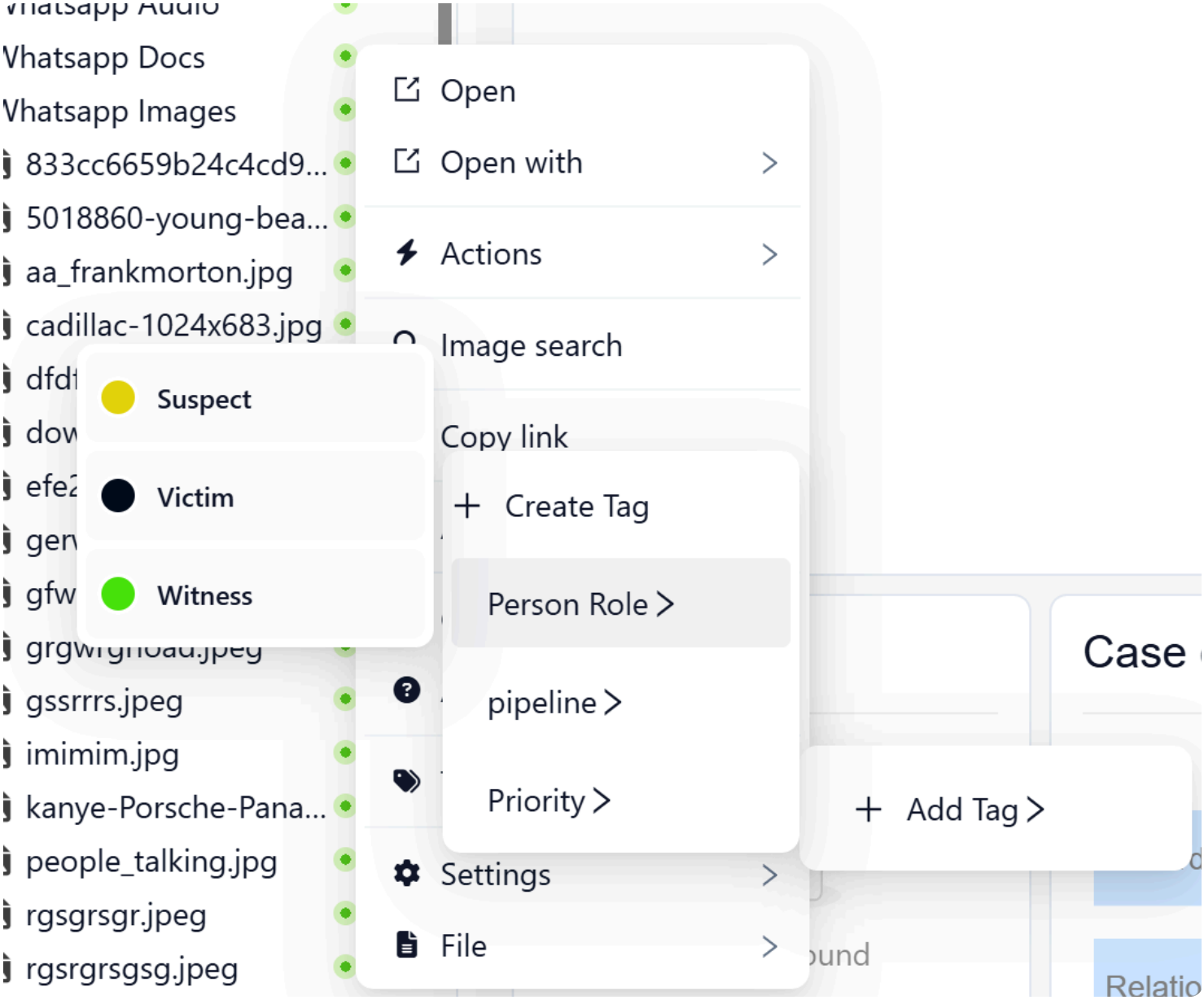
Tags are short, structured labels you attach to a file or record to communicate meaning at a glance. They provide a quick, visible signal about the status, relevance, or risk of an item — without needing to open it. Use tags to:

- Indicate **status** (e.g., `Reviewed`, `In review`)
- Highlight **priority** (e.g., `High priority`)
- Flag **risk or suspicion** (e.g., `Suspicious`, `POI`)
- Assign **next steps** (e.g., `Needs translation`, `Follow up`)

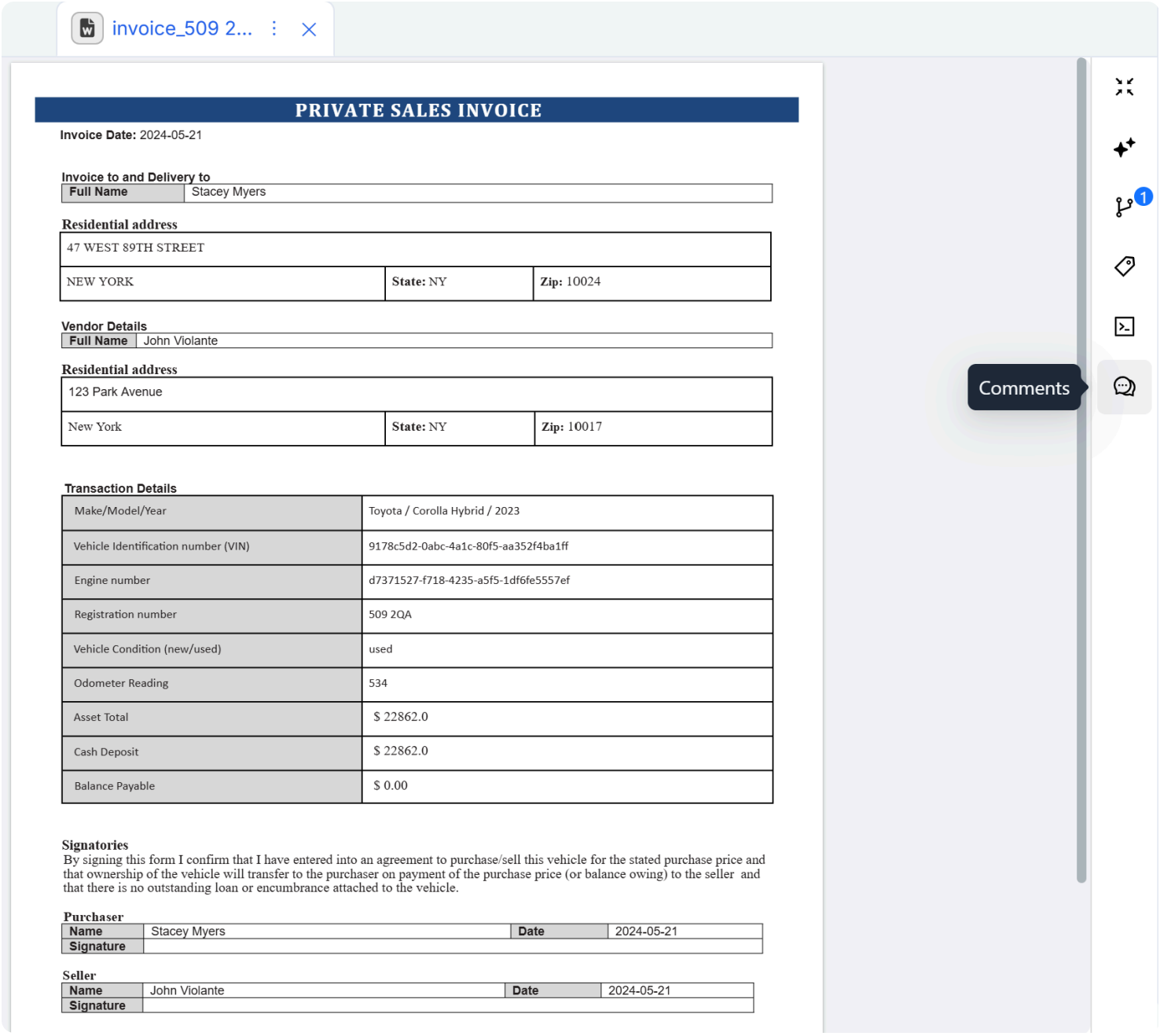
Tags solve those by making triage visible:

- **Reduce duplicate work**: “Reviewed” tells others not to re-check unnecessarily
- **Focus attention**: “High priority” and “Suspicious” surface the right items first
- **Coordinate teams**: “Needs translation” or “Needs peer review” routes work to the right people
- **Create an audit trail of collaboration** (especially when paired with comments): what was flagged, when, and by whom


Story 1 - Initial Analysis
Files
85
JViolante - Samsung S22+
Whatsapp Audio
Whatsapp Docs
Whatsapp Images
833cc6659b24c4cd9...
5018860-young-bea...
aa_frankmorton.jpg
cadillac-1024x683.jpg
dfdfdf oad.jpeg
downlorerer2343254...
efe23g3.jpeg
gerw_jack_d.jpeg
gfwrg.jpeg
grgwrgrload.jpeg
gssrrrs.jpeg
imimim.jpg
kanye-Porsche-Pana...
people_talking.jpg
rgsgrsgr.jpeg
rgsrgrsgsg.jpeg
Open
Open with
Actions
Image search
Copy link
Add Relation
Copy to...
AI Find by Question Test
Tags
Settings
File
Add Tag
Map
Case



Comments are free-text notes attached to a file or record to add context that a tag alone can't capture. Use a comment to explain *why* something was tagged or to provide instructions ("Flagged as suspicious because the metadata timestamp conflicts with chain-of-custody log").



Best practice: Use tags for clear, standardized signals. Use comments for details and justification.

Working with Records

Understanding and Working with Records in Octostar

In this section, you'll learn about **Records** and how to effectively manage them in Octostar. Here's what we'll cover:

- **Browsing Records**: Navigate through the record tree to find specific records.
- **Visualizers and Cards**: Understand how these tools present key information.
- **Creating and Using Sets**: Learn to create **Sets** of records that share the same visualizers.
- **Bulk Uploading**: Discover how to upload multiple records at once.
- **External Records**: Explore how records from external systems are integrated and function within the platform.

What is a Record?

A **record** in Octostar is a structured item used to investigate various entities, such as:

- Person
- Organization
- Vehicle
- Event
- Place
- Object

Creation and Structure

Records are created from the **Ontology**. Each record is an instance of a specific *concept*, featuring:

- **Fields (Properties):** Defined by the concept.
- **Links (Relationships):** Connect the record to other records.

Usage

- **Record Viewer:** View and edit records.
- **Search:** Use records as results.
- **Link Chart:** Explore connections between records.

Supporting Evidence

Records can reference supporting evidence through:

- **Attachments:** Include relevant files.
- **Provenance:** Trace key facts back to source files or mapped data.

Creating and Editing Records

Create a record when you need a structured profile for something you are investigating—such as a person, organization, vehicle, event, or place. Records follow the **Ontology**, so the available fields and relationship types depend on the concept you choose.

Before you begin

- You need **Write** access to the workspace to create or edit records.
- If your organization uses connected data sources, some records may be read-only because they are mapped from an external system.

Create a record

1. Open the workspace where you want to store the record.
2. In the workspace tree, open the folder where you keep case records (or create one).
3. Select **New Workspace Record** and choose the concept (for example **Person** or **Organization**).
4. Enter the required fields. Start with the identifiers you trust most (for example full name, document ID, phone number).
5. Select **Save**. **What you should see:** The record opens in the **Record Viewer** with its fields and related sections (relationships, attachments, tags/comments). The record is also available in Search under its concept.

Edit a record

1. Open the record (from Search results, the workspace tree, or a Link Chart).
2. Select **Edit**.
3. Update fields as needed.
4. Select **Save**.

What you should see: Updated values are visible in the record and reflected in Search and Link Charts.

> **Important:** If you cannot edit a record, you may have **Read** access only, or the record may come from a connected datasource and be configured as read-only. In that case, ask an administrator to confirm the mapping and write-back settings.

Tips

- Use consistent identifiers (for example a single canonical name format) to reduce duplicates.
- Add relationships as you confirm them so Link Charts reflect your latest understanding of the case.
- Attach key evidence files to the record to preserve provenance.

The Record Viewer

The **Record Viewer** is a tool for inspecting and, if permitted, updating a record’s details. It organizes the record into a structured profile based on the **Ontology**. Here's how it works:

Structure

- **Sections**: Fields are grouped into sections such as identifiers, attributes, and contact details.
- **Panels**: Related information is organized into dedicated panels like **Relationships**, **Attachments**, **Comments**, and **Tags**.

Features

- **Linked Records**: Open and explore linked records.
- **Relationships**: Add or edit relationships.
- **Attachments**: Attach supporting evidence.
- **Provenance Review**: Trace key facts back to source files or mapped data.

Access Levels

- **Write Access**: Switch to edit mode to update fields and save changes. **Read Access**: Review the record, open linked entities, and use the record as a starting point for Search, Link Charts, and AI Chat with context.

Record Properties and Fields

A record stores information as **fields** (also called *properties*) defined by the **Ontology**. The concept of the record (e.g., **Person** or **Organization**) determines:

- **Available Fields**: Specifies which fields are available.
- **Data Types**: Defines acceptable data types such as text, number, date, location, and lists.
- **Required Fields**: Identifies which fields are mandatory.

This structure ensures records remain consistent across the platform, allowing Search filters, dashboards, and apps to interpret data uniformly.

Record Viewer

In the **Record Viewer**, fields are typically grouped into sections for easy scanning and updating. Note the following:

- **Read-Only or Hidden Fields**: Some fields may be read-only or hidden based on your permissions and organizational configuration (e.g., rules set in the Schema Editor).
- **Missing Fields**: If a field is missing, it might not be part of the concept in the Ontology, could be configured as hidden, or the record might originate from a connected system with restricted editing capabilities.

Relationships Between Records

Relationships connect records to model how entities relate to each other in an investigation. These connections are essential for understanding complex interactions.

Definition

A relationship is a typed link defined by the **Ontology**. Examples include:

- `works_at`
- `located_at`
- `communicates_with`

These links connect a source record to a target record.

Functionality

Relationships power Link Charts and many cross-entity searches. They enable you to move consistently from “a person” to “their associates, organizations, locations, events, and assets.”

Creation

You can create relationships from contexts such as:

- **Record Viewer** (e.g., from a **Related** section)
- **Link Chart** (draw or add a link between nodes)
- App output (e.g., entity extraction proposing links from a document)

Attributes

Depending on configuration, a relationship may include attributes such as:

- Dates
- Roles

- Notes
- Confidence

Best Practices

Always link relationships back to evidence where possible (attachments, comments, or provenance) to ensure connections are traceable and auditable.

The Search Engine

The Search Engine enables you to quickly locate records and information across your workspace. This section explains how to perform **basic** searches, refine results using **filters**, and use **advanced** search options to narrow down results with greater precision.

Basic Search

The search engine is Octostar's interface for finding records across your data. It runs as a standalone application embedded inside Octostar. Under the hood, it is powered by OpenSearch, which indexes every record in the system for fast full-text and semantic retrieval.

Interface layout

When you open the search view, you see:

- A **search bar** at the top
- A collapsible **filter sidebar** on the left
- A **results area** filling the rest of the screen Type a query in the search bar and results appear with a short debounce delay.

Search modes

The search engine supports four modes:

Text (default)

Full-text search that breaks your query into terms and matches them across record fields using a layered relevance strategy:

1. Exact matches on the record's display name (highest score)
2. Phrase matches
3. All-terms-present matches
4. Broad search across all fields Wrap part of your query in quotes to search for an exact phrase.

Boolean

Use explicit operators (AND, OR, NOT) for precise query construction.

Semantic

Uses vector embeddings to find records conceptually similar to your query, even without exact word matches. The system converts your search text into a numerical embedding and runs a nearest-neighbor search against pre-indexed embeddings.

Auto

An AI model reads your query and selects the most appropriate mode. A natural-language question might route to semantic search, while a keyword-heavy query routes to text mode. Auto mode shows you which mode it selected, and you can override it.

View modes

Display results in three view modes:

- **Grid** -- cards in a responsive masonry layout that adjusts the number of columns to your screen width
- **List** -- one result per row with more detail per item
- **Table** -- traditional columnar format with default columns for label, Workspace, last modified, created by, size, and notes. Add extra columns from the Concept's properties, resize them, or remove them. Column preferences are saved locally and persist between sessions. Switch between views at any time using the segmented control above the results.

Result rendering

Each result is rendered using a template system that adapts to the record's Concept type. Templates control what information appears: display name, icon or thumbnail, Workspace badge, and Concept-specific properties. Search terms are highlighted in the results. Results are paginated at 20 items per page, with navigation controls at both the top and bottom of the results area.

Selecting results

Select one or more results by clicking their checkboxes (in grid and list views) or row selectors (in table view). Hold **Ctrl** or **Cmd** and drag across results to draw a selection rectangle. When you have a selection, a toolbar appears showing the count of selected items and offering actions, including an **Ask AI** button that opens a chat session with your selected records as context. If the search was opened in a modal context (for example, when another part of Octostar asks you to pick records), clicking a result selects it directly rather than opening it, and a submit button sends your selection back to the calling view.

Image-based search

When the image detection API is available, a camera button appears in the search bar. Click it to open a full-screen modal where you upload an image. The system finds visually similar records using face detection and image similarity algorithms.

Saved searches

Save a search by clicking the **Save** button. Octostar captures the current query (the full OpenSearch query that was last executed) and stores it as a Workspace item. An AI model generates a descriptive name for the saved search based on the query contents. Reopen saved searches later to re-execute the same query.

Filters and Advanced Search

The filter sidebar narrows results without changing your search text. It sits on the left side of the search view and can be collapsed when not needed.

Workspace Filter

Restrict results to one or more of the Workspaces you have access to. Select Workspaces at the top of the filter sidebar.

Concept Filters

Limit results to specific record types. Preferred Concepts (configured by your administrator) appear first for quick access. An expandable section reveals all available Concepts. Each Concept shows a match count so you can see at a glance where your data lives. Selecting a Concept automatically includes its child Concepts. For example, filtering by "Person" also matches "Employee" and any other specializations.

Property-level Filters

Within each Concept, drill down further with filters generated dynamically from the Concept's schema. The filter widget adapts to the property type:

- **Numeric properties** -- range sliders
- **Date properties** -- date-range pickers
- **Text or categorical properties** -- multi-select lists, tag clouds, or search-within-property inputs

Active Filters

Active filters appear as removable badges above the results area. Review them at any time to see which constraints are in play. Click a badge to remove that filter.

Combining Filters with Search Modes

Filters operate independently of the active search mode, allowing flexibility in refining your results. Here's how to effectively combine filters with various search modes:

1. **Apply Filters**:
    - Use Workspace, Concept, and property-level filters to refine your search.
    - These can be layered on top of text, boolean, semantic, or auto mode queries to progressively narrow down results.
2. **Boolean Mode for Precision**:
    - Switch to **Boolean** mode for precise query construction.
    - Combine operators (AND, OR, NOT) in the search bar.
    - Use the filter sidebar to constrain results by Workspace, Concept, or property values.

Examples:

- **Example 1**: You need to find all employees who joined after 2020 and work in the marketing department. Use a date-range filter for the joining date and a Concept filter for the department. In Boolean mode, combine these with an AND operator to get precise results.
- **Example 2**: Searching for documents related to both "Project X" and "Project Y" across multiple Workspaces. Apply Workspace filters to include only relevant Workspaces. Use a text query with OR operator in Boolean mode to capture documents related to either project.

The Link Chart

LinkChart provides an **interactive visual representation** of relationships between records, allowing you to explore and analyze connections within your workspace. This section explains how to open and load data into **LinkChart**, navigate the chart interface, add or remove nodes, expand relationships, apply filters and highlights, and customize styling and layout options.

Opening and Loading Data into the Link Chart

A *Link Chart* is an interactive graph visualization for exploring the connections between records. Instead of viewing records one at a time, a Link Chart places them on an infinite canvas as nodes and draws their Relationships as edges.

Link Charts as Workspace items

A Link Chart is itself a Workspace item. Save it, reopen it later, and share it with others through Workspace permissions, like any other file or record. When you save a Link Chart, Octostar persists the nodes and their positions, edges, groups, camera position, and active visual styles. When you reopen it, the chart appears exactly as you left it.

Draft Link Charts

Draft Link Charts are temporary, unsaved explorations. Work with a graph without committing it to the Workspace, and save it only if the exploration proves useful. Unsaved nodes (records you have created on the canvas but have not persisted yet) are visually distinguished so you always know what still needs saving.

Loading data

Build a Link Chart by adding records to the canvas. You can:

- Search for records from the toolbar
- Drag and drop records from a Workspace browser or search results
- Paste records from the clipboard When you add records, Octostar can automatically discover and draw the Relationships between the new nodes and any records already on the canvas, so the graph fills in around your starting point.

Saving Your Link Chart

To save your current work, click **Save** in the toolbar. Octostar will store the entire chart, including:

- Nodes
- Edges

- Positions
- Groups
- Camera settings
- Visual styles

Every significant action on the Link Chart—such as adding, deleting, moving, grouping, and laying out—is recorded in an undo stack. This allows you to navigate through your editing history using **Undo** and **Redo** options, which are also accessible via standard keyboard shortcuts.

Navigating the Chart: Pan, Zoom, Select

Pan and zoom

Navigate the canvas using standard pan and zoom controls. Floating controls in the corner provide zoom buttons and a **Fit to Canvas** option that reframes the view to show all nodes. A minimap provides an overview of the entire graph. Both node and edge labels follow a level-of-detail system: labels hide automatically when you zoom out far enough and reappear as you zoom back in.

Selecting items

- **Click** to select a single item
- **Shift + drag** to draw a rectangle and select everything inside it
- Use the **freeform lasso tool** to draw an arbitrary selection boundary
- Press **Ctrl+A** to select everything

Once you have a selection, you can:

- Move items by dragging
- Delete them
- Group them
- Inspect them in the sidebar If exactly two nodes are selected, right-click to create a new Relationship between them from the context menu.

Sidebar integration

When you select a node, the right sidebar can show that record's full details, Relationships, and metadata. A table view of the current selection is also available, with rows colored by Concept to match the graph's visual scheme. Selecting entities in a table or search results view can update the Link Chart's selection, making it easy to move between the graph and other views.

Context menu

The right-click context menu adjusts based on your selection:

- **Empty area** — Search for records or create a new one.
- **Single node** — Access standard record actions (open, edit, etc.) and a **Refresh** option to reload the entity's data.
- **Two selected nodes** — Access a submenu of available Relationship types to connect them directly.

Adding and Removing Nodes

Adding nodes

Add records to the canvas from the toolbar, by dragging and dropping from a Workspace browser or search results, or by pasting from the clipboard. Each node displays the record's display name and an icon resolved from its Concept. If the record has an associated image, the node can display that instead.

Creating records from the canvas

Right-click on an empty area of the canvas and select the option to create a new record. The new record appears as an unsaved node, visually distinguished from persisted nodes until you save.

Removing nodes

Select one or more nodes and delete them using the toolbar **Delete** button, the context menu, or the keyboard shortcut. Removing a node from the canvas does not delete the underlying record from the Workspace -- it only removes the node from the current Link Chart.

Creating Relationships between nodes

Select exactly two nodes, right-click, and choose a Relationship type from the context menu submenu. The new edge appears immediately on the canvas.

Expanding Relationships

Automatic Relationship discovery

When you add records to the canvas, Octostar can automatically discover and draw the Relationships between the new nodes and any records already present. This means the graph builds outward as you add data.

Edges

Edges are labeled with the Relationship name. When multiple Relationships exist between the same pair of records, they are drawn as arcs to avoid overlap. Edge labels follow the same level-of-detail system as node labels: they hide when you zoom out and reappear as you zoom in.

Browsing Relationships from the sidebar

Select a node to view its Relationships in the right sidebar. From there, you can add connected records to the canvas to expand the graph further.

Filtering and Highlighting

Groups (combos)

To organize complex graphs, group nodes into collapsible clusters called *combos*. You can:

- Create a single named group manually
- Let Octostar auto-group nodes by their Concept type Combos display a label, an icon, and a count of the nodes they contain. Collapse a combo to reduce visual noise or expand it to reveal its contents.

Cropping

Use the **Crop** button in the toolbar to reduce the graph to only the currently selected items. Everything outside the selection is removed from the canvas. This is useful for isolating a subgraph after selecting the nodes you want to focus on.

Interaction highlights

Node and edge styles respond to interaction:

- Hovered items gain a halo
- Selected items receive a larger halo
- Items highlighted for detailed inspection are wrapped in a hull decoration

Searching within the chart

Use the **Search** field in the toolbar to find nodes by name within the current Link Chart. Matching nodes are highlighted on the canvas.

Styling and Layout Options

Visual styling

Visual styling uses a system of *stylers*. The default styler, **Color by Type**, assigns a consistent color to each Concept so you can read the structure of the graph at a glance. All "Person" nodes share one hue, all "Organization" nodes share another, and so on. Custom stylers can be layered on top. Toggle them through the **Style Picker** in the toolbar.

Node and edge appearance

Each node shows the record's display name and an icon resolved from its Concept. If a record has an associated image, the node displays that instead. Edges are labeled with the Relationship name and drawn as arcs when multiple Relationships exist between the same pair of records.

Layout algorithms

Octostar offers several automatic layout algorithms:

- **Graceful** (default) -- a custom force-directed layout that produces balanced, readable graphs
- **Grid**
- **Hierarchical (Dagre)**
- **Force-directed variants**
- **Radial**
- **Random** Switch layout algorithms from the toolbar. For very large graphs, the system automatically optimizes its rendering strategy to maintain responsiveness.

Groups and layout

Groups (combos) interact with layout algorithms. When you apply a layout, grouped nodes are positioned together. Collapse a group before applying a layout to treat it as a single unit in the arrangement.

AI Chat Overview

AI Chat enables interaction with Workspace data using natural language. It retrieves relevant content from indexed documents and generates responses based on that evidence. Here's how you can leverage AI Chat:

- **Explore Information**: Delve into data to gain insights.
- **Understand Context**: Grasp the background and implications of data.
- **Identify Connections**: Discover relationships between different data points.
- **Guide Investigative Analysis**: Use AI Chat to support thorough investigations.

Key Sections

1. **How AI Chat Works**: Understand the mechanics behind AI Chat.
2. **Asking Effective Questions**: Learn techniques for obtaining the best responses.
3. **Capabilities and Limitations**: Know what AI Chat can and cannot do.
4. **Responsible Use**: Guidelines for using AI Chat within an investigative workflow.

Introduction to the AI Chatbot

Workspace Overview

Your **Workspace** can house various materials, including documents, images, videos, emails, chat exports, and structured **Records**. As the volume increases, locating the right evidence and understanding its relevance to your case becomes more challenging.

Introducing AI Chat

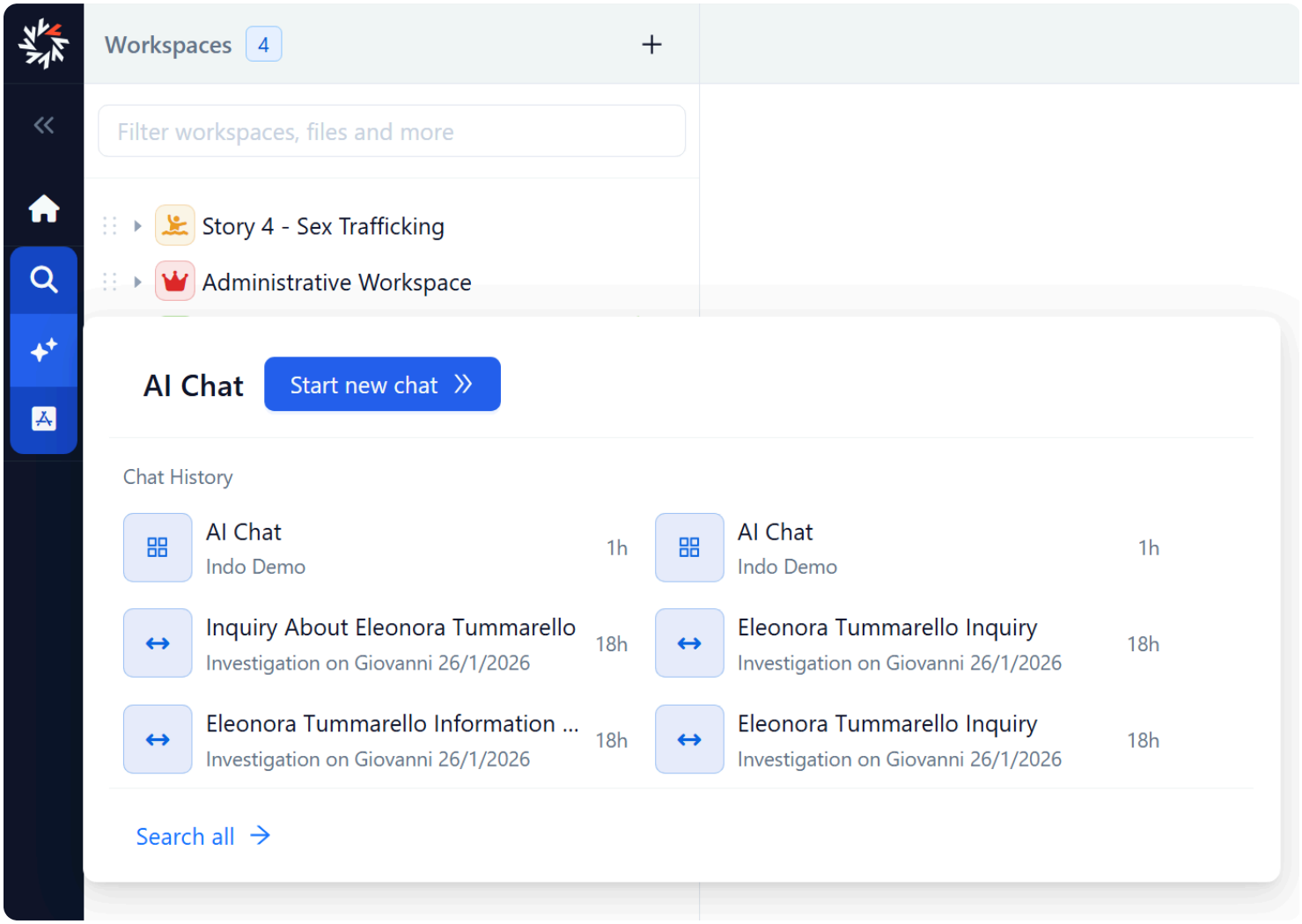
AI Chat allows you to interact with your data using natural language. It helps you to:

- Summarize material
- Extract key facts
- Compare sources
- Identify connections for review in a **Record** or a **Link Chart**

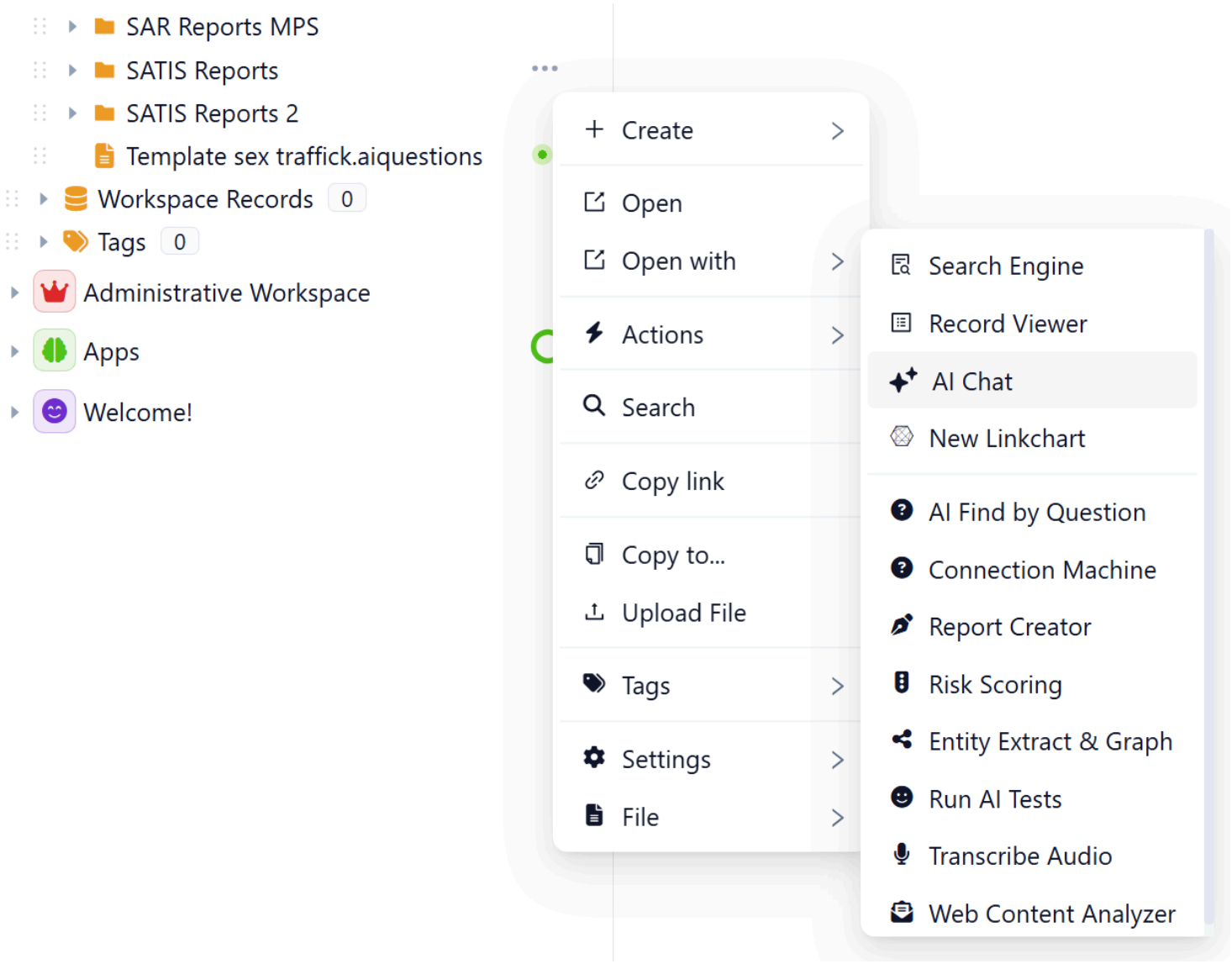
Accessing AI Chat

You can open **AI Chat** from:

- The sidebar on the left



- The context menu of folders or single documents



How AI Chat answers

AI Chat uses *Retrieval-Augmented Generation (RAG)*. When you enter a question, the system:

- Retrieves relevant content from OpenSearch using keyword and vector search
- Sends the retrieved content to the language model as context
- Generates a response based on that retrieved context

AI Chat respects permissions. It can only use content you are authorized to access. If you cannot open an item directly in Octostar, AI Chat should not expose it in its response.

How to use AI Chat safely

AI Chat supports investigative work. It does not replace professional judgment. Treat responses as a starting point for review.

When you rely on an answer:

- Review the response for scope and assumptions.
- Check the references or sources section.
- Ask for citations when you need traceability.
- Open the referenced documents and confirm the details.
- Validate critical information before you take action or include it in a report.

AI Chat links responses back to source documents to support traceability and follow-up. Use those links to move from a summary or claim to the original evidence.

How to Ask Effective Questions

The quality of an **AI Chat** answer depends on how clearly you define what you need. Use the guidance below to improve accuracy and make outputs easier to verify.

Be specific about scope

Avoid broad prompts that do not define a topic, time window, or source set.

- Avoid: “Tell me about this case.”
- Use: “Summarize the private vehicle sales mentioned in these WhatsApp invoices and identify the main individuals involved.”

Add details that help the system retrieve the right evidence:

- Names of people, companies, or organizations
- Date ranges (or a start and end date)
- Identifiers (phone numbers, IBANs, license plates, email addresses)
- File or folder context (for example “this folder” or “these selected documents”)

Examples:

- “In which documents does Valerio Simoni appear, and in what context?”
- “Which **Relationships** connect this company to luxury vehicle sales in 2023?”
- “List all phone numbers and IBANs mentioned in the selected files, grouped by source document.”

Define the output you want

State what you want AI Chat to produce. If you do not specify an output format, you may get a narrative answer that is harder to review.

Common output types:

- A summary (short or detailed)
- A structured list (entities, identifiers, events)
- A comparison (differences across documents or versions)
- A list of **Relationships** (who is connected to whom and why)
- A timeline (ordered by date)

- An “open questions” list (gaps, contradictions, missing evidence)

Examples:

- “Summarize all references to private vehicle sales in this folder. Output: bullet list with date, seller, buyer, amount, and source file.”
- “How are these two **Records** connected? Output: the shortest connection path and the supporting sources.”
- “What unanswered questions remain in this dataset? Output: top 10 questions and which document(s) might answer each one."

Use AI Chat for interpretation, not exact counting

AI Chat is strongest when you need interpretation and synthesis across sources. Use it to:

- Interpret context and meaning across documents
- Connect **Records**, **Concepts**, and **Relationships**
- Summarize long text or many documents
- Extract themes, patterns, and inconsistencies
- Propose follow-up lines of inquiry For exact counts or exhaustive totals across large datasets, use **Search** and filters where possible. Language models respond based on retrieved context and do not guarantee complete numerical aggregation unless the full relevant set is explicitly retrieved and reviewed.

Examples of safer phrasing:

- “Based on the retrieved sources, list the transactions you can cite. Do not estimate totals.”
- “If you cannot confirm a count from the sources, say so and list what you found.”

Ask for citations and verify For sensitive findings or anything that goes into a report, ask for traceability. Use prompts like:

- “Provide citations for each statement.”
- “For every name, amount, and date, cite the source document and the passage.” Then verify:
- Review the references section in the response.
- Open each cited file or **Record** and confirm the detail in the original content.
- Treat summaries as pointers. Do not rely on them without checking the underlying evidence.

Iterate in small steps

When a question is complex, break it into two passes:

1. Extract facts with citations
2. “List all vehicle sale transactions with date, parties, and amount, with citations.”
3. Analyze patterns

4. “Based on the cited transactions, identify the top three individuals involved and how they are connected.” This approach makes results easier to validate and reduces missed details.

What the Chatbot Can and Cannot Do

What AI Chat Can Do

AI Chat helps you work faster with the data that is already available to you in Octostar. It can:

- Retrieve and summarize indexed content in your **Workspace.**
- Use it to summarize documents, extract key facts, and highlight relevant passages.
- Analyze enriched evidence outputs.
- It can use transcripts, extracted text (for example OCR), metadata, and other structured annotations created during ingestion and processing.
- Connect structured data across the platform.
- It can reason over **Record**, **Concept**, and **Relationship** data to explain how entities relate and where supporting evidence appears.
- Perform semantic retrieval.
- It can retrieve relevant content using keyword search and vector search (embeddings), which helps when the wording varies across sources.
- Provide traceable answers.
- It can reference the source documents and records used to generate the answer, so you can open and verify the underlying evidence.
- Use enabled tools when configured.
- If your deployment supports agent workflows, AI Chat can use available tools (for example launching an app or preparing an export) within the limits of your permissions. Example prompts:
- “Summarize all WhatsApp conversations related to luxury vehicle sales and list recurring sellers. Provide citations.”
- “How is this **Record** connected to the suspect’s phone number? Show the **Relationship** path and cite sources.”
- “List all IBANs and phone numbers mentioned in these documents, grouped by file.”

What AI Chat cannot do

AI Chat operates within strict constraints. It cannot:

- Access systems that are not integrated or enabled.
- If a database, external service, or workspace content is not connected to Octostar, AI Chat cannot retrieve it.
- Override permissions.
- It cannot access content you do not have permission to open. If you cannot open a file or record, AI Chat should not expose it.
- Guarantee exact statistical counts across large datasets.
- It answers using retrieved context. It may miss items that were not retrieved, and it should not be treated as a reliable counting engine for totals unless you can verify the full set.
- Replace validation or legal review.
- It does not replace investigative judgment, chain-of-custody practices, or formal legal review.
- Analyze content that is not ingested and indexed.

Your responsibility

You remain responsible for the final interpretation and use of results. For critical findings:

- Ask for citations.
- Open the referenced documents or **Records**.
- Confirm names, dates, amounts, and identifiers directly in the source.
- Document how you validated the conclusion before acting on it.

Using the Chatbot in Investigative Workflows

Use **AI Chat** as a support tool during an investigation. It helps you summarize, connect, and prioritize evidence, but you still verify every critical detail in the source.

Start with an overview

Scenario: You review WhatsApp documents that contain private sales invoices for luxury vehicles linked to a suspect's phone number.

1. Open **AI Chat**.
2. Add the relevant folder or documents as context.
3. Ask a scoped question, for example:
4. "Summarize all references to private vehicle sales in these WhatsApp documents and identify the main individuals involved. Provide citations." Use the response to:

- Identify recurring names, companies, and identifiers (phone numbers, IBANs, plates).
- Pull out transaction details (amounts, dates, locations) that you want to verify.
- Detect patterns in conversation (repeated sellers, brokers, intermediaries).
- Flag potential financial **Relationships** that you should model as records. Then validate:
- Review the references section in the response.
- Open each cited file or **Record**.
- Confirm names, dates, and amounts directly in the source before you write notes or brief others.

Deepen the analysis with Records and Link Charts

After the overview, move from summary to structured investigation.

1. Create or open the relevant **Records** (for example people, organizations, vehicles).
2. Add or review **Relationships** that are supported by evidence.
3. Open a **Link Chart** to visualize connections and expand from key entities.

Use follow-up questions in **AI Chat** to narrow scope and test hypotheses:

- "Which **Relationships** connect these Records between `2023-01-01` and `2023-06-30`? Provide citations."

- “List the phone numbers and IBANs that appear in both the invoices and the chat messages, grouped by source.”
- “Which entities appear in multiple files, and what evidence supports each connection?”

Keep the loop tight:

- Ask one focused question.
- Review citations.
- Update records or the Link Chart.
- Repeat.

Use AI Find by Question Application for Document-By-Document Extraction

Use **AI Find by Question** when you need structured answers per document, not a single aggregated narrative. More information can be found AI Find By Question (AI Find By Question).

Use **AI Chat** for:

- A quick overview
- Cross-document reasoning
- Exploratory questions and hypothesis building
- Suggested connections to investigate

Use **AI Find by Question** for:

- High precision extraction
- Yes/No checks per document
- Structured outputs (columns per question)
- Document-by-document validation
- Exportable tables (CSV)

Example workflow:

1. Use **AI Chat** to identify what matters and which fields you need.
2. Switch to **AI Find by Question** and define questions as columns, such as “Is there a vehicle sale mentioned?”, “What is the sale amount (EUR)?”, “Who is the seller?”, “Who is the buyer?”, and “What phone numbers or IBANs are listed?”
3. Run extraction and review the table.
4. Export results and attach them to your case workspace.

For advanced usage, see the AI Find by Question section in this guide.

Verification checklist

Use the same verification steps in both tools:

- Review the references section.
- Ask for citations when the answer includes key claims.
- Open the original files and **Records** to confirm details.
- Treat conclusions as unverified until you can point to the source evidence.

Applications - User Guides

This section presents the **applications** available within Octostar and explains how they support different investigative and analytical workflows. Each application offers distinct functionality and features designed for specific tasks, allowing you to choose the right tool based on the scope and objectives of your investigation. Every application includes a dedicated guide in this section. All guides follow a consistent structure which is highlighted in the *How to Read an App Guide* section, so once you are familiar with one guide, you can easily navigate the others.

For detailed information about application architecture and development, please consult the **Developer Guide**.

How to Read an App Guide

Each application in Octostar has its own dedicated guide in this section. All guides follow the same structure, so once you are familiar with one, you will know how to navigate any of them.

Guide Structure

1. **Overview**

Explains in plain terms what the application does and what kind of output it produces.

2. **When to Use This Application**

Helps you quickly decide whether this is the right tool for your current task. Octostar includes many apps, and choosing the right one upfront saves time.

3. **Before You Begin**

Tells you what to prepare before opening the app: file types you will need, access permissions required, or data that should already exist in your workspace. Reading this section first can prevent interruptions mid-workflow.

4. **Step-by-Step Walkthrough**

Mirrors the sequence of actions you perform inside the application. Each step describes both what you need to do and what the application does in response, including any settings or decisions you will encounter along the way. You do not need to read the entire guide before using an app — the walkthrough is designed to be followed in parallel with the application itself.

5. **Understanding the Output** and **Saving and Exporting Results**

Explain what you see after the main action completes and how to preserve or share your work.

6. **Tips for Best Results** and **Known Limitations**

It is worth reading both, especially the limitations, before relying on any output for an official document or investigation record.

A note on AI-powered applications.

> Several apps in Octostar use artificial intelligence to extract, generate, or classify information. In all such cases, the output is a starting point, not a finished product. The relevant guide will tell you explicitly what requires human review and where the AI is most likely to make mistakes.

Entity Extract & Graph

Overview

Entity Extract & Graph reads one or more evidence files and extracts *entities* (for example people, organizations, locations, vehicles, and events) and their *relationships*. You use it to turn unstructured content into structured **Records** and an optional **Link Chart** you can review, edit, and save back into a **Workspace**. Inputs include documents, images, and audio files. Outputs include extracted entities/relationships, saved workspace records, and an exported graph. AI extraction is a starting point and requires human review, especially for relationships.

When to Use This Application

- You need a first-pass list of people, organizations, places, and identifiers mentioned in a document set.
- You want to turn a document (or folder of documents) into structured **Records** you can search and reuse.
- You want to generate a **Link Chart** quickly to explore connections without manual data entry.
- You need to compare evidence across multiple files and identify repeated entities and actions.
- You want to link extracted entities to existing records to reduce duplicates and improve data quality.

Before You Begin

- Confirm you have access to the **Workspace** where the source files live and where you plan to save outputs.
- Prepare the evidence you want to analyze:
  - Documents (for example PDFs and text files)
  - Images
  - Audio files
- If you plan to save results, decide which **Workspace** and folder path you will use for output.

Step-by-Step Walkthrough

Step 1 — Load your documents

Drag and drop content into the dropzone when the app opens. You can load:

- One or more files
- A folder (to process all files inside)
- A workspace (to process workspace-level content) After files load, the app shows a file list with a status for each item:
- ✓ — File loaded and ready to process
- ⚠ — File has no readable content
- ✗ — File could not be read If a file shows ⚠ or ✗, review that the file is correctly indexed, if it is not, use the **Force Re-Index** Action.

Step 2 — Configure extraction options (optional)

Before extraction, you can set two options to control how the app reads and prioritizes content.

- **Extraction Goal:** enter a short description of what you are looking for. This focuses the extraction on your investigative objective.
- Examples:
  - “Focus on financial transactions.”
  - “Find supply chain actors.”
- Leave this blank for a general extraction across all entity types.
- **Deep Mode:** enable this option when you want a more exhaustive extraction. Deep Mode can surface more entities and relationships, but it can also increase noise. Use it when completeness matters more than precision.

Step 3 — Run the extraction

Select **Extract Entities** to start processing. The application:

- Reads the loaded files
- Extracts entities and relationships
- Updates the graph visualization while extraction is running While extraction runs, you can monitor:
- A live counter of entities and relationships found
- The graph updating in real time To stop early, select **Stop**. Processing time depends on the number of files and their size.

Step 4 — Review entities

Open the **Entities** tab to review and refine extracted entities. Entities appear as cards (10 per page). Use filters to focus your review:

- Filter by concept type (for example Persons only, Vehicles only)
- Filter by keyword (search across entity data)

Each entity card shows:

- The entity name and type (color-coded)
- Saved status: ✅ saved, ⚠️ not saved
- Relationship count
- Link count to existing records

Expand a card to review details and make changes. Use the per-entity actions:

- **Edit** (✏️) — Change the entity name or type
- **Delete** (🗑️) — Remove the entity
- **Save** (💾) — Save this entity to your workspace

Review and fix relationships

Inside the entity card, open the relationships section to validate links. You can:

- Delete a relationship
- Change the source entity, target entity, or relationship label using dropdowns
- Link the extracted entity to an existing record using the search icon and a pasted record link

Treat relationships as suggestions. Confirm them against the evidence before saving.

Step 5 — Review the summary and graph

Open the **Summary** tab for a high-level view of the extraction. This tab includes:

- Statistics: total entities, relationships, and links to existing records
- Annotated text: the document text with highlighted entities
- Interactive graph: a node-edge graph of entities and relationships, color-coded by entity type
- Word clouds: frequent words and verbs

Use pan and zoom to navigate the graph. Saved entities appear in full color. Unsaved entities appear lighter.

Understanding the Output

After extraction completes, you work with two main outputs:

- **Entities list (cards):** use this to validate and correct the extracted data. The saved status icons (✅ and ⚠️) help you track what has been written back to the workspace.
- **Summary views:** use these to understand coverage and context:
    i. Statistics help you assess extraction size and completeness.
    ii. Annotated text helps you verify where each entity was found.
    iii. The interactive graph helps you spot clusters, key actors, and unexpected connections.
    iv. Word clouds help you identify recurring topics and actions. Color-coding indicates entity type. Relationship labels indicate the relationship type the app inferred. Review relationship labels carefully before saving.

Saving and Exporting Results

Use the left sidebar save controls after you review the extraction. Start by selecting:

- The target **Workspace**
- A folder path inside that workspace Then choose one or both save options:
- **Save Entities & Relationships:** saves extracted entities and relationships as permanent workspace records. Saved records become searchable and reusable across Octostar.
- **Create a Graph:** exports the entities and relationships as an interactive **Link Chart**. You can set a custom filename before creating it. The chart is saved to the selected workspace folder.

Save incrementally when possible. You can save individual entities from the **Entities** tab before saving the full set.

Tips for Best Results

- Write a specific **Extraction Goal** when you have a focused task. A narrow goal reduces noise.
- Review relationships before saving. Relationship extraction is the most error-prone part of the output.
- Use concept and keyword filters in **Entities** when the extraction is large.
- Link extracted entities to existing records to reduce duplicates and improve consistency across cases.
- Save high-confidence entities as you review them instead of waiting until the end.
- If a file shows ⚠ (no readable content), confirm the file contains selectable text or that OCR/transcription is enabled for your deployment.

Known Limitations

- Results can vary between runs. Processing the same document twice may produce different entities or relationships.
- Relationships require manual review. The app can infer incorrect or overly broad links.
- Very large graphs may not render interactively. Graphs with more than `5,000` combined nodes and edges do not display as an interactive visualization.
- Files may take a moment to be recognized after dropping them before extraction can begin.

AI Find By Question

Overview

AI Find By Question extracts answers from documents using questions you define in natural language. You select files or folders, define questions (each one becomes a column), and review results in a table you can export as CSV or save to a **Workspace**. Inputs are indexed documents from your workspace. Outputs are a structured results table, optional tags applied to selected items, and exported CSV files. AI results require review, especially when questions are vague or when documents contain multiple possible answers.

When to Use This Application

- You need to extract the same fields from many documents (for example vendor name, contract value, effective date).
- You want a structured table for review, triage, or downstream analysis instead of reading documents manually.
- You need to tag documents based on extracted values (for example “High value contract”).
- You want to reuse a standard set of questions across cases using templates.
- You need a CSV export for sharing or reporting.

Before You Begin

- Ensure your documents are available in Octostar and have indexed content.
- Confirm you have access to the source **Workspace** and the destination workspace if you plan to save templates or export results to a workspace.
- Decide what questions you need answered and what output type each answer should take (for example text, numeric, categorical).

Step-by-Step Walkthrough

Step 1 — Begin extraction

When the app opens, you see a welcome screen that summarizes the workflow. Select **Begin Extraction** to start.

Step 2 — Select documents

Drag and drop documents from the Octostar sidebar into the drop zone. You can add:

- Individual files
- Entire folders

After you add documents, the app shows a list with each document's name and type. Use the controls to manage the selection:

- Search for a document using the search bar
- Remove a document you do not want to include
- Review skipped files (documents without indexed content show a warning)

When your selection is ready, select **Next: Add Questions**.

Step 3 — Configure questions

Define the information you want extracted from each document. Each question becomes one column in the results table.

Add and manage questions

Use the question controls to build your set:

- Select **Add Question** to create a new question tab
- Enter a question in natural language (for example "What is the company name?")
- Press `Enter` in the prompt field to add another question
- Rename a column tab using the pencil icon
- Delete a column using the trash icon on the tab

Choose an output type

Select an output type for each question. The app tries to auto-detect the best type, but you can change it. A description of the selected type appears below the selector.

Configure categorical questions

If you choose the categorical type, define at least two categories. You can:

- Add categories manually (name + optional description), then select **Add**
- Select auto-detect to have the AI suggest categories from a sample of your documents
- Remove categories using the `X` on a category tag

Allow multiple values

For free text and numeric questions, you can enable **Allow multiple values** when a document may contain more than one valid answer. Results appear as semicolon-separated values. Examples:

- “List all authors”
- “What are the line item amounts?”

Save and load templates

Use templates to reuse question sets:

- Save the current questions as a template to any writable workspace
- Load a saved template to populate the questions instantly Templates are stored as `.aiquestions` files in your workspace and can be shared with your team.

Adjust advanced settings

Select **Settings** to tune performance:

- **Batch Size** — How many documents the AI processes in one batch
- Smaller batches improve accuracy but take longer. Larger batches run faster but may miss details.
- **Parallelism** — How many batches run at the same time
- Higher values increase speed without changing accuracy.

Test before full extraction

Select **Test Extraction** to run a trial on the first 5 documents. Use the test run to verify:

- Results look correct
- Output types match the answers you expect
- Questions are specific enough After testing, refine questions if needed. When you are ready, select **Start Full Extraction**.

Step 4 — Review selection and run full extraction

Before the run starts, you see a checklist of documents. Use the controls to manage the run set:

- Select or deselect individual documents
- Use **Select All** and **Deselect All** for bulk changes
- Review the summary for document count and question count To start, select **Run Full Extraction**.

Step 5 — Monitor extraction and review results

While extraction runs, the app shows:

- A progress bar
- Elapsed time and estimated time remaining
- A live results table that updates as documents complete You can select **Pause** to stop processing temporarily. When paused, you can review partial results, export what is complete, or select **Resume** to continue.

Step 6 — Tag selected entities (optional)

After you review results, you can tag items directly from the table.

1. Select rows using the checkboxes.
2. Select **Tag Selected Entities**.
3. Choose a workspace and select an existing tag, or create a new tag with a name and color.
4. Select **Apply Tag**.

Use this to categorize documents based on extracted results (for example “Contracts > 100k EUR” or “Needs review”).

Understanding the Output

The primary output is an interactive results table. Each row represents a document. Each question you defined becomes a column.

The table supports:

- Sorting and filtering by any column
  - Text search for free-form columns
  - Checkbox filters for categorical and simple question columns
- Resizing columns by dragging borders
- Renaming columns using the edit icon in a column header
- Opening a document by selecting its name
- Selecting rows with checkboxes for bulk actions (export or tagging)

Exports include:

- Document name
- All question columns
- Metadata fields such as entity ID, entity type, creation date, and creator

---

Saving and Exporting Results

Select **Save Table** to export results. You can export in two ways:

- **Download CSV** — Saves the CSV file to your computer.
- **Save to Workspace** — Uploads the CSV into an Octostar workspace so you can share it with your team.

Before saving to a workspace, select:

- Destination **Workspace**

Tips for Best Results

- Write specific questions. Include units and constraints (for example “total contract value in EUR”).
- Run **Test Extraction** before a full run to validate question wording and output types.
- Start with a smaller **Batch Size** when accuracy matters, then increase it for speed.
- Use categorical output types when you know the likely answers in advance.
- Enable **Allow multiple values** when a document can contain more than one valid answer.
- Save question templates for recurring tasks and share them within a workspace.

Known Limitations

- Answers vary by document quality and indexing. Documents without indexed content are skipped and flagged.
- Results depend on question phrasing. Vague questions produce inconsistent or incomplete answers.
- Multiple-answer cases require configuration. If you do not enable **Allow multiple values**, results may be truncated to a single value.

Connection Machine

Overview

Connection Machine compares evidence sources in your **Workspace** and shows how they relate through shared identifiers and semantic matches. You can compare documents, folders, and images to find overlaps such as names, organizations, phone numbers, and locations. Outputs include a match table with traceable references and a differential view that highlights what is discovered thanks to their comparison. AI features improve coverage but require review, especially for semantic matches.

When to Use This Application

- You need to find overlaps across two document sets (for example two folders of reports).
- You need to trace the same person or organization across multiple sources using shared identifiers.
- You need to connect images or media by face or image similarity when text is incomplete.
- You need to compare a focused source against a broader target (for example one file vs the entire workspace).
- You need to identify what is new in an updated dataset using differential analysis.

Before You Begin

- Confirm the items you want to compare are available in a **Workspace**:
  - Documents
  - Folders
  - Images
- Ensure the target set is defined (specific files, a folder, or the entire **Workspace**).
- If you plan to use AI options, confirm AI features are enabled for your deployment.

Step-by-Step Walkthrough

Step 1 — Select source and target

Select or upload the source you want to compare. Then define the target scope. Typical options include:

- Source: one or more files, a folder, or images
- Target: specific files, a folder, or the entire **Workspace**

After selection, the app prepares inputs for identifier extraction and comparison.

Step 2 — Review extracted identifiers

After you load source and target, the app extracts identifiers using standard parsing and shows them in tabs. Common identifier types include:

- Names
- Organizations
- Dates
- Email addresses
- Phone numbers
- Locations
- Other structured entities

Review the identifier tabs before you run the comparison. Remove or correct items that are irrelevant to your investigation.

Step 3 — Enable AI deep analysis (optional)

Enable **AI Deep Analysis** when documents are unstructured or when key identifiers are implicit. In this mode, the app analyzes full text to extract contextual identifiers that standard parsing may miss, such as:

- Implicit names
- Contextual organizations
- Variations of the same entity
- Less structured references

Use this option when you need more coverage before you run the comparison.

Step 4 — Enable Smart AI matching (optional)

Enable **Smart AI Matching** when identifiers are inconsistent across sources. This option uses semantic matching to detect equivalences beyond exact identifier matches. Examples include:

- Nicknames to formal names (for example “Johnny” → “John”)
- Abbreviations (for example “NYC” → “New York City”)
- Aliases and common misspellings Review these matches carefully. Treat them as suggested equivalences until you confirm them in the source content.

Step 5 — Run the comparison

Start the comparison after you review identifiers and choose optional AI settings. The app compares the source and target using:

- Extracted identifiers
- AI Deep Analysis (if enabled)
- Smart AI Matching (if enabled)

The app then produces a match table ranked by relevance.

Step 6 — Review match results

Review the results table to see identifiers shared between source and target. By default, the app shows the most relevant matches first. Select **View All** to display all matches. Use filters to narrow results:

- Filter by identifier type
- Filter by label Open a match to view details. The details view shows:
- Where the identifier appears in the source
- Where it appears in the target
- Context in each document

Use this view to validate each connection.

Step 7 — Run differential analysis (optional)

Use **Differential View** to find identifiers that appear in the target but not in the source. This view is available in the details section of a matched identifier. Use differential analysis when you need to:

- Compare document versions
- Review newly acquired data
- Identify newly surfaced connections in a broader dataset

Understanding the Output

Connection Machine produces three main outputs:

- **Match table**
- A table of identifiers shared between the source and target. Each entry includes identifier type, identifier label, and references to where it appears.
- **Match details**
- A traceable view that shows the exact locations and surrounding context for each identifier in both source and target. Use this to confirm that a match is meaningful.
- **Differential view**
- A list of identifiers present in the target but not present in the source. Use this to detect new information when comparing updates.

If you enable semantic matching, treat suggested equivalences as leads. Confirm them by opening the referenced documents or media.

Tips for Best Results

- Define source and target carefully before you run the comparison. A broad target increases noise.
- Review extracted identifiers before matching. Remove irrelevant identifiers to improve results.
- Enable **AI Deep Analysis** for messy, unstructured, or narrative documents.
- Enable **Smart AI Matching** when you expect aliases, abbreviations, or inconsistent naming.
- Use filters to focus on a specific identifier type (for example phone numbers first).
- Confirm semantic matches by opening the referenced sources before recording conclusions.

Known Limitations

- Standard parsing can miss contextual identifiers in unstructured text. Use **AI Deep Analysis** when coverage matters.
- Smart AI Matching suggests semantic equivalences, not confirmed facts. You must validate matches in the source material.
- Match quality depends on how well identifiers are extracted from the selected sources.

Data Lab

Overview

Data Lab is an interactive analytics app for structured analysis of large datasets. You upload CSV, Excel, or JSON files, define relationships using shared identifiers, and analyze connected data in tables and a **Link Chart**. Outputs include filtered tables, derived datasets, dashboards, and optional semantic **Record** creation aligned to the **Ontology**. AI features can accelerate matching and column creation, but you still review results and validate formulas.

When to Use This Application

- You need to correlate multiple datasets using a shared identifier (for example customer ID, phone number, company name).
- You need to explore relationships as a network using a **Link Chart** after starting in tables.
- You need to prepare investigative subsets from raw exports before importing or sharing them.
- You need dashboard-style analysis to identify outliers, key entities, or patterns quickly.
- You need to convert tabular data into semantic **Records** so other Octostar apps can use it.

Before You Begin

- Prepare one or more datasets in a supported format:
  - `.csv`
  - `.xls` / `.xlsx`
  - `.json`
- Confirm you have access to the **Workspace** where you plan to upload or save outputs.
- If you plan to create semantic entities, confirm you know the intended **Concept** types and key identifiers for each dataset.

Step-by-Step Walkthrough

Step 1 — Upload your data

Open *Data Lab* from the **App Launcher** or from the item context menu.

Load datasets using one of these methods:

- Drag and drop one or more files
- Use the upload function to select files from your device or **Workspace**

After upload, datasets appear in the **Table Chart**.

To rename a dataset or change its icon:

- Right-click the dataset, or
- Select the three dots on the dataset tab

Step 2 — Define relationships between datasets

Upload at least two datasets.

Then define a relationship:

1. Select two datasets.
2. Select the shared identifier field, or allow AI to suggest one.
3. Enter a relationship name.
4. Create the relationship.

After you create a relationship, you can enable **Associative Filter** to apply relational filtering across datasets. When **Associative Filter** is enabled:

- Only records connected across linked datasets remain visible.
- Unrelated records are temporarily excluded.
- All tables update together using the same analytical context.

When the relationship is active, an additional column appears in the table view. This column shows how many related records exist in the connected dataset.

Step 3 — Visualize connections in a Link Chart

Select a relationship and switch to visual exploration mode.

Open the **Link Chart** to explore network structure:

- Records appear as nodes.

- Connections appear as links.
- You can identify clusters, central nodes, and indirect links.

If you want to keep relationship counts visible while you work in tables, add the relationship column as fixed:

- Select the column header.
- Choose **Import Content**.

Step 4 — Apply filters and explore connected data

Apply filters to any dataset in table view.

Select values to include. Data Lab updates all related datasets automatically to reflect the selected context.

Use progressive filtering. Start broad and narrow as you confirm what matters.

Step 5 — Run dashboard analysis

Open the **Dashboard** view to analyze data using interactive widgets. Data Lab generates initial widgets automatically, and you can customize them. To filter from the dashboard:

- Select a value in any widget.
- All connected datasets update to reflect the selected context. To manage widgets and templates, open the dashboard context menu:
- Move the cursor to the top-right corner to open the context menu.
- Add widgets.
- Save widget templates.
- Load saved templates.

Step 6 — Extract a new table from a key

Use table extraction when you want a focused dataset centered on an identifier.

1. Open the column header menu for the primary key.
2. Select **Transform**.
3. Select additional columns to include.

Data Lab creates a new dataset based on the selected key. The original dataset remains unchanged. You can analyze the extracted dataset independently.

Step 7 — Use AI to accelerate analysis

Data Lab includes AI features for analysis support.

AI-assisted column creation

In the table view:

1. Open the top-right context menu.
2. Create a computed column.
3. Describe the calculation in natural language. Data Lab generates the formula automatically. Test the formula before you finalize it.

Advanced Data Chat

Use *Advanced Data Chat* to ask analytical questions about:

- The entire dataset session, or
- A specific table

Enter your query in natural language. You can optionally limit the number of returned records. Select an output format such as tabular results or aggregated summaries.

Step 8 — Export or create semantic entities

At any stage, export datasets or convert them into semantic entity records. AI can help by suggesting:

- Entity types (Concepts)
- Identifiers
- Relationships based on the **Ontology**
- Column-to-attribute mappings Review and adjust mappings before you create records. After creation, semantic entities:
- Appear in the investigation area
- Support tagging and annotation
- Remain available to other Octostar applications

Step 9 — Export the entire session

Export the full session to preserve your work and continue later. You can export to:

- Your local computer
- A **Workspace** You can re-import the session later to continue analysis.

Understanding the Output

Data Lab produces outputs that update as you define relationships and apply filters.

- **Filtered tables**
- When you filter one dataset, connected datasets update dynamically. Counts and summary metrics adjust to the current context.
- **Relational count columns**
- When relationships are enabled, Data Lab aligns records using distinct values of the selected key. Rows are not duplicated. The relationship column shows how many related records exist in the connected table. Repeated occurrences consolidate under the same identifier.
- **Link Chart view**
- The **Link Chart** shows records as nodes and their connections as links. Use it to identify clusters, bridging entities, and unusual network structures.
- **Semantic entity records**
- When you export to semantic entities, records persist beyond the Data Lab session. You can access them in other Octostar applications for investigation and workflows.

Saving and Exporting Results

Data Lab supports multiple ways to preserve and reuse outputs.

- **Export datasets**
- Export a table or derived dataset for sharing or downstream processing.
- Supported export formats for tables (CSV, Excel, JSON) and where the export is saved.]
- **Create semantic entities**
- Convert tabular rows into semantic **Records** aligned to the **Ontology**.
- Configure mappings before creation, including identifiers and relationships.
- **Export session**
- Save the entire analysis session so you can re-import it later.
- Configure whether the session saves locally or to a **Workspace**.

Tips for Best Results

- Select a stable shared identifier when you define relationships (for example normalized IDs instead of free-text names).
- Enable **Associative Filter** when you want all datasets to stay synchronized.

- Start broad and filter progressively to avoid hiding relevant connections early.
- Review relationship count columns. High counts often indicate key entities.
- Clean identifier formatting before linking datasets (for example trim whitespace and normalize case).
- Test AI-generated formulas before you rely on computed columns.

Known Limitations

- Identifier formatting differences can prevent relationships from forming (for example `COMPANY NAME` vs `company name`). Clean or normalize values before linking.
- Mixed data types can affect filtering and computed columns (for example numbers stored as text). Convert types where needed.
- Over-filtering early can hide relevant connections across datasets. Apply filters progressively.
- AI-assisted features can produce incorrect formulas or mismatched identifiers. Validate outputs before exporting or creating records.

Report Creator

Overview

Report Creator generates a draft police report or legal affidavit from your case materials. You load files from a **Workspace**, choose a document type, enter optional case details, and generate a `.docx` file you can edit. Inputs are case documents and supporting materials, and optional **Link Chart** data for entity selection. Outputs are a Word document saved to your computer or to a workspace location. AI output requires review and editing before you finalize or submit it.

When to Use This Application

- You need a first draft narrative from multiple case files for a report or affidavit.
- You need a consistent document structure for repeated report types (report, search warrant affidavit, arrest warrant affidavit).
- You need to include key entities from a **Link Chart** and focus the document on a defined set of people or subjects.
- You need to generate a Word document for further editing, approval, or conversion to PDF.
- You need to assemble a timeline-style narrative from mixed evidence sources.

Before You Begin

- Collect the case materials you want to use (documents and supporting files).
- Confirm you have access to the **Workspace** that contains the source files.
- If you plan to save the output to a workspace, confirm you have write access to the destination folder.
- If you plan to filter entities, confirm your case includes **Link Chart** files with the entities you want to reference.

Step-by-Step Walkthrough

Step 1 — Load your files

Load case files using one of these methods:

- From your **Workspace**: If you launch Report Creator from a folder, case, or **Link Chart**, the app loads those files automatically and lists them in **Currently selected files**.
- Manual upload: Use the file uploader to drag and drop additional files into the app. You can upload multiple files at once.

Review **Currently selected files** before you continue. The table shows each document's name, type, and folder so you can confirm the right materials are included.

Step 2 — Filter entities from Link Charts (optional)

Use this step when your case includes relationship graphs and you want to limit the report to specific entities.

1. Toggle **Filter entities from linkcharts**.
2. Review the entity list that appears.
3. Select the entities to include using checkboxes.
4. Navigate pages if the list is long.

Skip this step if you do not need entity-level control.

Step 3 — Choose your report type

Select the document type using the dropdown:

- **Report** — Standard case narrative.
- **Affidavit** — Legal affidavit. If you select **Affidavit**, a second dropdown appears. Select one of:
- **Search Warrant**
- **Arrest Warrant**
- **Subpoena Request**
- **Surveillance Warrant**
- **Emergency Order**
- **Consent Form**

Your selection controls the template and the form fields shown in the next step.

Step 4 — Fill in the form

After you select a document type, the app shows a form with fields relevant to that type. All fields are optional. You can fill missing details after generation.

Common fields include:

- For reports:
  - Officer name
  - Case number
- For affidavits:
  - Affiant name and residence
  - Type-specific details (dates, locations, surveillance means, witness name)

Use the narrative context text area to add background the files do not contain. Include constraints, priorities, or instructions that affect how the document should read.

Step 5 — Generate the document

Select **Generate** to start. The app:

- Reads and processes the loaded files
- Builds a chronological narrative from the materials
- Applies the selected template and formats the output as a Word document

A progress indicator shows while the document is generated. Duration depends on the amount of material.

When generation completes, the app shows a success message and two actions for saving.

Step 6 — Save or download your document

After generation, choose one of these options:

- **Download**
- Saves the `.docx` file to your computer. Open it in Word to edit. Export to PDF if needed.
- **Save and Open**
- Saves the `.docx` file to a location in your **Workspace** and opens it in the document editor. Configure:
  - Destination workspace
  - Folder path
  - Filename

Understanding the Output

The output is a Word document (`.docx`) generated from your files and your form inputs.

- Reports typically include:
  - A narrative summary of the case
  - Key dates formatted in italics
  - Main subjects formatted in bold
  - A header with current date and case number
  - Person profile cards on additional pages (with photos when available from graph data)
- Affidavits typically include:
  - An opening summary
  - A chronological list of events
  - A conclusion with the judicial reasoning
  - Affiant details and relevant dates populated from your form entries Treat the document as a draft. Verify facts, names, dates, and legal language before you distribute or submit it.

Saving and Exporting Results

Report Creator provides two output paths:

- **Download**
  - What it does: saves the generated `.docx` to your computer
  - Where it ends up: local download folder
  - Settings: none
- **Save and Open**
  - What it does: saves the generated `.docx` to a workspace location and opens it for editing
  - Where it ends up: selected **Workspace** folder path
  - Settings: destination workspace, folder path, filename If you need PDF output, export from Word after you review edits.

Tips for Best Results

- Load all relevant documents before generation. Missing materials reduce detail and can distort chronology.
- Use the narrative context field to add background and constraints the documents do not contain.
- Filter entities from **Link Charts** when the case contains many entities and you need a focused report.
- Treat AI output as a draft. Review and edit the document before final use.

- Enter officer and case identifiers when available, but fill them later if needed.

Known Limitations

- Generated text can omit details or misinterpret context when source material is incomplete or inconsistent.
- Entity filtering depends on **Link Chart** quality. Incorrect or missing entities in charts affect the selection list.
- Output requires human review for accuracy, formatting, and legal suitability.

Audio Transcription

Overview

Audio Transcription converts audio recordings into timestamped text. You open it from an audio file or folder in your **Workspace**, run transcription, and optionally generate a translation in a target language. Outputs include editable transcripts with timestamps and exported files in selectable formats. AI output requires review, especially for noisy audio and for names, addresses, and specialized terms.

When to Use This Application

- You need a written transcript of an interview, call, or recording for review or disclosure.
- You need searchable text from long recordings and want to filter for keywords before saving.
- You need both the original transcript and a translation for collaboration or reporting.
- You need an export format suitable for subtitles or downstream tools (for example `SRT`).
- You need to process multiple recordings in one batch by transcribing a folder.

Before You Begin

- Store the audio files in a **Workspace** folder you can access.
- Confirm your audio format is supported (for example `MP3`, `WAV`, `FLAC`, `M4A`, `OGG`).
- If you plan to save outputs, confirm you have write access to the destination folder.

Step-by-Step Walkthrough

Step 1 — Open the application

Open the app from your workspace in one of these ways:

- Single file: Right-click an audio file and select **Transcribe Audio**.

- Folder: Right-click a folder and select **Transcribe Audio** to load all audio files in that folder. After launch, the app loads the selected files and shows a status indicator for each file.

Step 2 — Review the main screen

After files load, the main screen shows:

- A file list for the loaded audio files
- An embedded audio player
- Transcription options Use the audio player to preview the recording before you start transcription.

Step 3 — Transcribe audio

To run transcription, select **Transcribe!**. The app:

- Detects the spoken language automatically
- Transcribes the audio
- Shows a progress bar while processing
- Displays the transcript with timestamps when complete Each timestamp links the text segment to the point in the audio where it was spoken.

Step 4 — Transcribe with Translation (Optional)

Use translation when you need the transcript in another language.

1. **Enable Translation**: Check the **Include audio translation** option.
2. **Select Language**: Choose a target language from the dropdown menu.
3. **Start Transcription**: Click **Transcribe!**.

The progress bar will show:

- **Transcription**: First half
- **Translation**: Second half

After processing, the app displays the original transcript alongside the translation.

Available Target Languages:

- English
- Spanish
- French
- German

- Italian
- Portuguese

If a direct translation path isn't available, the app automatically routes through English.

Step 5 — Review and edit the transcript

Use the timestamp buttons to verify content:

- Play the audio in the embedded player.
- Select a timestamp to jump to that moment in the recording.

Edit content directly in the transcript:

- Click into any transcription or translation text field.
- Enter corrections.

Edits remain in the current session and are included when you export.

Step 6 — Filter results (optional)

Use filtering to find specific topics or words in long recordings.

1. In the left sidebar, enter one or more keywords in the filter input.
2. Review the filtered list of segments that match.

You can filter in two modes:

- Per-file filtering: applies only to the current file
- Global filtering: applies the same filter across all loaded files

Toggle the global mode using the global filter switch in the sidebar. The filter uses smart keyword matching. For example, searching for “run” matches “running” and “ran”. When translation is enabled, matching works across both the transcription language and the translation language.

Step 7 — Save transcripts

Use the save options panel in the left sidebar.

Choose what to save

- Current file
- All files

Choose content type

- Transcription
- Translation (only if translation is enabled)

Choose sections

- All sections
- Filtered sections only

Choose output format

Select the output format from the format list.

Choose where to save

- Original folder: saves alongside the source audio file
- Custom workspace folder: select a folder path in your workspace

Save

Select **Save Transcripts**. The app shows a confirmation message after saving. The saved file is linked to the source audio file in the workspace.

Understanding the Output

After transcription, you work with a timestamped transcript displayed in a three-column layout. Key behaviors:

- Each row represents a sentence or spoken segment.
- Each row includes a timestamp you can select to jump playback to that moment.
- When translation is enabled, the app displays original and translated text side by side.
- You can edit text directly, and those edits carry into exported files.

Saving and Exporting Results

You export transcripts using the save options panel.

- What you save:
  - Current file or all files
  - Transcription and/or translation
  - All sections or filtered sections only
- Where output ends up:
  - Original folder, or

    - A custom folder path in your **Workspace**
- What formats you can export:
    - Selectable formats from the output format list

If you save filtered sections only, the exported file includes only the matching segments and their timestamps.

Tips for Best Results

- Use clear audio recordings. Background noise reduces accuracy.
- Use timestamps to verify names, addresses, and critical statements.
- Filter before saving when you only need a segment of a long recording.
- Use folder-level transcription for batch processing, then save **All files** once.
- Review translations for proper nouns and legal terminology. Replace terms manually where needed.
- Keep the original transcript alongside the translation when sharing outputs.

Known Limitations

- Transcription accuracy drops with noisy audio, overlapping speakers, and heavy accents.
- Automatic language detection can misclassify short clips or mixed-language recordings.
- Translation quality varies by language pair and domain terminology.
- Smart keyword matching can include related word forms that you did not intend. Validate filtered exports before sharing.

Patterns of Life

Overview

Patterns of Life analyzes movement and communication patterns from mobile network data, such as CDR and IPDR records. You load activity for one or more phone numbers over a time range and review trajectories on an interactive map. The app outputs clustered locations, anomalies, home/work hotspots, and short-horizon movement predictions. Results are model-driven and require analyst review, especially when you treat locations as sensitive conclusions.

When to Use This Application

- You need to understand a subject's routine and movement patterns over days or weeks.
- You need to identify significant locations, such as likely home or workplace.
- You need to detect unusual activity (time or location anomalies) for targeted follow-up.
- You need to compare activity patterns by weekday and hour to spot routines and deviations.
- You need short-horizon forecasts to support operational planning.

Before You Begin

- Ensure CDR/IPDR data is available and mapped so the app can query positions and activity.
- Decide the phone number identifiers you want to analyze (for example `MSISDN`).
- Decide the time range you want to load for analysis.

Step-by-Step Walkthrough

Step 1 — Load targets and time range

Select the phone number target(s) and a time range for analysis. After load, the app renders the activity on the map and populates the sidebar tabs.

Step 2 — Choose a playback mode

Use the map in one of two modes:

- Static mode: shows a snapshot around the current slider position.
- Playing mode: animates forward from the current slider position to the end of the loaded range.

In static mode, drag the time slider to jump to any moment. In playing mode, pause to freeze on a frame or stop to reset.

Step 3 — Set the track bucket and preserve positions

Configure how the map frames time and what points remain visible.

- **Track bucket**
- Sets the time window represented by each frame. Values range from `5 min` up to `48 h`. Smaller buckets show finer movement detail. Larger buckets condense activity into fewer frames. Default is `15 min`.
- **Preserve positions**
- Controls whether the map shows only positions inside the current frame window, or accumulates positions over time.
  - Off: shows only positions inside the current window.
  - On: accumulates all positions from the start of the range up to the current time.

Step 4 — Tune analysis settings (optional)

Open **Analysis Settings** to tune clustering, hotspot classification, and anomaly detection. Adjust these settings:

- Work start / end hour (default `09:00–17:00`)
- Sleep start / end hour (default `22:00–06:00`)
- Clustering radius in km (default `0.5 km`)
- Min stay duration in hours (default `2 h`)
- Weekend days (default Saturday and Sunday) After you change settings, select **Apply Settings** to recompute results.

Step 5 — Use the analysis tabs

Use the sidebar tabs to explore different results. All tabs respect your time range and weekday filters.

Summary tab

Use **Weekday filter** to include or exclude specific days. Review activity patterns in:

- Activity bar chart (drag a window to set a time range)

- Activity timeline (online and call activity by weekday)
- Activity heatmap (weekday × hour matrix) [CONFIRM UI LABEL: Summary] [CONFIRM UI LABEL: Weekday filter]

Tracks tab

Use this tab to understand movement trajectories.

- Review total distance statistics.
- Review the cell tower transitions table for frequent routes.
- Enable **Most common path** to draw the most frequent route as a red line with start/end markers.
- Review the daily distance chart to spot unusual movement days.

Anomalies tab

Use **Anomaly threshold** to control sensitivity. Review:

- Anomaly count by `MSISDN`
- Anomaly table (first 150 anomalies) with time, coordinates, score, and severity label

Select an anomaly from the table or the map to center the view and load details. You can then narrow the time range around that anomaly:

- **Apply filter**: focuses on a window around the anomaly using the track bucket interval.
- **Clear filter**: returns to the full view.

Predictions tab

Use **Prediction horizon** to choose how many hours ahead to forecast after the last known position. Use **Minimum confidence filter** to hide low-confidence results.

Predictions appear as:

- Purple markers on the map with hour offset and confidence
- A prediction table with coordinates, time offset, and confidence score

Clusters tab

Use this tab to identify frequently visited locations.

- Adjust minimum cluster size to control how many points are required to form a cluster.
- Review cluster circles on the map (sized by visit count).
- Open cluster details to see centroid coordinates, visit count, and visit periods.

Hotspots tab

Use this tab to classify recurring locations as home and work based on visit patterns.

- Home location appears as a green house-shaped marker.
- Work location appears as a blue briefcase-shaped marker.

- Hotspot markers appear as distinctive drop-shaped icons.

Hotspot scoring uses the work/sleep hours and weekend days defined in **Analysis Settings**.

Understanding the Output

Patterns of Life produces map-based and tabular outputs that you interpret together.

Map Playback

Shows points and trajectories over time. Static mode gives a snapshot around the slider time. Playing mode animates forward.

Tracks

Shows total distance traveled and common routes. Use these outputs to spot travel days that differ from the usual pattern.

Clusters

Groups positions into frequently visited locations and estimates time spent. Larger cluster circles indicate higher visit count.

Hotspots (Home/Work)

Labels the highest-scoring recurring clusters as home and work using configured work/sleep hours and weekend weighting.

Anomalies

Lists points that break routine by location novelty, time rarity, or time-location novelty. Anomaly scores range up to `1.0`.

- Below `0.2`: normal
- Above `0.8`: extremely anomalous

Predictions

- Displays forecast locations as purple markers and a table. Predictions include a confidence score. Use the minimum confidence filter to focus on reliable results.

Tips for Best Results

- Start with the default **Track bucket** (`15 min`), then increase it when you need a higher-level view.
- Enable **Preserve positions** when you need a cumulative trail. Disable it when you need a clean frame-by-frame view.
- Set work/sleep hours and weekend days to match the local context before you interpret home/work outputs.
- Use **Weekday filter** to separate routine weekdays from weekend behavior.
- Use the anomalies table to drive targeted review, then apply a narrow time window around a selected anomaly.
- Use the **Minimum confidence filter** in predictions to avoid over-interpreting low-confidence forecasts.

Known Limitations

- CDR/IPDR locations are approximations based on network events. They do not represent GPS ground truth.
- Home/work classification is heuristic and can be wrong for irregular schedules or sparse data.
- Anomaly detection depends on the chosen threshold and on historical coverage. Short time ranges reduce reliability.
- Predictions are probabilistic. Low confidence values indicate weak patterns and should not drive decisions without supporting evidence.

Anomaly Detection

Overview

Anomaly Detection flags unusual telecommunications behavior for individuals observed inside a defined geographic area. You draw one or more polygons on the map, set an analysis time range, and compare activity in that period against a historical baseline. Outputs include a results table of flagged individuals, summary statistics, and an entity tagging workflow for follow-up. The scoring is model-driven and requires analyst review before you treat results as risk or intent.

When to Use This Application

- You need to identify individuals whose presence or movement in an area stands out from historical patterns.
- You need to compare a specific event window against prior activity to surface changes in behavior.
- You need to triage large volumes of activity into a smaller set of high-risk candidates for follow-up.
- You need to tag flagged entities by risk level so other teams can review them in Octostar.
- You need map-based context to understand where and when anomalous activity occurs.

Before You Begin

- Ensure telecommunications activity data is already available in Octostar for the area and time range you want to investigate.
- Decide the analysis period (the window you want to evaluate) and the lookback period (the baseline window).
- Confirm you have access to the **Workspace** where you want to apply tags.

Step-by-Step Walkthrough

Step 1 — Define your area of interest

Draw one or more polygons on the map to define the geographic area you want to analyze.

1. Select **Draw Polygon**.
2. Click on the map to place polygon vertices.
3. Double-click, or select **Complete**, to finish the polygon.
4. Press `Escape`, or select **Cancel**, to abort drawing. To remove all polygons and start over, select **Clear Polygons**. You can draw multiple polygons to cover several areas.

Step 2 — Configure your analysis

Use the left sidebar to configure the analysis.

Select detection algorithms

Select up to three algorithms. Each algorithm detects a different type of anomaly.

Set the analysis period

Set **Time Range** to define the analysis period. This window is the activity you want to evaluate for anomalies.

Set the historical baseline

For each selected algorithm, set **Historical Lookback Period** in days (`1–90`). This baseline defines what "normal" looks like for comparison.

Set minimum frequency threshold (frequency-based only)

If you use the frequency-based algorithm, set **Minimum Frequency Threshold** (`2–100`). This sets the minimum number of visits required for an area to count as historically significant.

Step 3 — Run the analysis

After you draw polygons and set parameters, select **Run Analysis**. The app shows a progress bar in the sidebar with status and completion percentage.

Step 4 — Review results, statistics, and the map

When analysis completes, the results panel populates and the map updates.

Review the results table

Open the **Results** tab to see flagged individuals. Scores range from `0` to `1`:

- `0` indicates no anomaly.
- `1` indicates the highest anomaly level. You can sort by any column and change page size to `20`, `30`, or `50`.

Review summary statistics

Open the **Statistics** tab for an overview:

- Summary cards with counts (people, identifiers, call activity, online activity)
- Algorithm score cards (median, min, max per algorithm)

- Pie chart showing distribution across risk categories
- Histogram showing score distribution

Review map markers

After analysis, the map shows color-coded markers for detected positions. Hover over a marker to view a tooltip including:

- Owner name
- Identifier
- Coordinates
- Timestamp
- Anomaly score

Step 5 — Filter and refine results

Use filter controls in the sidebar to narrow results. Filters update the results table in real time. Clear a filter to return to the full results set.

Step 6 — Tag entities for follow-up

Use the **Entity Tagging** tab to apply tags to flagged individuals.

1. In the **Results** table, select the rows you want to tag using the checkboxes.
2. Open **Entity Tagging**.
3. Select the **Workspace** where tags will be applied.
4. Select an existing **Tag Group** or create a new one.
5. Apply tagging. The app categorizes tags by risk level (High, Medium, Low) and shows a confirmation notification when complete.

Understanding the Output

Anomaly Detection produces outputs in the results panel and on the map.

- **Anomaly score**
- A normalized value from `0` to `1` that represents how unusual an individual's activity is compared to the historical baseline.
- **Combined score**
- A weighted average across all active algorithm scores when you run multiple algorithms.
- **Results table**
- A list of flagged individuals with scores and risk categories. Use sorting and filters to prioritize follow-up.
- **Statistics dashboard**

- Summary counts and score distributions that help you validate whether the run produces a focused set of results or too many candidates.
- **Map markers**
- Spatial context for where anomalies occur. Use the tooltip to verify time and location before you draw conclusions.

Tips for Best Results

- Use multiple algorithms together to reduce false positives and improve combined scoring.
- Start with a longer **Historical Lookback Period** (for example `60–90` days) for a more stable baseline.
- Use the distance-based algorithm when you investigate individuals far outside their usual territory.
- Use “New Location” when you focus on entities new to the system.
- Apply filters before tagging so you tag only the most relevant candidates.
- Review map tooltips for time and coordinates before you treat a result as actionable.

Known Limitations

- Results depend on the quality and coverage of historical data. Short baselines reduce reliability.
- Risk categories summarize scores. They do not confirm intent or threat without supporting evidence.
- Multiple algorithms can produce different rankings. Use the combined score as a triage signal, not a conclusion.
- Polygons define scope. Incorrect polygon placement can include irrelevant activity or miss key positions.

Communication Analysis

Overview

Communications Analysis helps you explore communication records such as phone calls, internet sessions, and SMS messages. You select record types and data sources, define focus identifiers, and then use maps, network graphs, timelines, and tables to find patterns and relationships. Inputs are telecom datasets already available in Octostar. Outputs are interactive visualizations, filtered record tables, and saved analysis configurations you can reload later.

When to Use This Application

- You need to analyze **CDR**, **IPDR**, or **SMS** data for a subject or group of subjects.
- You need to identify frequent contacts and expand an investigation from primary to secondary identifiers.
- You need to review movement and geographic hotspots from location-enabled records.
- You need to compare activity across time windows using multiple time filter bands.
- You need to save and reload an analysis configuration for collaboration or continuity.

Before You Begin

- Confirm the relevant telecom datasets are available in one or more workspaces.
- Decide which record types you need: **CDR** (calls), **IPDR** (internet sessions), or **SMS** (messages).
- Collect starting identifiers for your primary subject, such as a phone number, IMSI, IMEI, or IP address.

Step-by-Step Walkthrough

Step 1 — Select your data

Use the left sidebar to define which records the app analyzes.

Choose record types

Toggle record types on or off:

- Call Detail Records (CDR)
- IP Detail Records (IPDR)
- SMS Each type shows a record count.

Select workspaces

Select the workspace(s) that contain relevant datasets by checking the boxes next to their names.

Select data sources

Within each workspace, select the data sources you want to include. Each data source shows a record count. Use these selections to narrow the analysis scope.

Step 2 — Define your focus identifiers

Use **Focus Identifiers** to define the subjects you investigate. Each focus identifier receives a unique color used across the map, graph, timeline, and tables.

Supported identifier types include:

- Phone number
- IMSI
- IMEI
- IP address
- Email address
- Domain name To add an identifier:

1. Select the `+` button in **Focus Identifiers**.
2. Enter the identifier value.
3. Select the identifier type.
4. Confirm.

To manage the list:

- Check or uncheck an identifier to include or exclude it without deleting it.
- Select the `X` next to an identifier to remove it.
- Use the search field to filter the list.
- Toggle the type view to switch between identifier categories.

If targets are pre-configured by an administrator, use the **Targets** section to toggle them on or off. Targets populate focus identifiers automatically.

Step 3 — Apply time filters

Use **Time Filters** to restrict analysis to specific periods. You can:

- Drag on the temporal bar chart to select a date/time range.
- Create up to four separate time filter bands with distinct colors for side-by-side comparison.
- Edit or remove time filters.

Step 4 — Apply geo filters

Use **Geo Filters** to restrict analysis to records inside a geographic area.

1. Select the polygon drawing tool on the map.
2. Draw a shape around the area of interest.
3. Name the filter and confirm. The geo filter appears in the list. Enable, disable, or delete geo filters as needed. The filter applies only to records with location data inside the drawn area.

Step 5 — Explore the main panels

The main content area contains resizable panels. You can expand a panel to full screen using the expand icon in the panel title bar. You can also collapse the left sidebar to increase screen space.

Positions map (top-left)

Use the map to review geographic locations derived from records:

- Pan and zoom to explore.
- Select markers to see which identifiers are present.
- Draw polygons on the map to create geo filters. Focus identifiers use consistent colors across the interface.

Relationships network graph (top-right of map area)

Use the graph to explore who communicates with whom:

- Nodes represent identifiers.
- Lines represent communication links.
- Drag nodes to adjust layout.
- Hover over nodes and links to view details.

Records table (center)

Use the table to review individual records filtered by your selections. Tabs switch between CDR, IPDR, and SMS. Typical columns include:

- Date and time

- From / To phone number
- From / To IMSI and IMEI
- From / To IP address
- Call or session duration
- Data volume, service, protocol (IPDR)
- Associated persons (if linked) I nteractions:
- Select a column header to sort (select again to reverse).
- Select an identifier value in the table to add it to your focus list.
- Focus identifiers are highlighted in their assigned colors.

Top identifiers chart (center-right)

Use the bar chart to identify the most frequent contacts of your focus identifiers.

- Use the dropdown to choose how many identifiers to show (default is 10).
- Select a bar to add that identifier to your focus list.

Timeline (right panel)

The timeline has two tabs:

- Timeline tab (scatter plot): each dot is one event. Use it to spot bursts and synchronized activity. Use the duration slider to filter events by call duration.
- Temporal filters tab (stacked bar chart): shows record volume over time. Use it to select time ranges and create time filters.

Step 6 — Save and reload an analysis

To save your current analysis state, select **Save** (disk icon). The saved state includes:

- Focus identifiers
- Selected workspaces and data sources
- Time and geo filters
- Selected targets Enter a name when prompted. You can reload the analysis later by opening the saved analysis record.

Understanding the Output

Communications Analysis outputs are interactive and update immediately when you change filters or focus identifiers.

- **Map output** shows positions derived from records. Colors correspond to focus identifiers.
- **Network graph** shows communication links between focus identifiers and related identifiers.

- **Records table** shows raw events filtered by record type, time, geo, workspace, and data source selections.
- **Top identifiers** highlights the most frequent contacts so you can expand focus.
- **Timeline views** show activity over time and support time-range selection and comparison.

Filters combine across types. If you use time and geo filters together, the app shows only records that satisfy all active filters. Within a single filter type, selections are inclusive (for example selecting two data sources includes records from either one).

Saving and Exporting Results

The app supports saving your analysis configuration for reuse.

- **Save**
  - What it does: saves your current analysis state (focus identifiers, sources, and filters)
  - Where it ends up: stored as a saved analysis record you can reopen
  - Settings: analysis name

Tips for Best Results

- Start with one primary identifier, then expand using **Top Identifiers** and table click-to-add.
- Use time filters early to reduce noise, then add geo filters for specific locations.
- Use multiple time filter bands (up to four) to compare periods side by side.
- Collapse the sidebar and expand panels when you need more space for map or timeline review.
- Treat map outputs as dependent on source location quality. Validate against record details.

Known Limitations

- The app supports up to four simultaneous time filter bands.
- Geo analysis depends on location data being present. Some records do not have positions.
- Combining multiple filters can hide records. Clear filters when results look unexpectedly empty.

Event-Based Analysis

Overview

Event-Based Analysis finds which entities were present in a geographic area during specific time windows. You draw one or more event polygons on a map, set a time range for each event, select an analysis preset, and run the analysis to discover subjects such as people, devices, or vehicles. Outputs include a paginated subject list with reasoning chains, plus optional subject position history on a map and timeline. Results depend on configured reasoning paths and require review before you treat presence as confirmation.

When to Use This Application

- You need to identify who or what was present at a location during a defined time window.
- You need to compare multiple events and find subjects present in all of them.
- You need to run a preset investigation strategy (for example “Find People” or “Find IMEIs”) without writing queries.
- You need to save events and results to a **Workspace** so other analysts can review them.
- You need map and timeline views for detections tied to a reasoning chain.

Before You Begin

- Ensure position-style geo-event data is available in Octostar for the area and time windows you plan to analyze.
- Confirm you have access to the destination **Workspace** folder where you will save event files and results.
- Confirm presets are configured and available. Presets are defined in `config.yaml`.
- If you plan to visualize subject positions after analysis, avoid changing `config.yaml` paths between running and visualizing the analysis.

Step-by-Step Walkthrough

Step 1 — Choose create mode or load mode

Use the Start Analysis section in one of two modes:

- Create: define new events by drawing polygons and setting time ranges.
- Load: load saved events from `.ebas` files stored in a workspace folder.

Step 2 — Create events (create mode)

Step	Action	Details
1	Draw a polygon on the map	Use the drawing tools to draw one or more polygons for the event.
2	Set the time range	Select start date/time and end date/time.
3	Select a preset from the dropdown	Presets define which tables and reasoning paths the analysis uses.
4	Preview the event	Select the preview action to see how many positions match your polygon and time range.
5	Add the event	Enter an event name and add it to the event list.

Repeat these steps to create multiple events.

Step 3 — Load events (load mode)

Load events when you already have `.ebas` files saved in a workspace.

Step	Action	Description
1	Browse workspace folders	The app lists folders that contain `.ebas` files.
2	Select a folder and load its events	Loaded events appear in the same event list used in create mode.

After loading, run the analysis using the same steps as create mode.

Step 4 — Run the analysis

Select one or more events from the event list and select **Run**. The app can run analysis in two ways:

- In-app execution with real-time progress streaming

- Background execution as a job Use in-app execution when you want to monitor progress live. Use background execution for longer runs.

Step 5 — Visualize results

Open the Visualize Results section to explore the output.

Load an analysis to visualize

Use the dropdown at the top of the page to choose what to visualize:

- In-app results from the most recent run
- Saved `.eba` files found in your workspaces

Browse subjects

After the analysis loads, the app shows a paginated list of subject cards. Each card includes:

- Label and type (subject label and concept type)
- Event chips (events where the subject appears)
- Tags (classification tags)
- Identifiers (raw identifiers linked to the subject)
- Reasoning chain (ontology path from geo-event to subject) Use pagination controls to move through results.

Filter subjects

Use filters to narrow the subject list:

- Event selection using event chips
- Subject concept type
- Identifier concept type
- Label regex
- Identifier label regex
- Event label regex
- Reasoning regex Event chip filtering shows only subjects present in all selected events.

View subject positions (map and timeline)

Expand a subject card to load position history:

- Map view showing the geo-event locations where the identifier was detected
- Timeline chart showing detections over time Position retrieval depends on `paths` in `config.yaml`. If the analysis reasoning chains do not match current `paths`, the app shows a warning and does not load positions.

Understanding the Output

Event-Based Analysis produces:

- **Event list**
- A list of named events, each defined by one or more polygons, a time range, and a preset.
- **Subject list**
- A paginated list of discovered subjects (people, devices, vehicles, and other concept types). Each subject includes identifiers and a reasoning chain.
- **Event chips**
- Colored indicators that show which events a subject appears in. Use chips to filter to subjects shared across events.
- **Reasoning chain**
- The ontology path used to connect a raw geo-event to the subject (for example `position_of_identifier → phone_number → is_owned_by → legal_person`). This explains why the subject appears in results.
- **Subject position history**
- A map and timeline for detections, when path matching succeeds. If `config.yaml` paths change after analysis, position history may not load.

Saving and Exporting Results

You can persist both events and analysis results to a workspace folder.

- Save events as `.ebas` files
  - What it does: saves each event as an individual `.ebas` file
  - Where it ends up: selected **Workspace** folder
  - Settings: destination folder path
- Save analysis results as a `.eba` archive
  - What it does: saves analysis output as a single `.eba` file for later visualization
  - Where it ends up: selected **Workspace** folder
  - Settings: destination folder path, filename

Tips for Best Results

- Create events with precise polygons. Large polygons increase noise and reduce interpretability.
- Use separate events for separate places or time windows, then use event chips to find intersections.
- Preview each event before you add it to confirm the time window and polygon match expected record counts.

- Select presets that match your investigative goal. Presets control which tables and reasoning paths are active.
- Keep `config.yaml` paths stable between running an analysis and visualizing positions from that analysis.
- Use regex filters to narrow subject lists when labels vary in formatting.

Known Limitations

- Results depend on presets and reasoning paths defined in `config.yaml`. Misconfigured paths reduce recall or produce incomplete subject sets.
- Subject position history can fail to load when `config.yaml` paths change after analysis, because stored reasoning chains no longer match.
- Presence in results indicates a match through configured reasoning paths. It does not confirm identity without supporting evidence.
- Large event lists and broad presets can produce very large subject sets, which may reduce usability without filtering.

Meetings Finder

Overview

Meetings Finder identifies meetings between people or identifiers based on recorded positions in a time range. You select one or more target identifiers, set proximity parameters, and run the analysis to find co-located positions. Results appear on an interactive map and a timeline chart so you can review who was near whom, where, and when. Results depend on the available position data and require review before you treat a co-location as a confirmed meeting.

When to Use This Application

- You need to find which identifiers were physically near a target during a time range.
- You need to check whether a group of selected identifiers met each other and when.
- You need a map view and a time view to review co-location evidence quickly.
- You need to rank neighbors by meeting count and meeting duration for triage.

Before You Begin

- Ensure position data exists for the identifiers you want to analyze.
- Decide whether you want to discover unknown neighbors (single target) or test co-location between known targets (multiple targets).
- Decide the time range and proximity settings you will use.

Step-by-Step Walkthrough

Step 1 — Select target identifiers

Select **Search Entities** and search for people or identifiers. Select the identifiers you want to analyze as targets. After you select targets, the app uses their position data during the selected time range to prepare the meeting analysis.

Step 2 — Set the time range

Set a start and end time for the analysis. The app searches for positions for the selected targets within this time range. If the app finds no positions for a target in the selected range, the analysis cannot produce meetings for that target.

Step 3 — Choose analysis mode

Meetings Finder runs in one of two modes, based on how many identifiers you select.

- *Neighbor discovery mode* (single target)
- You select exactly one target. The app finds other identifiers that were near the target during the time range.
- *Co-location mode* (multiple targets)
- You select two or more targets. The app finds meetings only between the selected targets.

Step 4 — Set proximity parameters

Set the parameters used in both modes:

- **Max distance (meters)**
- Two identifiers count as near each other when their positions are within this radius. Default is `15` meters.
- **Time window (minutes)**
- Positions are compared only when they fall within this time window of each other. Default is `10` minutes.
- **Time granularity (minutes)**
- Positions are sampled at this interval to reduce data volume and speed up analysis. Default is `5` minutes.

Step 5 — Run the analysis

Start the analysis after targets, time range, and parameters are set.

- In neighbor discovery mode, the app scans position records in the time range and returns identifiers that match distance and time criteria.
- In co-location mode, the app compares positions between selected targets only.

Step 6 — Review results in the visualizations page

Open the visualizations page to explore the results. The page includes:

- Identifiers legend
- Map

- Timeline scatter plot

Understanding the Output

Meetings Finder presents results in three coordinated views.

Identifiers legend

The legend is a scrollable list of identifier cards. Each card includes:

- A color-coded label used across map and timeline
- A role indicator (target or neighbor)
- Meeting count
- Total meeting duration with targets (for neighbors)
- **Show / Hide** toggle
- **Focus** toggle
- Tags Use the legend to limit how many identifiers appear on the map and to filter the list.

Map

The map shows meeting-related positions for visible identifiers:

- Target positions use crosshair-style markers (hollow circle with inner dot).
- Neighbor positions use filled circles. When you enable **Focus** for an identifier, the map draws its full trajectory as a line connecting all positions in time order, not only meeting positions.

Timeline scatter plot

The timeline scatter plot shows positions over time:

- Time appears on the horizontal axis.
- Identifiers appear on the vertical axis. Drag on the plot to select a sub time period. The map, focused trajectories, and legend update to match the selected period. Select **Clear time range** to return to the full view.

Tips for Best Results

- Start with the default parameters, then tighten **Max distance (meters)** when you need stricter co-location.
- Increase **Time window (minutes)** when position sampling is sparse and you want to catch near-simultaneous detections.
- Use co-location mode when you already have a defined group and want meetings only among them.

- Use the legend filters and identifier limit to keep the map readable.
- Use timeline brushing to isolate a specific period, then review the map for that window.

Known Limitations

- Results depend on the quality and coverage of position data. Missing or sparse positions reduce recall.
- Larger time ranges can increase runtime and produce crowded maps and timelines.
- Time granularity sampling can omit short meetings in neighbor discovery mode when buckets are coarse.
- In co-location mode, identifiers that never meet another selected target do not appear in results.

Online Face Recognition

Overview

Online Face Search detects faces in an image and searches the web for visually similar faces. You upload an image or open one from a **Workspace**, select a detected face, and run a search that returns matches with similarity scores. Outputs include a grid of best matches and a downloadable ZIP of matched images. Results require review because similarity is not identity confirmation.

When to Use This Application

- You have a face image and need to find visually similar images on public web sources.
- You need lead generation when you have limited identifiers beyond an image.
- You need to compare multiple faces in one image by selecting each detected face.
- You need a ZIP package of candidate matches for offline review or disclosure workflows.
- You need to store the input image in a workspace as part of an investigation record.

Before You Begin

- Prepare an image that contains at least one clear, visible face.
- If you plan to save the image to a workspace, confirm you have write access to the destination folder.
- Confirm your environment allows outbound access to the online face search service.

Step-by-Step Walkthrough

Step 1 — Provide an image

Provide an image using one of these methods:

- Upload a file using the file upload button.
- Open from workspace by right-clicking an image file and launching the app from the context menu. After the image loads, it appears in the main area.

Step 2 — Detect faces

Select **Detect Faces** in the left sidebar. The app scans the image and highlights detected faces:

- Each face is outlined with a red border.
- Each face receives a number (Face 1, Face 2, …).
- A message confirms how many faces were found. If no faces are detected, the app shows an error message. Load a different image and try again.

Step 3 — Select a face

If the image contains more than one detected face, use the dropdown selector to choose which face you want to search.

Step 4 — Configure search settings

Configure the search settings in the left sidebar before you run the search.

Step 5 — Save image to workspace (optional)

Save the input image to a workspace when you want to preserve it as evidence or share it with your team.

1. Enable **Save image to workspace**.
2. Select the destination **Workspace** from the dropdown.
3. Enter the file path where the image should be stored, for example `Faces/my-image.png`. The app saves workspace images in `PNG` format.

Step 6 — Search the web for a face

Select **Search Web for Face**. The app:

- Analyzes the selected face
- Sends it to the online face search service
- Shows a loading spinner with a scanning message

- Displays results when the search completes The search can take up to ~100 seconds depending on service response time.

Step 7 — Review results

Results appear under **Best Matches** in a grid. Each match includes:

- A face thumbnail
- The source URL
- A similarity score Use the per-result context menu (⋮) to interact with an individual match.

Understanding the Output

Online Face Search outputs a ranked grid of web matches.

- **Face detection overlay**
- Red borders and face numbers show which faces the app detected in the input image.
- **Best matches grid**
- Each result includes a thumbnail, source URL, and a similarity score. Higher similarity indicates a closer visual match, but it does not confirm the person's identity.
- **Similarity score**
- Use the score to prioritize review. Confirm candidates by opening the source images and checking contextual evidence.

Saving and Exporting Results

Online Face Search supports two output actions:

- Save the input image to a **Workspace** (optional)
  - What it does: stores the input image in a workspace folder
  - Where it ends up: specified folder path in the selected workspace
  - Settings: workspace selection and file path
  - Note: saved images are stored as `PNG`
- Download matched images as ZIP
  - What it does: downloads all matched face images as a single ZIP file
  - Where it ends up: your local device downloads
  - Settings: none Select **Download Images** to download the ZIP.

Tips for Best Results

- Use a clear, well-lit image with faces that are not heavily obscured.
- Run **Detect Faces** before adjusting search settings so you confirm face selection.
- Use `Facenet` as a default model. Try `ArcFace` or `Facenet512` when you need a different trade-off in matching behavior.
- Increase **Max Results** when you need broader coverage and the first run returns few matches.
- Save the input image to a workspace when you need auditability and team review.
- Retry after timeouts. The online search relies on an external service and can fail due to connectivity or service delays.

Known Limitations

- Web results depend on an external service. Timeouts and connectivity failures can occur.
- Similarity is not identity confirmation. False positives can occur, especially with low-quality images.
- If no faces are detected, you must provide a different image or improve image quality.
- Workspace-saved images are stored as `PNG`, which may change file format from the original upload.

Phone & Email Lookup

Overview

Phone & Email Lookup finds profile information linked to a phone number or email address by querying multiple sources and presenting results in one view. You enter one or more phone numbers or email addresses, run a lookup, and review results grouped by source. Outputs include collapsible source sections, optional raw responses, and an exportable JSON file saved to a **Workspace**. Results vary by source coverage and require review before you treat them as verified identity data.

When to Use This Application

- You need quick enrichment for a phone number or email address during triage.
- You need to compare results from multiple sources in one place instead of checking them one by one.
- You need to identify linked identifiers (emails, addresses, usernames) for follow-up searches.
- You need to export lookup results to a workspace for later review or sharing.
- You launch from a person, phone number, or email entity and want fields pre-filled from existing context.

Before You Begin

- Prepare the identifiers you want to look up:
  - Phone numbers with international prefix (for example `+44`, `+1`, `+33`)
  - Email addresses in standard format (for example `name@domain.com`)
- Confirm you have write access to at least one **Workspace** if you plan to export results.
- Confirm outbound access to the configured lookup sources is available in your environment.

Step-by-Step Walkthrough

Step 1 — Open the application

Open *Phone & Email Lookup* from Octostar. If you launch the app from a person, phone number, or email entity, the sidebar fields are pre-populated with contact details already linked to that entity.

Step 2 — Enter phone numbers (optional)

In the sidebar, use the **Phone Numbers** section. Enter one or more phone numbers:

- Separate values using commas or new lines.
- Include the international prefix for each number (for example `+1`, `+44`, `+33`).
- Use spaces, hyphens, or dots if needed. Example formats:
- `+1 555-867-5309`
- `+44.20.7946.0958`
- `+33 6 12 34 56 78`

Then select **Lookup Phone Numbers**.

The app validates formatting. If a number is missing the international prefix or is malformed, the app shows an error and does not start the lookup until you correct it.

Step 3 — Enter email addresses (optional)

In the sidebar, use the **Email Addresses** section. Enter one or more email addresses:

- Separate values using commas or new lines.
- Use standard email format (for example `name@domain.com`).
- Treat values as case-insensitive.

Then select **Lookup Email Addresses**.

The app validates each address and alerts you if any are invalid. After validation, it queries the available sources and displays results.

Step 4 — Review results by source

Results appear in the main content area. Each source appears in its own collapsible section.

- Expand or collapse a section to view results from that source.
- Use source logos to identify where each profile comes from.

- If a source returns multiple profiles for the same input, the app shows numbered entries.
- Review profile images when available.

A **Raw Data** section appears at the bottom of each result when you need to inspect the unprocessed response. If a source has no result for an input, it does not appear.

Step 5 — Switch between previous lookups (optional)

Use the sidebar history selectors to switch between lookups you ran in the current session:

- **Previous phone number searches**
- **Previous email address searches**

Switching history entries does not re-run the lookup.

Step 6 — Save results to a workspace

Use the export controls in the sidebar.

1. Select a destination **Workspace** from the **Workspace** dropdown. Only workspaces where you have write permission appear.
2. Enter a folder path in **Folder path**. Default is `Extracted/`.
3. Select **Save JSON**.

The app shows a confirmation message with the save location. The exported file contains the full structured lookup results.

Understanding the Output

Phone & Email Lookup displays results grouped by source.

- **Source sections**
- Each data source appears as a collapsible section. A section can contain one or more profiles for the same input.
- **Profile content**
- Depending on source and input type, results may include identity details, account status, activity indicators, carrier/location details, and linked identifiers.
- **Raw Data**
- Use this section when you need the complete response for audit or deeper inspection. Coverage varies by source. A missing section does not indicate a failed lookup. It indicates no data returned from that source.

Saving and Exporting Results

The app supports exporting lookup results as JSON.

- **Save JSON**
  - What it does: exports the full lookup response in structured JSON format
  - Where it ends up: selected **Workspace** and folder path (default `Extracted/`)
  - Settings: **Workspace** selection and **Folder path**

Tips for Best Results

- Always include the full international prefix for phone numbers. Missing prefixes are the most common validation error.
- Enter multiple phone numbers or emails at once using commas or new lines.
- Use the session history selectors to switch between lookups without re-running queries.
- Use `Extracted/Cases/2026/`-style folder paths to keep exports organized.
- Treat results as leads. Verify identity claims against other evidence before acting on them.
- If a lookup returns fewer sources than expected, rerun later. Source availability can vary.

Known Limitations

- Results vary by source availability. Not every source returns data for every input.
- Output can contain conflicting profile data across sources. Validate before using in reports.
- The app depends on multiple external sources. Connectivity or source outages can reduce coverage.

Forensic Importer

Overview

Forensic Importer imports UFDR mobile extraction reports and turns them into searchable views such as dashboards, timelines, maps, and communication graphs. You upload a UFDR report or package, select one or more devices, and then explore artifacts such as calls, chats, locations, and media. Inputs are UFDR XML or UFDR package files (including ZIP archives). Outputs are interactive analysis views and exportable tables and files. Results reflect what exists in the extraction and do not replace forensic validation.

When to Use This Application

- You need to review a UFDR extraction from Cellebrite and quickly locate relevant artifacts.
- You need a unified timeline to reconstruct a sequence of events across multiple data types.
- You need to identify key contacts using a communication network view before reading message content.
- You need to compare two or more devices for shared contacts, co-presence, or shared Wi-Fi networks.
- You need CSV exports of calls, contacts, web history, wireless networks, or credentials for reporting.

Before You Begin

- Prepare a supported UFDR input:
  - UFDR XML report file
  - UFDR package file (ZIP including report and media)
  - Standard ZIP archive containing UFDR reports
- Confirm you have sufficient session storage quota to upload and process the report.
- If you work with sensitive credentials, ensure you follow your organization’s screen and sharing policy before revealing passwords.

Step-by-Step Walkthrough

Step 1 — Upload a UFDR report

Open the app in your browser. In the left sidebar, select **Upload UFDR Report** and choose a file. The app accepts:

- UFDR XML report files
- UFDR package files (ZIP archives including report and media)
- Standard ZIP archives containing UFDR reports A progress indicator shows while processing runs. When processing completes, the device appears in the sidebar under your uploaded phones list. Repeat this step to upload multiple devices.

Step 2 — Select devices for analysis

In the sidebar, each uploaded device appears as a selectable item.

- Select a device to include it in analysis. A checkmark (✓) appears next to selected devices.
- Select one device to enter single phone analysis.
- Select two or more devices to enter multi-phone analysis.
- To remove a device, select the trash icon (🗑️) next to its name.

Step 3 — Explore single phone analysis

When you select one device, the left panel shows **Insights** and **Artifacts**.

Review insights

Use **Insights** for high-level summaries derived from the full report.

- Overview dashboard
  - 24-hour activity clock
  - Top contacts (top 10)
  - Movement profile (frequently visited locations, time-of-day coloring, mobility classification)
  - Communication breakdown (calls, messages, emails)
- Graph analytics
  - Network diagram where nodes are contacts or phone numbers and links are communications
  - Hubs are visually emphasized
  - Filter by message type, direction, and time range
- Unified timeline
  - Scrollable timeline combining calls, messages, web visits, locations, and system events
  - Events are color-coded by type

Review artifacts

Use **Artifacts** to access specific extracted categories. Common artifact sections include:

- Device info
- Calls
- Contacts
- Chats
- Emails
- Media
- Web history
- Locations
- Passwords & credentials
- Wireless networks

Use built-in filters and sorting within each section to narrow the scope (for example date filters in Locations and Web History).

Step 4 — Explore multi-phone analysis

Select two or more devices to switch to multi-phone analysis. Use the multi-phone views to correlate across devices:

- Communication network
  - Highlights contacts that appear on multiple phones
  - Hubs: contacts on three or more devices
  - Bridges: contacts on exactly two devices
  - Visualizes communication volume between devices
- Location correlation
  - Overlays GPS points for all selected devices
  - Helps identify shared locations and possible co-presence (same place, same time)
- Wi-Fi correlation
  - Finds Wi-Fi networks that appear across devices
  - Supports shared-location inference when GPS is missing
- Unified timeline
  - Single timeline combining events from all selected devices
- Media gallery
  - Combined media view across all selected devices

Step 5 — Manage session storage

Monitor storage usage at the bottom of the sidebar. The app shows:

- Current session storage usage
- Maximum allowed storage When you finish analysis or storage is low, select **Clear Cache**. Clearing the cache removes uploaded files and processed data from your session.

Step 6 — Export artifacts for reporting

Use export options in artifact views when you need outputs outside the app.

- CSV exports are available for:
  - Contacts
  - Calls
  - Web history
  - Wireless networks
  - Passwords (redacted and full)
- Media files can be downloaded individually from the media gallery. Credentials export supports:
- Redacted exports (safe for sharing)
- Full exports (sensitive)

Understanding the Output

Forensic Importer organizes results into two layers:

- **Insights**
- Aggregated summaries and visualizations. Use these to identify time windows, key contacts, and movement patterns quickly.
- **Artifacts**
- Raw extracted categories. Use these to validate findings and retrieve exact messages, timestamps, and metadata. Key behaviors:
- **Single phone analysis** focuses on one device and shows insights and artifacts for that report.
- **Multi-phone analysis** correlates contacts, locations, Wi-Fi networks, and timelines across selected devices.
- **Credentials** are redacted by default. You must explicitly reveal them before viewing full values. Map and timeline outputs depend on what is present in the UFDR report. For large location datasets, the map displays up to `2,000` location points for performance.

Saving and Exporting Results

Forensic Importer supports exporting artifacts, primarily as CSV or individual file downloads.

- CSV exports
  - What it does: exports tabular artifacts for reporting
  - Settings: section-specific filters (date ranges, types) apply to the exported data
- Media downloads
  - What it does: downloads individual media files from the gallery
  - Where it ends up: local download
  - Settings: per-file selection
- Password exports
  - What it does: exports credentials in redacted or full format
  - Settings: choose redacted vs full

Tips for Best Results

- Start with the overview dashboard to identify key time windows and high-volume contacts.
- Use the unified timeline to reconstruct a specific incident window before reading all chats.
- Load all relevant devices before selecting them so you can switch between single and multi-phone views.
- Apply date filters in locations and web history to reduce noise.
- Use graph analytics to identify hubs before you triage message threads.
- Select **Clear Cache** at the end of a session to free storage and remove session data.

Known Limitations

- Only UFDR inputs are supported. Other mobile extraction formats are out of scope.
- Location maps depend on extracted GPS records. Some devices or apps may provide sparse or no location data.
- Large location datasets are capped at `2,000` points on the map for performance.
- Session storage is quota-limited. Uploading multiple large UFDR packages can exhaust available space.
- Clearing the cache removes all uploaded devices and processed results from the session.

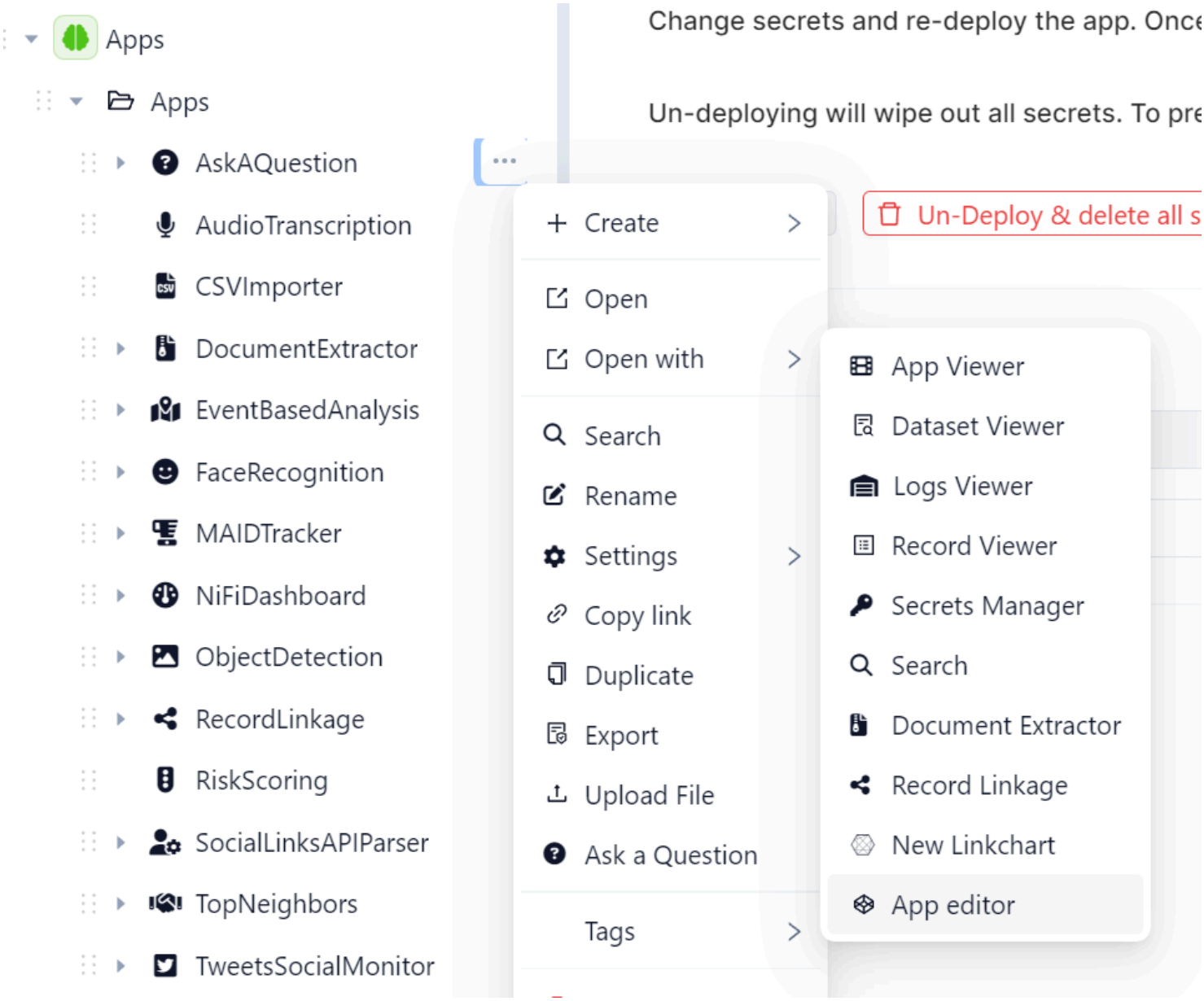
Developer Guide

The Developer Guide provides **technical documentation** for building and extending functionality within Octostar. It is intended for developers and technical users who want to understand the platform's application framework and implementation approach.

Writing Applications

What are Applications?

Apps are a way of providing custom functionality into the Octostar platform through seamless integration with existing or new software and systems. Apps can be specifically tailored to use cases where the dashboard, linkcharts and templates provided by Octostar do not suffice, such as for complex and dynamic user interfaces, long-running computations, data analysis, and so on. Apps are integrated into the platform like an Operating System does, where they are accessible both via a dedicated list of apps as well as by opening files, folders or entities with the **Open With...** option.



From the developer point of view, an App is a special folder inside any Octostar workspace. This folder must be:

- Placed under the `Apps/` root folder of the workspace
- Specially flagged as an app by setting the folder's `os_item_content_type` field to `os_app` Ready-to-go apps can be created by selecting **Create → App** from the context menu of a workspace.

VISIBLE APPS Much like files and entities, only apps defined in open workspaces are visible, whether from the list of deployed apps, the context menu of an entity or as a transform. Closing a workspace will hide all associated apps from all platform interactions.

App Interface in Octostar

Apps have dedicated options when their context menu is opened, such as:

- App Viewer
- App Editor
- Logs Viewer
- Secrets Manager The **App Editor**, in particular, opens a tab where the user can select code files to be opened with the Octostar in-built code editor (Monaco Editor), as well as showing options to **Deploy App**, **Undeploy App**, and more. The **Secrets Manager** *(accessible in the App Editor mode → Actions → Secrets Manager)* is used to write sensitive informations that apps require to run, such as secret tokens or authorizations to connect to external or internal services. Apps can then fetch these secrets via function `octostar.utils.jobs.apps.get_app_secret` in the `octostar-python-client` library. The **Logs Viewer** is particularly useful to display the logs emitted by the app's container, whenever the app is deploying or already deployed.

App Structure

Apps must have, in order to be deployed:

- - a `manifest.yaml` file. This file defines the appearance and characteristics of an app, as well as the transforms it exposes to other apps and components of the platform, including which files or entities can be opened with said app. Crucially, it also specifies the base Docker image in which to run the app. More details on the manifest can be read here: The Manifest (The Manifest)
- A file acting as the entrypoint, typically a `main.sh` or a `main.py` (a custom entrypoint can be specified via the manifest)
- If the app contains a `requirements.txt`, that file will be pip installed when the app is deployed (before the entrypoint)
- any other file which the app uses (e.g. code, libraries, files) Apps can be deployed by opening them with the **App Editor** and selecting the **Deploy App** option. Doing so will:
- create an `app.zip` file containing all the files in the folder and which is uploaded to S3
- extract and deploy the above zip in a kubernetes pod (whose image can be specified from the manifest)
- generate an octostar URL for the app and store it in the record of the zip file. This acts as an endpoint for the app itself, therefore **an app can serve any web framework written in any language** (commonly streamlit, flask, fastapi). The app will locally serve from port 8080 to the Octostar platform. If multiple servers from the app need to be interfacing with Octostar (e.g. fastapi back-end + streamlit front-end) a reverse proxy like nginx can be installed and launched.

Interactions between Apps & Octostar

Apps can interact with Octostar through the **Octostar API**(s). The main tools for integration we have at disposal are:

- A REST API internally available at `{OCTOSTAR_URL}/api/octostar/`. Documentation on available endpoints is via OpenAPI at `{OCTOSTAR_URL}/api/octostar/meta`
- A front-end package for python + streamlit called `streamlit_octostar_research` and available on pypi

- A back-end SDK for python called `octostar_python_client` which is packaged with the Octostar installation and available at `{OCTOSTAR_URL}/api/octostar/meta/octostar-python-client.tar.gz`
- A utility library for python + streamlit called `streamlit-octostar-utils` and available on pypi
- Tuwo front-end APIs: `OntologyAPI` and `DesktopAPI`

The Manifest

The App manifest file

The `manifest.yaml` is a required file which defines how the app integrates closely with the user's context and actions. The manifest is a `yaml` file listing some simple key/value settings (such as `title` and `description`) and possibly a set of functionalities: `filters` , `transforms` , and `services` as described below. These functionalities work together to expose powerful context-based extensions.

A Simple manifest.yaml

```yaml
name: Hello World
version: 1.0.0
description: A very simple iframe type application
icon: fa-hand
```

The manifest above provides a custom name, title, and icon. This app uses the default docker for a context-free app.

A Fully Featured manifest.yaml

```yaml
name: Credit Check
version: 1.0.0
image: octostar/code-streamlit
description: Shows a live credit check for the selected person
icon: fa-magnifying-glass-dollar
engine: k8s
entry_point: bash /app/main.sh

memory_limit: 8Gi # App will be killed for OOM and restarted by K8s
memory_requested: 250Mi  # App won't even deploy if less memory is available
storage_limit: 10Gi # App will hit "No space left on device"
storage_requested: 1Gi # App will not be even deployed if less storage is available

semantic_bindings:
    record_viewer:
        - person

watchers:
    - name: "check_updates_for_person"
      entry_point: bash /app/watcher.sh x
      description: "Check if a person has new credit events & notify"
      semantically_bound: [ person ]
      interval: "20m"

filters:
  persons:
    type: semantically_bound
    concepts:
      - person

transforms:
  open_credit_check_iframe:
    description: opens an iframe
    type: iframe
    title: '{{ app.folder.os_item_name }}'
    icon: fa-magnifying-glass-dollar

  enhance_item:
    description: Adds some_new_attribute to item
    type: javascript
    code: |
      function transform(item){
        item.some_new_attribute = 'hello'
        return {item};
      }

services:
  open_background_check:
    role: context_menu
    group: open
    label: Credit Check
    icon: fa-magnifying-glass-dollar
    description: Opens the streamlit app passing the person as context for credit check
    accepts:
      - persons
    transforms:
      - enhance_item
      - open_credit_check_iframe
```

Dissecting the manifest.yaml

Simple Key/Value Settings

```yaml
name: credit_check
version: 1.0.0
description: Shows a live credit check for the selected person
icon: fa-magnifying-glass-dollar
image: scarduzio/code-streamlit
entry_point: bash /app/main.sh
```

The `name` , `title` and `icon` for the app are provided. **Other options include:**

- `image` to specify a specific docker image used for the app.
- `entry_point` the command to bootstrap the frontend app
- `no_archive` if true the app zip file is not uploaded/expanded on the server. The commands from `main.sh` will be directly executed in the container.
- `url_from_logs` if true, the octostar web application will try to extract a url (maybe it contains a token) from the app logs, and include it on the app url
- `engine` the default is `k8s`; other options include `html` and `react` which require no kubernetes job
- `run_as_user` can give the numeric user_id to use if not the default.
- `startup_timeout` can give the number of seconds to wait if different from the default
- `container_port` can give the docker container http port if different from the default (8080)
- `template` the name of a template to open as the frontend. This can be useful to give a user interface to a fastAPI backend, for example. The template must exist within the templates folder of the app.

Semantic Binding

The optional `semantic_binding` section provides a way to register the app to feature within the some contexts. Beneath each context are listed the ontology concepts supported by the app.

The following contexts are supported:

- `record_viewer` : The app will appear as a tab in the record viewer.

In this example for a map app, the concepts of `person` , `vehicle` , `mobile_phone` and `place` are semantically bound to the `record_viewer` The semantic binding for `person` includes `priority` and `maximize` properties recognised by the record viewer. In this case if the binding has the highest priority (lower number means higher priority), the app will show in full tab view due to the `maximize: true` property. (Note for the record_viewer the record details have a priority of 200 and media gallery has priority of 100)

```yaml
semantic_bindings:
    record_viewer:
        - person:
            priority: 50
            maximize: true
        - vehicle
        - mobile_phone
        - place
```

Watchers

The `watchers` section contains the list of watcher functions the app exposes. When an app offers a watcher function, users are free to create local entities of type `watch_intent` in the workspaces containing a reference to the specific watcher function.

```yaml
watchers:
    - name: "check_updates_for_person"
      entry_point: bash /app/watcher.sh x
      description: "Check new credit events & notify"
      semantically_bound: [ person ]
      interval: "20m"
```

In this case, the watcher function named `check_updates_for_person`, and it can process only entity of type "person". It will run every 20 minutes (minimum interval is 5 minutes).

An example of a `watch_intent` local entity created by an app or by octostar to reserve the periodic execution of the above watcher function about an entity of type person.

```
 {
            "entity_id": "3a96da2f-96ee-40d6-9c5b-882a99829c56",
            "entity_type": "os_watch_intent",
            "entity_label": "Check new credit events & notify - credit_check - Thomas Jefferson",

            "description": "<Normally null, if value is 'STOPPED', this execution will be skipped>",
            "previous_run_output": "<base64 encoded value, useful to compare results and finding for differences
(optional)>",
            "os_last_updated_at": "2024-03-20 11:25:56",
            "os_created_by": "operator@octostar.com",
            "app_id": "oj-c2-ea-e9e",
            "os_last_updated_by": "operator@octostar.com",
            "os_workspace": "c43ae201-d175-406e-8f04-f8d1d8b7e402",
            "last_run": null,
            "watcher_name": "credit_check",
            "os_item_type": "os_watch_intent",
            "os_entity_uid": "3a96da2f-96ee-40d6-9c5b-882a99829c56",
            "arguments": "<base64 encoded JSON representing the arguments for the watcher function (optional), e.g.
twitter handle>",
            "interval": "20m",
            "os_created_at": "2024-03-20 11:25:56"
        },
```

The `octostar-python-client` pip dependency offers a practical helper to implement watcher functions without worrying much about the details, here is how to use it with an example:

```
from octostar.utils.watchers import process


def monitor_watcher(w_intent:WatcherIntent) -> Optional[BaseException]:
    if w_intent.app_name == "social_monitor":
        search_filter = base64_to_dict(w_intent.arguments)
        last_ts = base64_to_dict(w_intent.previous_run_output)["last_ts"]
        if last_ts:
            last_day = last_ts.split(" ")[0]
            search_filter["from_date"] = last_day
        tweets = get_tweets_from_sl(search_filter)
        if tweets:
            tweets_df = tweets_records_to_df(tweets, parser=True)
            if last_ts:
                tweets_df = tweets_df[tweets_df["creation_date"] > last_ts]
            last_ts = max(tweets_df["creation_date"])
            save_records(tweets_df, [w_intent.os_workspace], w_intent.watcher_name)
            if search_filter["notify"]:
                notify_about_new_tweets(w_intent, len(tweets_df))
        save_new_latest_timestamp.sync(w_intent, last_ts)

process.sync(processor=monitor_watcher)
```

Filters

The `filters` section, if provided, lists one or more named re-usable selectors which return a true/false value to accept an offered item/items. The property names for a filter depend on the filter `type` . In the example, a `semantically_bound` filter has a `concepts` property, listing the concepts which are accepted by the filter. In this case, just one concept is provided: `person` . Any person record will be accepted by the filter, including records of a type which are a subclass of `person` , for example `student` or `fisherman`.

```
  persons:
    type: semantically_bound
    concepts:
      - person
```

The following filter types are supported:

semantically_bound

As noted above, the `semantically_bound` filter has a `concepts` property, which lists one or more accepted concepts.

javascript

Defines an `accepts` function in javascript code.

```
  is_requirements_txt:
    type: javascript
    code: |
      function accepts(item) {
        return item.os_item_name === "requirements.txt";
      }
```

A javascript filter has a `code` attribute which defines an `accepts` function which is evaluated in the browser. In the example above we see the filter returns a true value for an `item` if the `os_item_name` attribute equals `requirements.txt`

python

Defines an `accepts` function in python code.

```
is_a_folder:
  type: python
  code: |
    def accepts(item):
      return item["os_item_type"] == "os_folder"
```

A python filter has a `code` attribute which defines an `accepts` function which is evaluated in the browser. In the example above we see the filter returns a true value for an `item` if the `os_item_type` attribute equals `os_folder` .

For both `javascript` and `python` filters, the possible parameters that may be used by the `accepts` function include:

- **`item`**: The `workspace_item` under consideration.
- **`items`**: An array of `workspace_item` objects under consideration.
- **`entities`**: A list of `entities` under consideration.
- **`workspace`**: The workspace to which the `item` or `items` belong. The full list of items in the workspace can be found in the `workspace.items` property.
- **`ee`**: The `EventEmitter`. **`graph`**: An object having `nodes` (records/entities) and `edges`.

Transforms

Transforms are operations taking the accepted context as input, and producing as output either a modification to the context, a visible change to the user interface, or both. From the example:

```
enhance_item:
  description: Adds some_new_attribute to item
  type: javascript
  code: |
    function transform(item){
      item.some_new_attribute = 'hello'
      return {item};
    }
```

Above we see a `transform` named `enhance_item` . The transform defines in javascript (`type: javascript`) a `transform` function which adds `some_new_attribute` to the `item` parameter, then returned as the `item` property of the output. In this way the `item` is effectively and silently transformed: the context to any downstream transform is modified with the updated `item` parameter.

Also from the example:

```
open_credit_check_iframe:
  description: opens an iframe
  type: iframe
  title: '{{ app.folder.os_item_name }}'
  icon: fa-magnifying-glass-dollar
```

Above we see a transform of type `iframe` . The transform opens a new tab in which the app is opened with the default app url (an alternate url could be provided). An icon and title for the tab are given. The `iframe` transform allows powerful integration with the user interface, for example by using streamlit apps which can communicate directly with the Octostar desktop or server using our APIs.

The full list of available transforms are below:

Python

Executes a python `transform` function in the browser. The function may be provided either inline using a `code` attribute, or in a file referenced using the `file` attribute.

Javascript

Executes a javascript `transform` function in the browser. The function may be provided either inline using a `code` attribute, or in a file referenced using the `file` attribute.

Nunjucks

Executes a nunjucks template `transform` in the browser. The template may be provided either inline using a `template` attribute, or in a file referenced using the `file` attribute.

Iframe

Opens a tab to the given `url` attribute, or the default app url. Exposes the browser api to the app using `window.message` event api as exposed with the octostar streamlit api.

- Note the item(s) exposed within the iframe are referenced as `record` and `records` .
- Each record will open in a different tab. To use the same tab include `reuse: true`

HTTP

Without opening a tab, invokes a server http operation which can be a service offered by the app (for example a service in a flask app), or an absolute url to an external service. The following optional parameters may be provided:

- `headers` an object for which any attribute can be a nunjucks template or a string
- `data` an object for which any attribute can be a nunjucks template or a string
  - note that if the parameter is a string and matches an entry in the transform context, then that context entry is returned. The example below sends the `item` object as the body of the http POST request.
- `url` can be relative to the app url `/credit_rating` or an absolute url eg:. `https;` `//example.com/credit_rating`
- `method` default is GET, but POST, PUT etc may be used.

Here is an example of posting an item to a flask service

```
  post_to_flask:
    description: Upload the file to flask
    type: http
    method: POST
    url: "{{ app.url }}/upload"
    headers:
      Content-Type: "application/json"
    data: item
```

Below is an example of a server side flask service which receives the item from the above transform, and returns a notification object:

```
@app.route('/upload', methods=['POST'])
def notification():
    item = request.get_json()
    return jsonify({
            "notification": {
                    "message": "Upload Successful!",
                    "description": f"""The item '{item["entity_label"]}' was successfully uploaded""",
                    "level": "info"
                }
    })
```

Parameters

The following parameters may be available to the accept filter and transform functions. The accept filter must check that the parameters it needs are present and acceptable then return true, otherwise it must return false.

- - `item`: the workspace item under consideration
- `items`: the list of workspace items under consideration
- `workspace`: the relevant Workspace object
- `graph`: the graph of nodes (Entity[]) and edges (EdgeSpec[])
- `dataTransfer`: a dataTransfer like object
- `entities`: the Entity[] to be considered The following parameters are available to the transforms (but not to the accept filter)
- - `OntologyAPI`: the OntologyAPI (todo documentation)
- `DesktopAPI`: the DesktopAPI (todo documentation)
- `onTaskCompleted`: a function which can be invoked with the task result
- `onTaskCanceled`: a function which can be invoked to cancel the task
- any other attributes assigned by previous tranform(s)

Services

Services connect the Octostar user interface with the transforms by the app. From the example:

```yaml
open_background_check:
  role: context_menu
  group: open
  label: Credit Check
  icon: fa-magnifying-glass-dollar
  description: Opens the streamlit app passing the person as context for credit check
  accepts:
    - persons
    - legal_person
  transforms:
    - enhance_item
    - open_credit_check_iframe
```

Here we have a service `open_background_check` in the `context_menu` role (the default). The service appears under the `open` actions on the context menu, having the label `Credit Check`.

The service is available whenever the context (what was right clicked on) is accepted by any of the listed filters in the `accepts` section. Each name given in the `accepts` section must be either either:

- - The name (key) of a filter defined in the `filters` section of the manifest
- OR the name of a concept, eg `legal_person` in the example (a shortcut semantic binding filter).

When the user select the operation, two transforms are performed in sequence. First the `enhance_transform` then the `open_credit_check_iframe`

Other roles (including `record_viewer` and `set_viewer` will become available soon).

The `group` attribute defines where the service appears in the context menu

- - open
- edit
- transform
- general
- danger

The `subgroup` attribute may be included for `transform` services.

The optional `show_progress` parameter can be set to `false` (default is `true`). When `false` the service progress bar is not displayed.

A `params` key can also be included on the service (not shown in the example) to include any additional parameters to be passed to the transform.

This example provides a transform service which uses the http example above to post the item to flask, then send the result from flask to the event emitter.

```yaml
name: Send Folder To Flask
version: 1.0.0
description: Example posting from context menu to flask

filters:
  is_a_folder:
    type: semantically_bound
    concepts:
      - os_folder

transforms:

  post_to_flask:
    description: Upload the file to flask
    type: http
    method: POST
    url: "{{ app.url }}/upload"
    headers:
      Content-Type: "application/json"
    data: item

  emit_notification:
    description: Emit notification received from the flask app
    type: emit_event
    topic: octostar:desktop:builtins:showNotification
    args: [ notification ]

services:
  send_to_flask:
    group: transform
    subgroup: Folder
    label: Post to Flask
    icon: fa-car
    description: Posts it to flask
    accepts:
      - is_a_folder
    transforms:
      - post_to_flask
      - emit_notification
```

Apps, Jobs

An Octostar App is executed in its own container, optionally using a custom image, within Kubernetes. But how does it work in detail?

App deployment

In Kubernetes, a `Job` is a controller object that represents a finite, one-off task that runs to completion. It creates one a `Pod` and ensures that it successfully terminates. Restarting it multiple times if crashes. Well, when you "**deploy**" your app in Octostar, our API defines and instantiates a Kubernetes `Job` that typically never completes (unless you instruct it to). An Octostar App is a resilient process, with the following guarantees:

- Will be restarted on crash, yet avoiding crash loops with a backoff strategy.
- Will get executed on another Kubernetes cluster node in case of failures
- A unique URL will be associated to the app
- HTTP connections will be forwarded to the internal port 8088 TCP
- Secrets can be associated to the app (API keys, SSH keys, etc)
- The necessary environmental variables will be provided automatically for the Octostar Python SDK autoconfiguration

Subordinate Jobs

An Octostar app can consume the Octostar API directly, or via the Python SDK.

One of the features in the API is the ability to request the on-demand execution of another instance of the current app. Potentially specifying a custom entry point command. For example a specific Python script, with some specific arguments.

This is useful when the app at hand intends to allocate extra compute/memory resources to parallelise resource intensive executions.

Important

Refrain from spawning subordinate jobs if you intend to parallelise IO bound operations. You won't need more memory or CPU in this case, just find a way to parallelise your code within your main app container.

Subordinate jobs will all be terminated when you un-deploy your app. Alternatively, kill them one by one in the JobsManager UI.

Here is an example of app code (using `octostar-python-client`) that spawns subordinate jobs.

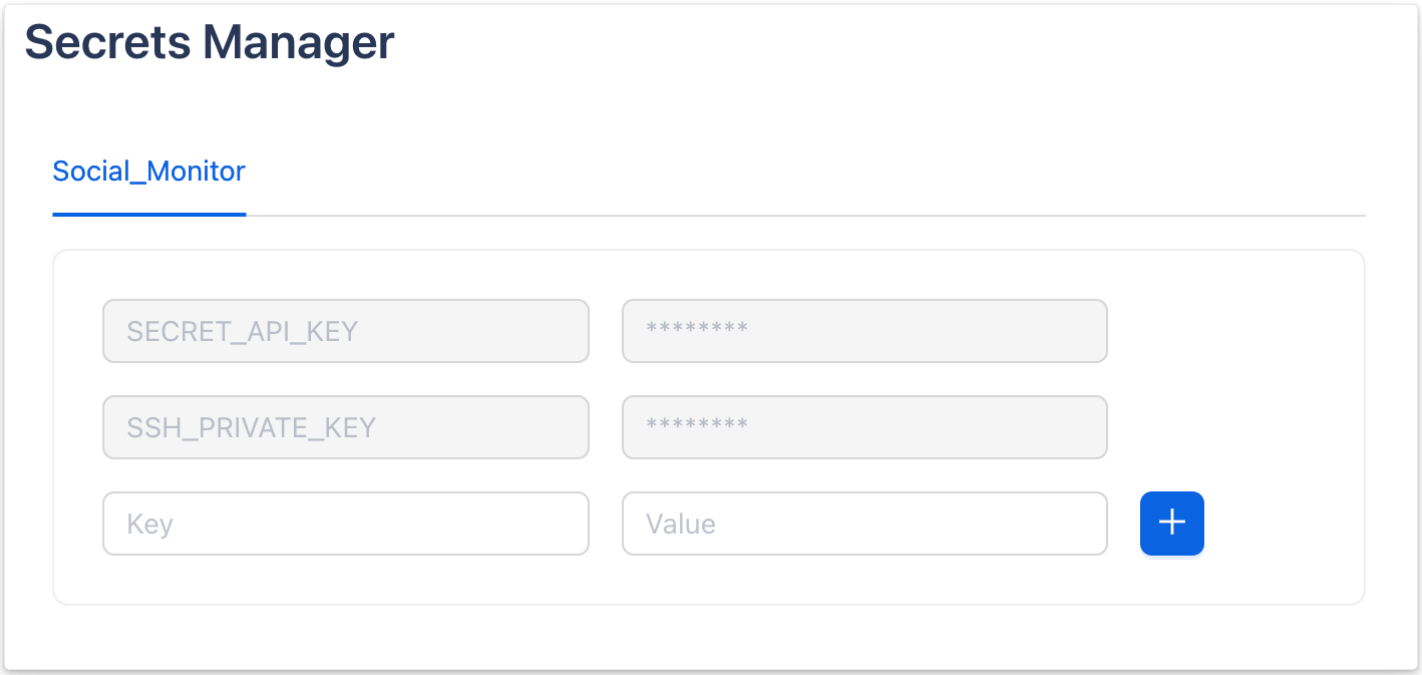
```
from octostar.utils.jobs import execute_new_job
from octostar.utils.notifications import toast

for id in to_process:
  execute_new_job.sync(commands=[f"bash heavy_worker.sh -i {id}"])
  toast.sync(message=f"🚀 New job launched for id {id}")
```

Secrets

Using "Secrets Manager" You can save app-specific secrets using Octostar App editor. This key-value store is backed by the equivalent data structure in Kubernetes, called **Secret.**

This is an object that contains sensitive data such as passwords, OAuth tokens, SSH keys, etc. Secrets provide more control over how sensitive information is used and reduce the risk of accidental exposure.



Writing Python Apps

Python offers an easy-to-read, flexible programming language, with the full power of AI and data analysis. This is why Python is a first-class citizen for Octostar. The platform supports two flavors of Python programming for apps:

- Streamlit, a full-stack solution which can be used to create highly customizable and interactive apps, especially for data analysis and visualization
- FastAPI/Flask, to create REST APIs that other apps or components in the Octostar platform can use, such as manifest transforms and endpoints for the automated data processing pipeline. These apps can make use of pure HTML+CSS+JS, ReactLive, or even another Streamlit server (via nginx) to also expose user interfaces

---

Quickstart Developer Guide

To develop your first app, from any workspace select **Create → App** and choose either the **Streamlit** or **Flask** templates. Next, give a name to the app and, from the **App Editor**, select **Deploy App**.

DO NOT PANIC After deploying your app, attempting to open it will likely result in a 503 or 502 from nginx. This is normal behavior, as the app is deploying (unpacking, installing dependencies, etcetera) but not yet fully deployed. You can monitor the installation state of the app via the Logs Viewer.

From this point onwards, you can begin making changes to the template files, or even begin anew. In general, development can either be:

- **Within the Octostar Platform**, inserting files using the built-in editor and using the **Deploy App** function. This method is useful for quick development of small-scale applications or for production-ready testing. All functionalities provided by Octostar will work out-of-the-box, without the hassle of working outside of the platform's context. However, it is slow to iterate upon, as each change requires to save the new files and redeploy the app in its entirety
- **Developing locally**. This allows the developer to use the tools and IDE they are most familiar with, with a much faster code-test-debug cycle. However, it does require some tweaks, which are explained in the next section.

MAKING CHANGES AND DEPLOYMENT Remember that, whenever changes are applied to any files in the app, as well as changes in the Secrets Manager, the app ***MUST*** be redeployed for the changes to take into effect. This rule does not apply to changes in the manifest (except those related to deployment) and to applications served directly from the front-end, that is, those using an `engine!=k8s`.

Developing Locally in your own IDE

When developing a complex app, debugging it with your familiar development environment can be very valuable. The steps to develop locally are as follows:

Create a Virtual Environment [Optional, Recommended]

A virtual environment is always recommended when working with Python projects. This can be per-app or globally for multiple apps. You can use your favorite tools to create a venv, such as `virtualenv`, `conda`... For example, using `conda`:

```
conda create -n octostar_env python=3.11
conda activate
```

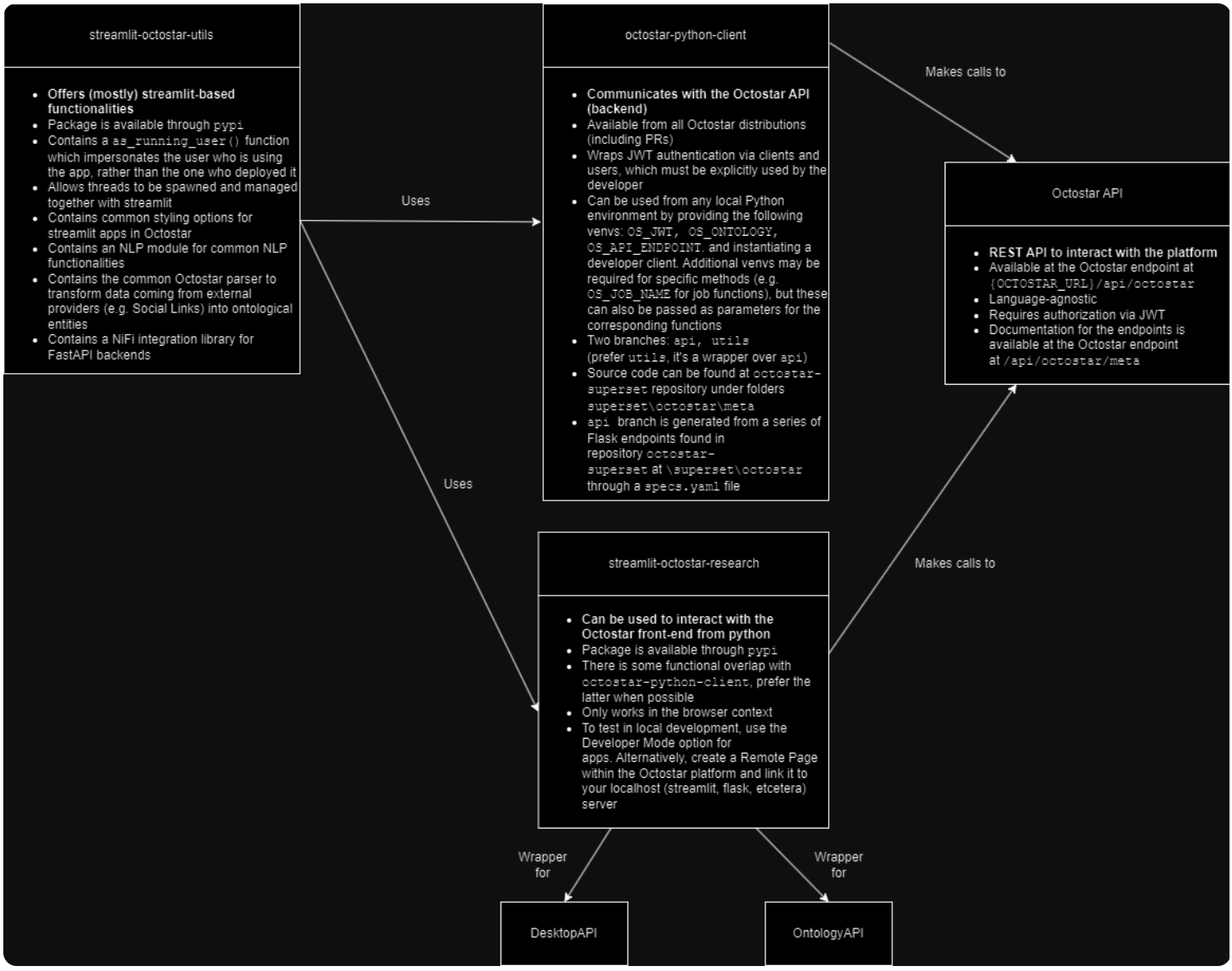
Install the Required Libraries

```
pip install octostar-streamlit
pip install streamlit-octostar-utils
pip install "${OCTOSTAR_URL}/api/octostar/meta/octostar-python-client.tar.gz"
```

As the packages are still in development, it is a good idea to uninstall and purge cache before re-installing them, for any reason:

```
pip uninstall octostar-python-client
pip uninstall octostar-streamlit
pip uninstall streamlit-octostar-utils
pip cache purge
```

A general idea of what these libraries do and how they are interconnected can be explained by this diagram:



Running the App

Develop code as normal in your IDE. To test the app you can either:

- Run and debug it locally, if there are no interactions with Octostar
- Run it via a Remote Page, if there are browser interactions
- Run with a developer client, if there are back-end interactions More on these points in the following sections.

Back-End Calls

Operations like querying for data, read/writes on files, the jobs system and notification systems, app secrets and many more pass through the back-end. In general, all operations that do not require an active browser session can be done as a back-end call. For the full list of supported operations, you can look at:

- The Octostar API at `{OS_API_ENDPOINT}/api/octostar/meta`

- The docstrings in `octostar-python-client`, particularly under the modules under `octostar.utils`. For instance, reading files is supported with function `octostar.utils.workspace.read_file()`, while reading an app's secrets is done via `octostar.utils.jobs.apps.get_app_secret()`. To be able to execute back-end calls against Octostar (in particular via the `octostar_python_client` library), run your app from your IDE as usual but define at least these environmental variables:

Environment Variable	Description
OS_JWT	Your personal JWT token as seen in the "whoami" endpoint at `{OS_API_ENDPOINT}/api/octostar/meta/whoami`. This token is used for authentication when calling any Octostar API.
OS_API_ENDPOINT	The public URL of your Octostar installation.
OS_ONTOLOGY	The name of the ontology as configured in Octostar UI.
OS_DEV_MODE	Set it to the string "true" to enable developer mode. Under developer mode, the behavior of some functions changes (such as the `dev_mode` decorator and `get_app_secrets`).

Some back-end functions may also require additional environment variables, such as **OS_JOB_NAME**, but these can typically be passed as parameters to the corresponding functions.

Next, instantiate a developer client and call the back-end functions as follows:

```
from octostar.client import as_developer_user
from octostar.utils.ontology import query_ontology
with as_developer_user() as client:
  client.execute(query_ontology.sync, "SELECT * FROM dtimbr.person")
```

You can also explicitly pass the environment variables as parameters to `as_developer_user()`, if you prefer. In fact, there are several ways to define clients (and the users they impersonate), both for development and production. In particular, you may decorate a function like this:

```
from octostar.client import impersonating_launching_user, dev_mode
@impersonating_launching_user()
@dev_mode(os.getenv('OS_DEV_MODE'))
def my_function(client):
    ...
```

This will dynamically swap the client in use (development or production) based on the `OS_DEV_MODE` flag, rather than having to manually change the client at multiple points in code. The development client always uses the environment variables defined above, including the authorization JWT (which will eventually expire), while production clients will refresh the authorization based on the user role.

When this flag is enabled, the behavior of `octostar.utils.jobs.apps.get_app_secret` also changes: it will look for a `.env` file in your app and, if it contains the key for the secret, it will use the value contained in that file. This is useful to separate sensitive credentials during local development from the rest of the app, assuming the `.env` file is set to be

ignored by git.

You can check the full details about clients and users here: Authentication & Impersonation (Authentication & Impersonation)

Browser Interactions

To be able to also execute browser-related functionality (e.g. from `streamlit_octostar_research`), enable **Developer Mode** in the Octostar platform from the app of your choice (you can create an **Empty App** if you don't have one), and pass it the local URL of your running app server. This will allow to test the full functionality of the app, including context and browser interactions. Note that back-end interactions using Developer Mode still need to happen via a developer client.

Alternatively, you can create a **Remote Page** in your Octostar deployment and connect it to a locally running server (e.g. `streamlit`). Ensure that under the **Remote Page Editor** your page has permissions to execute the functions you are running. With Remote Pages, browser functionality will work, but you will not be able to test manifest-related functionality.

A Complete (and Minimal) Startup Script Example

```
import os
from dotenv import load_dotenv
from octostar.client import impersonating_launching_user, dev_mode
from octostar.utils.ontology import query_ontology

def runcommand(cmd):
    try:
        os.system(cmd)
    except Exception as e:
        print(e)

def inject_env():
    os.environ['OS_DEV_MODE'] = 'true'
    # Load environmental variables from a .env file. This is the recommended approach
    load_dotenv()
    # Alternatively, specify them manually
    os.environ["OS_API_ENDPOINT"] = 'your-endpoint'
    os.environ["OS_JWT"] = \
        "your-jwt"
    os.environ["OS_USER"] = "your_username"
    os.environ["OS_ONTOLOGY"] = "os_ontology_v1"
    print("Env injected.")

inject_env()

@impersonating_launching_user()
@dev_mode(os.getenv('OS_DEV_MODE'))
def start_app(client):
    ### YOUR CODE HERE
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")
    ### OR RUN SOME ALREADY DEFINED APP (streamlit, flask, etc.). You can even run main.sh from here
    runcommand("python main.py")

if __name__ == '__main__':
    while True:
        answer = input("Reinstall libs? ")
        match answer:
            case "y":
                runcommand("pip uninstall -y octostar-python-client")
                runcommand("pip uninstall -y streamlit-octostar-research")
                runcommand("pip uninstall -y streamlit-octostar-utils")
                runcommand("pip cache purge")
                download_endpoint = os.environ["OS_API_ENDPOINT"]
                runcommand(f"pip install {download_endpoint}/api/octostar/meta/octostar-python-client.tar.gz")
                runcommand("pip install streamlit-octostar-research")
                runcommand("pip install streamlit-octostar-utils")
                break
            case _:
                break
    start_app()
```

Authentication & Impersonation

The `octostar-python-client` library mandates to define an explicit client to call functions from the Octostar API. Clients allow for security and permissions to be respected whenever applications wish to perform CRUD operations against the platform, as well as any other type of request. App developers should take great care in ensuring they impersonate the user with the correct level of access.

Which users can an app impersonate?

Clients have an authorization (a JWT) with which they make requests against the Octostar API. These clients represent users, which are a high-level abstraction describing a user of the platform. Apps can impersonate the following users:

- The Launching User, who is the one who **DEPLOYED** the app;
- The Running User, who is the one **OPENING** and using the app via its user interface or transforms;
- The Developer User, who is the user **TESTING** the app, only accessible during local development

YOU ARE WARNED You should see clear warnings in your logs/console whenever back-end functions have been called with a developer client or without an explicit client invocation. In both cases, the library will attempt to fallback to a JWT contained in the app files or in an environment variable. This is highly unsafe if performed in a production setting.

There are three ways to define a user:

- At a function-level, using `impersonating_XXX_user()` decorator
- At a context-level, using `as_XXX_user()` context manager
- Manually, using `User(make_client())`

See the sections below for some examples of these.

Impersonating the Launching User

By default, when deploying an application, `octostar-python-client` will create a default client which is set to the user **DEPLOYING** the app (usually with some admin role).

You can therefore run the functions of the library as they are, however this will cause warnings, as it is much preferrable to explicitly instantiate it:

```
from octostar.client import as_launching_user, impersonating_launching_user

with as_launching_user() as client:
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

@impersonating_launching_user()
def my_func(client):
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

my_func()
```

LAUNCHING USER = ADMIN ACCESS Be careful! The launching user is the user who ***DEPLOYS*** the app, not the user who is currently interacting with it! Make sure any calls done by this user cannot contain information with privileged access or, if it does, that the user interacting with the app cannot see it.

The credentials of the user who launched the app are in the file `/etc/secrets/OS_JWT`, which changes over time.

Impersonating the Running User (streamlit)

To impersonate the user running the app, there is a corresponding set of functions in `streamlit-octostar-utils`

```
from streamlit_octostar_utils.octostar.client import as_running_user, impersonating_running_user

with as_running_user() as client:
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

@impersonating_running_user()
def my_func(client):
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

my_func()
```

The credentials of the running user are updated from the browser and stored in streamlit's session state, and therefore require browser interactions to be working.

Impersonating the Running User (flask, fastapi)

Since these servers are stateless, the JWT of the running user must be supplied manually and a corresponding client be instantiated within the endpoint:

```
from octostar.client import make_client
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def my_func(jwt: str):
    client = make_client(jwt)
    return client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")
```

Alternatively, the launching user can be used (with care!).

Impersonating the Developer User

The developer user can be impersonated in a similar manner as the launching user, however the necessary parameters must be supplied, either as environment variables or as parameters:

```
from octostar.client import as_developer_user, impersonating_developer_user

import os
from dotenv import load_dotenv

# Option #1: Load from an .env file used exclusively in local development
load_dotenv()
# Option #2: Explicitly set the environment variables
os.environ['OS_DEV_MODE'] = 'true'
os.environ["OS_API_ENDPOINT"] = 'your-endpoint'
os.environ["OS_JWT"] = "your-jwt"
os.environ["OS_USER"] = "your-username"
os.environ["OS_ONTOLOGY"] = "your-ontology"

with as_developer_user() as client:
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

@impersonating_developer_user()
def my_func(client):
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

@impersonating_running_user()
@dev_mode(os.getenv('OS_DEV_MODE'))
def my_func2(client):
    client.execute(query_ontology.sync, "SELECT count() FROM dtimbr.person")

my_func()
my_func2()
```

This user will print a warning whenever it is called.

THE @dev_mode DECORATOR Using the dev_mode decorator allows an app to switch between using a production client (defined via an @impersonating_XXX_user decorator) and the developer client based on the OS_DEV_MODE environment variable. This is handy as it does not require modifying the app code to test locally.

Additional Notes

It is recommended to always double-check that the JWT is properly refreshed, unless expiration of the JWT is desirable (e.g. for time-bound operations).

DO NOT PASS CLIENTS AROUND IN STREAMLIT Streamlit shares imported modules across threads (= user sessions), which means that importing a client from a module in a streamlit page, e.g. a utility python file from your app, will cause a singular client to be used for **ALL** users interacting with that page. In other words, **clients are not thread-safe!**

❌ DON'T

```
## importing a running-user client from another module
from my_utils import my_client
import streamlit as st
from octostar.utils.ontology import query_ontology

st.write(my_client.execute(query_ontology.sync, "SELECT * FROM dtimbr.person"))
```

✅ DO

```
## importing the function creating the client, not the client itself
from streamlit_octostar_utils.octostar.client import as_running_user
import streamlit as st
from octostar.utils.ontology import query_ontology

with as_running_user() as client:
    st.write(client.execute(query_ontology.sync, "SELECT * FROM dtimbr.person"))
```

Writing Data Drivers

As an entity-centric platform, Octostar fully supports integrating data from external providers, on-demand, into your local workspaces. You may, for example, search the internet for additional information about a person, an IP address, an email, or use a proprietary service that is available inside your organization and which Octostar is able to connect to.

To facilitate data integration into the ontology Octostar uses, and therefore make the external data fully compatible with dashboards, apps, templates and the filesystem, we provide a utility library for raw API response conversions.

The API Parser Module

The API Parser module is available in `streamlit-octostar-utils` with no additional dependencies. It contains an engine to parse dict-like data (e.g. JSON) into a set of entities and relationships, according to rulesets which are to be defined by the user and which are read top-to-bottom, in a similar fashion to **Declarative Programming** (that is, pairs of rules and consequences).

The output of the engine parsing is a list of entities and relationships that Octostar can understand in agreement with the business ontology in use. The parser has a rich and customizable set of `Rules` and `Matches` which allows to:

- Match fuzzy keys into entities and entity properties
- Define nested entities with associated relationships if a pattern is matched in the data
- Unroll nested lists and dictionaries
- Combine multiple fields into the same entity property (e.g. merge first and last name into a `name` property via string concatenation)
- Consolidate similar entities into a singular one

Minimal Usage Example

```
from api_parser.entities_parser import EntitiesParser, EntitiesParserRuleset
from api_parser.parameters import ConsolidationParameters
import yaml
import json
import os

### Provide some functions to indicate the rulesets to use, which are the entries in the JSON and the likely entity
type
from vetric_functions import guess_raw_vetric_ruleset as guess_ruleset_fn, \
                             guess_vetric_entries as guess_entries_fn, \
                             guess_raw_vetric_entity_type as guess_entity_fn

with open("my_raw_data.json", "r") as file:
    response = json.load(file)
entities = list()
relationships = list()
entries = guess_entries_fn(endpoint, response)
for entry in entries:
    ruleset_files, ruleset_parameters = guess_ruleset_fn(endpoint, entry)
    guessed_entity = guess_entity_fn(endpoint, entry)
    parser = EntitiesParser([EntitiesParserRuleset(os.path.join("rulesets", ruleset_files[i]), ruleset_parameters[i])
for i in range(len(ruleset_files))])
    new_entities, new_relationships = parser.apply_rules(entry, endpoint, guessed_entity)
    entities.extend(new_entities)
    relationships.extend(new_relationships)

print("ENTITIES (" + str(len(consolidated_entities)) + ")")
for entity in consolidated_entities:
    print(entity.__dict__)
    print()
print("RELATIONSHIPS")
for rel in consolidated_relationships:
    print(rel.__dict__)
    print()
```

The example assumes the developer has defined:

- A function to return the list of entries to parse from a JSON (because e.g. in the JSON file the raw entities are contained in a list keyed `results`)
- A function to return the ruleset (and params) to apply per entry
- A function to optionally guess the entity type from the entry
- The rulesets themselves

Minimal Ruleset Example

An example of a ruleset file is as follows:

```
from api_parser.matches import EntityMatch, DiscardMatch
from api_parser.rules import BoolRule, RegexRule

def _is_instagram_account(field, entry):
    conditions = list()
    entry = entry.value
    conditions.append('class' in entry and 'instagram' in entry['class'].lower())
    conditions.append('guessed_entity' in entry and entry['guessed_entity'] == 'instagram_account')
    return any(conditions)

CONCEPTS = {
    BoolRule(_is_instagram_account): EntityMatch(
        concept_name="instagram_account",
        inherits_rules_from_concepts=["account"],
        is_root=True,
        keep=False,
        push_context=True,
        parse_rules={
            RegexRule("photo_id"): DiscardMatch()
        }
    )
}
```

The file defines that an entry should be processed by this ruleset if `_is_instagram_account()` returns `True`, and if so it should create an entity of type `instagram_account` with fields obtained by parsing the entry as follows:

- discarding a field named `photo_id` if there is any
- inheriting the rules from another ruleset (which defines rules for the more generic concept of `account`) and applying those to the entry

Writing Streamlit Apps

Writing streamlit apps is as simple as writing any other streamlit app and following its best practices, except for a couple of differences. For basic streamlit knowledge we link here to its documentation:

Streamlit Docs

In addition, the Octostar developer should be aware of:

- Initialization logic
- Browser Interactions
- Parallelism & User Session Management
- As a Back-end Service

Initialization Logic

It is good practice to have some initialization logic in a streamlit application. By convention, apps should follow this skeleton structure:

```
import streamlit as st

def initialize():
    st.session_state = {
        'state': dict(),
        'input': {
            'user': dict(),
            'workspaces': None
        },
        'initialized': True
    }

def loop():
    st.write("Hello World!")

if not st.session_state.get('initialized', False):
    initialize()
if st.session_state.get('initialized', False):
    loop()
```

Browser Interactions

Functions involving the front-end (from `streamlit-octostar-research`) cannot be put in `initialize()` and must be put in `loop()` instead (with if-guarded statements), as they return data asynchronously (initially they just return `None`) and cause a rerun of streamlit's code when their data changes, exactly as if a user had interacted with a widget. ❌ **DON'T**

```
from streamlit_octostar_research.desktop import get_open_workspace_ids

def initialize():
    st.session_state = {
        'state': dict(),
        'input': {
            'user': dict(),
            'workspaces': get_open_workspace_ids() # Will be set to None
        },
        'initialized': True
    }

def loop():
  st.write(st.session_state['input']['workspaces'])
```

✅ **DO**

```
from streamlit_octostar_research.desktop import get_open_workspace_ids

def initialize():
    st.session_state = {
        'state': dict(),
        'input': {
            'user': dict(),
            'workspaces': None
        },
        'initialized': True
    }

def loop():
  if not st.session_state['input']['workspaces']:
    ## Will be set to None initially but eventually rerun with a valid value
    st.session_state['input']['workspaces'] = get_open_workspace_ids()
  if not st.session_state['input']['workspaces']:
    st.stop() # prevent further code execution until we get a valid result
  st.write(st.session_state['input']['workspaces'])
```

Parallelism & User Session Management

Streamlit does not natively handle parallel computations and using **asyncio** is generally awkward and error-prone, therefore strongly discouraged. Indeed, whenever a streamlit rerun is invoked, if the event loop is currently running at that time it will crash and be left in an inconsistent state.

Using **threads** is a much more robust mechanism. However, since they are detached from the main thread, streamlit will not generally be aware of them. Furthermore, we want to use a shared pool of threads among user sessions to avoid starving the app of resources. To resolve these issues, there are two module in `streamlit-octostar-utils`:

- `threading.async_task_manager`: This module defines an `AsyncTaskManager` which can be cached globally in an app via `st.cache_resource()`. Tasks can be sent to the pool as a combination of an ID and a callable using method `submit()`, and a (intermediate or final) result can be polled via `get_result()` . Tasks can also be (co-operatively) cancelled via `cancel()`
- `threading.session_callback_manager`: This module defines a singleton object which runs in a separate threads and listens for the following events:
  - user session start
  - user session heartbeat (every 2 seconds)
  - user session end

This can be used in conjunction with the `AsyncTaskManager` to cancel pending tasks when a user session is ended. It can also be used standalone, e.g. to shut down some service if no user is connected to the streamlit server.

As a Back-End Service

Streamlit is a stateful server, allowing heavy computations in the same codeplace as the user interface. However, streamlit does not natively offer a solution to expose reusable functions as APIs to other services. Instead, what is recommended is to use a python REST API server, such as Flask or FastAPI (Writing FastAPI/Flask apps (Writing FastAPI/Flask apps)), which are typically used in conjunction with a JS front-end. If there is a requirement to develop an API together with a streamlit interface, the recommended approach is to deploy two servers (e.g. FastAPI + streamlit) in the `main.sh` file of the app, coordinated by **nginx** as a reverse proxy to route incoming requests to the correct server. For example:

```
## install nginx
apt-get update
apt-get install -y nginx
## run streamlit server on port 8501
streamlit run --server.port 8501 --server.address 127.0.0.1 --server.enableCORS true streamlit_app.py &
## run flask server on port 8502
python fastapi_app.py &
## run nginx to respond to requests on 8080 (default port for apps)
nginx -g 'daemon off;' -c $(pwd)/nginx.conf
```

And in an `nginx.conf` file:

```
events {}

http {
    server {
        listen 8080;

        location /docs {
            proxy_pass http://127.0.0.1:8502/docs;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
        }

        location /openapi.json {
            proxy_pass http://127.0.0.1:8502/openapi.json;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
        }

        location /api {
            proxy_pass http://127.0.0.1:8502/api;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
            client_max_body_size 128M;
        }

        location / {
            proxy_pass http://127.0.0.1:8501;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_set_header Host $host;
        }
    }
}
```

Streamlit Styling Rules

Include the following file `.streamlit/config.toml` in every Streamlit app:

```
[theme]
primaryColor="#0C66E4"
backgroundColor="#FFFFFF"
secondaryBackgroundColor="#F7F8F9"
textColor="#091E42"
font="sans serif"

[browser]
gatherUsageStats=false
```

This will ensure the correct font and colors are used, aligned with those in use by Octostar.

Furthermore, include the following Streamlit function to be executed as **the first Streamlit command, ONCE PER PAGE**:

```
import streamlit as st
st.set_page_config(layout="wide")
```

The command can be customized as needed.

Finally, this other utility function (usually at the beginning of the streamlit `loop()` function) should be called:

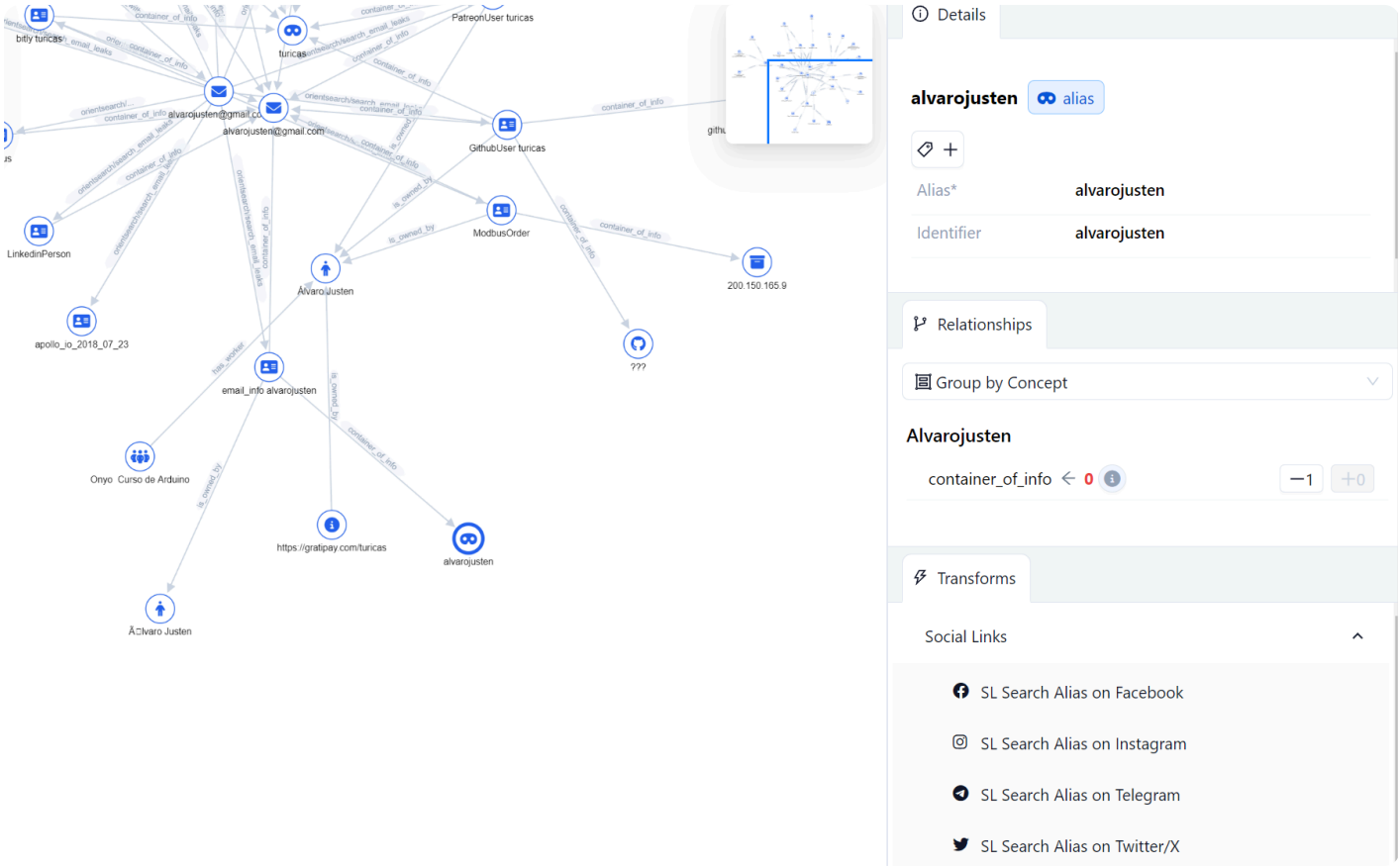
```
hide_streamlit_header()
```

Which will hide the default header used by streamlit, replacing it with white space (this gives the app a cleaner appearance)

Writing FastAPI/Flask apps

Writing REST APIs inside apps can be extremely useful in Octostar. They can be used:

- Together with the manifest (usually as transforms of type `http` The Manifest (The Manifest)), to expose these functions as services in Octostar's context menus or dedicated transform menus
- To use in conjunction with a front-end (e.g. ReactLive, or even Streamlit itself),
- To be called by other apps
- To be called by the automated data processing pipeline



An example of transforms in the linkchart on a node of type `alias`

The structure, inputs and expected outputs of the API will depend on the context in which the API is expected to be used, as in the cases described above.

BROWSER INTERACTIONS If you are developing a back-end API, browser functionalities (e.g. from `streamlit-octostar-research`) will not be available. Instead, this data should be passed from the browser itself, e.g. via the manifest, or as inputs to the endpoint(s).

Endpoint Conventions

The App Viewer and other components in Octostar assume that a (back-end) app exposes its user interface/main page under their localhost at port 8080. By convention, especially if an app has both back-end and front-end components:

- The back-end exposes endpoint `/docs` for the OpenAPI documentation, endpoint `/openapi.json` for the JSON format of the same documentation, and /`/api` for the REST API endpoints themselves. In particular, by convention the endpoints for the NiFi processing pipeline are expected at `/api/nifi`
- The front-end exposes endpoint `/` for the main application page and any other endpoints if it is a multi-page app, except for those reserved above

You can clearly see the application of these rules from an example nginx.conf file.

Utilities

Under library `streamlit-octostar-utils` there are some utility modules for the creation of APIs under the `api_crafter` package.

For FastAPI-based solutions, the module can be imported as such:

```
from streamlit_octostar_utils.api_crafter import fastapi
```

Some examples of utilities include a `CommonModels` for some base pydantic models useful for FastAPI requests and responses, a `DefaultErrorRoute` to respond in a standardized way when an exception is raised, a `RequestCancelledMiddleware` to raise a `CancellationError` if the client disconnects during a request instead of proceeding, and finally a `Route` class which allows developer to separate endpoint definitions from the router they should be associated with, allowing more modular code. More details are here Utilities for FastAPI (Utilities for FastAPI).

These utilities currently do not have corresponding versions for other back-ends such as Flask or Django. Future versions of Octostar may include better support for these (and other) back-ends.

Utilities for FastAPI

The Route Class

A server built with FastAPI typically defines its functionality by:

1. **Defining an `app` instance**: This may include a custom lifecycle.
2. **Defining functions as HTTP endpoints**: These are attached to a router using the `@router.method` decorator.
3. **Including routers in other routers**: Eventually, these are included in the app.
4. **Starting the app**: Use `uvicorn` (or equivalents) to start the app within `__main__`.

`streamlit-octostar-utils.api_crafter.fastapi` enhances points 2 and 3 by:

- **Abstract `Route` Class**: Developers can inherit from this class to define one or more endpoints. This separation keeps the main file clean.
- **Extended Decorator**: The `@Route.route` decorator extends the `@router.method` decorator. It takes a `Route` instance (typically `self`), a FastAPI router `router`, and the same arguments as the FastAPI decorator. This allows dynamic binding of functions as FastAPI routes, directing routes inside the same `Route` to different FastAPI routers. Binding occurs for all functions decorated with `@Route.route` inside the `define_routes()` method when `include_routes()` is called on the `Route` instance, following a syntax similar to FastAPI's `router.include_router()`.
- **Additional `@Route.include()` Decorator**: This sets the underlying function as an attribute in the class instance, allowing instance functions defined inside another function to be exposed as class attributes.

Structure of an Endpoint

An endpoint is structured as a class of type `Route` with several methods:

- **`define_routes()`**: Native to the Route class, this method defines one or more FastAPI routes (usually variants on the same computation) using the `@Route.route` decorator. Routes defined here bind to associated routers when `include_routes()` is called.
- **`__init__()`**: This method defines the base attributes for the `Route`. The new `@Route.route` decorator allows endpoints to refer to `self` and any attributes defined in this constructor.

Implementation

With this class, you can:

1. Instantiate an app and one or more routers.
2. Instantiate a `Route`.
3. Call `include_routes()` on the route instance to bind its endpoints to the router(s).
4. Call `include_router()` on the app to include the routers into the whole app.

Writing JavaScript Apps

In this section, developers find information regarding how to write a JavaScript application and its integration with the Octostar Platform.

Find all the relevant information regarding APIs in APIs Docs and Typscript API.

Project Scaffolding

Recommended Directory Structure

The template ships with a minimal, working scaffold. Directories marked `[add]` are not included but represent the recommended layout as your app grows.

```
your-app/
├── manifest.yaml           # App registration with Octostar
├── Dockerfile              # Based on Octostar base image
├── build.sh                # Build script (frontend + backend)
├── main.sh                 # Runtime entrypoint
├── requirements.txt        # Backend Python dependencies
├── .env.example            # Environment variable template
│
├── frontend/
│   ├── package.json        # @octostar/platform-react + your deps
│   ├── tsconfig.json
│   ├── vite.config.ts
│   └── src/
│       ├── App.tsx          # OctostarContextProvider + StateProvider root
│       ├── main.tsx         # React DOM entry point
│       ├── state/
│       │   └── StateProvider.tsx   # JWT initialization, app-wide state hooks
│       ├── services/
│       │   └── api.ts       # Authenticated fetch wrapper (token management)
│       ├── utils/
│       │   └── env.ts       # isStandalone() helper
│       ├── hooks/           # [add] Custom hooks
│       ├── features/        # [add] Feature-based components
│       ├── components/      # [add] Shared UI components
│       └── types/           # [add] TypeScript definitions
│
└── backend/
    └── main.py              # FastAPI app, serves frontend static files
```

Key Dependencies

Frontend (`package.json`):

```
{
  "dependencies": {
    "@octostar/platform-react": "^0.5.45",
    "@octostar/platform-types": "^0.5.37",
    "@octostar/platform-api": "^0.5.32",
    "react": "^18.3.1",
    "antd": "^5.x"
  }
}
```

The template also includes a few example UI libraries (`recharts`, `mermaid`, etc.) that you can remove when building your own app. **Backend** (`requirements.txt`):

```
fastapi>=0.115.0
uvicorn[standard]>=0.32.0
python-dotenv>=1.0.0
```

The Octostar Python SDK (`octostar.client`, `octostar.utils.*`) is **not** listed in requirements.txt — it comes pre-installed in the base Docker image.

The App Manifest

Every Octostar app needs a `manifest.yaml` that registers it with the platform. It controls how the app appears in the UI, what resources it receives, which entity types it accepts, and how it's launched.

```yaml
# ── Identity ──────────────────────────────────────────────────────
name: My Octostar App               # Display name shown in the platform
description: Short description of what the app does
icon: fa-circle-question            # FontAwesome icon name (fa-*)
alias: my-octostar-app              # Unique URL-safe slug — no spaces
image: octostar/app.my-octostar-app # Docker image (org/app.your-slug)

# ── Resources ─────────────────────────────────────────────────────
storage_limit: 2Gi
memory_limit: 2Gi
cpu_priority: normal                # normal | high

# ── Entity filters (optional) ─────────────────────────────────────
# Restrict which entity types can open this app.
# Omit this section entirely if your app doesn't target specific entity types.
filters:
  is_file:
    type: semantically_bound
    concepts:
      - os_file       # any file
      # - os_image    # images only
      # - os_document # documents only
  is_folder:
    type: semantically_bound
    concepts:
      - os_folder

# ── Transforms ────────────────────────────────────────────────────
# Define how the app is opened. "iframe" embeds it as a panel inside Octostar.
transforms:
  open_iframe:
    description: Opens the app as an iframe panel
    type: iframe
    title: My Octostar App
    icon: fa-circle-question

# ── Services (optional) ───────────────────────────────────────────
# Adds an entry to the right-click context menu for matching entities.
# Omit this section if your app doesn't need a context menu entry.
services:
  open_my_app:
    order: 60
    role: context_menu
    group: general
    label: My Octostar App
    icon: fa-circle-question
    show_progress: false
    description: What the app does in one sentence
    accepts:
      - is_file
      - is_folder
    transforms:
      - open_iframe
```

Key fields

Field	Description
`alias`	Unique identifier used in URLs. Must be URL-safe (hyphens, no spaces).
`image`	Docker Hub image in `org/app.your-slug` format. Must match your CI/CD pipeline.

Field	Description
`filters`	Named entity selectors. Each filter defines an Octostar concept type that can trigger this app. Optional.
`transforms`	Defines launch modes. `type: iframe` embeds the app as a panel. The transform name (e.g. `open_iframe`) is referenced by services.
`services`	Registers entries in the right-click context menu. Optional.
`accepts`	List of filter names (defined in `filters`) that control when a context menu entry is visible.
`transforms` (under a service)	List of transform names to invoke — references the keys defined in the top-level `transforms` block.

The Octostar Platform React SDK

The `@octostar/platform-react` package provides React components and hooks for integrating with the Octostar shell.

OctostarContextProvider

Wraps your entire app. It establishes the connection with the Octostar shell and makes platform APIs available to all child components via context.

```
import { OctostarContextProvider } from '@octostar/platform-react'

export default function App() {
  return (
    <OctostarContextProvider>
      {/* rest of your app */}
    </OctostarContextProvider>
  )
}
```

> The template places `OctostarContextProvider` at the root in App.tsx. It is safe to nest multiple instances — the provider is idempotent.

useOctostarContext

The primary hook for accessing Octostar platform APIs. Must be called inside an `OctostarContextProvider`.

```
import { useOctostarContext } from '@octostar/platform-react'

const { ContextAPI, OntologyAPI, DesktopAPI } = useOctostarContext()
```

The template uses `ContextAPI` to retrieve the session JWT on startup:

```
// from StateProvider.tsx
const { ContextAPI } = useOctostarContext()

ContextAPI.getContext().then((ctx) => {
  if (ctx?.token) api.setToken(ctx.token)
})
```

`OntologyAPI` and `DesktopAPI` are available for querying the knowledge graph and interacting with the Octostar desktop respectively — see the TypeScript API reference for their full signatures.

OsDropzone

A drag-and-drop target that accepts entities from the Octostar sidebar:

```
import { OsDropzone } from '@octostar/platform-react'

<OsDropzone
  accept={{ entities: true }}
  onDropEntities={(entities) => handleDrop(entities)}
  texts={{
    idle: 'Drag files here from the sidebar',
    active: 'Release to drop',
  }}
/>
```

> *Find all the other information regarding Typescript API in Typescript API Documentation.*

Authentication and the JWT Token

Octostar apps authenticate via a JWT token injected by the platform when the app is launched in the iframe. The token must be forwarded to all backend API calls via an `Authorization: Bearer` header.

> **Note:** Tokens expire hourly. In local dev, refresh your token from the Octostar UI under **Profile → Developer Tokens → Out Token** when calls start returning 401

Frontend: Getting the Token

The token is initialized once at startup in StateProvider.tsx, before any child component renders. It is sourced from two places depending on the environment:

```
1. Octostar context  — production (running inside the platform iframe)
2. VITE_OS_JWT env   — local dev / standalone mode
// state/StateProvider.tsx
const { ContextAPI } = useOctostarContext()

useEffect(() => {
  // Standalone / local dev: read token from VITE_OS_JWT
  if (!withOctostar || isStandalone() || !ContextAPI) {
    api.setTokenFromEnv()
    return
  }
  // Production: get token from the Octostar shell
  ContextAPI.getContext()
    .then((ctx) => { if (ctx?.token) api.setToken(ctx.token) })
    .catch(() => api.setTokenFromEnv())  // fallback on error
}, [withOctostar, ContextAPI])
```

The API service stores the token in memory and attaches it to every request. `sessionStorage` and `localStorage` are not available inside Octostar iframes, so the token must live in-process for the lifetime of the page.

```typescript
// services/api.ts
class ApiService {
  private token: string | null = null

  setToken(token: string) { this.token = token }
  setTokenFromEnv() {
    const t = import.meta.env.VITE_OS_JWT
    if (t) this.token = t
  }

  async request<T>(endpoint: string, options: RequestInit = {}): Promise<T> {
    const headers = {
      'Content-Type': 'application/json',
      ...(this.token && { Authorization: `Bearer ${this.token}` }),
      ...options.headers,
    }
    const res = await fetch(`./api${endpoint}`, { ...options, headers })
    if (!res.ok) throw new Error((await res.json()).detail ?? `HTTP ${res.status}`)
    return res.json()
  }
}

export const api = new ApiService()
```

Backend: Extracting the Token

The minimal template (`backend/main.py`) does not include JWT extraction. For a full app, the recommended pattern is a FastAPI dependency:

```python
# dependencies.py (add this to your backend)
from fastapi import Header, HTTPException, Depends
from typing import Optional

def get_token(
    authorization: Optional[str] = Header(None, alias="Authorization")
) -> str:
    """Extract JWT from the Authorization header."""
    if authorization and authorization.startswith("Bearer "):
        return authorization[7:].strip()
    # Dev/standalone fallback
    import os
    token = os.getenv("OS_JWT")
    if token:
        return token
    raise HTTPException(status_code=401, detail="Missing Authorization header")


def get_client(token: str = Depends(get_token)):
    """Create an authenticated OctostarClient for this request."""
    from octostar.client import OctostarClient
    return OctostarClient(
        token=token,
        api_endpoint=os.getenv("OS_API_ENDPOINT"),
        ontology=os.getenv("OS_ONTOLOGY"),
    )
```

`OctostarClient` is not in `requirements.txt` — it comes pre-installed in the base Docker image.

Environment Variables

The project uses two separate `.env` files — one for the backend, one for the Vite dev server: `.env` (backend, project root):

```
OS_API_ENDPOINT=https://your-octostar-instance.com
OS_ONTOLOGY=os_ontology_v1
OS_USER=your_username_here
OS_JWT=your_jwt_token_here   # dev/standalone only — platform injects it in production
OS_STANDALONE=true           # set to true for local dev (skips platform auth)
```

`frontend/.env` (frontend, Vite dev server only):

```
VITE_OS_JWT=your_jwt_token_here    # same token as OS_JWT above
VITE_OS_STANDALONE=true            # skips Octostar cont
```

Working with Entities

Entities are the fundamental data objects in Octostar — files, folders, people, organizations, or any custom concept type.

Entity Structure

Every entity has a common set of fields:

Field	Description
`os_entity_uid`	Primary unique identifier for the entity
`entity_id`	Alternative identifier, used by drag-and-drop APIs
`os_concept`	Concept type name (e.g. `os_file`, `os_folder`, `os_image`)
`entity_type`	Synonym for `os_concept` on some API responses
`os_item_name`	Human-readable display name

> The identifier field name varies by context: `getContext()` returns `record.os_entity_uid`, while `OsDropzone` entities expose `entity_id`. Always check which API you're reading from.

Loading Entities from the Octostar Context

When the app launches from a context menu action, the entity the user right-clicked on is available via `ContextAPI.getContext()`:

```
const { ContextAPI } = useOctostarContext()

useEffect(() => {
  ContextAPI?.getContext().then((ctx) => {
    const entityUid = (ctx as any)?.record?.os_entity_uid
    // use entityUid to call your backend...
  })
}, [ContextAPI])
```

Receiving Entities via Drag-and-Drop

Entities dragged in from the sidebar are delivered with `entity_id`:

```tsx
import { OsDropzone } from '@octostar/platform-react'
import type { Entity } from '@octostar/platform-types'

<OsDropzone
  accept={{ entities: true }}
  onDropEntities={(entities: Entity[]) => {
    const uid = entities[0]?.entity_id
  }}
  texts={{ idle: 'Drop an entity from the sidebar', active: 'Release to load' }}
/>
```

Looking Up Entities by UID (Backend)

On the backend, query a specific entity by UID using the ontology SQL client. The generic workspace item table covers all entity types:

```python
async def lookup_entity_by_uid(client: OctostarClient, uid: str):
    escaped = uid.replace("'", "''")
    sql = f"""
        SELECT * FROM etimbr.os_workspace_item
        WHERE os_entity_uid = '{escaped}'
        LIMIT 1
    """
    results = await client.query_ontology(sql)
    return results[0] if results else None
```

Entity-specific concept tables (e.g. `timbr.person`, `timbr.os_file`) can also be queried directly when you need concept-specific fields.

Distinguishing Entity Types

Use `os_concept` (or `entity_type` as a fallback) to check what kind of entity you're working with:

```python
concept = record.get("os_concept") or record.get("entity_type")
is_folder    = concept == "os_folder"
is_workspace = concept == "os_workspace"
is_file      = not (is_folder or is_workspace)
```

Expanding Folders Recursively

When a user provides a folder entity, retrieve all files inside it. Track visited UIDs to prevent cycles:

```
def is_folder(item: dict) -> bool:
    concept = item.get("os_concept") or item.get("entity_type")
    return concept == "os_folder"


async def expand_folder(client, workspace_id, folder_uid, visited=None):
    if visited is None:
        visited = set()
    if folder_uid in visited:
        return []
    visited.add(folder_uid)

    items = await client.get_folder_files(workspace_id, folder_uid)
    files = []
    for item in items:
        if is_folder(item):
            files.extend(
                await expand_folder(client, workspace_id, item["os_entity_uid"], visited)
            )
        else:
            files.append(item)
    return files
```

Querying the Ontology

The Octostar ontology is a semantic data layer backed by Timbr. You query it using SQL through platform-specific table namespaces.

Table Namespaces

Namespace	Contents	Example use
`timbr`	Concept-specific tables — one table per concept type	`timbr.os_file`, `timbr.os_tag`, `timbr.person`
`etimbr`	Extended workspace views — filesystem objects and workspace items across all types	`etimbr.os_workspace_item`, `etimbr.os_wsfs_object`
`dtimbr`	Denormalized/display views — pre-joined data suited for listing and browsing	`dtimbr.os_workspace`

From the Frontend (`OntologyAPI`)

```
const { OntologyAPI } = useOctostarContext()

const workspaces = await OntologyAPI.sendQueryT<WorkspaceRow>(
  "SELECT os_entity_uid, entity_label FROM dtimbr.os_workspace"
)
```

From the Backend (Octostar SDK)

The Octostar Python client is not in `requirements.txt` — it is downloaded at container startup by `main.sh` directly from your platform instance, so it always matches the running version.

```
import asyncio
from octostar.utils.ontology import query_ontology

result = await asyncio.to_thread(
    client.execute,
    query_ontology.sync,
    "SELECT os_entity_uid, entity_label FROM dtimbr.os_workspace"
)
```

> The SDK is synchronous. Wrap all SDK calls with `asyncio.to_thread` to avoid blocking the FastAPI event loop.

Common Queries

List workspaces:

```
SELECT os_entity_uid, entity_label, os_item_content
FROM dtimbr.os_workspace
```

List files in a workspace:

```
SELECT os_entity_uid, entity_id, entity_label, entity_type,
       os_item_content_type, os_content_size
FROM timbr.os_file
WHERE os_workspace = '{workspace_id}'
```

List all items (files + folders) in a workspace:

```
SELECT os_entity_uid, entity_id, entity_label, entity_type,
       os_item_content_type, os_content_size
FROM etimbr.os_workspace_item
WHERE os_workspace = '{workspace_id}'
```

Get folder contents:

```
SELECT os_entity_uid, entity_id, entity_label, entity_type
FROM etimbr.os_wsfs_object
WHERE os_workspace = '{workspace_id}'
AND (os_parent_folder = '{folder_id}' OR os_parent_folder_uid = '{folder_id}')
```

List tags:

```
SELECT os_entity_uid, entity_label, os_workspace, color
FROM timbr.os_tag
```

List template files (`.aiquestions`) in a workspace:

```
SELECT os_entity_uid, os_parent_folder, os_workspace, entity_label
FROM timbr.os_file
WHERE entity_label LIKE '%.aiquestions'
AND os_workspace = '{workspace_id}'
```

SQL Safety

Always escape user-provided values before interpolating into SQL strings:

```
def escape_sql_string(value: str) -> str:
    return str(value).replace("'", "''") if value
```

Workspace Operations

Workspaces are shared file systems in Octostar. Your app can read and write files and check what the current user is permitted to do.

Permission Levels

Value	Level	Description
`1`	Read	User can view and download files
`2`	Write	User can read, create, and overwrite files
`3`	Admin	Full control, including workspace management

Checking Permissions

```
from octostar.utils.workspace.permissions import get_permissions

permissions = await asyncio.to_thread(
    client.execute,
    get_permissions.sync,
    workspace_ids  # list of workspace UIDs
)
# Returns: {"workspace_uid_1": 2, "workspace_uid_2": 1, ...}
```

On the frontend, filter to writable workspaces before letting the user save:

```
const writableWorkspaces = workspaces.filter(ws => ws.permission >= 2)
```

> `get_permissions` returns a flat dict keyed by workspace UID. If you are displaying workspaces fetched from `dtimbr.os_workspace`, merge the permission values into those records before sending them to the frontend.

Writing Files to a Workspace

```
from io import BytesIO
from octostar.utils.workspace import write_file

content_bytes = json.dumps(data).encode("utf-8")

result = await asyncio.to_thread(
    client.execute,
    write_file.sync,
    workspace_id,
    "path/to/file.json",  # destination path within the workspace
    "application/json",   # MIME type
    BytesIO(content_bytes),
)
```

Reading Files from a Workspace

```
from octostar.utils.workspace import read_file

raw: bytes = await asyncio.to_thread(
    client.execute,
    read_file.sync,
    workspace_id,
    "path/to/file.json",
)
data = json.loads(raw)
```

Getting Open Workspaces (Frontend)

`DesktopAPI` exposes what the user currently has open in Octostar:

```
const { DesktopAPI } = useOctostarContext()

const openWorkspaces  = await DesktopAPI.getOpenWorkspaces()
const activeWorkspace = await DesktopAPI.getActiveWorkspace()
```

These are useful for defaulting a workspace picker to the context the user is already working in, rather than asking them to select one.

Standalone and Development Modes

Your app should work in three modes:

Mode	When	Token Source	Octostar APIs
Production	Running inside Octostar iframe	`ContextAPI.getContext()`	Fully available
Development	Local dev with Octostar backend	`OS_JWT` env var + `OS_DEV_MODE=true`	Available via SDK
Standalone	Local dev without Octostar	`OS_STANDALONE=true`	Unavailable — use fallbacks

Iframe Detection

Disable React StrictMode inside iframes to prevent PostRobot connection issues:

```
// main.tsx
const isInIframe = (() => {
  try { return window.self !== window.top; }
  catch { return true; }
})();

const AppWrapper = isInIframe
  ? App
  : () => <StrictMode><App /></StrictMode>;
```

Conditional Octostar Features

Always check for API availability before using Octostar-specific features:

```
const { DesktopAPI } = useOctostarContext();

// Octostar environment
if (DesktopAPI) {
  const workspaces = await DesktopAPI.getOpenWorkspaces();
}
// Standalone fallback
else {
  const workspaces = await api.getWorkspaces();
}
```

On the backend, wrap SDK imports in try/except:

```
try:
    import octostar.client
    OCTOSTAR_SDK_AVAILABLE = True
except ImportError:
    OCTOSTAR_SDK_AVAILABLE = False
```

Deployment

Dockerfile

```
FROM octostar/streamlit-apps-no-gpu:latest

WORKDIR /app
COPY . /app

# Build frontend + install backend dependencies
RUN bash build.sh

ENTRYPOINT ["/bin/bash", "/app/main.sh"]
EXPOSE $CODE_SERVER_PORT
```

The base image `octostar/streamlit-apps-no-gpu:latest` includes:

- Python 3.x with the Octostar SDK pre-installed
- System libraries for common data processing tasks

Build Script

Your `build.sh` should:

1. Install Node.js (if not in base image)
2. Install Python dependencies (`pip install` or `uv`)
3. Build the React frontend (`npm run build`)
4. Output static files to a known directory (e.g., `frontend/dist/`)

Runtime

`main.sh` typically starts FastAPI with uvicorn, serving both the API and the frontend static files:

```
uvicorn app.main:app --host 0.0.0.0 --port ${CODE_SERVER_PORT:-8080}
```

In your FastAPI app, mount the frontend build output:

```
from fastapi.staticfiles import StaticFiles

# API routes first
app.include_router(api_router, prefix="/api/v1")

# Then serve frontend as fallback (SPA routing)
app.mount("/", StaticFiles(directory="frontend/dist", html=True)
```

Writing Core Octostar Apps

Core Octostar apps are no different than other apps, as they use the same framework, conventions and libraries detailed in the rest of this Section. However, to ensure these apps are maintanable, portable and reliable, especially given that different versions of these apps could be deployed under different Octostar environments, there are a few recommendations to be followed.

App Repository Structure

The basic structure for a core app should be as follows:

```
.
├── main.sh # entry point for the app
├── localdev.py # entry point for testing the app locally
├── .env # sensitive credentials for testing the app locally. MUST NOT BE TRACKED
├── manifest.yaml # octostar app manifest
├── config.yaml # exposed configs for the app (optional)
├── Dockerfile # app image definition
├── requirements.txt # python requirements file (optional)
├── build.sh # all requirements and downloads should be executed here
├── README.md # provide clear docs to users
├── .gitignore # ignore common python by-products and the .env file
├── .github
    ├── workflows
        ├── tag.yaml # create and push a new image for the app
```

The `main.sh` file is the entry point for the app, which should define the running process(es) (e.g. streamlit server, fastapi server, etc.). App developers should test the app locally by running the `localdev.py` file (or equivalent in other languages), which should read environment variables from the `.env` file and set the `OS_DEV_MODE` flag to true before launching either `main.sh` or a specific app process/main file. The `manifest.yaml` file should contain at least title, icon and description of the app. It must specify a unique alias among all apps and it must define the app itself as its image, e.g.

```
alias: my-app
image: octostar/app.my-app:version
```

This is because core Octostar apps should be versioned and built as Docker images (More on this in the **Creation of a Docker Image** section). The `build.sh` file should refer to all external dependencies and install them, including python libraries, javascript packages and downloadable contents. It will be executed as part of the `Dockerfile` definition, ensuring the application comes packaged with all its dependencies and therefore will be reliable and can be installed

without internet access. This file should also install the `requirements.txt`, if present. It is good practice for python dependencies to be frozen (`pip freeze`), and the same applies to `apt-get` calls (which should refer to OS-specific versions according to the base image defined in the `Dockerfile`).

PRE-DOWNLOADING PYTHON MODELS Often, python libraries related to data science will download models or contents when they are imported or specific functions are called for the first time. In order to force this download so that it is part of the built image, it is perfectly okay for the `build.sh` file to run a python snippet, like so:

```
python -c "f
from spacy_download import load_spacy
load_spacy('en_core_web_sm') # will download the en_core_web_sm if not found
"
```

A `tag.yaml` file should be also provided for a GitHub action which will deploy a new Docker image for the app (same name as the one defined in the `manifest.yaml`) to the Octostar Docker Registry. This action should trigger whenever a commit is tagged (where a tag = a version of the app). Finally, the `README.md` should provide a user-friendly guide to the app in terms of its features, basic and advanced usage, installation and developer guide (More on this in the **Readme** section). In particular, it should refer to the config.ini file and how its settings can change the behavior of the app.

APP SECRETS It is crucial that sensitive secrets (such as API keys) are not committed to the repository, ever. The basic setup for developers should be to always rely on the `.env` file to store app secrets locally, or the **Secrets Manager** in production. Therefore the `.env` file should **NEVER** be tracked, and secrets should **NEVER** be written directly into source files.

Creation of a Docker Image

The Dockerfile should have the following structure:

```
FROM {your_image}
WORKDIR /app
RUN apt-get update && apt-get install -y python3-venv
RUN python3 -m venv /app/venv
COPY . /app
RUN chmod +x build.sh
RUN source /app/venv/bin/activate && ./build.sh
ENTRYPOINT ["/bin/bash", "-c", "source /app/venv/bin/activate && /app/main.sh"]
EXPOSE $CODE_SERVER_PORT
```

Which will copy all app files tracked by git, execute the `build.sh` in a virtual environment and set `main.sh` as the app entrypoint. Note this image should be launched only via a GitHub action and NOT locally, as it could include files that are not tracked by git (like the `.env` file). Typically (but this is not mandatory) the **FROM** image is one of:

- `octostar/streamlit-apps-gpu` (python 3.10 w/ CUDA support)
- `octostar/streamlit-apps-no-gpu` (python 3.11)

BE CAREFUL OF THE PYTHON VERSION The python version of the base image is binding, particularly in the case of GPU-enabled Docker images. The application should be tested locally on the same python version as the image version to ensure compatibility of all libraries.

Versioning and GitHub Integration

GitHub integration allows to automate the creation of a Docker image every time a new version of the app is ready. To do so, we need to:

- Make sure the `tag.yaml` workflow file is included in the repository and points to the correct app name. You can see the template for this file here:tag.yaml (tag.yaml)
- tag to the remote a new version of the app, like so:

```
cd /path/to/the/app
git tag 1.2.3-dev4 # or any other version
git push origin --tags
```

The action will trigger upon pushing a new tag to the remote, building and uploading a new docker image for the app, which is then referenced in the `manifest.yaml` file. Typically, the two main branches for a core app should be `dev` (default branch) and `main`, where the former releases stable versions (e.g. `v0.1.2`) and the latter development/experimental versions (e.g. `v0.1.1-dev.1`).

The Apps Workspace

Once an app has been pushed as a docker image to the Octostar registry, in order to install and launch the app we only need to provide three files:

```
.
├── manifest.yaml # octostar app manifest
├── config.yaml # exposed configs for the app (optional)
├── README.md # provide clear docs to users
```

In particular, it is good practice that the app version in the manifest is frozen to a specific one, e.g.:

```
image: octostar/app.my-app:1.2.3
```

Core apps are packaged together (as a workspace with folders each with the above files) into the `ws.apps` repository. Upon creating a new app (or updating an existing core app), the workspace itself should be updated with one or more commits and tagged with a new version, exactly as one does for an app. This will create a new docker image for the workspace itself, such that newly deployed Octostar environments will fetch the new version of this workspace and, therefore, of the core apps.

Readme

The `README.md` is a crucial documentation tool for Octostar applications, as it allows fellow developers and analysts to understand how your application works, what features it provides and how it is meant to be deployed and used. It is recommended that all readmes follow the same general structure (which of course can be expanded upon):

```
# App Title
Two-three sentences slogan for the app.

## Features
Dotted list of the main app features.

## Installation (app version)
Concise install guide for production, including required app secrets. Conclude with how to open the app (e.g. if it has context menu interactions).

## User Guide
User-friendly description of how the app should be opened, on what entities/files it works and what are the main operations/flows that can be performed with it.

## FAQ
Question/Answer pairs to troubleshoot common issues or limitations of the application.

## Developer Guide
Description of the core structure of the app from a developer's point of view; what to do and what not to do when working with the app.

## Known Limitations
List of known bugs, edge cases or missing features under which the app will not work as expected.
```

For an example of documentation, see here: Example `README.md` (Example README.md)

tag.yaml

```yaml
name: Build and Push Docker Image

on:
  push:
    tags:
      - '*'

env:
  IMAGE_NAME: octostar/app.my-app
  CACHE_NAME: octostar/cache:app.my-app

jobs:

  docker-build:
    name: docker-build
    runs-on: ${{ matrix.runner_platform.runner }}
    strategy:
      matrix:
        runner_platform: [ { runner: linux_150gb_runner, platform: linux/amd64, architecture: amd64 }, { runner:
github_linux_arm64, platform: linux/arm64, architecture: arm64 } ]

    steps:
      - name: "Checkout ${{ github.ref }} ( ${{ github.sha }} )"
        uses: actions/checkout@v4
        with:
          persist-credentials: false
          fetch-depth: 0

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Build and push Docker Image
        id: docker_build
        uses: docker/build-push-action@v6
        with:
          context: .
          platforms: ${{ matrix.runner_platform.platform }}
          cache-from: type=registry,ref=${{ env.CACHE_NAME }}-${{ matrix.runner_platform.architecture }}
          cache-to: type=registry,ref=${{ env.CACHE_NAME }}-${{ matrix.runner_platform.architecture }}
          push: true
          outputs: type=image,name=${{ env.IMAGE_NAME }},push-by-digest=true,name-canonical=true,push=true


      - name: Export digest
        run: |
          mkdir -p /tmp/digests/${{ matrix.runner_platform.architecture }}/
          digest="${{ steps.docker_build.outputs.digest }}"
          touch "/tmp/digests/${{ matrix.runner_platform.architecture }}/${digest#sha256:}"

      - name: Upload digest
        uses: actions/upload-artifact@v4
        with:
          name: digests-${{ matrix.runner_platform.architecture }}
          path: /tmp/digests/*
          if-no-files-found: error
```

```yaml
        retention-days: 1

merge_docker_images:
  needs: [ docker-build ]
  runs-on: ubuntu-latest
  steps:
    - name: Download digests
      uses: actions/download-artifact@v4
      with:
        path: /tmp/digests
        pattern: digests-*
        merge-multiple: true

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}

    - name: Create manifest list for ${{ env.IMAGE_NAME }} and push
      working-directory: /tmp/digests
      run: |
        digest_amd64=$(ls amd64)
        digest_arm64=$(ls arm64)
        docker buildx imagetools create \
        --tag ${{ env.IMAGE_NAME }}:latest \
        --tag ${{ env.IMAGE_NAME }}:${{ github.ref_name }} \
        ${{ env.IMAGE_NAME }}@sha256:${digest_amd64} \
        ${{ env.IMAGE_NAME }}@sha256:${digest_arm64}

    - name: Display Docker Image Tag
      run: "echo Docker Image Tag: ${{ env.IMAGE_NAME }}:${{ github.ref_name }}"
```

Example README.md

```
# 🦜 Audio Transcription
This application transcribes audio files into text using open source AI tools. It supports noisy inputs, timestamps and translation all-in-one!

## Features
* Process any amount of audio files using Whisper + argostranslate
* Transcribe from a wide array of languages, with auto-detection of the source language
* Translate into many languages
* Smart filter by keyword only the relevant parts of your audio
* Timestamps for each sentence!
* Exposes a processor for the Octostar data processing pipeline to transcribe or translate audio files at `/api/nifi`

## Installation (v0.1.0-dev.2)
The application requires an **App folder** (*Your Workspace→Create→App→Empty App Folder*) with the following minimal file structure:
```

. ├── manifest.yaml ├── readme.MD (this file)

```
Please ensure that the manifest points to the right application and declares the correct application version, like so:
```

image: octostar/app.entity-extract-and-graph:0.1.0-dev.1

The app can be deployed from the App Editor (*App Folder→Open With→App Editor→Deploy App)*.

To open the application, open it as-is or via the context menu option *Open With→* **🎤Transcribe Audio** on an audio file or a folder.

User Guide
The app is straightforward to use. Once opened, it will show the list of files among the input ones which are supported for transcription. The user can select the checkbox **Include Audio Translation** to perform both transcription and translation at the same time.

Once the Transcribe! button is pressed, the AI model will begin transcribing the audio files. The time necessary for this process varies by hardware, typically around 1s per second of audio unless a GPU is available, which expedites transcription greatly.

Once the transcription process is complete, the application will show the transcription (and optionally translation) for the first audio file. The audio itself can be replayed from the player at the top of the screen, and below it are a series of transcriptions paired with timestamps. The transcriptions are fully editable to account for manual editing of mistakes, and clicking on a timestamp will play the audio from the beginning of that timestamp. At the bottom there is also a list of entities extracted with Named Entity Recognition, and which can be saved alongside the transcription.

On the left sidebar, there is a filtering tool. The user can insert any amount of keywords and the audio file(s) will be filtered accordingly on a per-sentence basis. Finally, the user can save either transcription or translation back into the original audio file or as a separate file, with our without timestamps.

FAQ
I cannot open the app! "App not Available", "404 not found", "500 service not available"
Please ensure the application is correctly deployed. If you are not an admin, please contact one.

My files are not loaded into the application?
Files may take a while to be recognized by the application, before which the application will still ask the user for files to be uploaded.

The transcription quality is poor/Detected language is wrong/There are missing sentences!
This application uses an AI model to detect the source language of the audio, and is therefore subject to variance in quality. Supported languages are (according to https://platform.openai.com/docs/guides/speech-to-text/supported-languages):

> Afrikaans, Arabic, Armenian, Azerbaijani, Belarusian, Bosnian,
> Bulgarian, Catalan, Chinese, Croatian, Czech, Danish, Dutch, English,
> Estonian, Finnish, French, Galician, German, Greek, Hebrew, Hindi,
> Hungarian, Icelandic, Indonesian, Italian, Japanese, Kannada, Kazakh,
> Korean, Latvian, Lithuanian, Macedonian, Malay, Marathi, Maori,
> Nepali, Norwegian, Persian, Polish, Portuguese, Romanian, Russian,
> Serbian, Slovak, Slovenian, Spanish, Swahili, Swedish, Tagalog, Tamil,
> Thai, Turkish, Ukrainian, Urdu, Vietnamese, and Welsh.

Any language outside of these will not be recognized and transcription may outright fail. Furthermore, supported languages may still be recognized incorrectly, in which case it is likely most of the audio will not be transcribed, or will be transcribed incorrectly.

Developer Guide
TODO

Known Limitations
* No support for multi-language: the application may outright fail, ignore sentences or produce garbage text when auto-detecting language on an audio in which multiple languages are spoken
* No support outside the supported language list: the application may fail, produce garbage text or detect a different language (typically English)
* No support for multi-speaker: the application will transcribe all speakers as if they were one

Implementer Guide

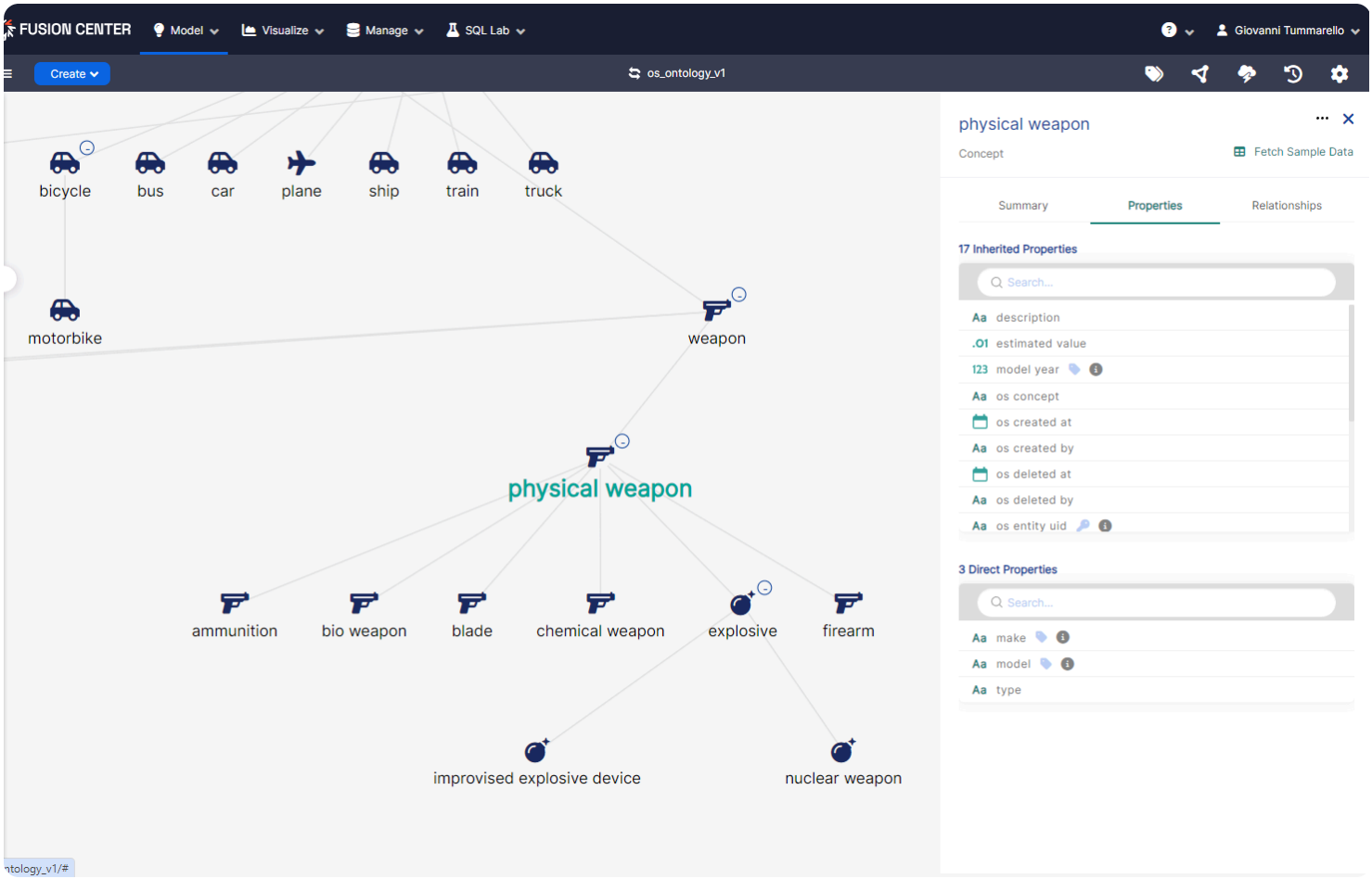
The Implementer Guide documents the configuration and modeling layer of Octostar. It is designed for technical implementers responsible for defining ontologies and managing mappings within the **Octostar Fusion Centre**. Proper **ontology** design and mapping configuration are essential to ensuring that data is structured, interoperable, and analytically meaningful across the platform.

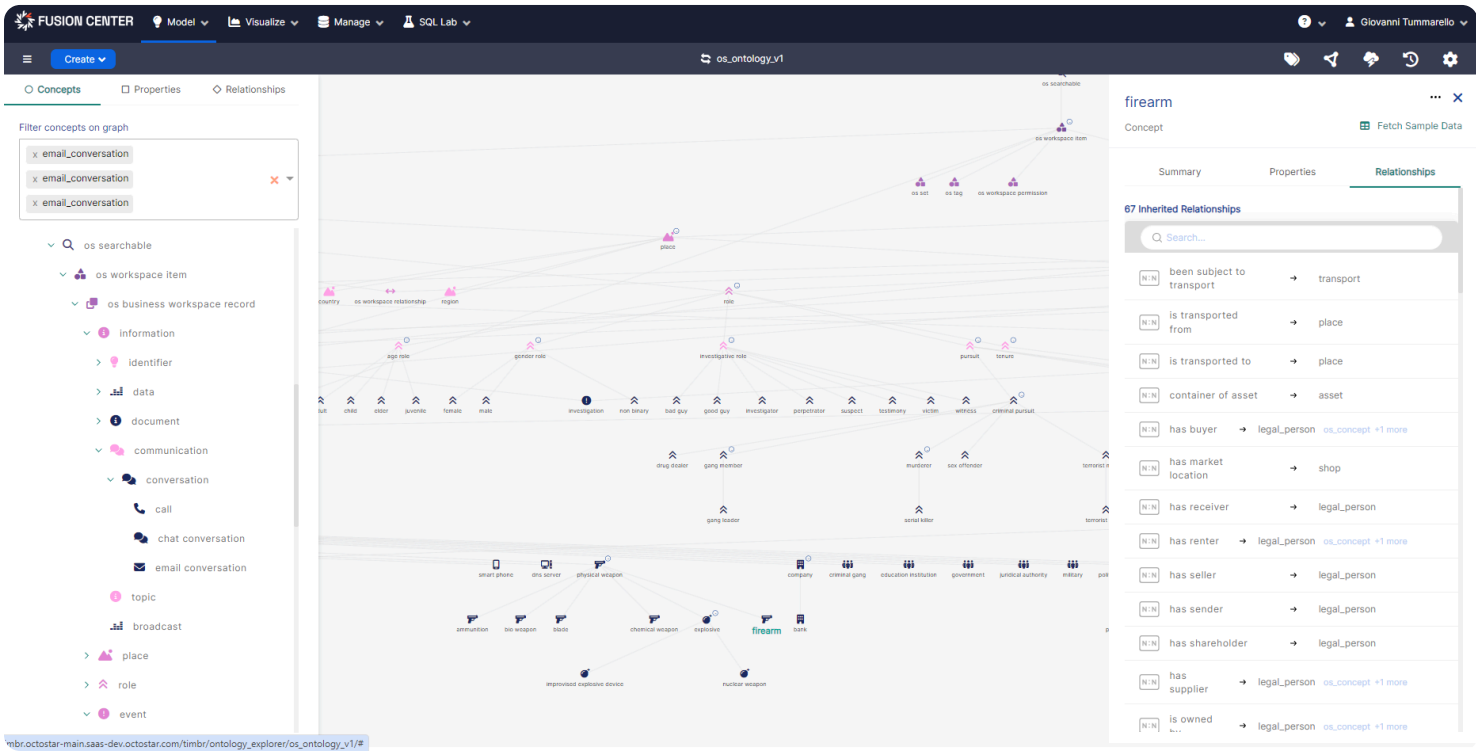
Ontology definition and mappings: the Octostar Fusion Centre

The *Ontology* (also known as *Data Model*) is the key for Octostar to operate. All the *records* (also known as *entities*) loaded or created within Octostar will fit into one (or more) of the ontology *concepts*.

The Octostar Standard Investigative Ontology (OSIO).

Octostar typically comes with the **Octostar Standard Investigative Ontology** (OSIO) - this contains typical concepts useful in general law enforcement, national security, and (to some extent) financial crimes. However, *this ontology can be changed* (by removing or adding new concepts, properties), or a new ontology can be created altogether.





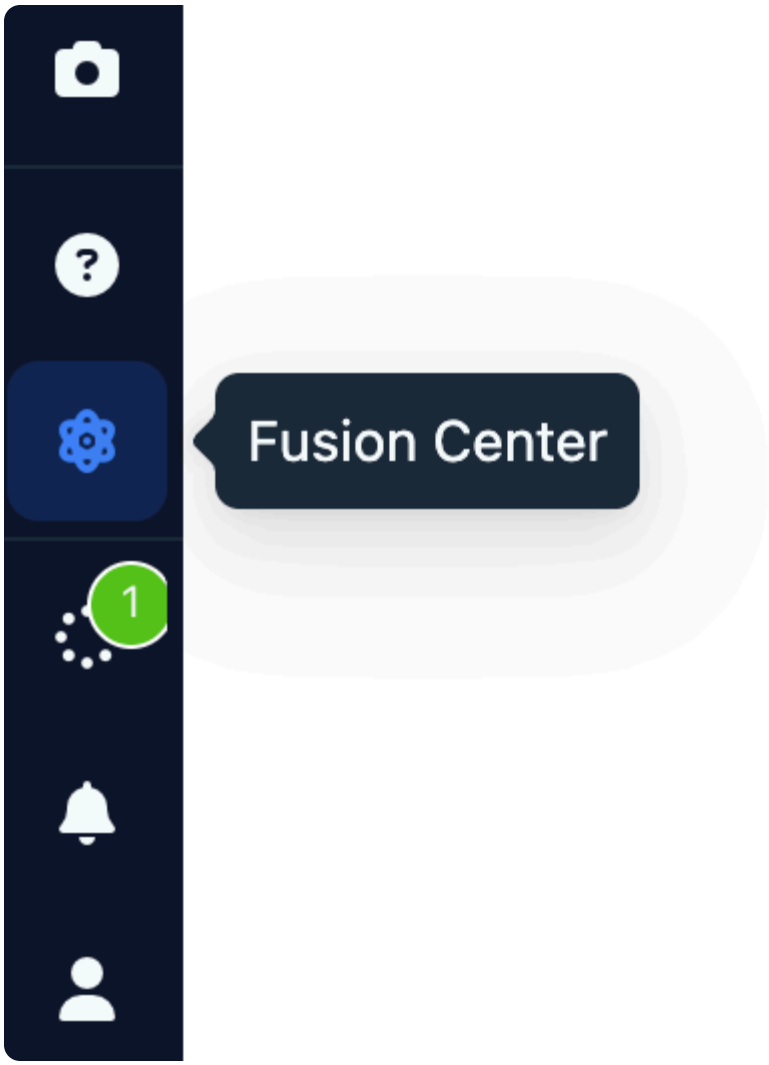
Aside from naming Concepts (e.g. Vehicle, Person), the Ontology defines:

- **properties:** these are attributes of the Concepts (E.g. Date of Birth)
- **relationships:** these are the possible relationships between Concepts (E.g. Person *spouse_of* Person). Relationships don't only have a name but can also have attributes themselves.

The **Octostar Fusion Center** is where:

- The data models (ontologies) are created and modified
- Live data sources are mapped (E.g. a new database table, mapped to the concept *person*) giving access to remote data
- Security is managed, with respect to data sources (not individual workspaces)
- Other administrative functions such as logging, auditing, performance monitoring and others.

The Octostar Fusion center can be accessed from the UI using this icon.



The Octostar Fusion center has a full documentation (see top level of this guide) which is key to learn to:

- Connect Concepts to Database tables (e.g. Oracle, Mysql, Clickhouse) so that the table rows become "entities" and can be visualized and investigated in Octostar

- Handle the security and access control aspects - who can see which data

---

The Octostar Base Ontology

The Octostar Standard Investigative Ontology (OSIO) is composed of 2 parts:

- the **base ontology:** this base ontology is a set of octostar-reserved entities necessary for the platform to work correctly. These entities are prepended with the `os_` prefix and so do their attributes. These should **never be changed** or customized by the end user.
- The other domain concepts (also known as as the *business part*) - these are "mounted" on top of the base ontology (which means these concepts inherit from some of the base classes).

There are a number of reserved fields and special properties in the ontology.

All items in Octostar Standard Investigative Ontology (but in general of any other business ontology) must then inherit from the `os_business_workspace_record` concept, so that it has the fields required from security and normal Octostar Operations.

Note that in general one could have other concepts in the ontology which do not derive from these but in this case they would not be able to be "stored" (appear to be stored) within specific investigations - since the `os_workspace` field is key for this.

Special Concepts, Fields and Relationships in the Standard Octostar Ontology

Special Concepts

- `os_thing:` A generic Octostar entity, with a universally unique ID, a concept type and a description
- `os_workspace_item:` Any item that can belong to a workspace. This also includes entities NOT belonging to a workspace (global entities), where the `os_workspace` field is null
- `os_attachable:` Any item in Octostar that has a file attachment. Files are kept externally w.r.t. records (e.g. S3), and are stored based on a combination of workspace ID and entity ID
- `os_business_workspace_record:` A generic Octostar entity which is of interest for an investigation. Anything inheriting from this entity (and not beginning with `os_`) can be created, edited, and visualized in many different ways in the platform. Also, all such concepts are customizable in the ontology by admins
- `os_message:` An annotation-like objects in Octostar, that is, an entity that augments the information about another entity. These include annotations and comments
- `os_tag:` A tag which can be applied to entities. Tags are local to workspaces, meaning they have a non-null `os_workspace` field. Tags can also have a color, a name, and belong to a group, such that only one tag in the group can be applied to an entity
- **os_workspace_permission**: A record detailing the level of access a user has for a specific workspace. These rules override the general role permissions
- **os_wsfs_object**: A workspace item which is visible in the Octostar workspace filesystem (the left sidebar). Items of this type can be moved, copied, exported and imported into workspaces with a classical filesystem structure
- **os_file**: An os_wsfs_object which behaves much like a file in a typical Operating System
- **os_folder**: An os_wsfs_object which behaves like a folder in a typical Operating System
- **os_workspace**: A special type of os_folder. Workspaces are separate filesystem spaces, each containing files, entities and even apps, dashboards, and templates.
- **os_watch_intent**: An entity associated with another entity which is being watched for updates. Periodically, the presence of an os_watch_intent triggers a job in k8s to perform some customizable operation on the associated entity
- **os_dashboard_visualization**: A template for a dashboard. The presence of this entity in an open workspaces makes that dashboard type visible when inspecting the concept types associated with the dashboard itself

- **os_linkchart**: An entity representing an Octostar linkchart. A linkchart is a graph made of nodes (entities) and links (relationships). Users can interact with and edit these linkchart from the Linkchart Editor.

Special Properties

Here is a brief explanation for some of the most important OS-reserved properties in the ontology :

- `os_entity_uid`: the globally unique identifier for a record.
- `os_concept`: the concept type for the record (same as the `entity_type` in **most** cases)
- `os_icon`: the icon shown whenever this record is seen in the octostar platform. If not specified, the icon shown is based on the concept type
- `os_workspace`: the workspace to which this record belongs. If empty, the entity is considered part of the global data
- `os_parent_folder`: for workspace file records, the direct parent folder of this file
- `os_has_attachment:` for file records, whether the record has an associated S3 attachment (which is the file proper)
- `os_item_content:` for workspace file records, any contents for the file which have been written directly in the record rather than off-loaded to S3. Commonly used for metadata, thumbnails or annotations
- `os_item_content_type`: for workspace file records, the MIME type of the attachment of this file record. For app folders, set to `os_app`. For saved searches files, set to `os_saved_search`

Special Relationships

Here is a brief explanation for some of the most important OS-reserved relationships in the ontology:

- `in_workspace (OTM):` Indicates that an entity belongs to a workspace. Foreign key is the `os_entity_uid` ↔ `os_workspace` fields between the workspace and the entity
- `featured_in (MTM):` Indicates that an entity is shown in a certain `os_file`. Previews for that entity will show that file
- `has_tag (MTM):` Indicates that an entity is associated with a certain `os_tag`. The platform will show tags associated with an entity in most viewers.
- `annotated_with (OTM):` Indicates that an entity has a certain `annotation` associated with it. Annotations are written by apps and the pipeline to augment knowledge about an entity. Foreign keys are the `os_entity_uid` ↔ `source_entity_uid` between the entity and the annotation
- `same_as (MTM):` Indicates that two entities have been resolved to be the same one via some Entity Resolution technique, e.g. Senzing
- `watcher_for (MTM):` Indicates that an entity is being watched by a given `os_watch_intent`. A watcher will periodically perform operations with that entity as an input and, upon changes of the output, will notify the user(s) subscribed to said watcher
- `has_comment (OTM):` Indicates that an entity has a `comment` associated with it. Comments can be added to any entity and are shown in a dedicated tab when looking at them through the Record/File Viewer. Foreign keys are the `os_entity_uid` ↔ `source_entity_uid` between the entity and the comment

The Semantic Graph Platform

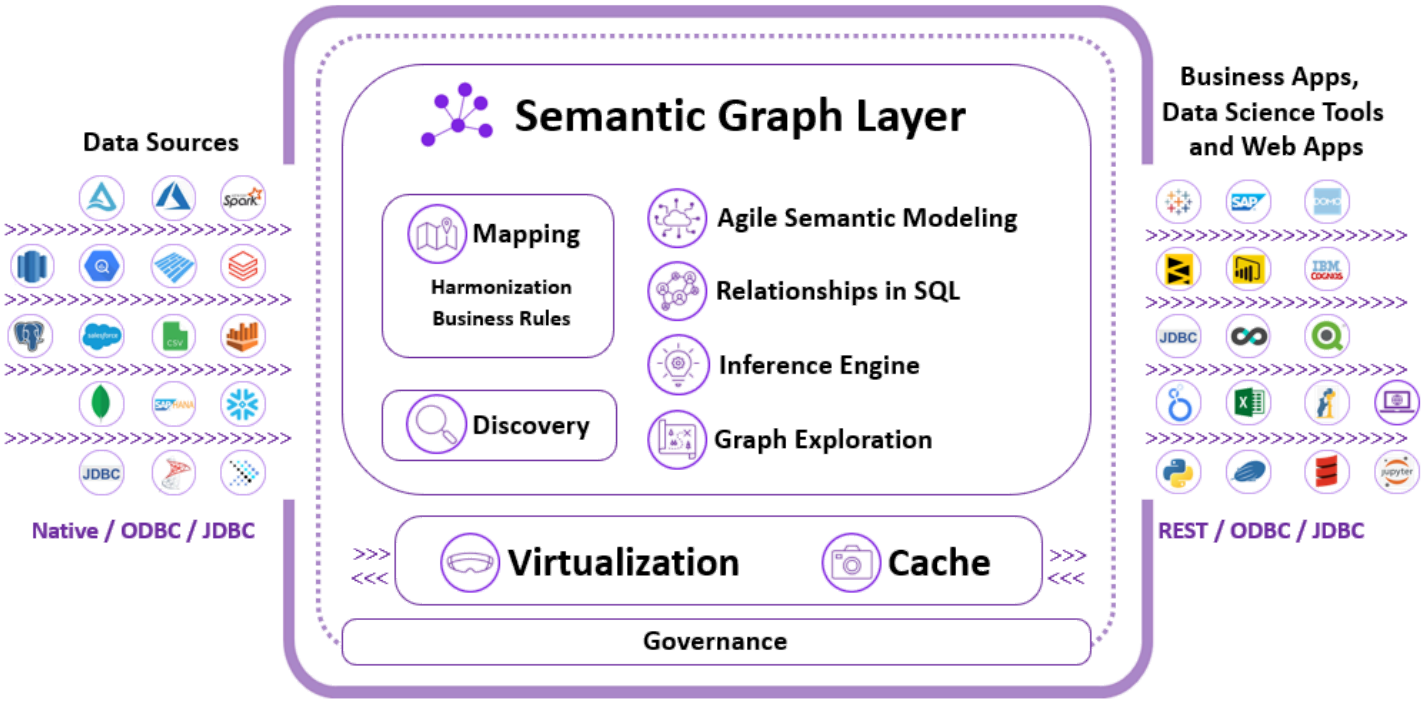
Timbr is an SQL-native knowledge graph that turns your databases into reasoning machines.

Timbr delivers the powers of knowledge graphs to the SQL data ecosystem with an easy-to-use no-code UI or with standard SQL statements.

The platform installs on top of existing databases to enable the creation of virtual semantic models mapped to the underlying data and provides seamless connectivity to the popular BI and data science tools in use by organizations.

Architecture

Timbr is accessible using JDBC, ODBC, REST API:



Platform components

Model, **Visualize**, **Manage**, and **Query** your data as a connected semantic graph

Model	Visualize	Manage	Query
Ontology Explorer	Graph Explorer	Knowledge Graphs	SQL Editor
Data Mapper	Knowledge Lineage	Datasources	Query Search
Ontology Views	Performance Dashboard	Master Data Management	Saved Queries
	Charts	Scheduled Jobs	
	Dashboards	Access Manager	
	CSS Templates	Recent Activity	
	Saved Explorations		

Features

- **Ontology Modeling:** represent an abstract, simplified view of the world with conceptual schemas.
- **Business Rules:** embed business logic and rules into the data model.
- **Virtualization:** no ETL required. Distributed JOINs/UNIONs and Push-down optimizations.
- **Graph Traversals:** graph traversals in standard SQL without the need to explicitly write joins.
- **Inheritance:** is-a relationships provides higher level of abstraction than SQL views.
- **Inference:** infer new knowledge and relationships based on a set of rules on the data.
- **Data Exploration:** an intuitive interface for exploring and visualizing data.
- **NoSQL Capabilities:** allowing a relatively flexible schema declaration and evolution.
- **Integration:** supports most SQL Engines, NoSQL, and data formats.
- **Materialization:** 3-tier cache engine - Datalake, SSD, OLAP In-memory.
- **Master Data Management:** Import and edit master-data live in the Timbr platform.
- **Apache Spark and Databricks Native:** available in DataFrames on Java, Scala, R and Python.

Resources

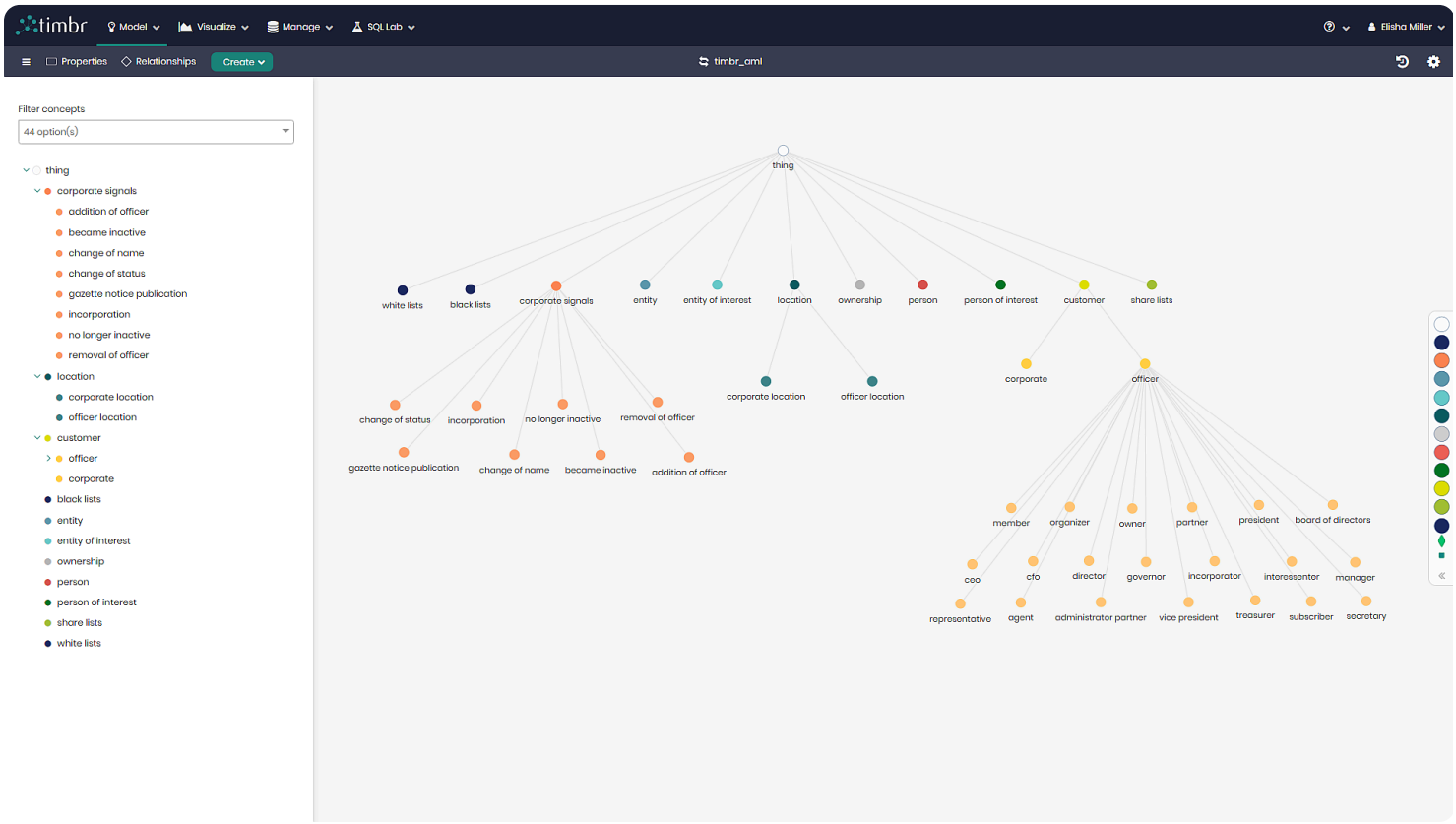
- timbr.ai Website
- License

- About Us
- Follow Us on LinkedIn

Ontology Explorer

The Ontology Explorer page enables users to create, edit, consolidate, and expose data concepts with business terminology. The data model serves as an ontology that allows integrating datasets as business concepts with semantic relationships, hierarchies of classifications, hard-coded business rules, and calculated metrics while consolidating data from multiple sources into a single semantic layer. The Ontology Explorer is used by both editors and viewers and is equipped with granular access controls, a full version control functionality to easily roll back and edit the data model, and search functionality of meta-data and relationships across the model to enable easy and intuitive data discovery and exploration.

Ontology Explorer Page

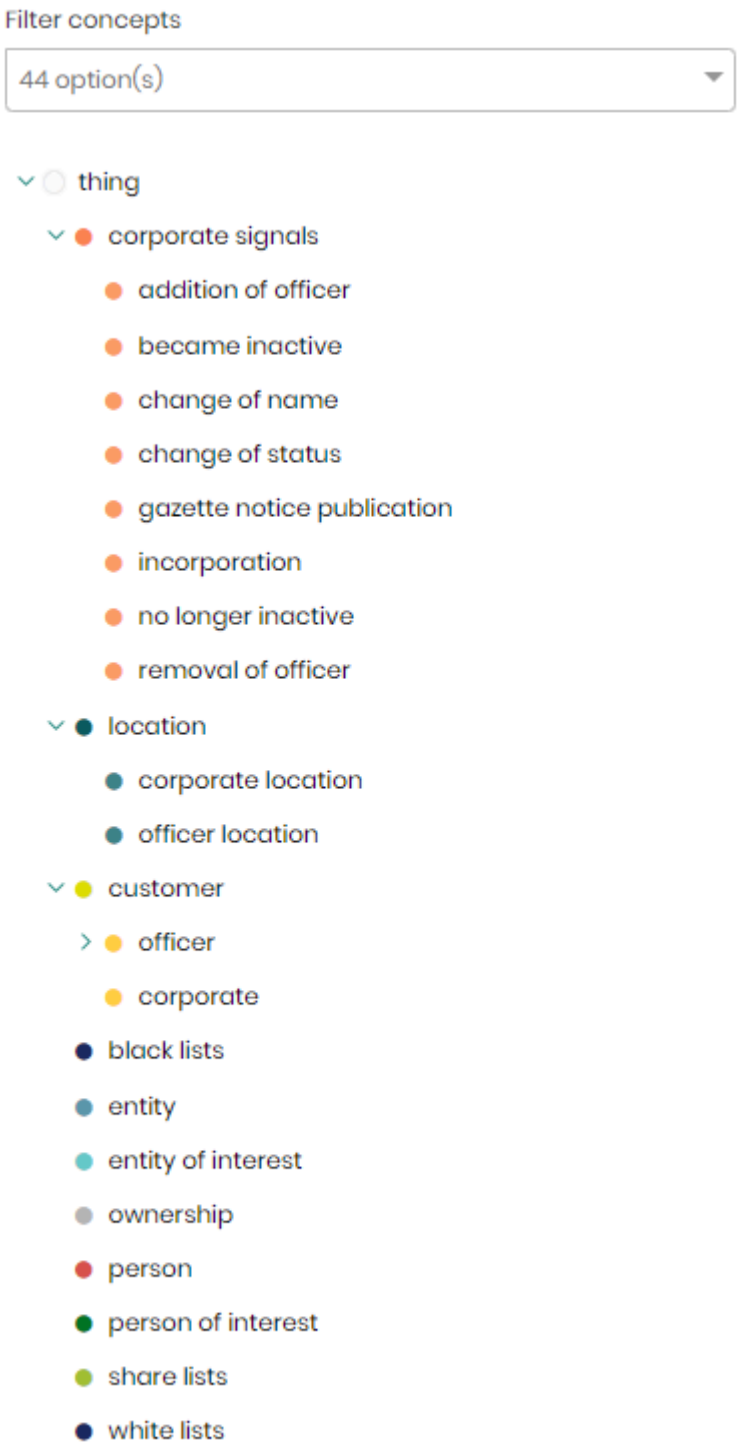


Timbr's *Ontology Explorer* page can be accessed through the **Model tab** by clicking on **Ontology Explorer** and selecting the desired knowledge graph.

The ontology explorer consists of different sections to explore the ontology graph:

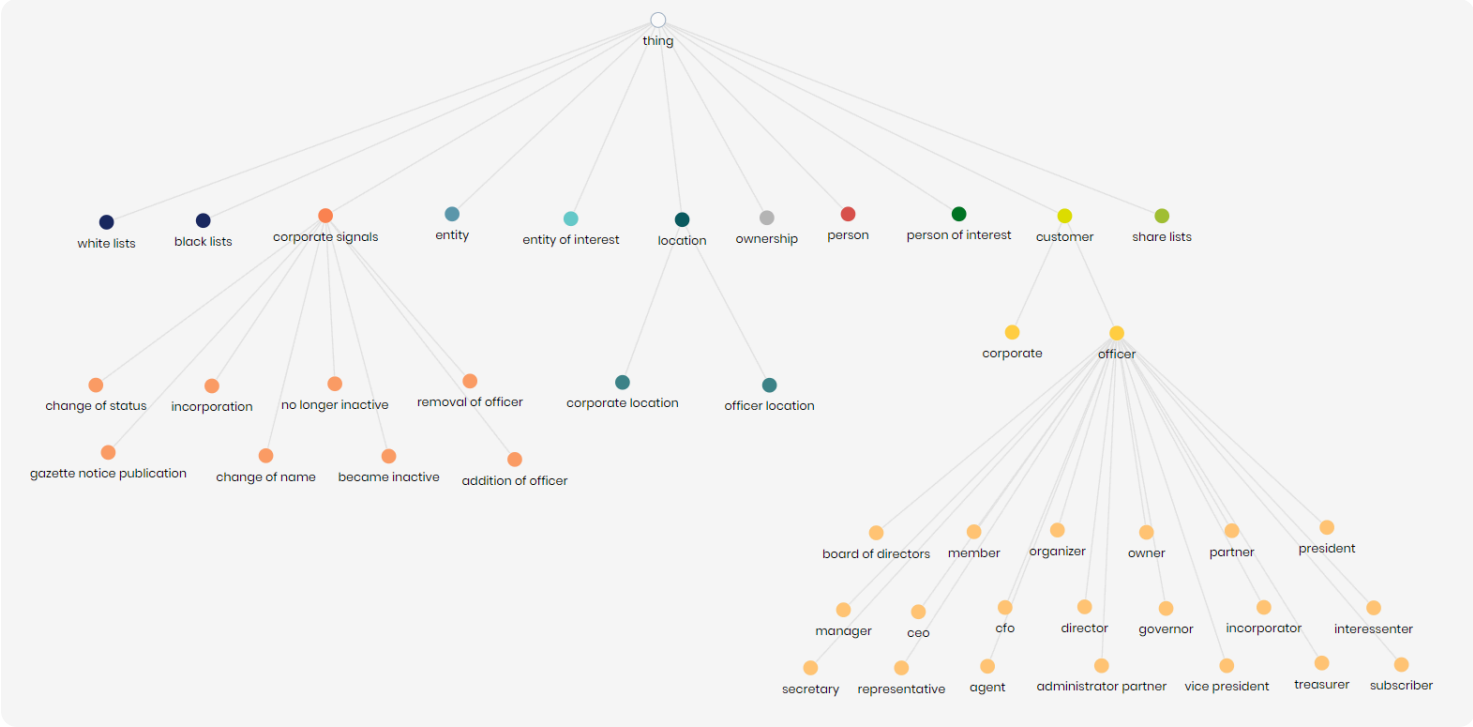
> **Ontology Tree** - A list of concepts and options to filter the graph and get more information about a certain concept.
>
> **Graph Viewer** - An interactive graph representing the ontology.
>
> **Main Graph Control** - A list of control options for the graph. It can be accessed above the Ontology.
>
> **Concept Menu** - A list of actions that can be performed on a single concept. It can be accessed by right-clicking on a concept.
>
> **Concept Details** - A right-side panel that opens up when you click on a concept in the graph. Provides additional information about a concept.

The Concepts Tree



In this section, you can see the hierarchy of the concepts in your ontology and you have the option to filter or get more information about a concept in the graph.

Graph Viewer



An interactive graph representing your ontology. The graph can be filtered to show the properties and references each individual concept has. More information on how to control the graph can be found in the next section below.

You can zoom-in and zoom-out of the graph view by using the mouse wheel.

Main Graph Control

The graph controls above the ontology explorer control the behavior and presentation of the graph. The following options are available:

Horizontal Bars - Opens a small dropdown menu with the options to:

- Save the current exploration.
- Open a saved exploration.
- Enter the current ontologies performance dashboard.

Properties - Opens a menu to indicate which properties to show on the graph, with the following options:

- **Show all** - A toggle to show all the properties in the graph.
- **Filter box** - A filter box for selecting specific properties to be displayed on the graph.

- **Filter current graph** - Filters the concepts on the current graph to only show the concepts and their connected concepts that contain the selected properties.
- **Add to current graph** - Adds the selected properties to the relevant concepts, leaving the other concepts on the graph untouched.
- **New graph** - Renders a new graph showing only the direct concepts that contain the selected properties.

Relationships - Opens a menu to indicate which relationships to show on the graph, with the following options:

- **Show all** - A toggle to show all the relationships in the graph.
- **Filter box** - A filter box for selecting specific relationships to be displayed on the graph.
- **Filter current graph** - Filters the concepts on the current graph to only show the concepts and their connected concepts that contain the selected relationships.
- **Add to current graph** - Adds the selected relationships to the relevant concepts, leaving the other concepts on the graph untouched.
- **New graph** - Renders a new graph showing only the direct concepts that contain the selected relationships.

Create - Opens a small dropdown menu with the options to:

- Create a new concept.
- Create one or more concepts using an existing concepts properties.
- Create a new property to assign to an existing concept.
- Create a new concept data mapping.

Change Knowledge Graph - Opens a dropdown with all the available knowledge graphs to select from and transfer to.

Cached Resources - Opens a window with all the cached resources connected to the current knowledge graph.

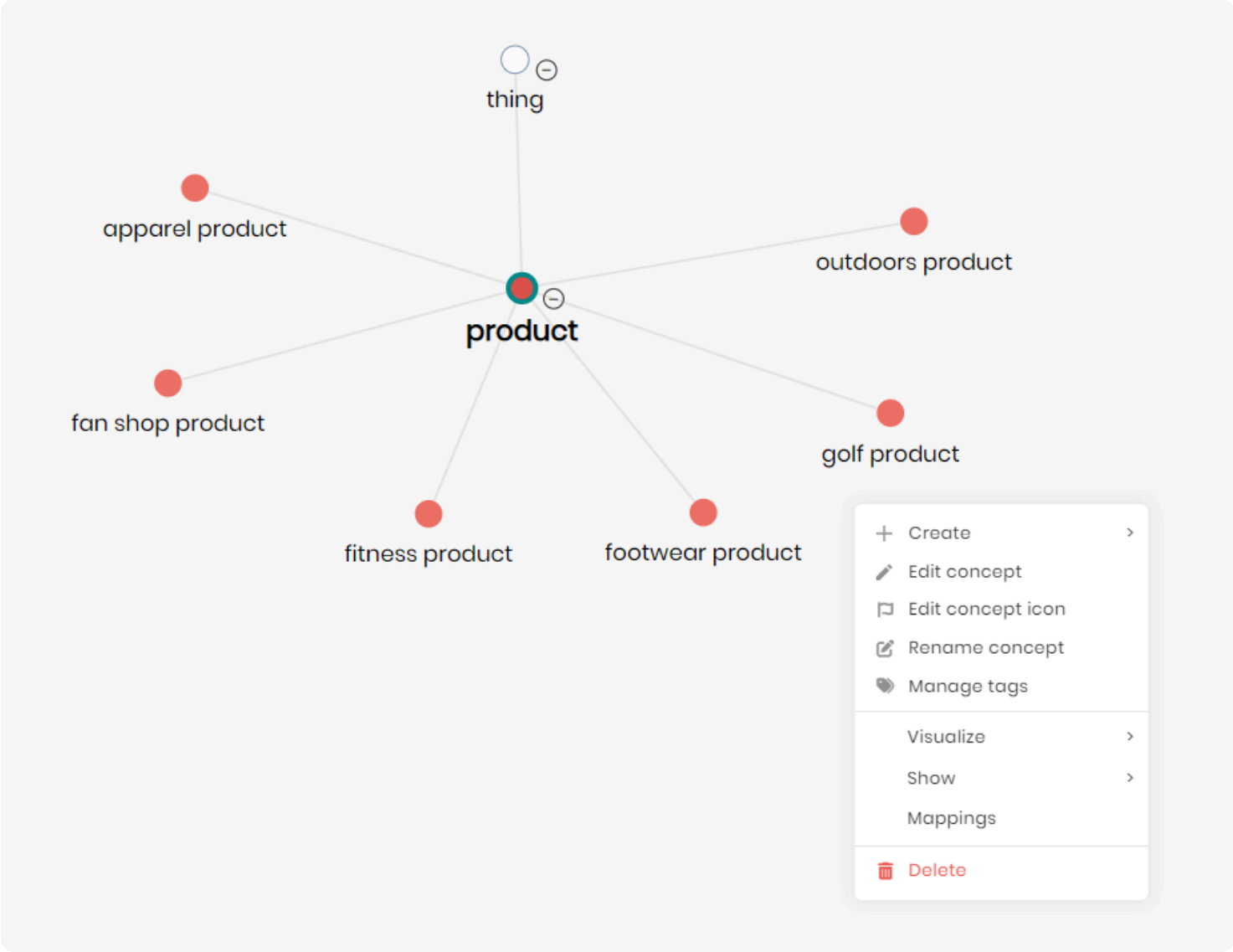
Ontology History - Opens a window containing details on all the actions performed on the current Ontology, with the ability to undo previous actions.

Graph Settings - Opens a settings window with the following options:

- **Node Sizing** - Allows changing the sizing of the concept nodes on the graph.
- **Node coloring** - Allows changing the colors of the concept nodes on the graph based on different methods such as by Inheritance level, Branches and leaves, Mapped and unmapped concepts, etc.
- **Arrange graph** - Makes nodes on the graph automatically return to the default position.
- **Auto arrange toggle** - When the toggle is switched on, concept nodes on the graph will automatically rearrange after each action. When the toggle is switched off, concept nodes on the graph will remain unmoved.
- **Extended view** - When the toggle is switched on, the extended view of the entire ontology will be presented. When the toggle is switched off, only the top first layer of concept nodes will be represented on the graph without their children concepts.

- **Node description** - When the toggle is switched on, concept node descriptions will appear when hovering over the concept. When the toggle is switched off, concept node descriptions will no longer appear when hovering over the concept.
- **Highlight graph** - When the toggle is switched on, concept nodes and their edges will appear highlighted when hovering over the concepts or their edges. When the toggle is switched off, concept nodes and their edges will no longer appear highlighted when hovering over the concepts or their edges.
- **Horizontal layout** - When the toggle is switched on, the graph will be presented in a horizontal layout. When the toggle is switched off, the graph will return to the normal default layout.
- **Legacy layout** - When the toggle is switched on, the graph will be presented using a legacy algorithm layout. When the toggle is switched off, the graph will return to the normal default layout.

Concept Menu



When you right-click on a concept in the graph viewer, a menu will open with actions that correspond to the concept.

Create - When hovering over Create the following options will appear:

- Create a child concept for the selected concept.
- Create one or more concepts using the selected concepts existing properties.

Edit concept - Opens an edit concept window in order to edit all aspects of the selected concept. This includes editing the concept's primary key, entity label, properties, relationship, business logic, etc.

Edit concept icon - Opens a window in order to choose a custom icon to represent the selected concept.

Rename concept - Opens a window in order to rename the selected concept.

Manage tags - Opens a window in order to add or edit the tags of the selected concept.

Visualize - When hovering over Visualize the following options will appear:

- **Visualize as chart** - Opens Timbr's built-in BI module in order to visualize the selected concept's underlying data using charts and graphs.
- **Explore as graph** - Opens Timbr's Graph Explorer module in order to visualize the selected concept's underlying data as nodes on a graph.
- **View lineage** - Opens a window with the selected concept's data lineage, showing its connected data sources, tables, data mappings, views and other concepts in the hierarchy.
- **Fetch Sample Data** - Retrieves a sample of 50 rows of data for the selected concept. The data retrieved does not include any graph traversal and is used just for plain data exploration.

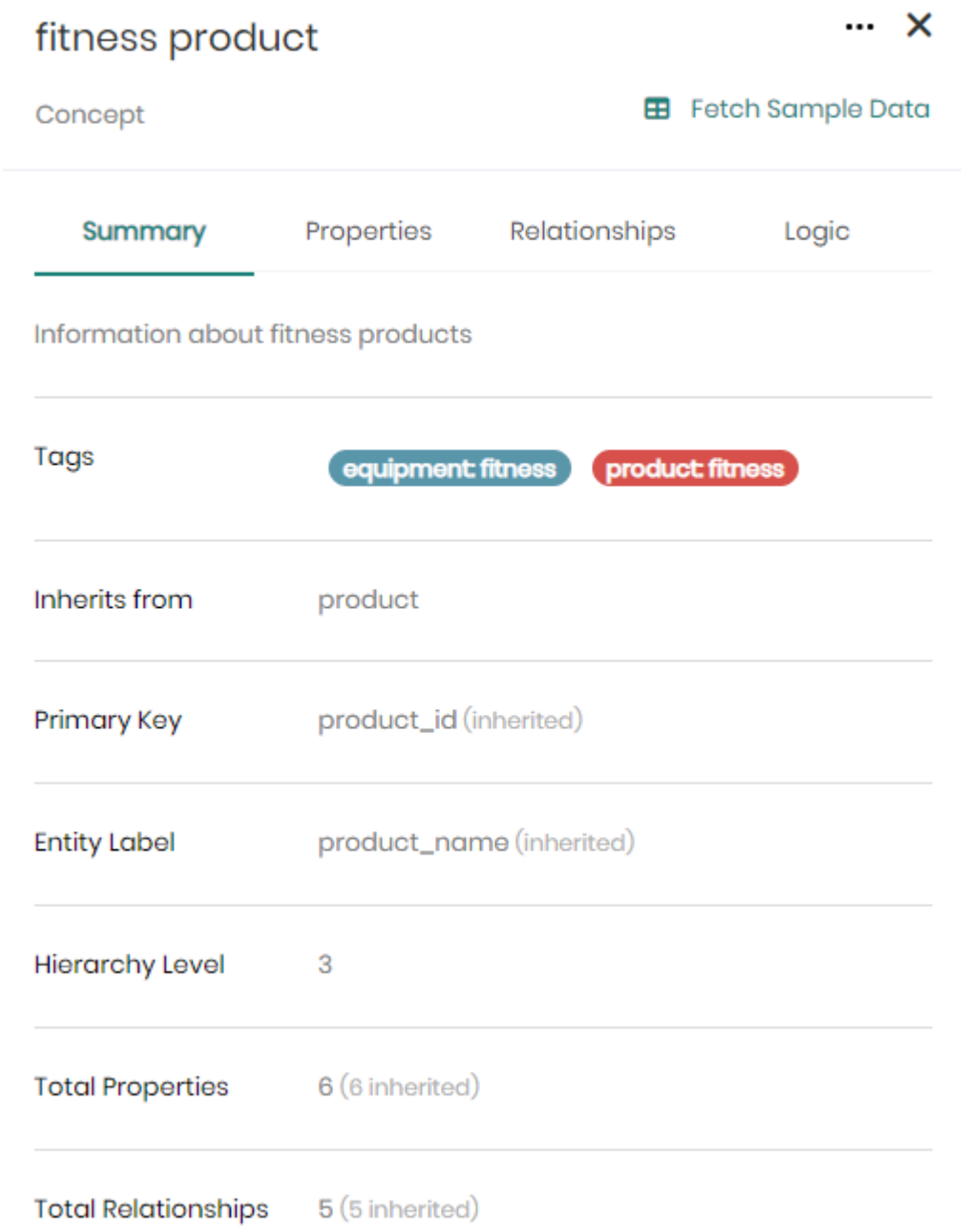
Show - When hovering over Show the following options will appear:

- **Expand/Collapse child concepts** - Choosing to expand a concept will expand and present the child concepts, whereas choosing to collapse a concept will collapse and hide the child concepts.
- **Show relationships** - Presents all the chosen concept's relationships with other concepts on the graph.
- **Show properties** - Presents all the chosen concept's properties on the graph.
- **Filter graph** - Filters the graph presenting only the chosen concept and its hierarchy.
- **Information** - Opens the concept details menu on the right side of the screen, showing a detailed summary of the selected concept.

Mappings - Opens a window that enables the mapping of data to the chosen concept in a few simple steps.

Delete - When clicked the chosen concept will be deleted from the Ontology model.

Concept Details



This menu opens from the right-side of the screen when you click on a concept in the graph. The menu shows a detailed summary for the selected concept, and includes:

Fetch Sample Data - Retrieves a sample of 50 rows of data for the selected concept. The data retrieved does not include any graph traversal and is used just for plain data exploration.

Summary Tab

> **Description** - Presents the concept's description.

> **Tags** - Presents the tags given to the chosen concept.
>
> **Inherits from** - Presents the concept the chosen concept inherits from.
>
> **Primary Key** - The primary key of the concept.
>
> **Entity Label** - The entity label of the concept, where there can be one or multiple entity labels.
>
> **Hierarchy Level** - The hierarchy level of the concept in the ontology.
>
> **Total Properties** - How many properties (direct and inherited) are in the concept.
>
> **Total Relationships** - How many relationships (direct and inherited) are in the concept.

Properties Tab

> **Inherited Properties** - A list of the concept's inherited properties with their name and type if any exist.
>
> **Direct Properties** - A list of the concept's direct properties with their name and type if any exist.

Relationships Tab

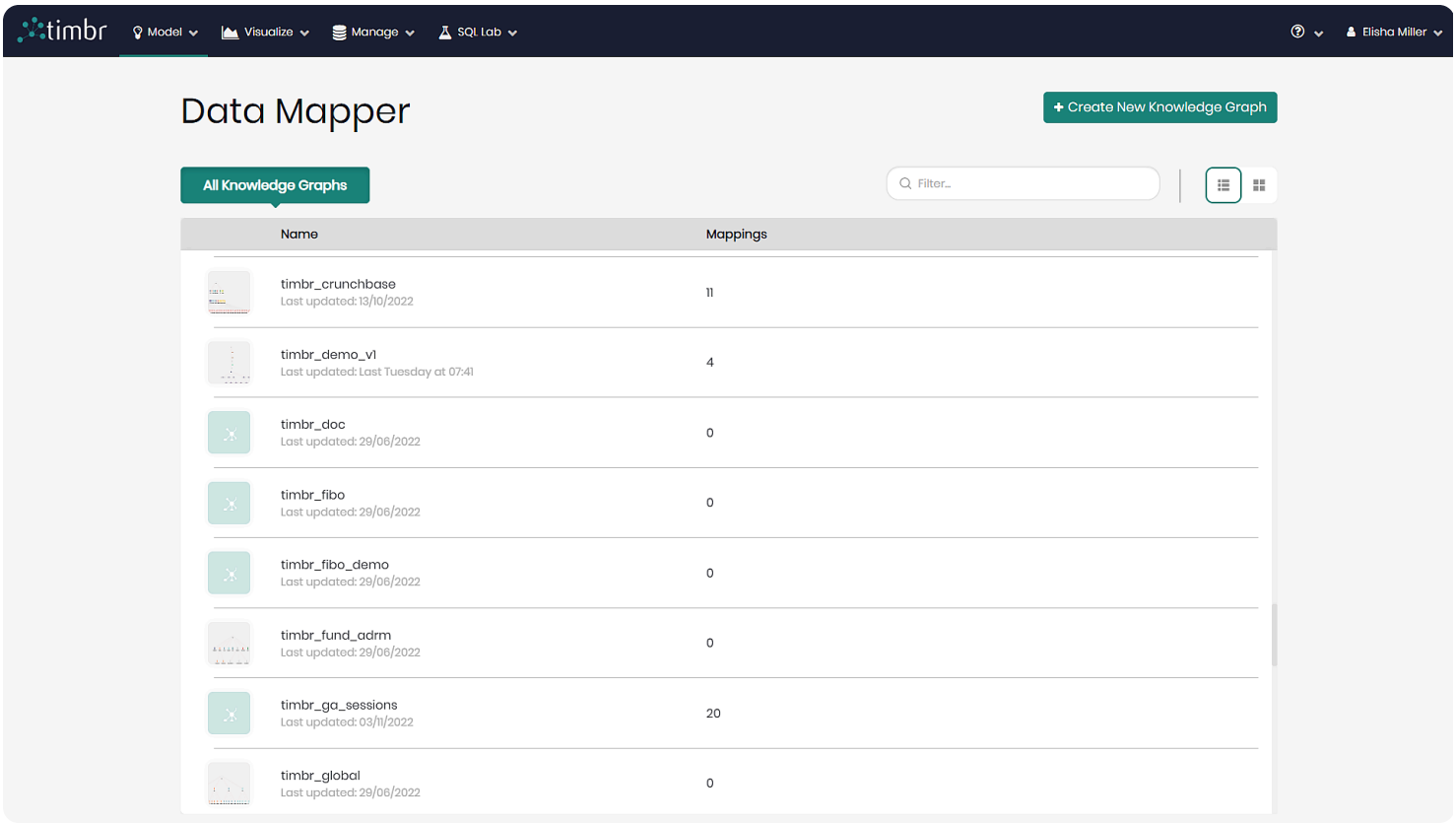
> **Inherited Properties** - A list of the concept's inherited relationships including the relationship name and the concept the relationship is connected to if any exist.
>
> **Direct Properties** - A list of the concept's direct relationships including the relationship name and the concept the relationship is connected to if any exist.

Logic Tab

For concepts that contain predefined business logic, the logic will be presented as an SQL statement in the logic tab.

Data Mapper

Timbr's Data Mapper page enables users to manage, create, edit, and remove data mappings from the data model to the underlying data sources. The data mappings can include calculated fields and any other function the underlying database supports (`JOIN`, `UNION`, `CAST`, `CONCAT`, etc.). The Data Mapper has complete version control and a simple-to-use cache functionality to optimize query performance. The data mappings can be performed using either the no-code UI or standard SQL for more complex scenarios.



Timbr's *Data Mapper* component can be accessed through the **Model tab** by clicking on **Data Mapper** and selecting the desired knowledge graph.

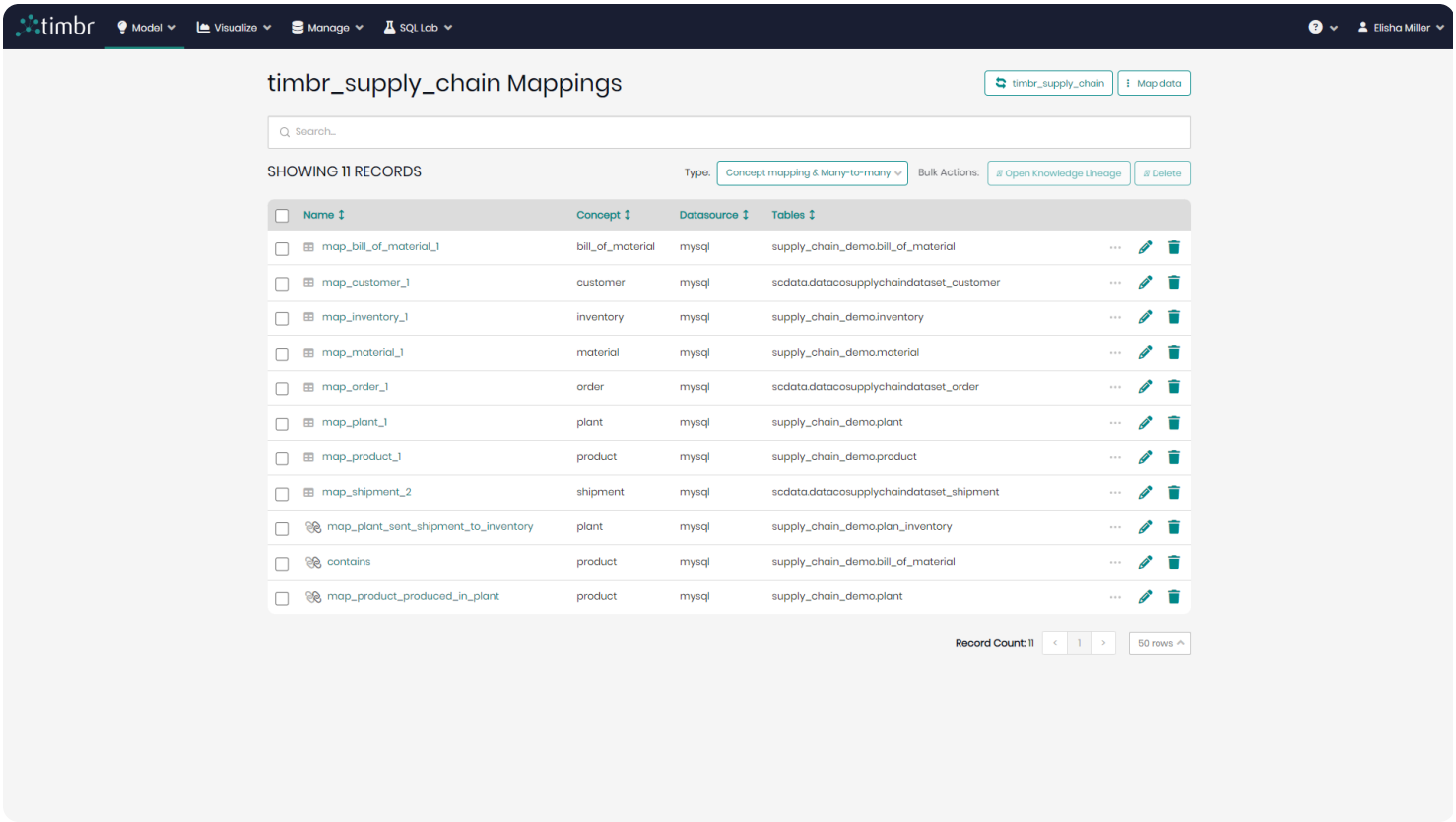
Getting Started

There are two ways to begin mapping data in Timbr's Data Mapper. The two ways are:

Create New Knowledge Graph - This button can be found on the top right side when entering the data mapper, enabling users to create a new knowledge graph on the spot and map data to it.

Choose a Knowledge Graph - Users can select any knowledge graph from the existing knowledge graphs on the list and begin mapping data to the selected knowledge graph.

Once the knowledge graph is chosen, all existing mappings which include concept mappings, many-to-many mappings and multi-value mappings will appear.



On the top right next to the mapping name there are two options:

Change Current Knowledge Graph - When clicked on, a pop-up window will appear enabling users to choose any other existing knowledge graph from the list.

Map Data - Opens a pop-up window offering users to create a mapping of one of three types (concept mapping, many-to-many mapping, multi-value mapping).

Below the previous two options is the search bar enabling users to search for specific mappings from the list.

Beneath the search bar is the number of current mappings. To the right there are the following options:

Type - Allows users to filter the list of mappings by the different types of mappings.

Bulk Actions - Any mappings selected from the list can have bulk actions performed on them, which include:

- **Open knowledge Linage** - Opens a knowledge lineage graph so users can visualize the selected mappings and their full lineage on the graph.

- **Delete** - Enables users to delete any mappings they'd like.

Each Mapping on the list contains a name, a concept it belongs to from the knowledge graph, a data source it's connected to, and the table the mapping is using.

To the right of that information, each individual mapping has its own functionalities which include:

Cache Resource - Enables users to cache the data behind the selected mapping.

Copy Query - Enables users to see and copy the SQL syntax to the clipboard that created the mapping.

Open Knowledge Lineage - Opens a pop-up with a knowledge lineage graph containing the selected mapping.

History - Presents users with a history log containing all the changes and actions performed on the mapping.

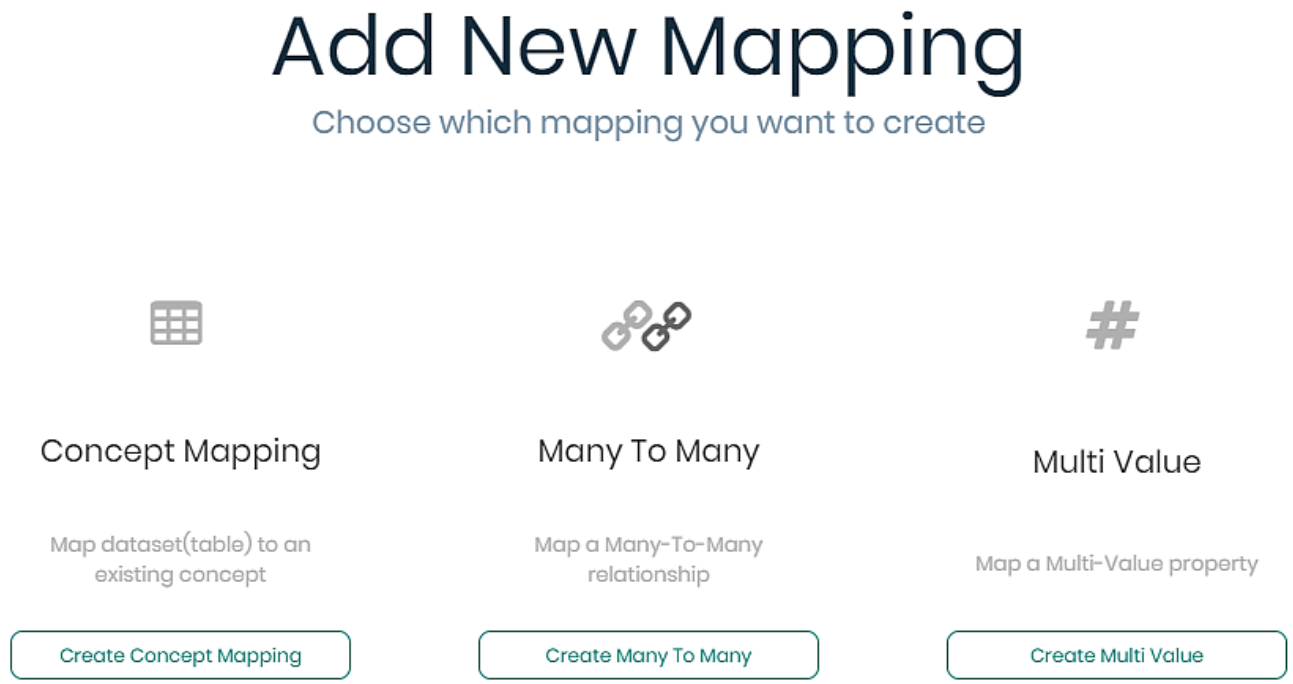
Edit Mapping - Allows users to make changes to the existing mapping.

Drop Mapping - Drops and deletes the selected mapping from the knowledge graph.

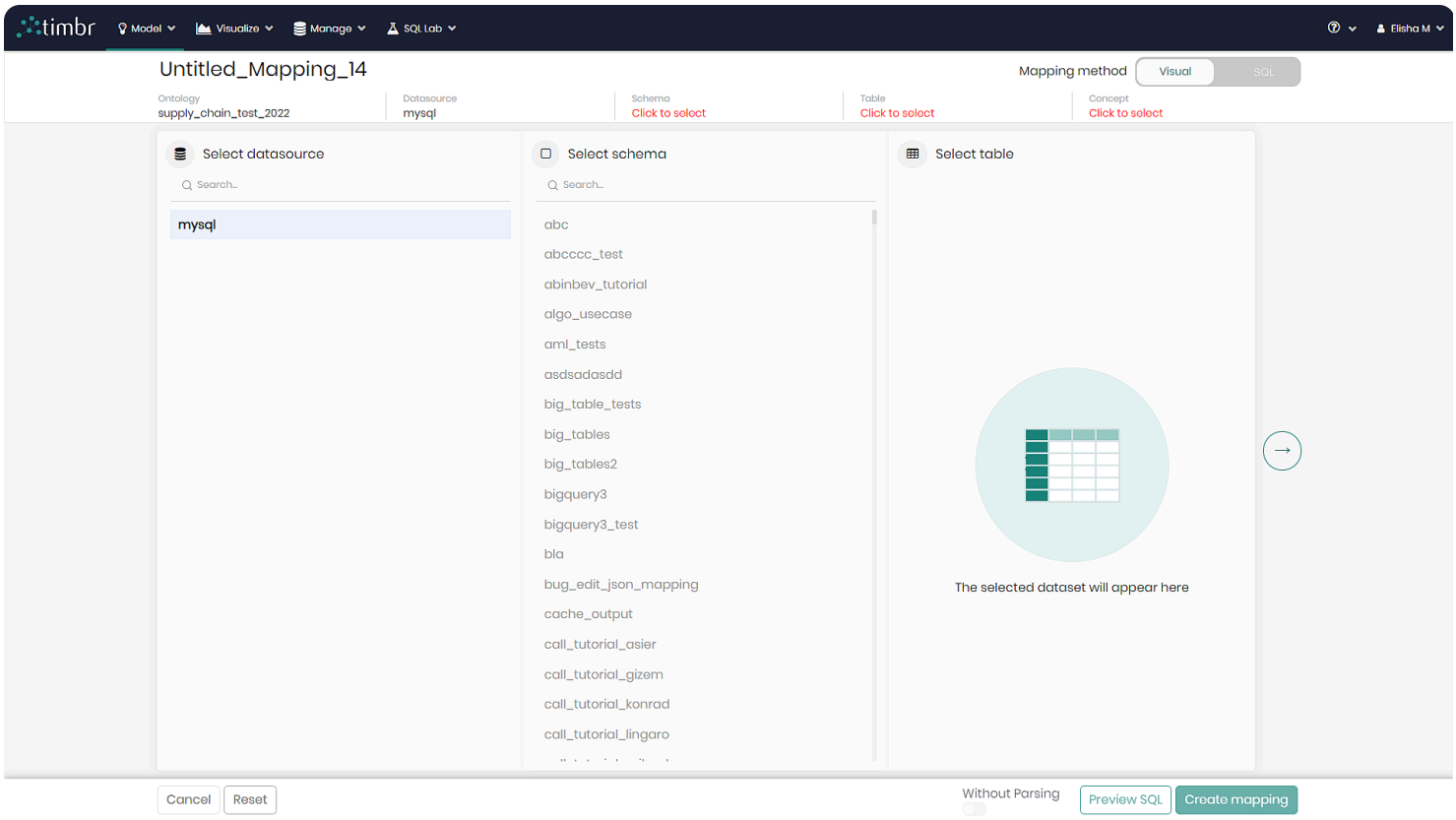
Mapping Data to a Concept Visually

Step 1 - Click on Map Data on the top right side of the Data Mapper screen.

Step 2 - Click on Create Concept Mapping.



Once clicked a concept mapping window will appear in order to visually map data to a knowledge graph concept.

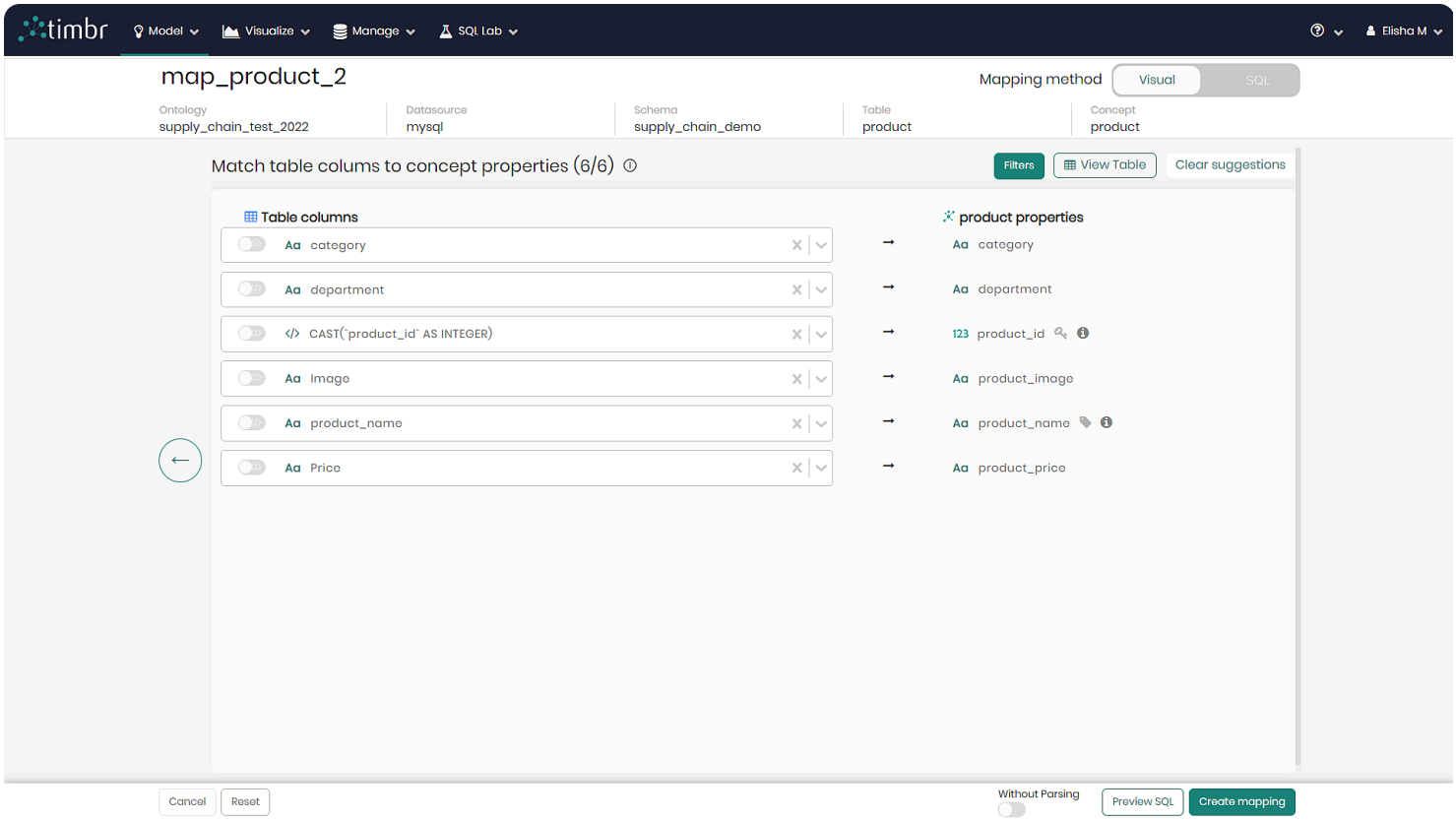


On the top right, users can switch the mapping method toggle from Visual to SQL in order to perform a concept mapping using SQL DDL statements.

Step 3 - On the top left, give your mapping a name.

Step 4 - Beneath the mapping name, the ontology name will appear. Alongside that, choose the relevant Datasource, Schema, Table, and Concept to which you wish to map data.

Step 5 - Match the column names from the datasource to the concept properties. The matching can be done manually or instead can be done automatically by Timbr by clicking on Add all suggestions on the top right of the screen. If matching is done automatically, it's worth checking to see if all the matches have been matched as intended.



At this point, users can perform any SQL functions on the table columns (CAST, SUM, etc.), as well as add desired filters.

When done matching, at the bottom of the screen users can choose the following actions:

Cancel - Cancels the mapping and returns to the main page of the Data Mapper.

Reset - Resets the mapping process and returns to rechoosing datasource, schema, table, and concept.

Parsing - A toggle for deciding whether to parse or not to parse the SQL in Timbr during the mapping.

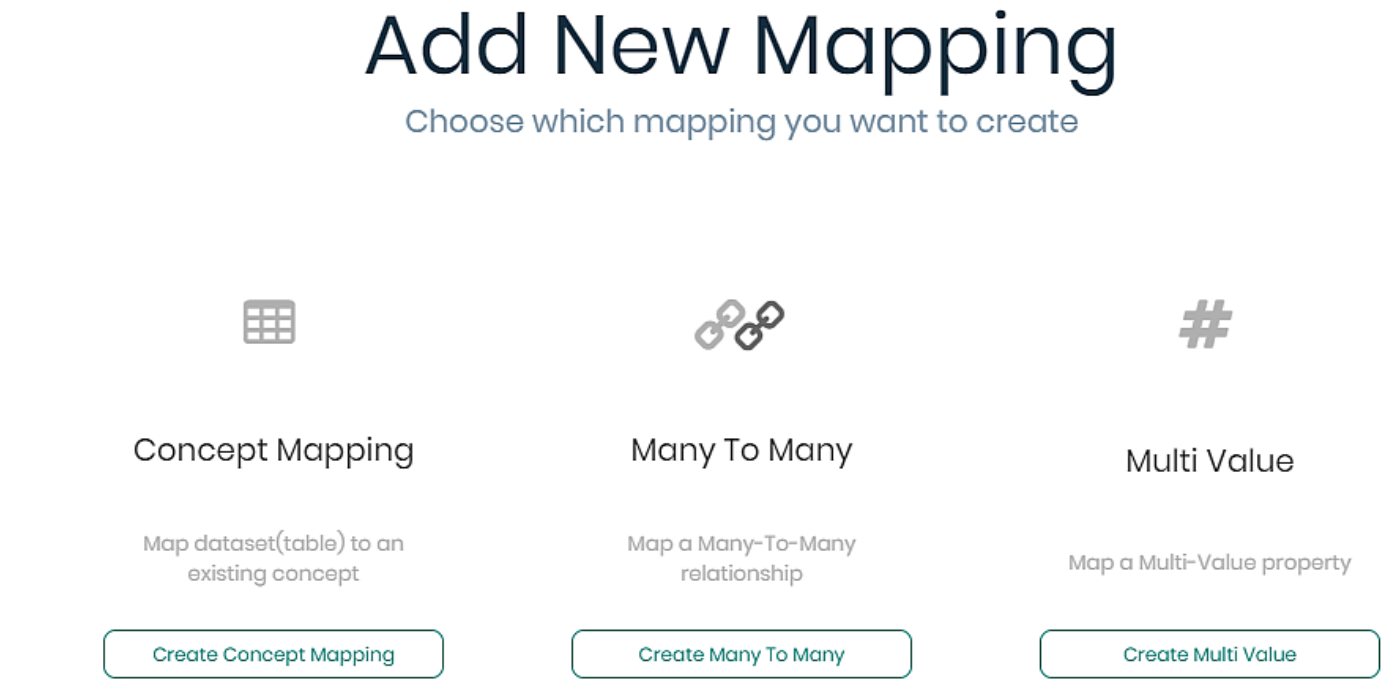
Preview SQL - Opens a slider with an SQL preview of the current mapping.

Create mapping - Creates and saves the mapping in the knowledge graph.

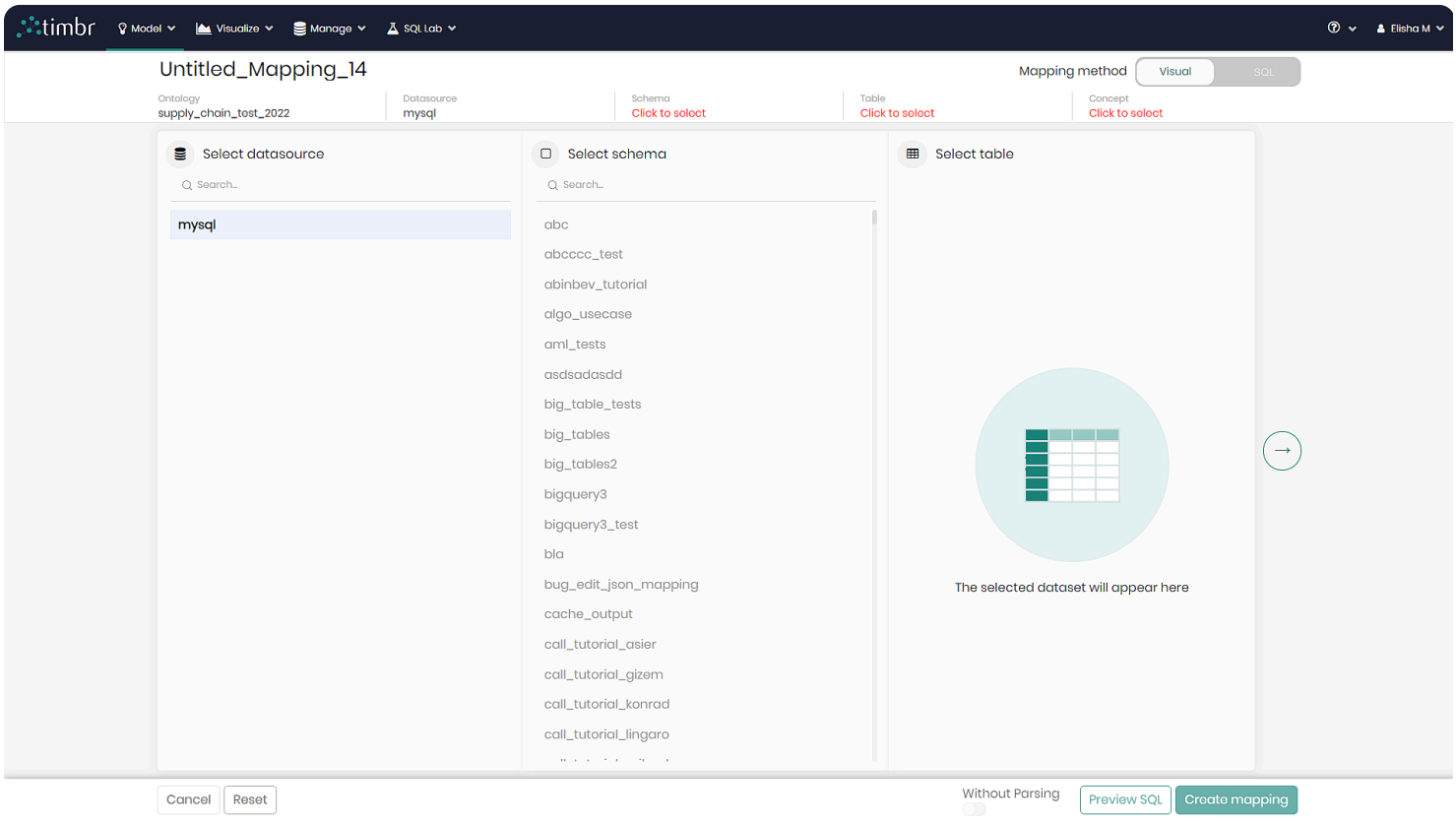
Mapping Data to a Concept with SQL

Step 1 - Click on Map Data on the top right side of the Data Mapper screen.

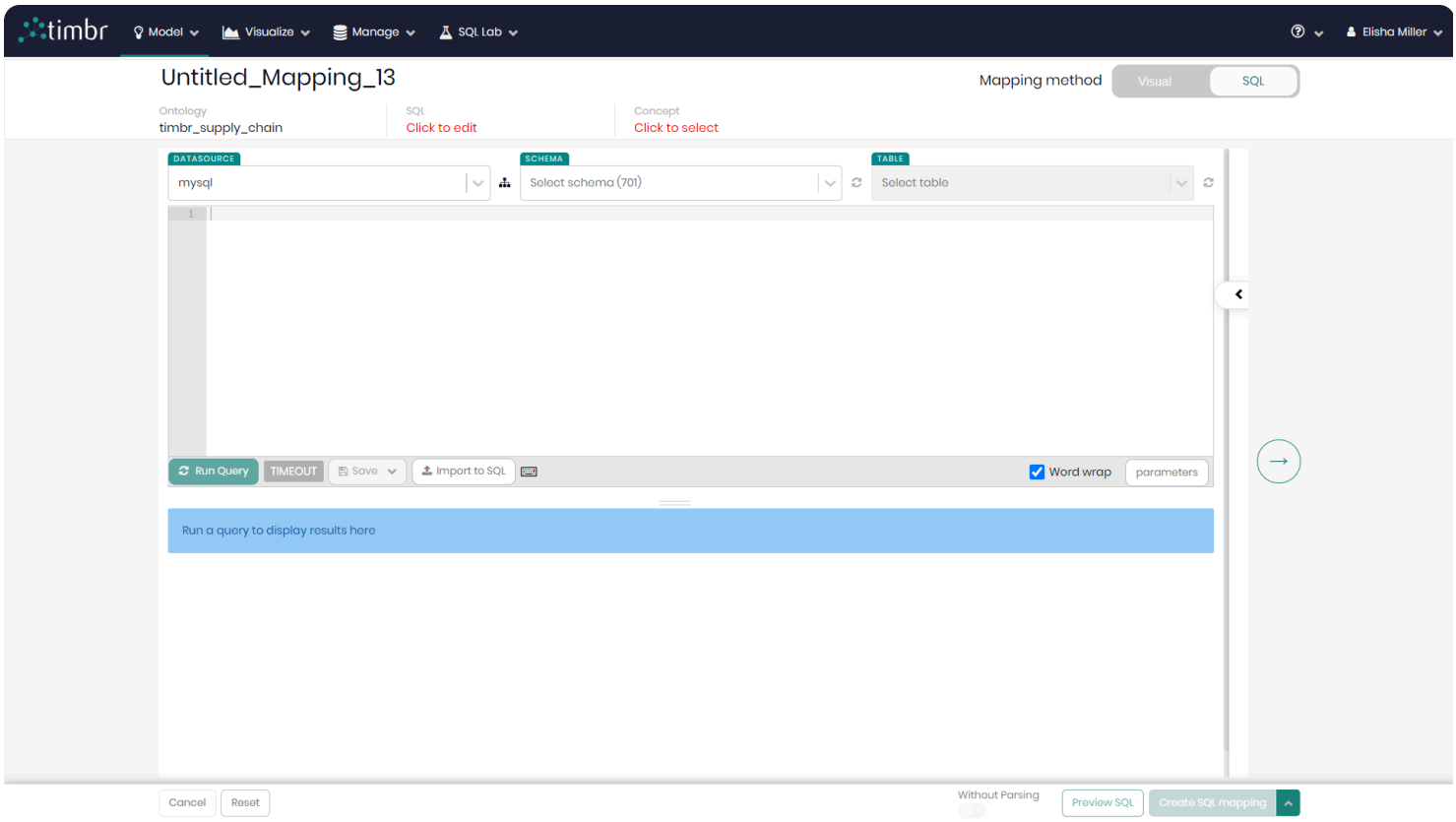
Step 2 - Click on Create Concept Mapping.



Once clicked a concept mapping window will appear in order to visually map data to a knowledge graph concept.



Step 3 - On the top right, change the Mapping method toggle from Visual to SQL.



Step 4 - On the top left, give your mapping a name.

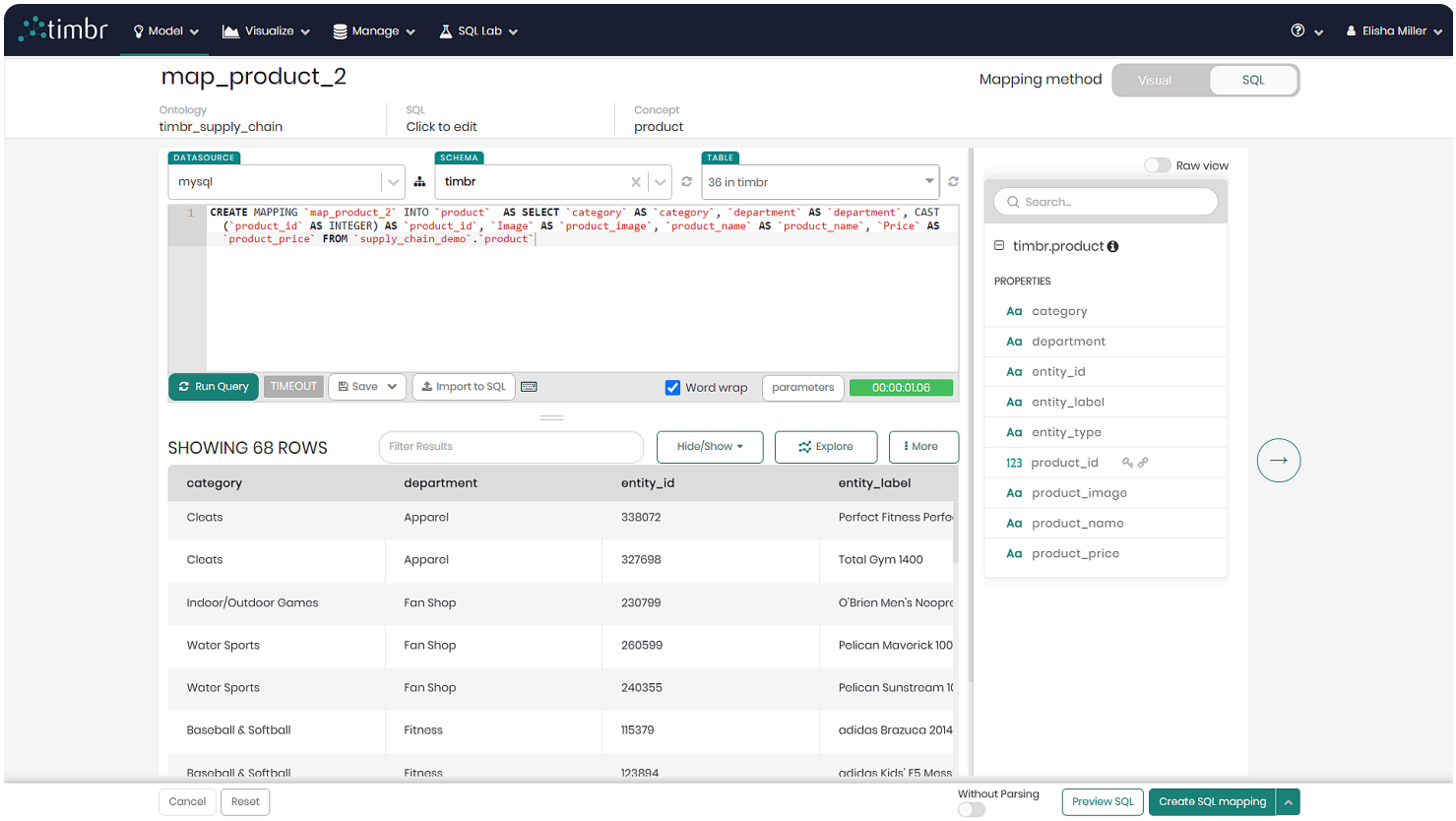
Step 5 - Beneath the mapping name, the ontology name will appear. Alongside that, click on SQL to begin editing the SQL mapping statements.

Step 6 - Next to edit SQL, once the mapping statements are in the main query box, choose the concept you wish to map the data to.

At this point, users can click on Create SQL mapping on the bottom right to create and save the mapping in the knowledge graph.

Users can also choose to query the different tables of their datasource as well as existing knowledge graph concepts to assist in creating the new mapping. This is done by simply writing the desired query in the main query box and running the query to see the results.

In addition, users can be assisted by selecting the datasource, schema, and table, which will slide in a panel with the metadata of the chosen datasource.



Beneath the query box there is a toolbar that includes the following functions:

Run Query - Runs the query in the main query box.

Timeout - Determining the query timeout in seconds. When left empty default timeout will be in effect.

Save - When clicked on, a dropdown opens with the 2 following options:

- **Save query** - Saves the current query which can later be found in the SQL lab tab under Saved Queries.
- **Create view** - Enables to create a view above the knowledge graph using the query results.

Import to SQL - This enables uploading a CSV file and having Timbr convert the CSV file into an SQL query that can then be mapped to the knowledge graph.

New table name - This enables typing a new table name and saving the query results to the new table being created.

Keyboard shortcuts and snippets - Presents various shortcuts and snippets in the SQL editor helping users write and run queries faster.

Word wrap - When the word wrap checkbox is marked, word wrap will be activated and any text that exceeds the length of any specific row will be pushed down to the next row. When the word wrap checkbox is unmarked text on any row can continue until specified otherwise.

Parameters - This enables creating and editing templates using the Jinja templating language which allows for using macros in the SQL code.

Query timer - Shows the query run time once run query is clicked.

When actions are done regarding the query, at the bottom of the screen users can choose the following actions:

Cancel - Cancels the mapping and returns to the main page of the data mapper.

Reset - Resets the mapping process and returns to rechoosing the mapping method.

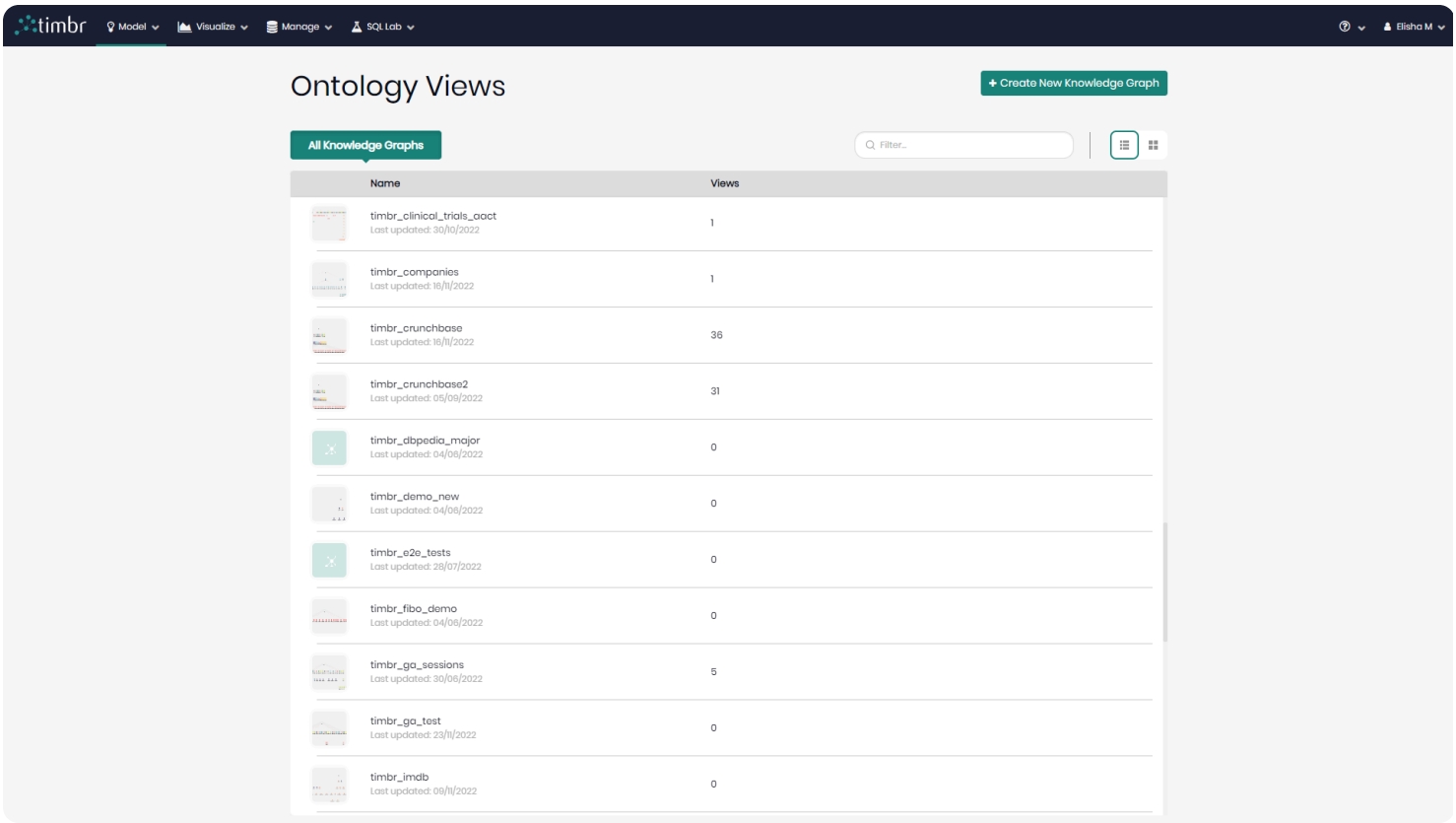
Parsing - A toggle for deciding whether to parse or not to parse the SQL in Timbr during the mapping.

Preview SQL - Opens a slider with an SQL preview of the current mapping.

Create mapping - Creates and saves the mapping in the knowledge graph.

Ontology Views

The Ontology Views in Timbr enable users to create, manage, and roll back views in a user-friendly manner. Views can be defined on top of the physical tables (unrelated to the Ontology Model), or on the business concepts and their relationships defined in the knowledge graph, or on top of other Ontology Views. Views can include calculations and any data function that the underlying database supports. Views can be cached for optimizing performance and include granular access controls for exposing views to different users with different access permissions.



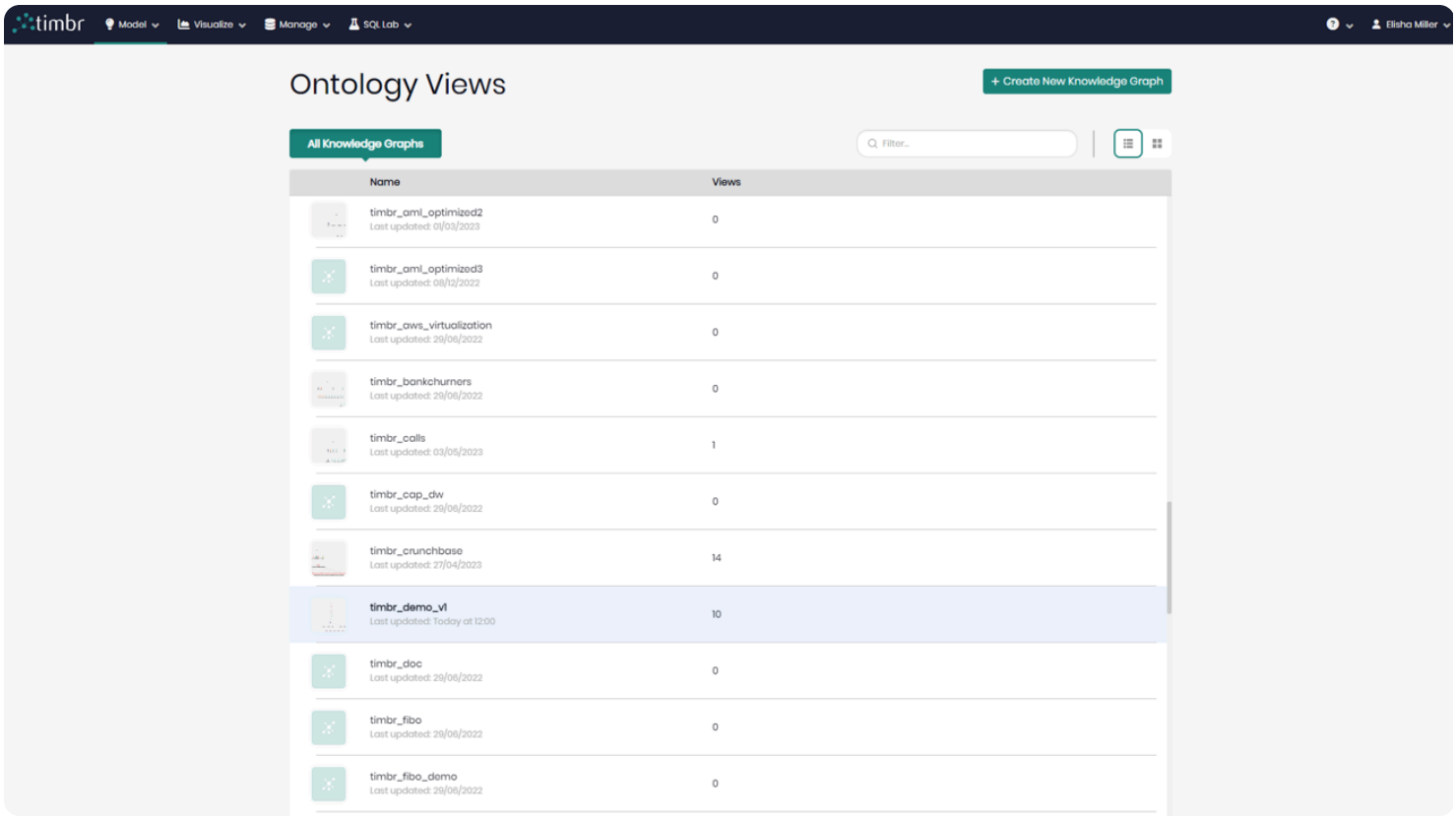
Timbr's *Views* component can be accessed through the **Model tab** by clicking on **Ontology Views**. Views can also be accessed through Timbr's *SQL Editor* by saving the results of queries as views.

There are a few ways and reasons as why to create *Views* in Timbr. And those are:

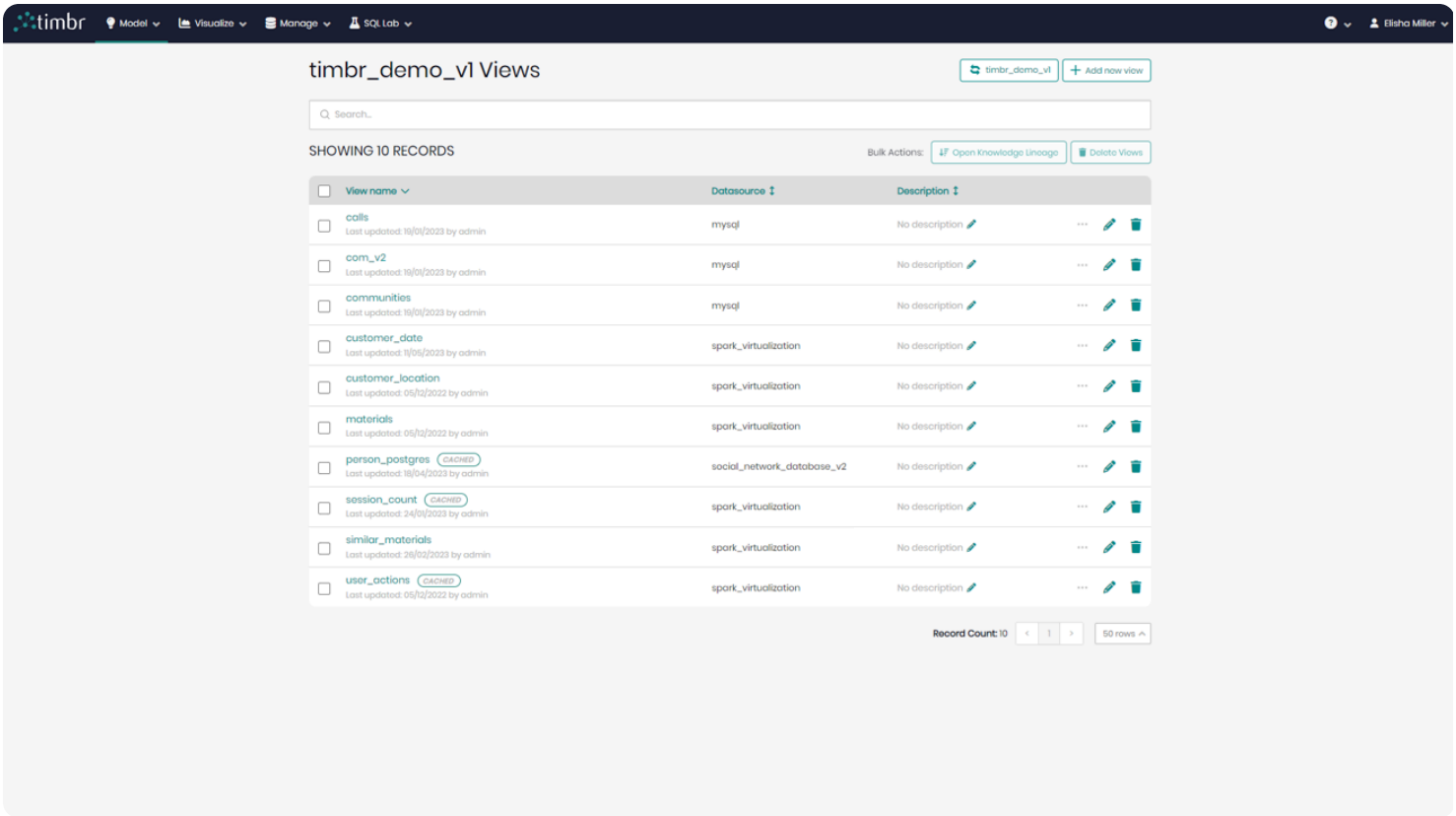
- To create a **view using the physical tables in the database**. This option enables users to import their existing views into Timbr for easier access and control management, together with a full lineage down to the source.
- To create **views on top of the knowledge graph concepts**, which enables users to combine aggregations using the knowledge graph relationships.

- To create **views above existing views** which enables users to combine aggregations using the knowledge graph relationships, where they can then add them to existing views when done.

Once *Ontology Views* is clicked on in the *Model* tab you will be asked to select an existing knowledge graph you wish to create or edit views on. Another option on the top right is to click on Create New Knowledge Graph in order to create a new knowledge graph to form the new views on.



Once the Knowledge Graph is chosen, a list of the existing Knowledge Graph views will appear (if any exist).



Above the list of views on the top right users can switch between knowledge graphs at any time to enter different views created by clicking on the arrows and Knowledge Graph name. Next to that, users can click on **+ Add new view** to add one or more views to the Knowledge Graph.

Beneath those 2 buttons is a search bar in cases where there are many views to search through.

Beneath the search bar users can find two bulk action buttons.

- **Open Knowledge Lineage** - Opens the Knowledge lineage component for all the selected views on the list.
- **Delete Views** - Deletes all the selected views on the list.

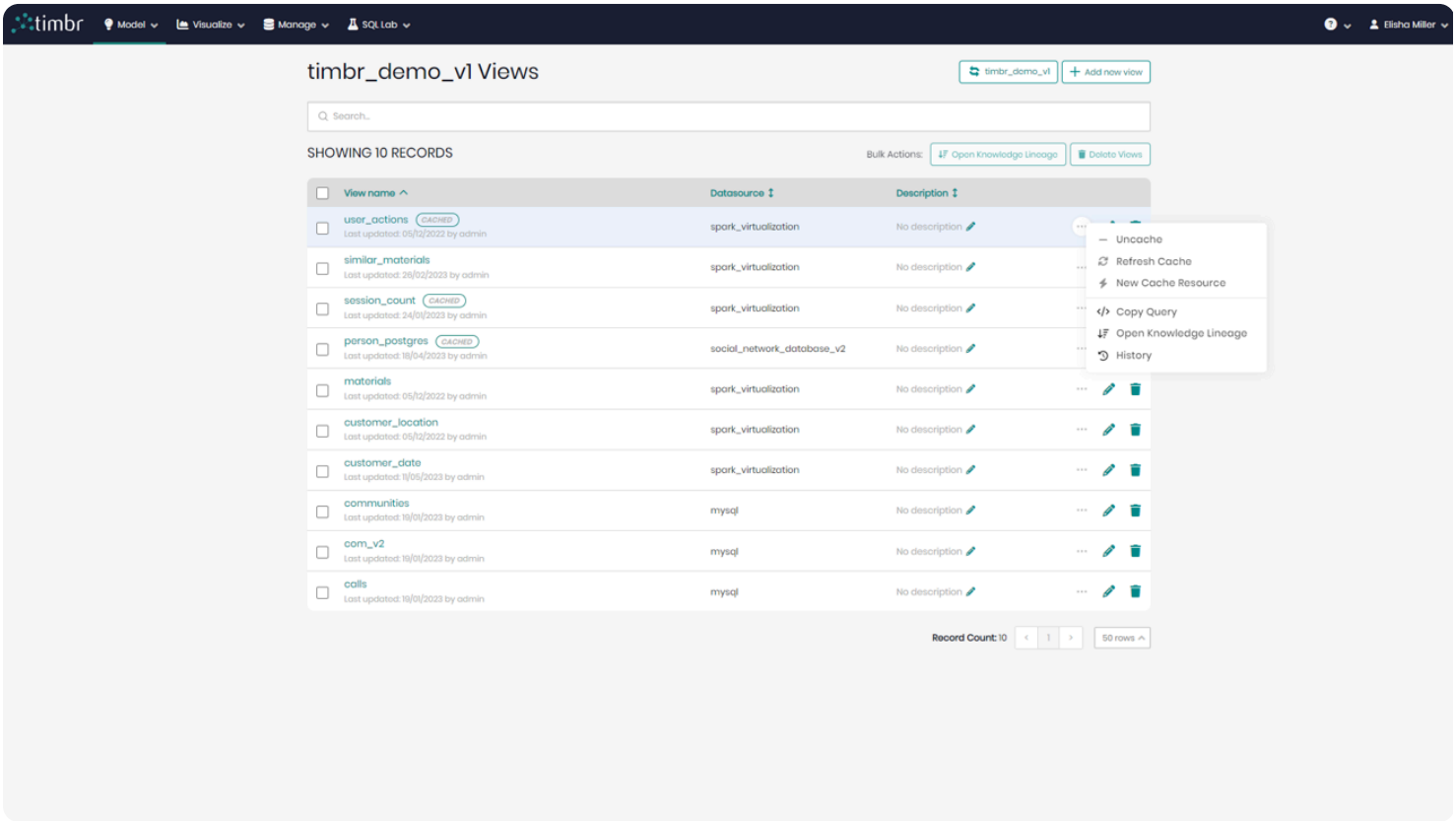
There are specific actions that can be performed on each view on the list. Those actions can be seen to the right of each view and its datasource and description, and they include:

The 3 Dots – When clicking on the 3 horizontal dots, a pop up will appear with the following options:

- **Uncache** – Removes the caching from the selected view if any caching exists.
- **Refresh Cache** – Refreshes the cache assigned to the selected view.
- **New Cache Resource** – opens a pop-up window to create a new cache for the selected view.
- **Copy Query** - Copies the SQL query of the selected view to the clipboard.
- **Open Knowledge Lineage** - Opens a window presenting the selected views lineage on a graph so users can really understand how the view was created and what it's connected to.
- **History** - Opens a window with the selected views history, stating when the view was created, edited and when different actions took place on the view.

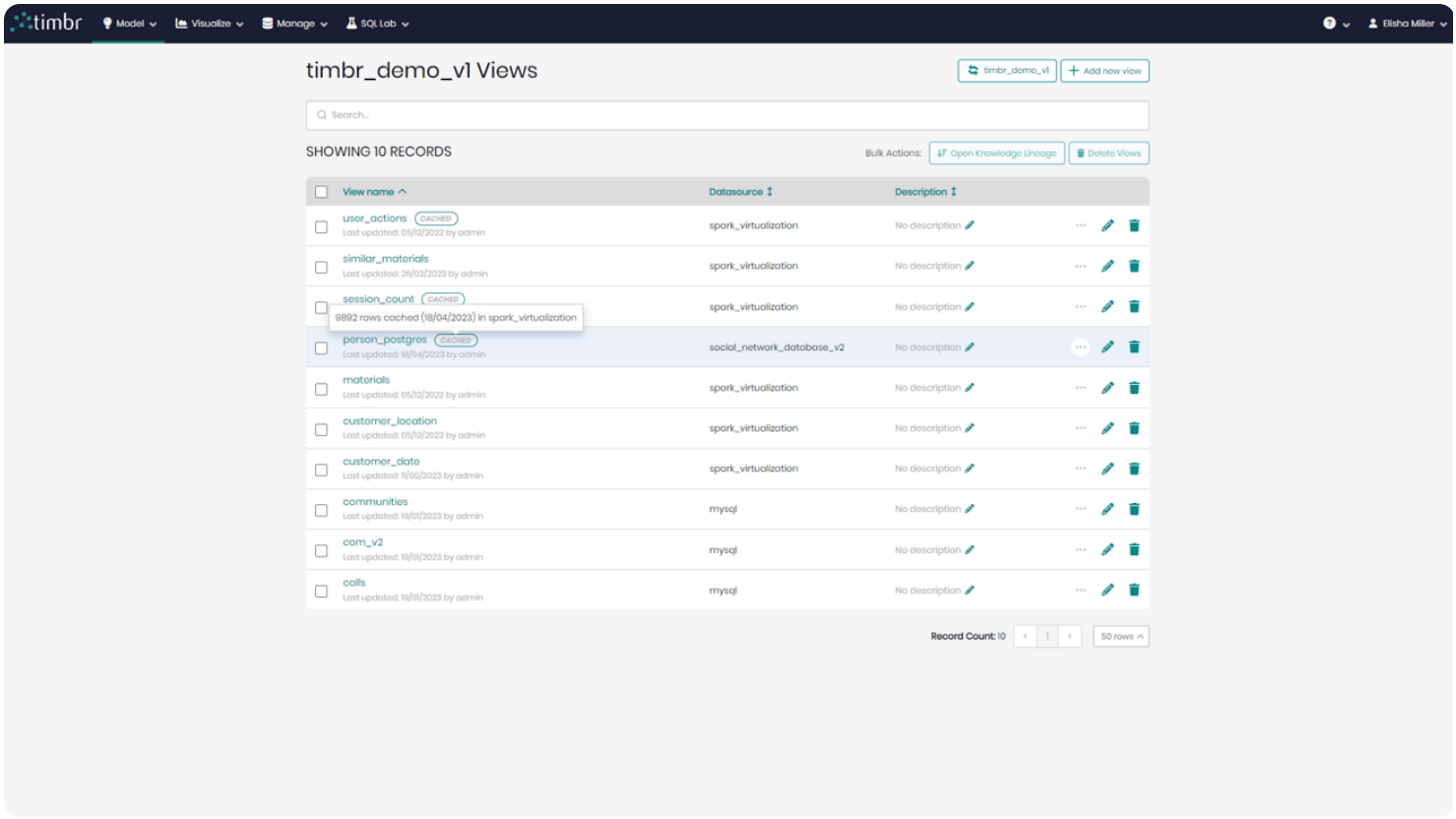
Edit View - When the *edit view* pencil icon is clicked, a window opens enabling users to edit the selected view.

Remove - When the *remove* trash can icon is clicked, the selected view will be removed and deleted from the list.

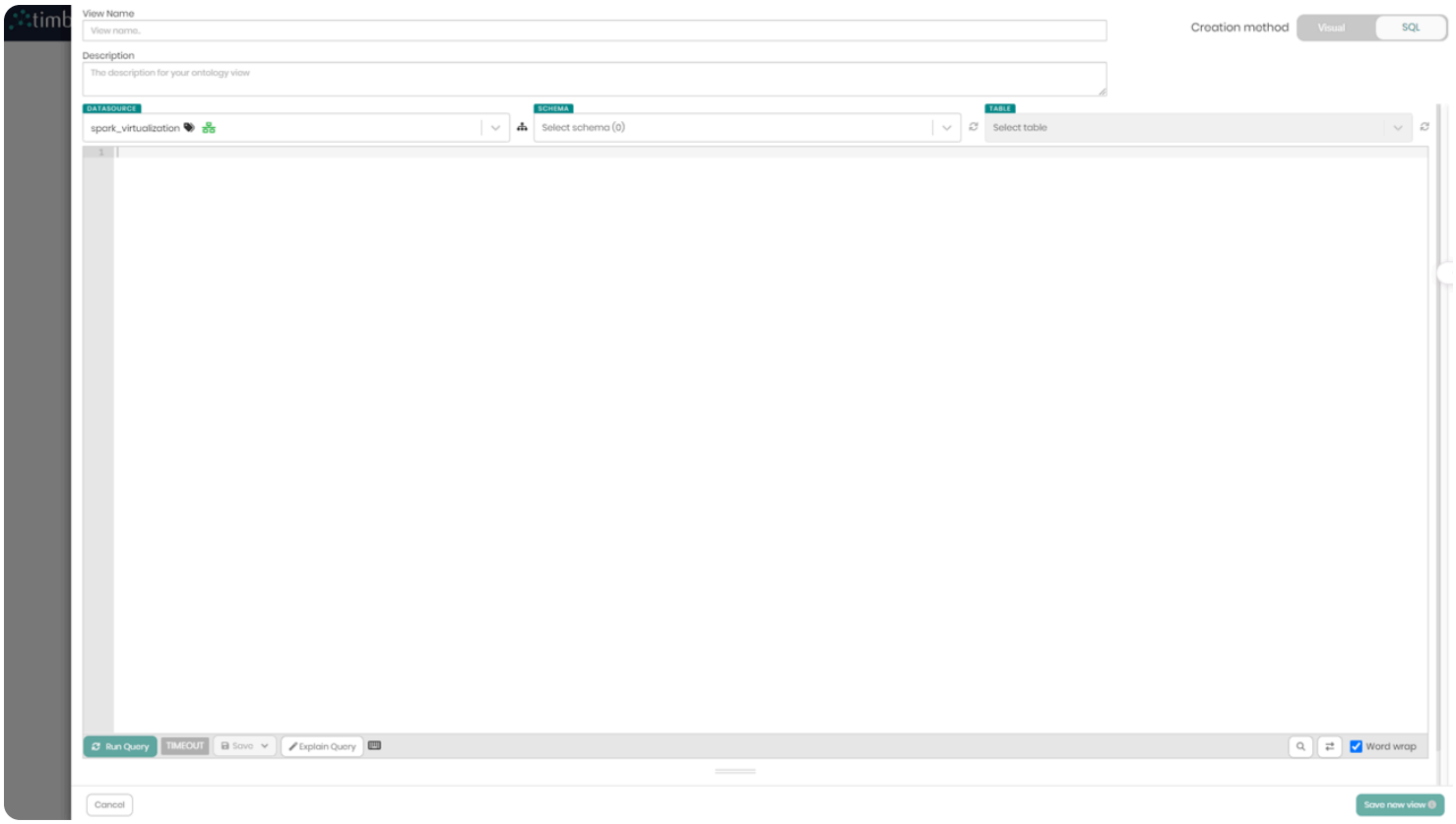


CACHED VIEWS VS. NON CACHED VIEWS

Timbr enables users to differentiate between views that are not cached with those who are, by adding a cached logo next to the view name for views that are cached. When hovering over the logo users can see the details of the caching which include *Total cached rows*, *Time and date of caching* as well as *datasources used in caching*.



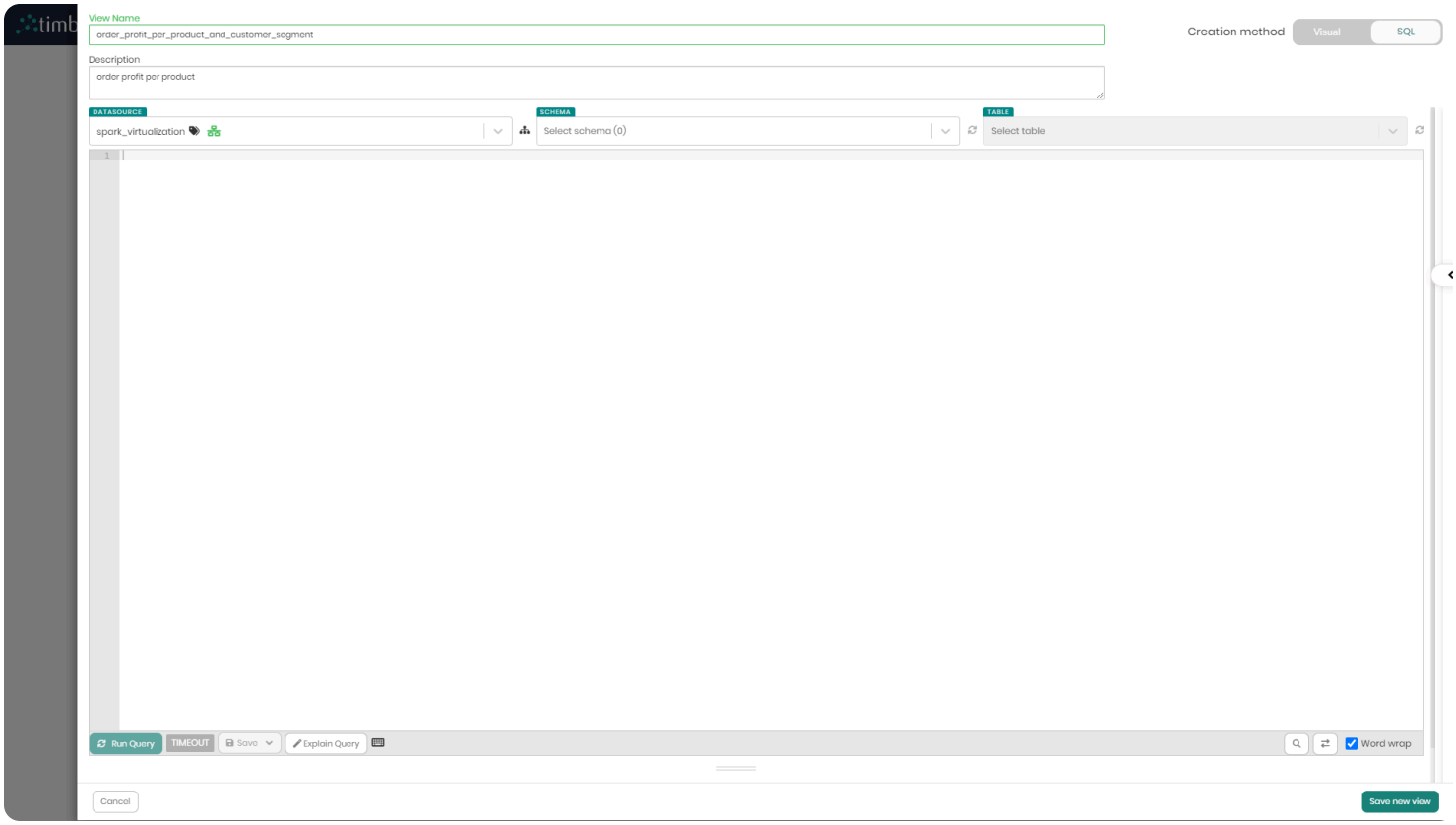
Once **+ Add new view** is clicked on the top right, a window will appear similar to the SQL Editor in order for users to enter their SQL statements that will create their view.



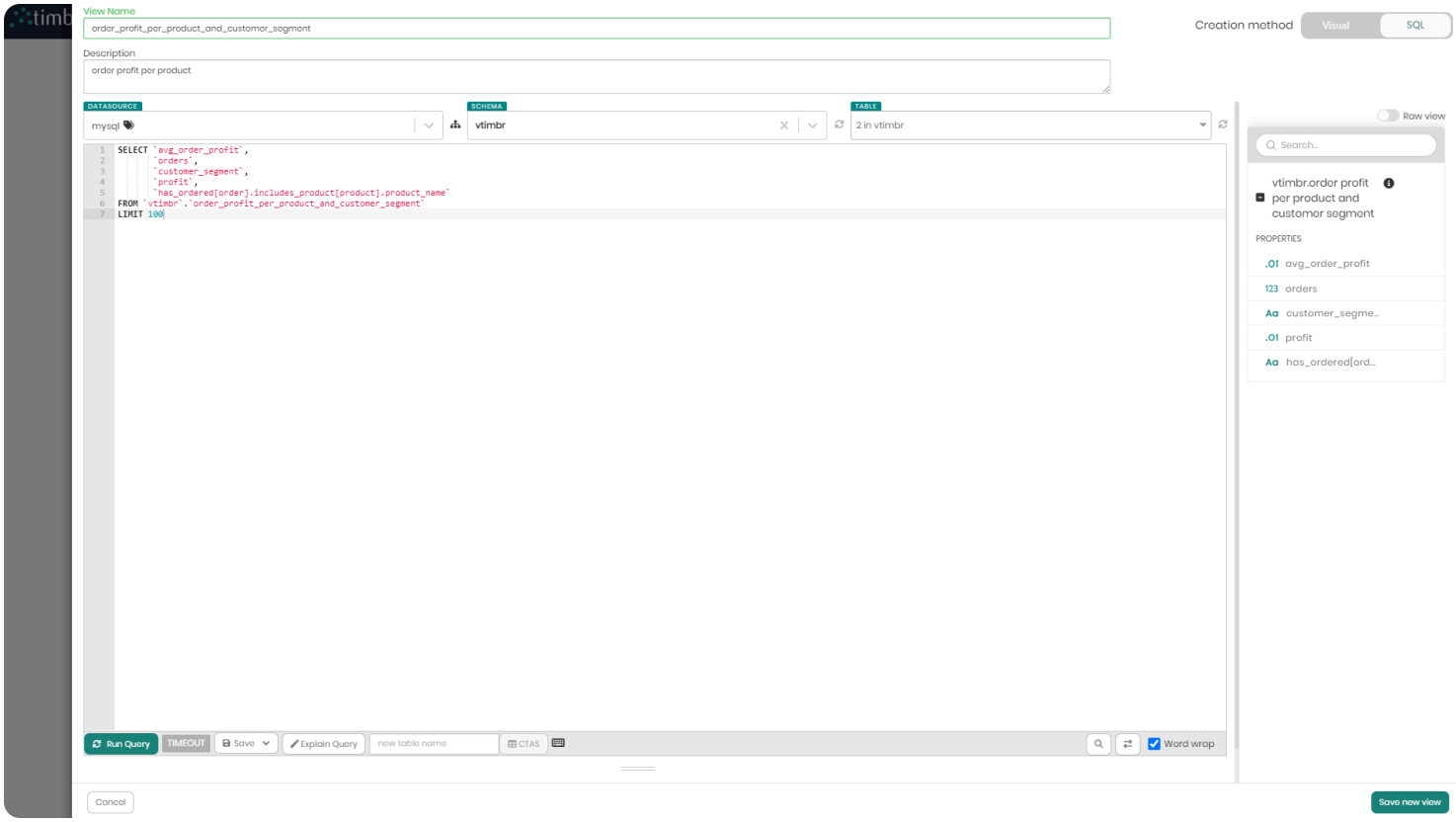
VISUAL MODELING

On the top right users can switch the *Creation Method* toggle to create the view in a visual manner without the use of SQL.

1. In the main SQL screen, the first step is to give the view a **Name** and an optional **Description**.



2. Next, the **SQL statement** should be entered in the main box. Users can choose their *Datasource*, *Schema* and *Tables* to assist them in building their views.



3. Once the SQL statement is prepared, at this stage users can simply click on **Save new view** on the bottom right, to complete the view and save it.

There are additional options before saving the view, which can be found beneath the query box in the query toolbar.

Those options include:

Run Query - Runs the query in the main query box. This enables users to see what the query results behind the view look like.

Timeout - Determining the query timeout in seconds. When left empty default timeout will be in effect.

Save - When clicked on, a dropdown opens with the 2 following options:

- **Save query** - Saves the current query which can later be found in the SQL lab tab under Saved Queries.
- **Map to concept** - Enables to map the query results to a chosen concept in the knowledge graph.

Explain Query - This opens an explain query window which presents the query that was pushed down behind the scenes of our current query to the datasource. This enables users to see what their query would look like without Timbr's relationships and business logic, where users would need to write up to 90% more code.

New table name - This enables typing a new table name and saving the query results to the new table being created.

CTAS - Creates a new table using the query results.

Keyboard shortcuts and snippets - Presents various shortcuts and snippets helping users write and run queries faster.

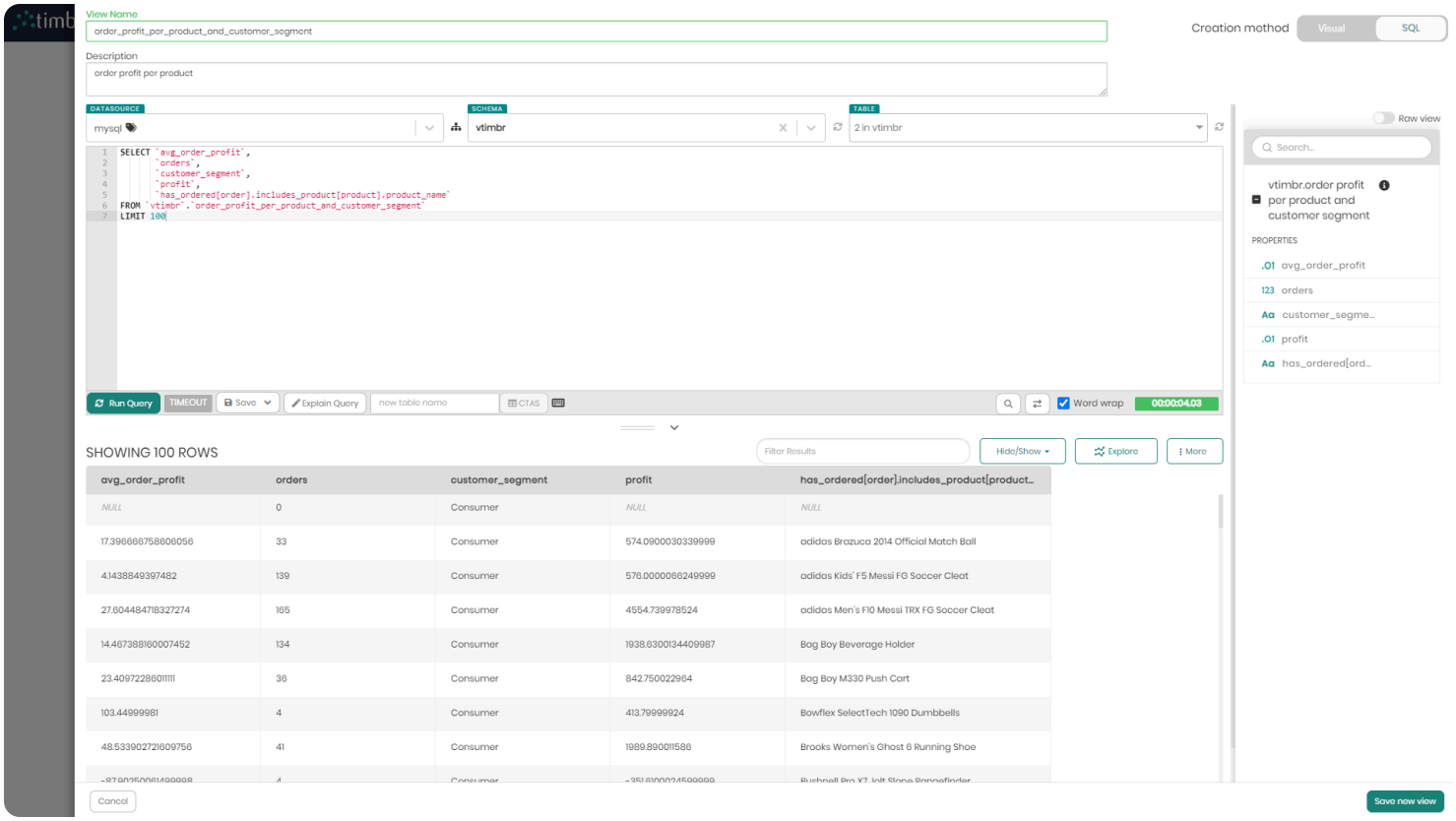
Search for - This enables users to search for specific words or phrases used within the query written above, which can come in handy when writing longer queries.

Replace with - This enables users to replace specific words or phrases used in the query written above with other words or phrases of their choice.

Word wrap - When the word wrap checkbox is marked, word wrap will be activated and any text that exceeds the length of any specific row will be pushed down to the next row. When the word wrap checkbox is unmarked text on any row can continue until specified otherwise.

Query timer - Shows the query run time once run query is clicked.

In the results section below, users are offered additional features which can be found right above the results box on the upper right side.



These options include:

Filter Results - A search bar to type and filter for specific results.

Hide/Show - Choosing which result columns to hide and which to show.

Explore - Enables to transfer the query results to Timbr's built-in BI module to explore and present the results using any of the various charts offered by Timbr.

More - The more button gives users additional options which include:

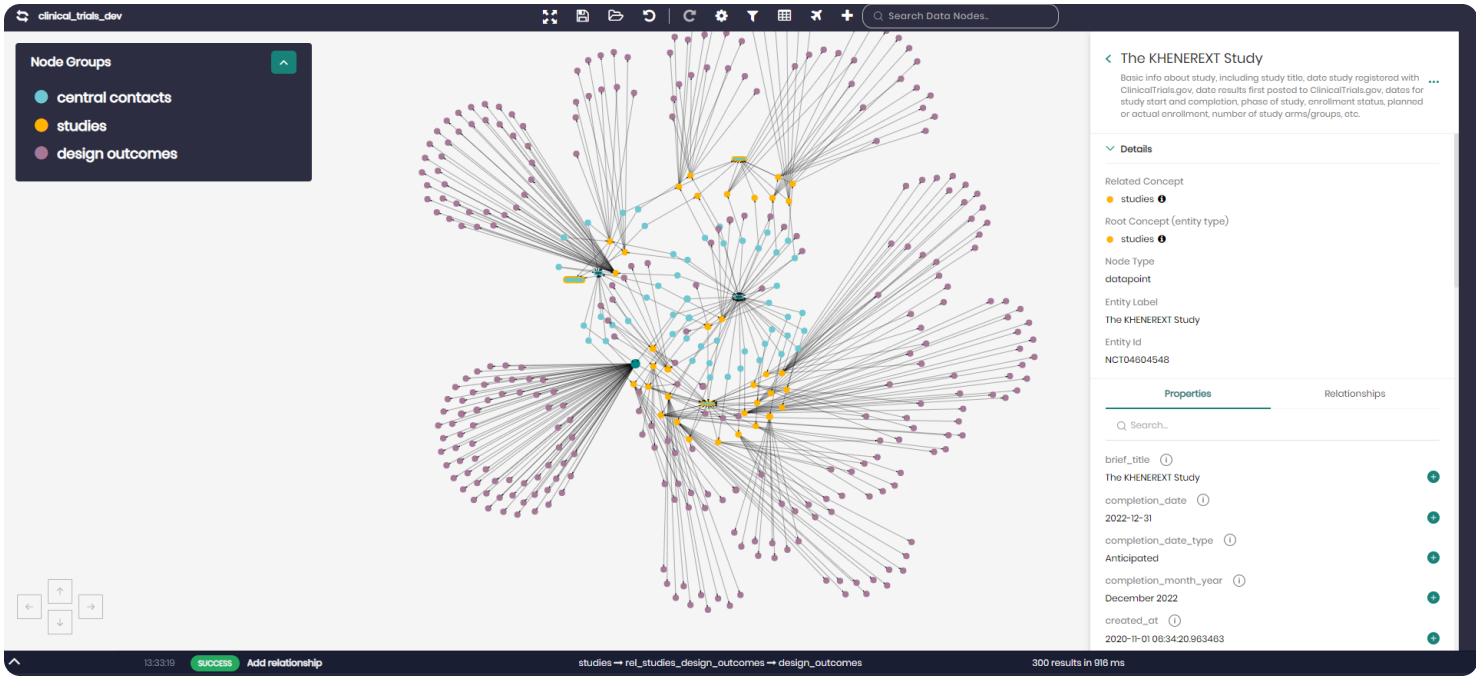
- **Remove null columns** - Removes all the null columns in the query results.
- **Export result set** - Exports the query results as a CSV, JSON or TEXT file.
- **Results Stats** - Presents different stats of the query such as total rows, columns, cells, existing values, nulls, query limit, query runtime, query start and end date.
- **Clipboard** - Copies the entire results to the clipboard to be used anywhere.
- **Show column stats** - Adds stats above the different columns such as unique values, null values, Min, Max, Avg and more.

Graph Explorer

The Graph Explorer page is a graph-based visualization tool for users to explore data as connected entities with direct or indirect relationships, commonalities, and dependencies. The Graph Explorer lets users intuitively explore hidden patterns and connections between data points. The data points are expressed as nodes, and the relationships connecting them are expressed as edges/vectors.

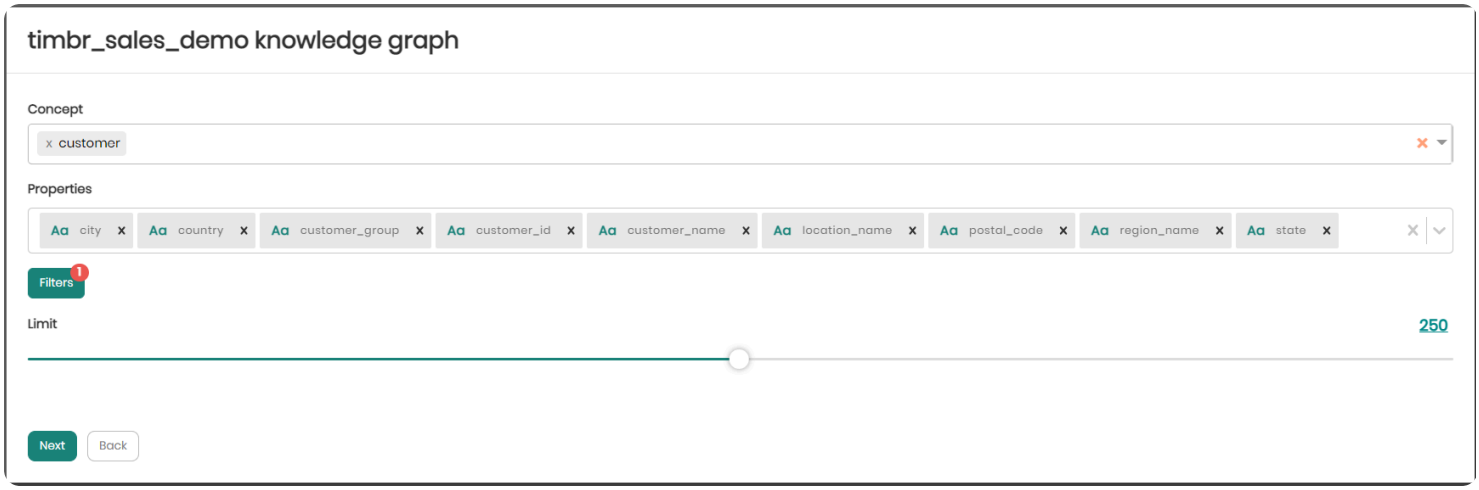
The Graph Explorer allows users to visualize the data and better understand how the data is related, gain insights on specific data points, and express complex scenarios or sequences of events.

Graph Explorer Page



The Graph Explorer can be accessed by clicking on the **Visualize** tab and selecting **Graph Explorer**.

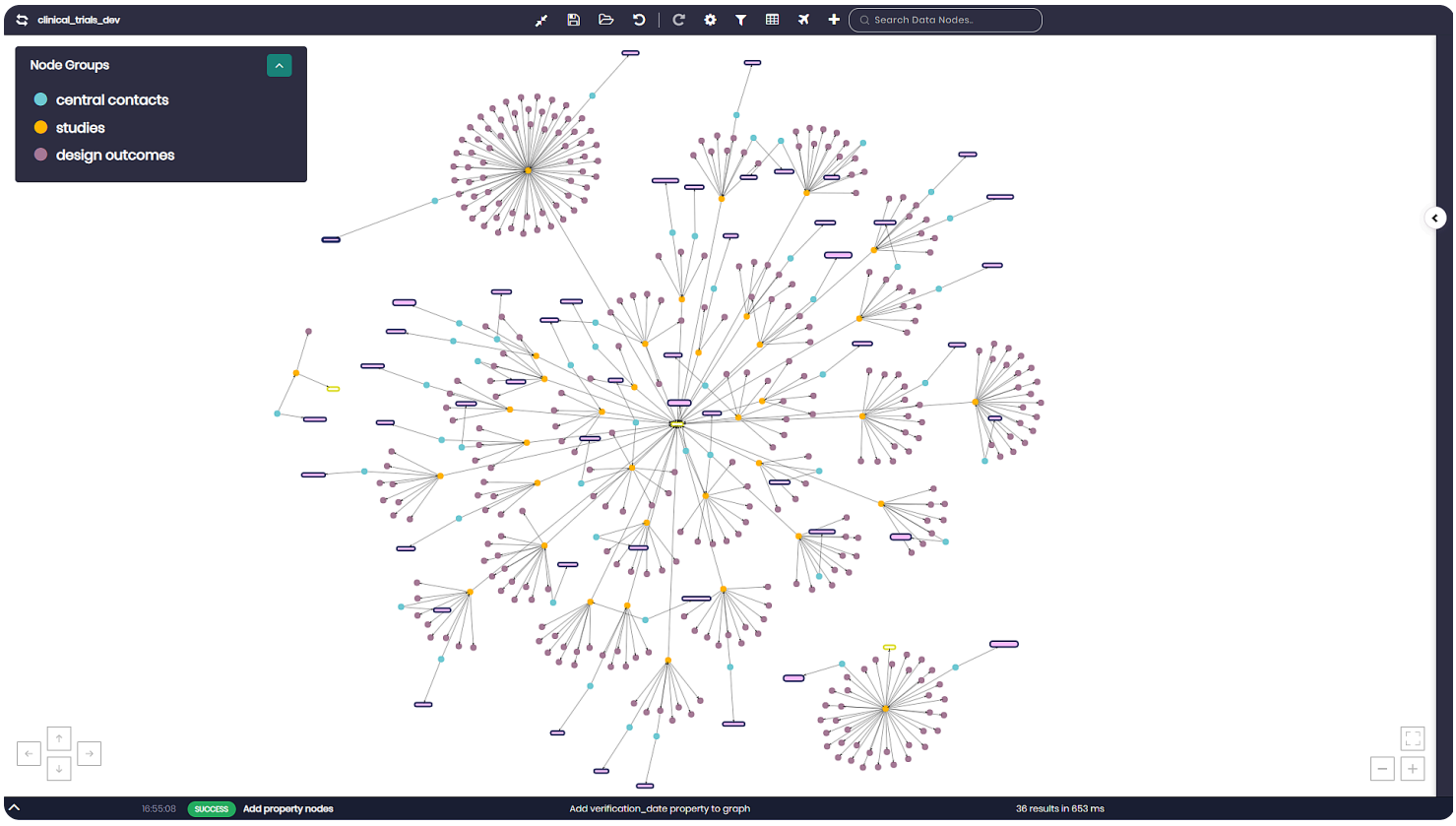
The starting point in the Graph Explorer page after selecting the desired knowledge graph is to create your initial query. The query is created with a user-friendly SQL query builder with dropdown menus to choose a **concept**, its **properties**, **filters**, and a **limit** to the query.



After the initial query, the resulting data points are shown as nodes on the graph. What this means is that a node in the graph represents a datapoint or a datapoint property (attribute).

The components of the graph explorer are as follows:

Main Graph View



An interactive graph representing the nodes as datapoints and relationships as edges. You can zoom-in and zoom-out of the graph view by using the mouse wheel.

Main Graph Control

The main graph control can be found above the graph and controls all things related to the ontology and the graph as a whole. This includes:

Change knowledge graph - Allowing to transfer and explore any other knowledge graph.

Fullscreen/Exit Fullscreen - For entering and exiting full-screen mode.

Save Exploration - Saves the status of the current exploration.

Open a saved graph - Opens a list of all the saved explorations to choose from. Once chosen the saved exploration will be added to the graph.

Undo/Redo - Undo the graph state to a previous state or Redo the following action.

Graph settings - Opens a window with different graph settings including:

1. **Physics** - This setting can be set to Auto, On or Off, determining whether the nodes on the graph will automatically stop moving based on the timeout chosen right below physics, move constantly or perhaps not at all.
2. **Automatic graph functions** - These functions run automatically on the graph and include the following functions:

> - **None** - Graph Explorer will not run any queries or try to automatically infer any relationships between the nodes in the graph.
> - **Infer similar properties and relationships** - Graph Explorer will automatically add nodes and edges to all nodes that have at least one or more relationships to other nodes in the graph.
> - **Retraverse similar properties and relationships** - Graph Explorer will automatically traverse the same relationships of existing nodes in the graph when introducing new nodes of the same group that exist in the graph.
> - **Retraverse and infer similar properties and relationships** - Perform both automatic functions as described above.

3. **Node Shape** - Choose the shape of the nodes, current options are dot, box, circle, star, ellipse and square.
4. **Node Scaling** - Choosing to scale the node sizes based on different parameters, the current options are:
    - Incoming and outgoing relationships
    - Incoming relationships
    - Outgoing relationships
    - By a specific property
    - Constant (default)
    - Page rank algorithm
    - Between centrality algorithm

5. **Node size** - The node sizes can be manually adjusted using the slider.
6. **Layout** - Choose the layout of the graph from:

1. **Free-form** - The default layout view of the graph.
2. **Hierarchy** - A hierarchy layout view of the graph that when chosen has the following options available:

> **Direction** - The direction of the relationships in the graph should be one of the following options:
>
> - Up -> Down
> - Left -> Right
> - Down -> Up
> - Right -> Left

Sort - The algorithm used to ascertain the levels of the nodes based on the data. Sort contains two options:

> - **Hubsize** - Takes the nodes that have the most relationships and puts them at the top. From that point, the rest of the hierarchy is evaluated.
> - **Directed** - Adhere to the relationships in the data (the edges).

Shake To - Controls what is the source in the directed layout. There are two options:

> - **Roots** - When all the root nodes are lined up at the top and their child nodes are as close to their root parents as possible.
> - **Leaves** - When all the leaves are lined up at the bottom and their parent nodes are as close to their children as possible.

Level Space - The spacing between the node levels on the graph.

Node Space - The spacing between nodes in the graph.

Tree Space - The spacing between trees (a group of connected nodes) in the graph.

Once settings are modified Save must be clicked to save the changes. To return to the graph's default settings, Reset to default should be selected on the top right corner of the graph settings window.

Filter Graph - Enables to filter the graph by relationships, properties and other general attributes, with the ability to show or hide any attribute on the graph.

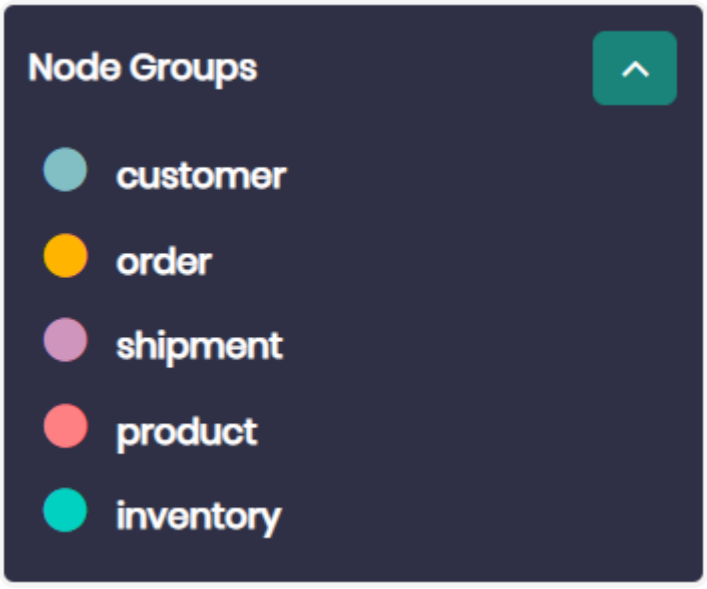
Fetch all groups properties - Adds all the properties to all the node groups on the graph, if they haven't already been added using the query builder earlier.

Find data connections - This enables traversing the graph using relationships and properties.

Add data nodes - Add data points as nodes to the graph like in the initial query screen. Choose a concept, properties, filters and limit to add to the graph.

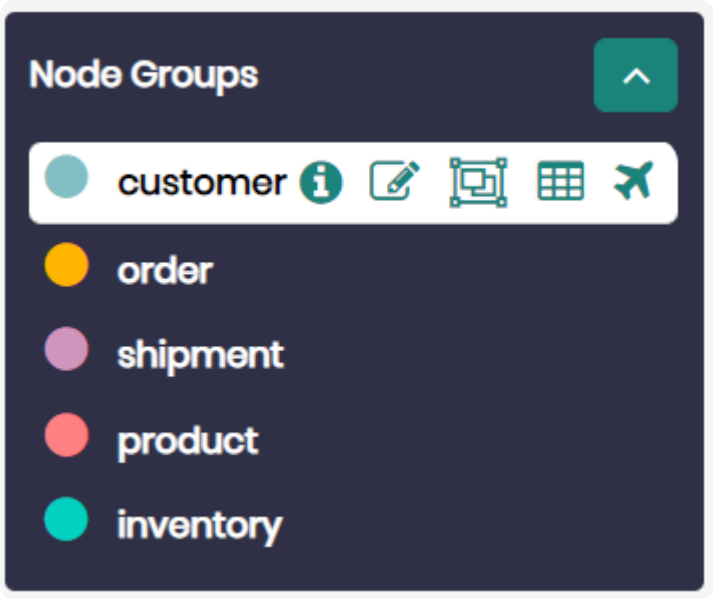
Search data nodes - A text search bar for searching specific data nodes on the graph.

Node Groups Control



The node groups control can be found on the top left of the graph explorer page and contains a list of the node groups (concepts) presented in the graph.

In the node groups control, you can change node group settings, cluster nodes to a group, add properties to concept groups, or traverse the graph using relationships and properties from a chosen node group.



When hovering over a concepts node group the following options will reveal themselves:

Information - displays the description of the concept group that was defined when creating the concept.

Node Group Settings - Opens a window with different node group settings including:

1. **Node Label** - This allows you to change and choose the label you would like to have representing your node group's data points.
2. **Node background color** - Choosing the background color of the nodes in the selected group.
3. **Node border color** - Choosing the border color of the nodes in the selected group.
4. **Node label color** - Choosing the label color for the nodes in the selected group.
5. **Node Shape** - Choosing the shape of the nodes in the chosen group, current options are dot, box, circle, star, ellipse and square.
6. **Node Scaling** - Choosing to scale the group node sizes based on different parameters, the current options are:
    - Incoming and outgoing relationships
    - Incoming relationships
    - Outgoing relationships
    - By a specific property
    - Constant (default)
    - Page rank algorithm
    - Between centrality algorithm

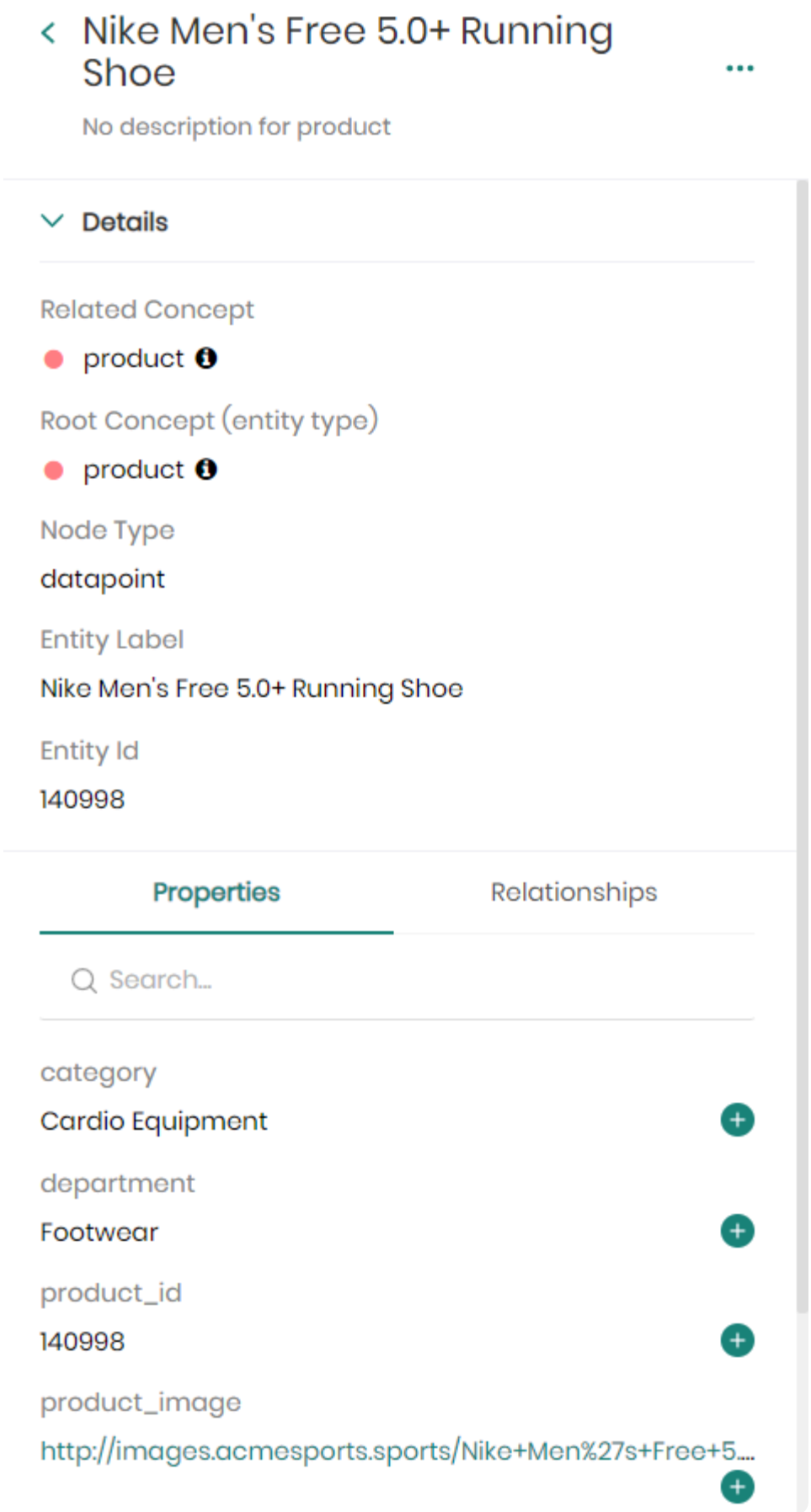
Once the node group settings are modified Apply must be clicked to apply the changes. To return to the default settings, Reset to default should be selected on the bottom of the node group settings window.

Cluster Nodes - Cluster a group of nodes into a single node, and then count the number of nodes in that group. To un-cluster the nodes, click on the cluster.

Fetch all group properties - Adds all the properties to the data points of the selected node group, if they haven't already been added using the query builder earlier.

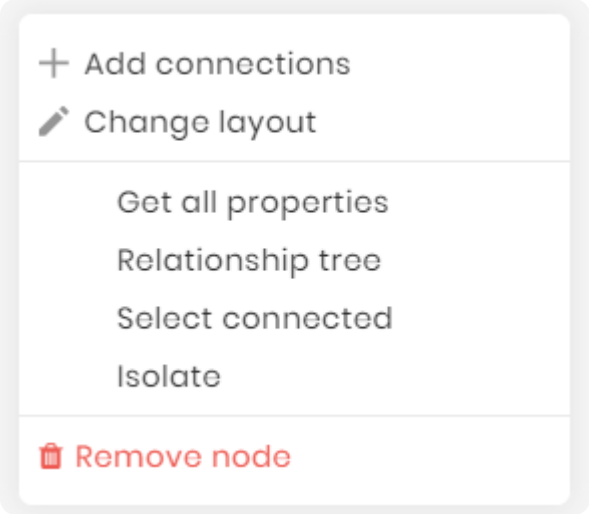
Find data connections - This enables traversing the graph using relationships and properties of the selected node group.

Node Properties


Nike Men's Free 5.0+ Running Shoe
No description for product
Details
Related Concept
product
Root Concept (entity type)
product
Node Type
datapoint
Entity Label
Nike Men's Free 5.0+ Running Shoe
Entity Id
140998
Properties
Relationships
Search...
category
Cardio Equipment
department
Footwear
product_id
140998
product_image
http://images.acmesports.sports/Nike+Men%27s+Free+5....

A right-side panel that opens when you click on a node in the graph. Provides additional information about a node (datapoint).

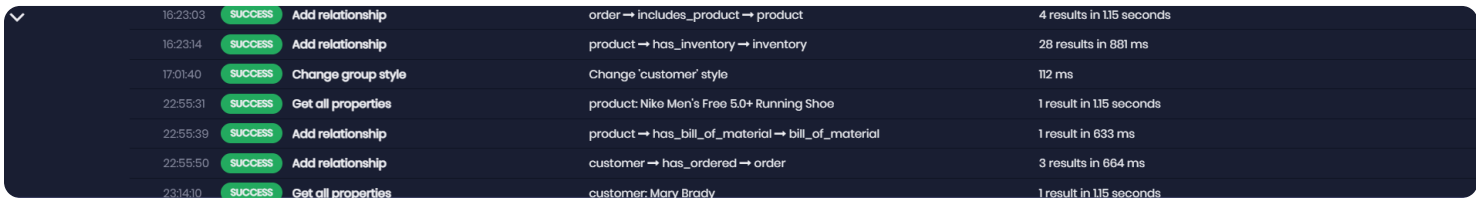
Node Action Menu



A list of actions that can be performed on a single node. This can be accessed by right-clicking on a node in the graph. The actions include:

Add connections - Allows to add connected relationships and properties to the selected node. **Change layout** - The ability to change or edit the selected node's label name, color and node background color. **Get all properties** - Adds all the properties to the selected data point if they haven't already been added using the query builder earlier. **Relationship tree** - Opens a tree view of the edges (relationships) of the selected node in the node properties side panel on the right. **Select connected** - Highlights all the connected nodes of the selected node. **Isolate** - Allows isolating a single node or all its connected nodes leaving only them on the graph. **Remove node** - Removes the selected node or all its connected nodes on the graph.

Query Action Log



A log of actions performed during the exploration. It can be accessed on the bottom side of the screen.

The log enables users to see the timestamp, status, action name and runtime of each action performed on the graph. In addition, the log offers the following options:

Copy query - The ability to view and copy any query behind any action performed on the graph.

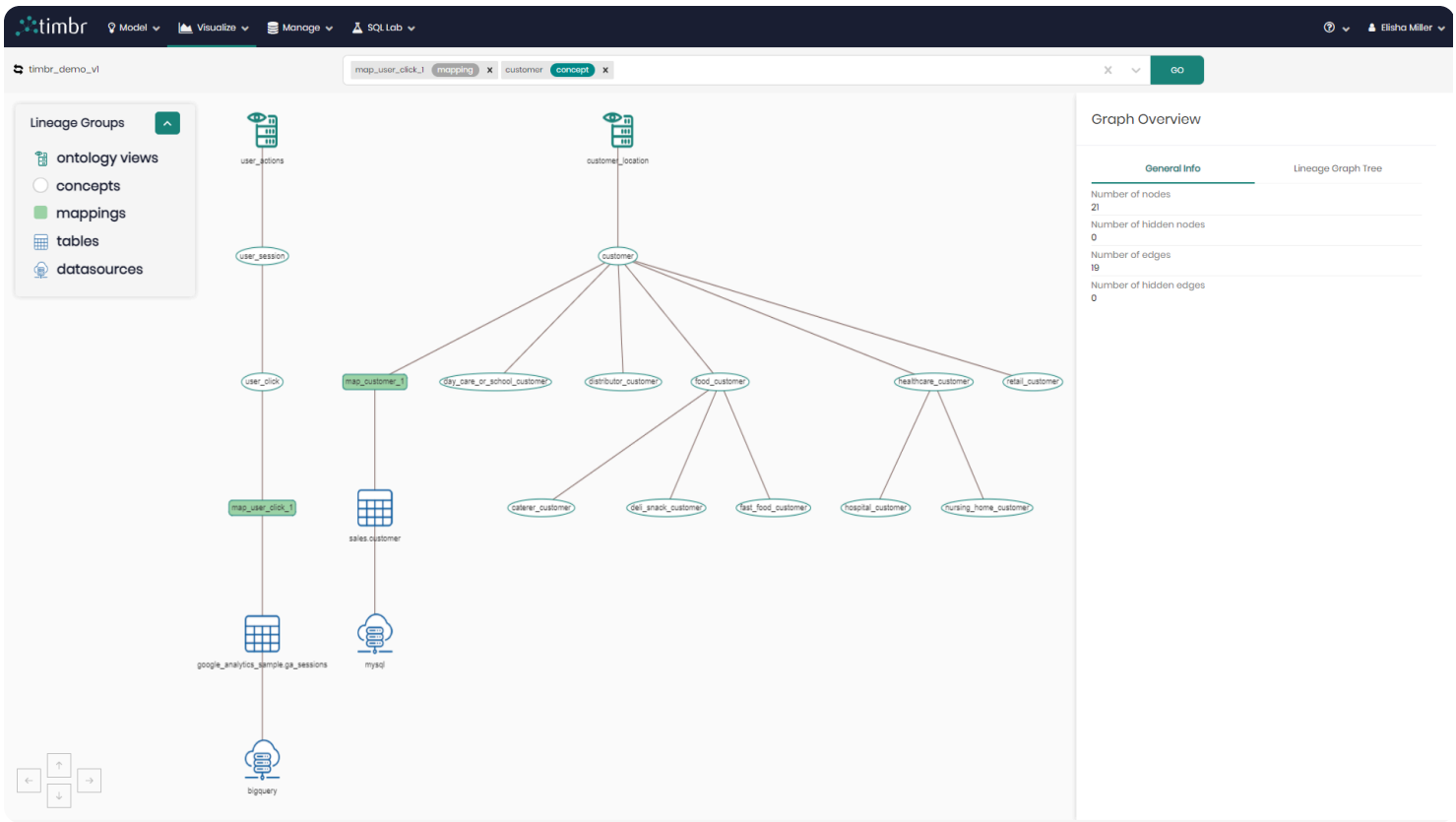
Delete log - Deletes any action from the action log.

Download JSON log - Downloads the chosen action as a JSON file.

Knowledge Lineage

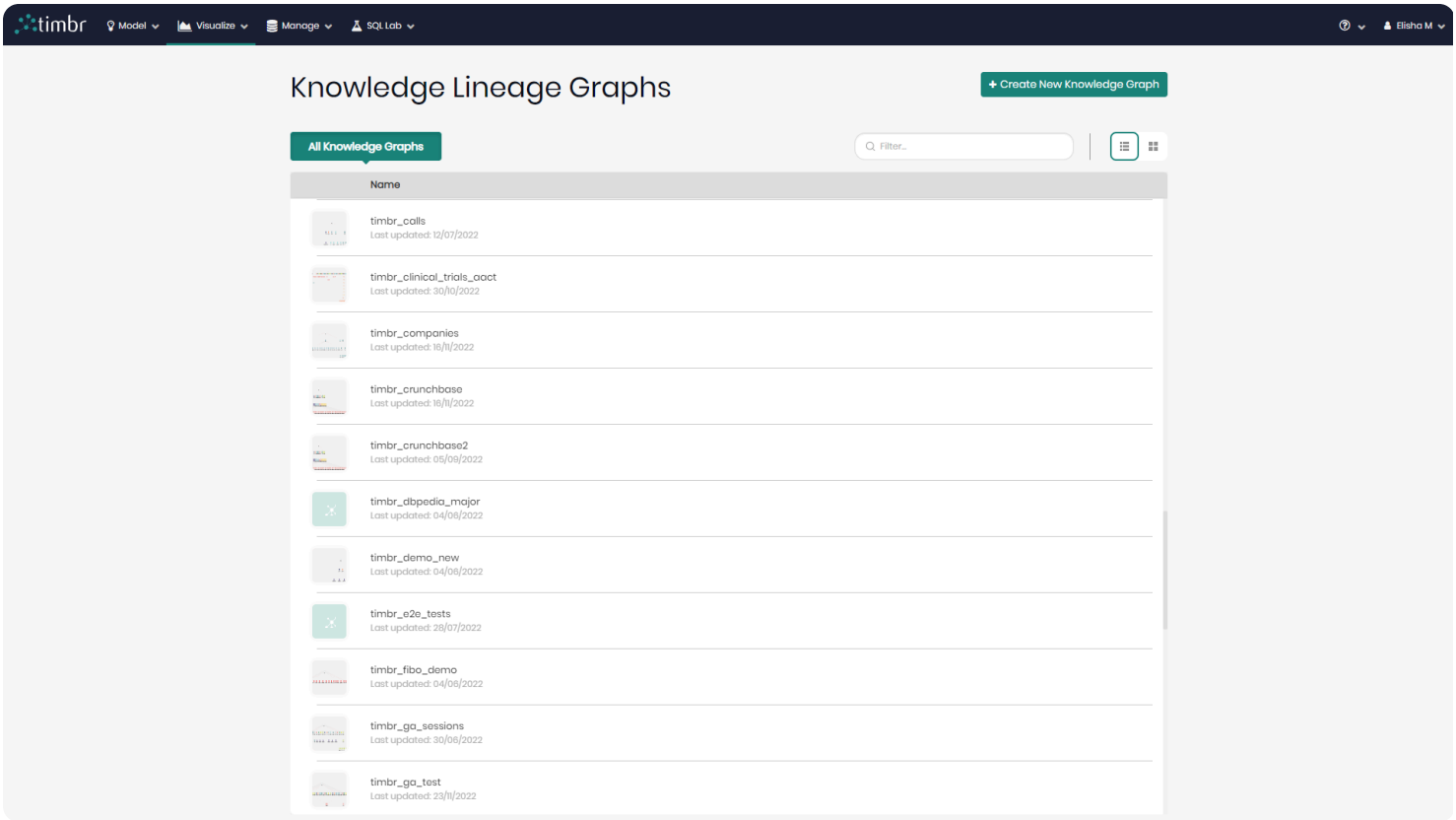
Timbr's Knowledge Lineage page enables users to easily understand data dependencies between the physical data sources and the virtual data model assets. Users can efficiently perform impact analysis before changing or removing data assets. The Knowledge Lineage component also helps data consumers trust the data and understand where it comes from, what information was integrated from each source, and how the data was used to define the business entities in the semantic knowledge graph.

Knowledge Lineage Page



The Knowledge Lineage can be accessed by clicking on the **Visualize** tab and selecting **Knowledge Lineage**.

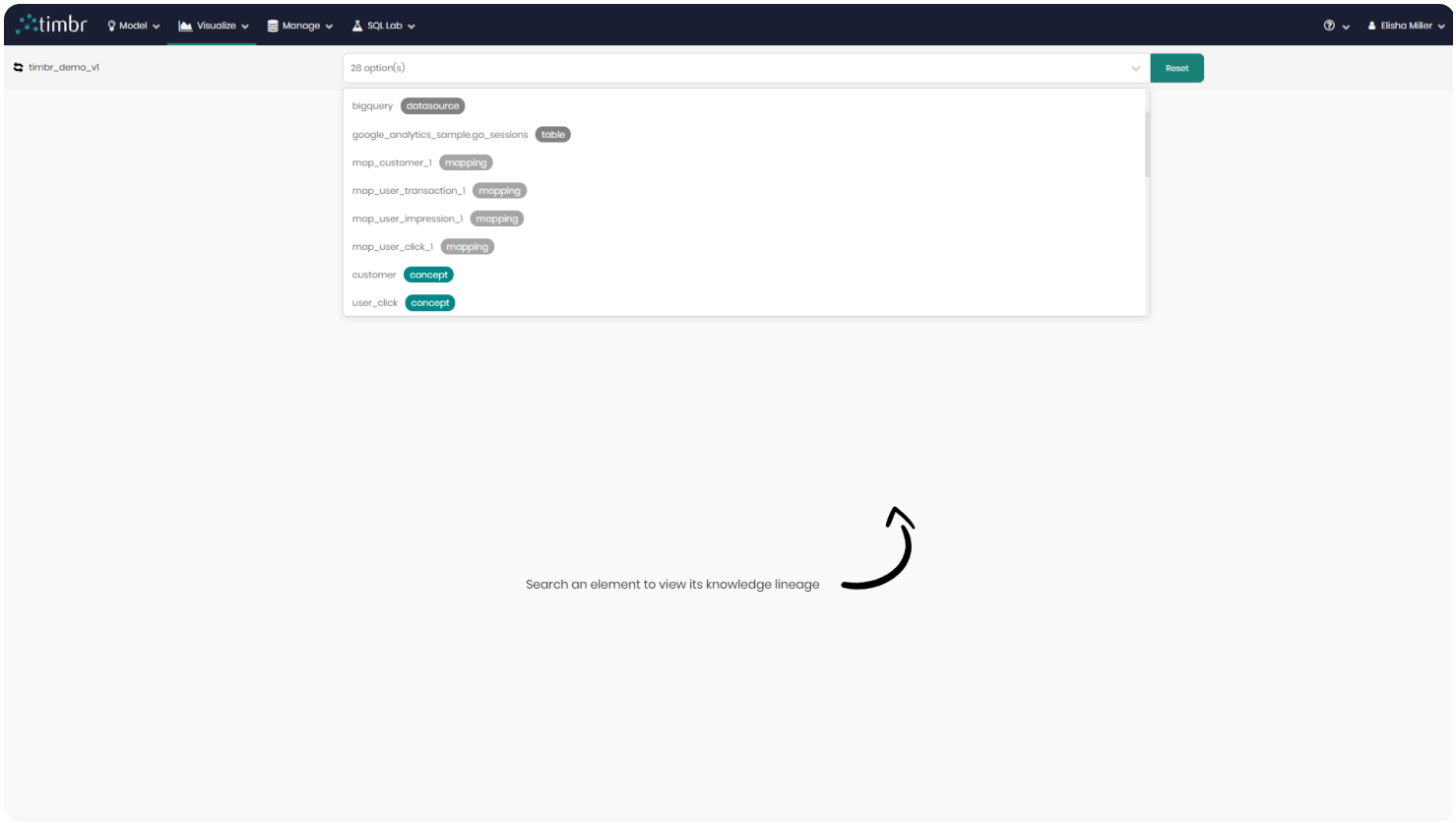
Once Knowledge Lineage is clicked on in the *Visualize* tab you will be asked to select an existing knowledge graph you wish to load and view. Another option on the top right is to click on *Create New Knowledge Graph* in order to create a new knowledge graph to load and view.



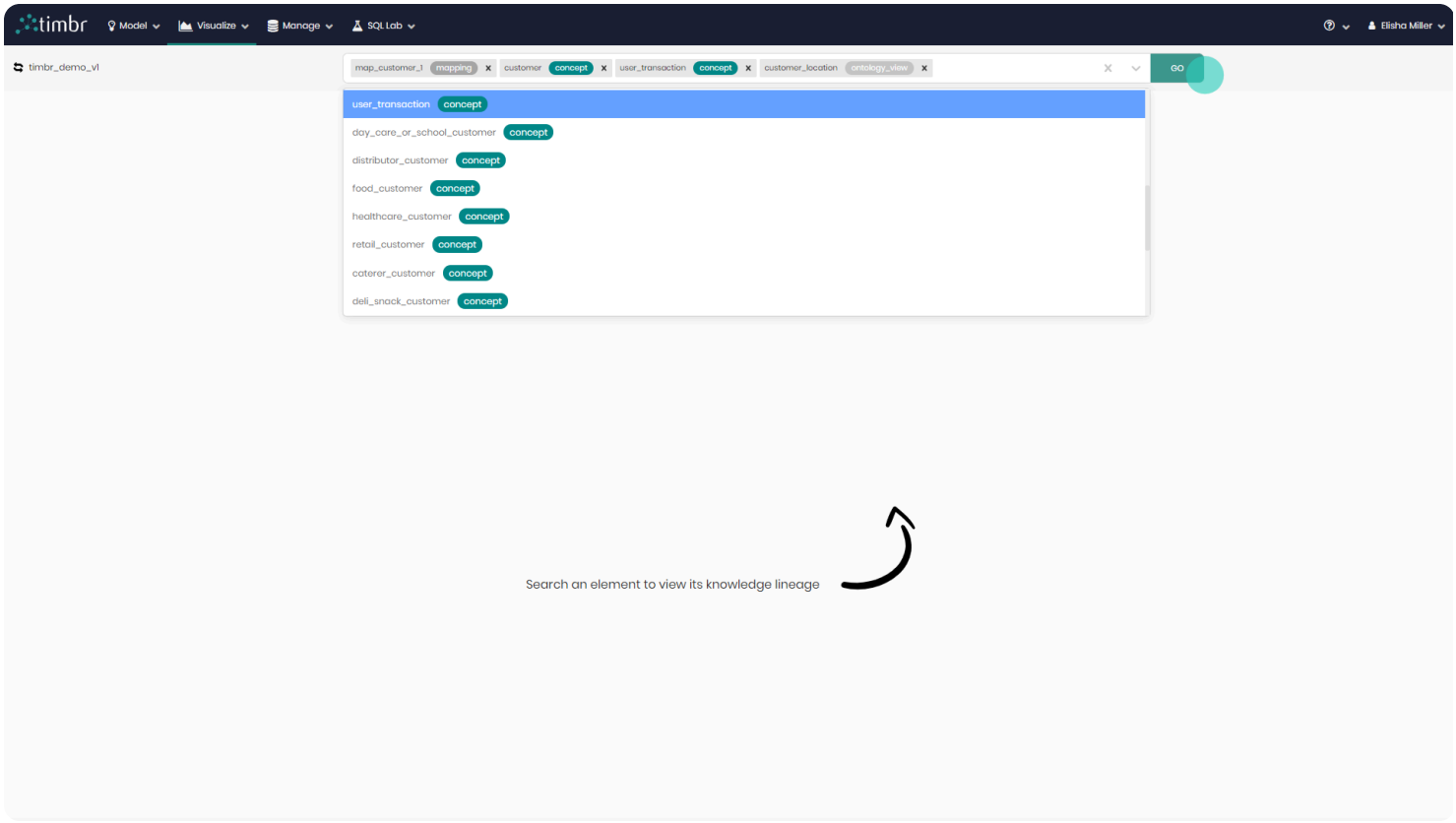
The starting point in the Knowledge Lineage after choosing a Knowledge Graph, is to select an element or multiple elements from the dropdown on the upper part of the screen in order to view them on the graph.

The dropdown includes the following elements:

- **Datasources** - Datasources that are connected to the Knowledge Graph.
- **Tables** - Tables from the different datasources connected to the Knowledge Graph.
- **Mappings** - Data Mappings in the Knowledge Graph.
- **Concepts** - Business Concepts of the Knowledge Graph.
- **Ontology Views** - Views associated with the Knowledge Graph.



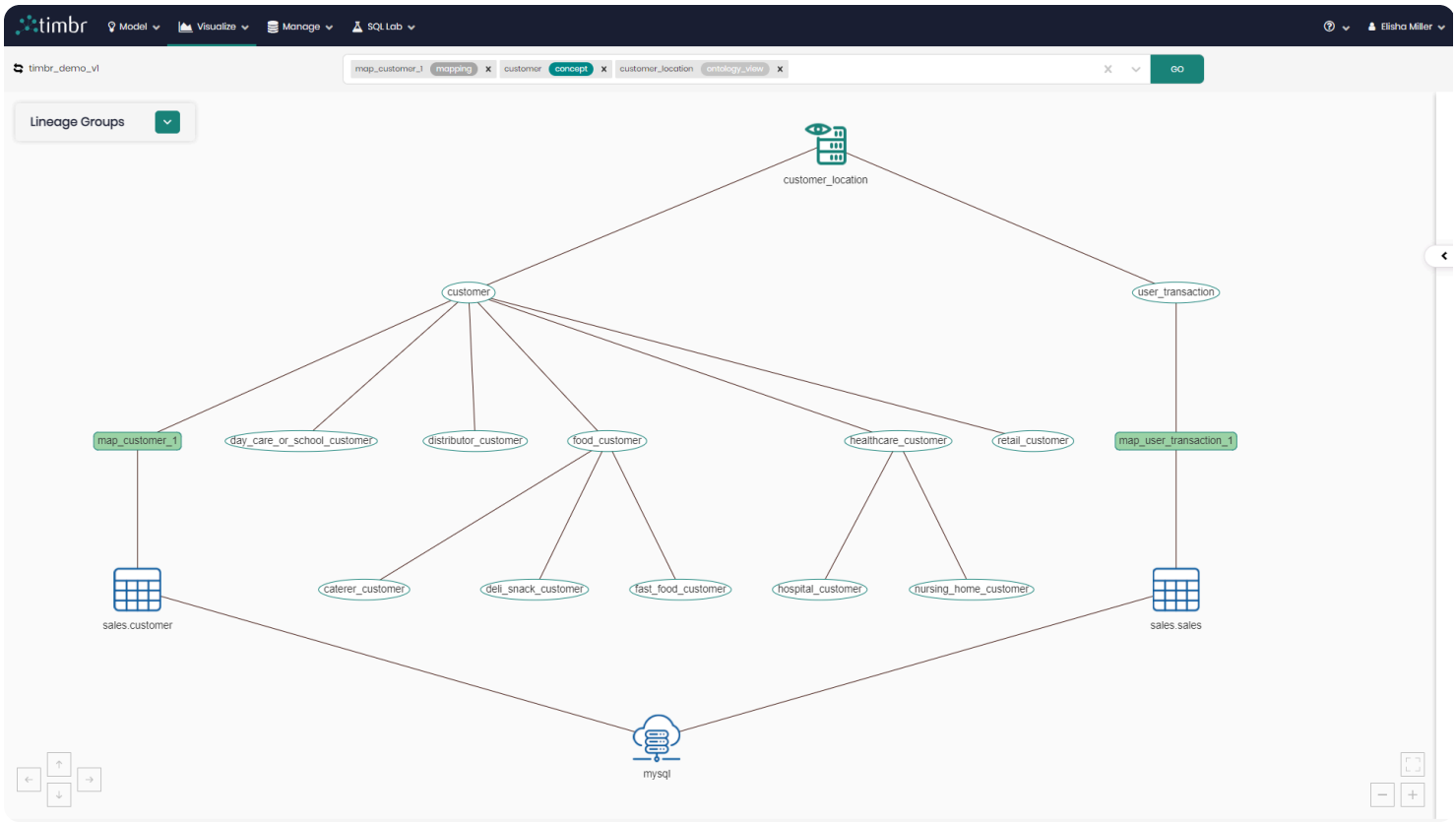
Once the elements are selected **GO** must be clicked in order to present the elements on the graph. At any point, users can click on the arrows on the top left of the screen and change to any other knowledge graph from the list that pops up.



After clicking *GO*, the selected elements should appear on the graph, each element will appear with its name and type which can be found in the *Lineage Group Menu* on the top left.

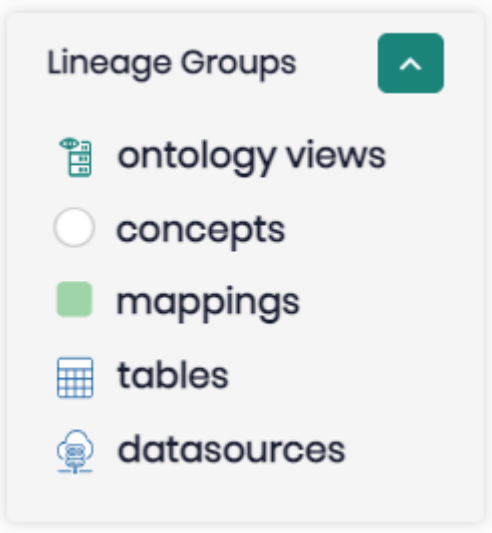
The components of the Knowledge Lineage Graph are as follows:

Main Graph View



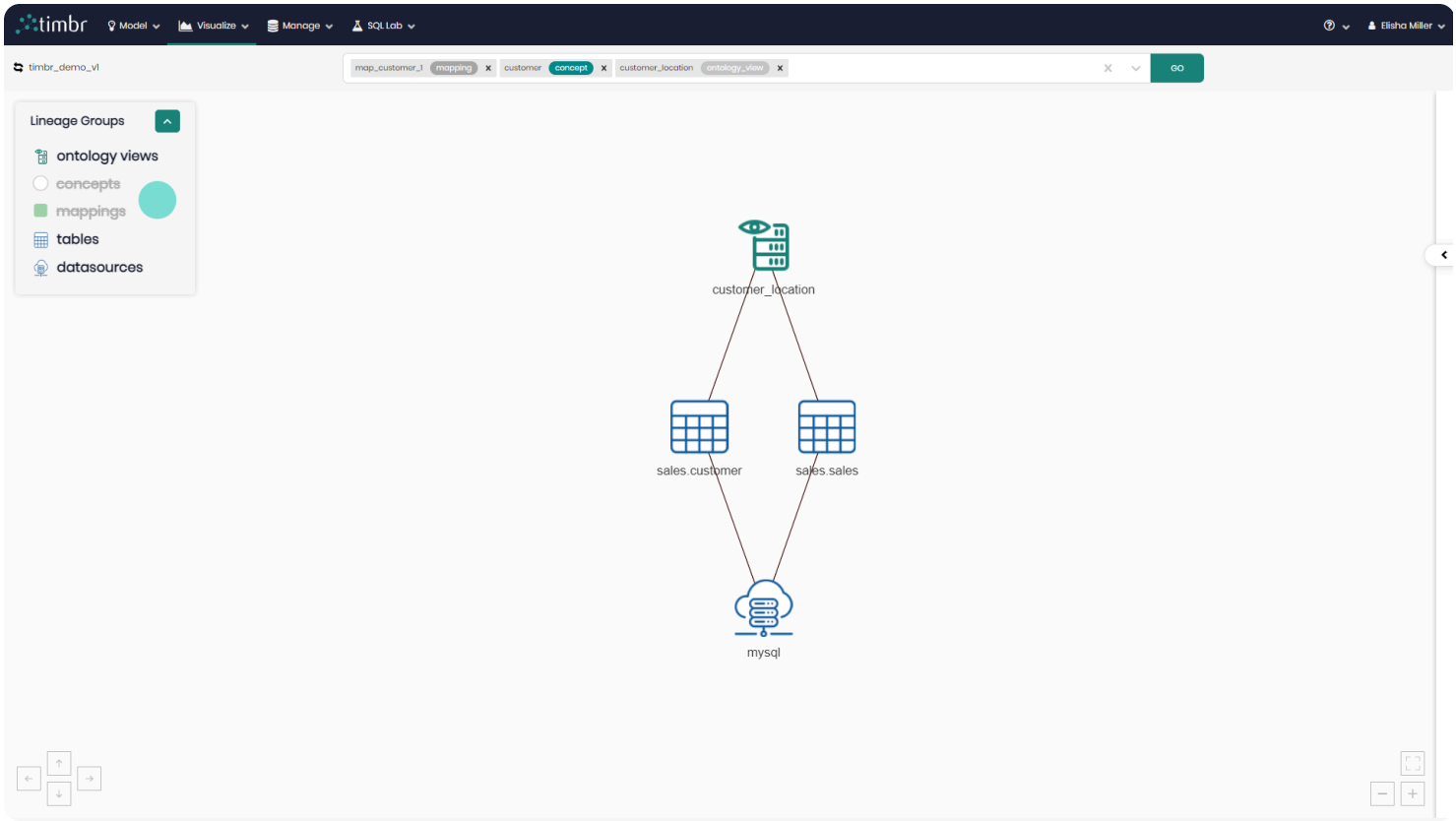
An interactive graph representing the selected elements and their connections. You can zoom-in and zoom-out of the graph view by using the mouse wheel or the + and - buttons on the bottom right, as well as drag the graph and its elements by holding down the mouse or by using the arrows on the bottom left of the screen.

Lineage Group Menu

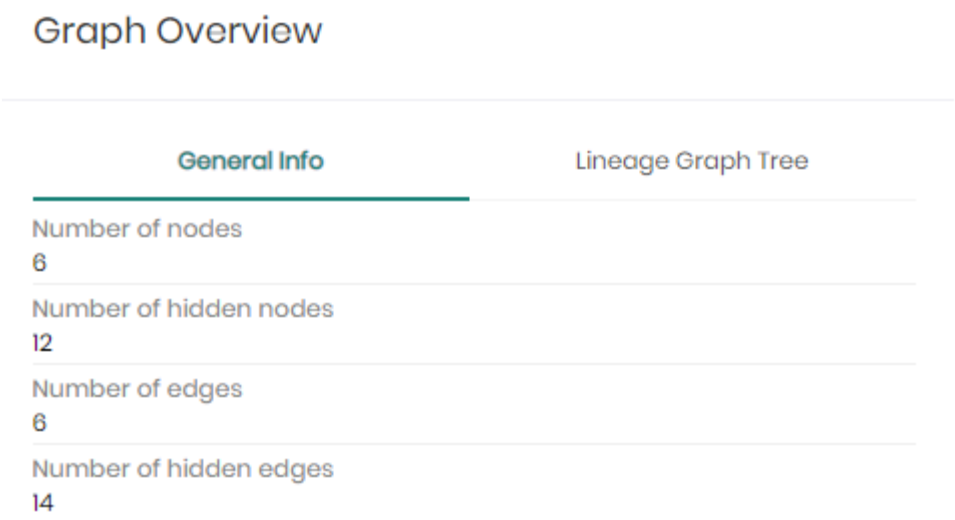


The Lineage Group Menu can be found on the top left of the Knowledge Lineage Page and contains a list of the chosen and presented elements in the graph.

Each element group is represented by its own icon in order to differentiate it from the rest of the elements on the graph. Any element group can be clicked on in order to hide the selected elements from the graph, and when clicked on again the elements of the group will return to the graph.



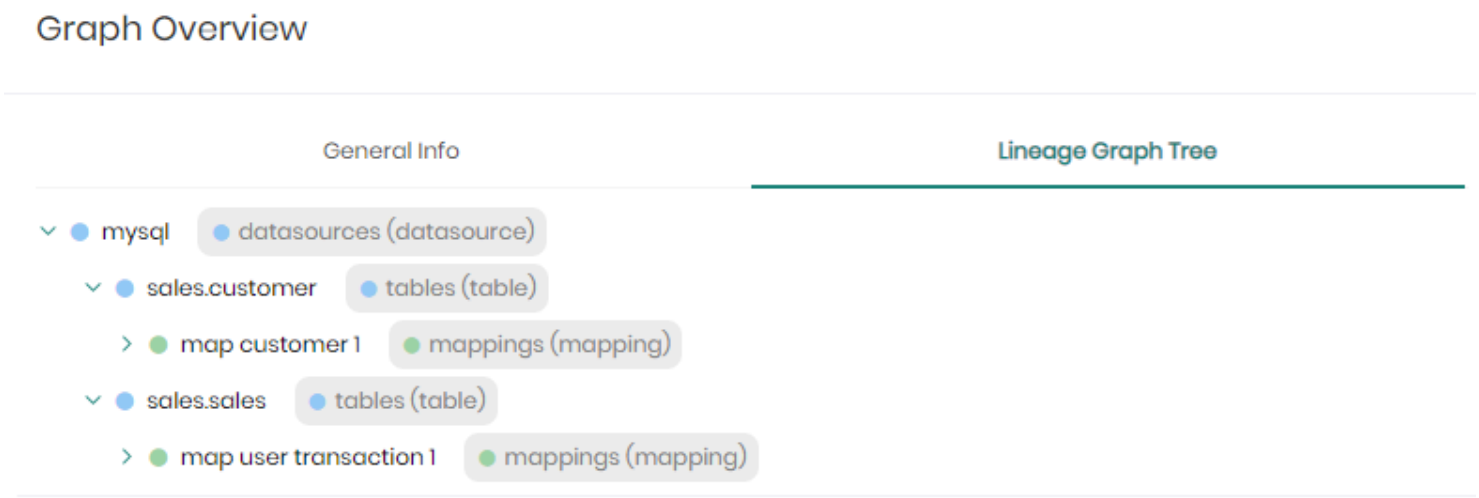
Graph Overview



A right-side panel that provides an overview of the graph and its elements. The Graph Overview contains the following two tabs:

General Info - States the *Number of nodes* that represent the elements, as well as the *Number of hidden nodes*, *Number of edges* and *Number of hidden edges*.

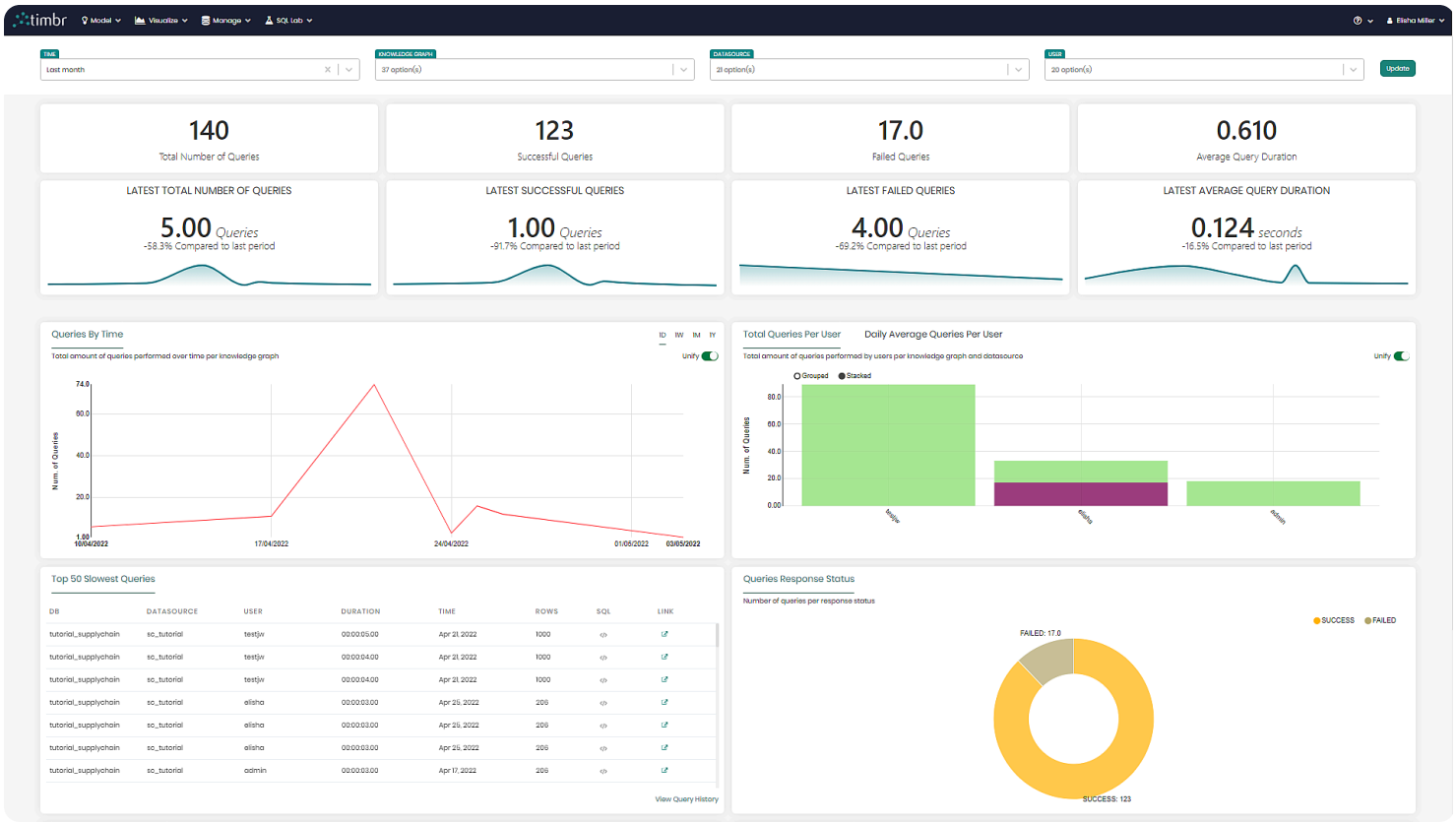
Lineage Graph Tree - Presents the elements on the graph as a tree containing detailed hierarchies.



At any point elements can be added or removed from the drop-down above, Once the changes are made and *GO* is clicked on, the graph will rearrange itself presenting the new chosen elements.

Performance Dashboard

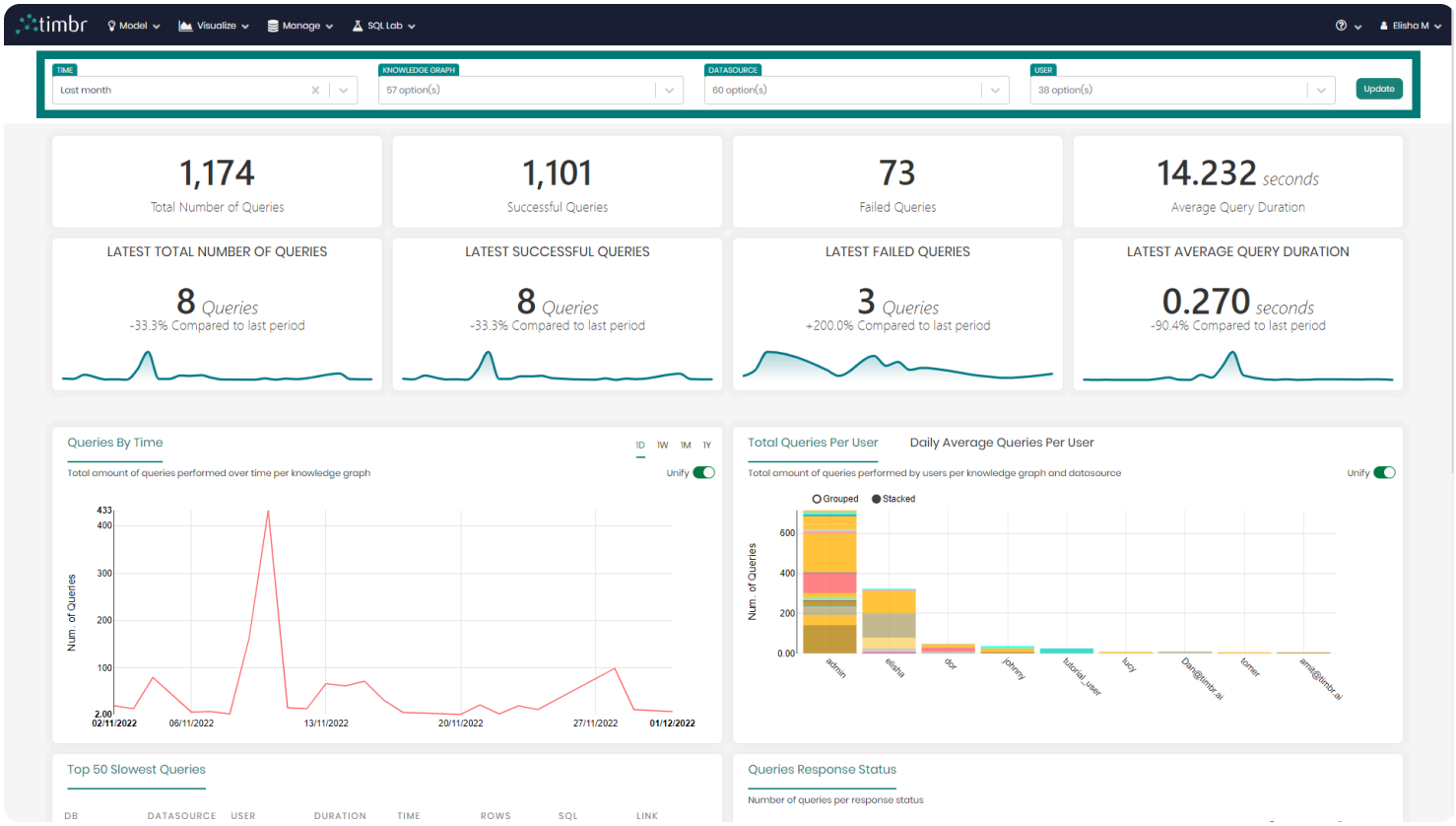
The Performance Dashboard page is a built-in interactive auditing tool that helps users get a clear picture of the usage and performance of the platform knowledge graphs. The dashboard monitors the entire query activity in the platform and includes predefined charts to highlight slow queries, high workloads, general activity, and timeframe comparisons. Users can filter and use the Performance Dashboard to analyze the data consumption and gain a better understanding of how data is used and if there are any issues to address.



Performance Dashboard components

The Performance Dashboard can be accessed through the **Visualize** tab and contains the following components:

Dashboard Filters



The dashboard filters above the different charts and metrics, enable users to drill down and get a deeper analysis of their knowledge graphs.

The current four filters include:

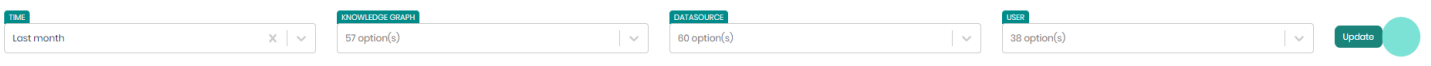
Time - Enables users to choose the time frame of the charts presented in the dashboard.

Knowledge Graph - Allows users to choose the specific knowledge graphs they want to analyze on the dashboard.

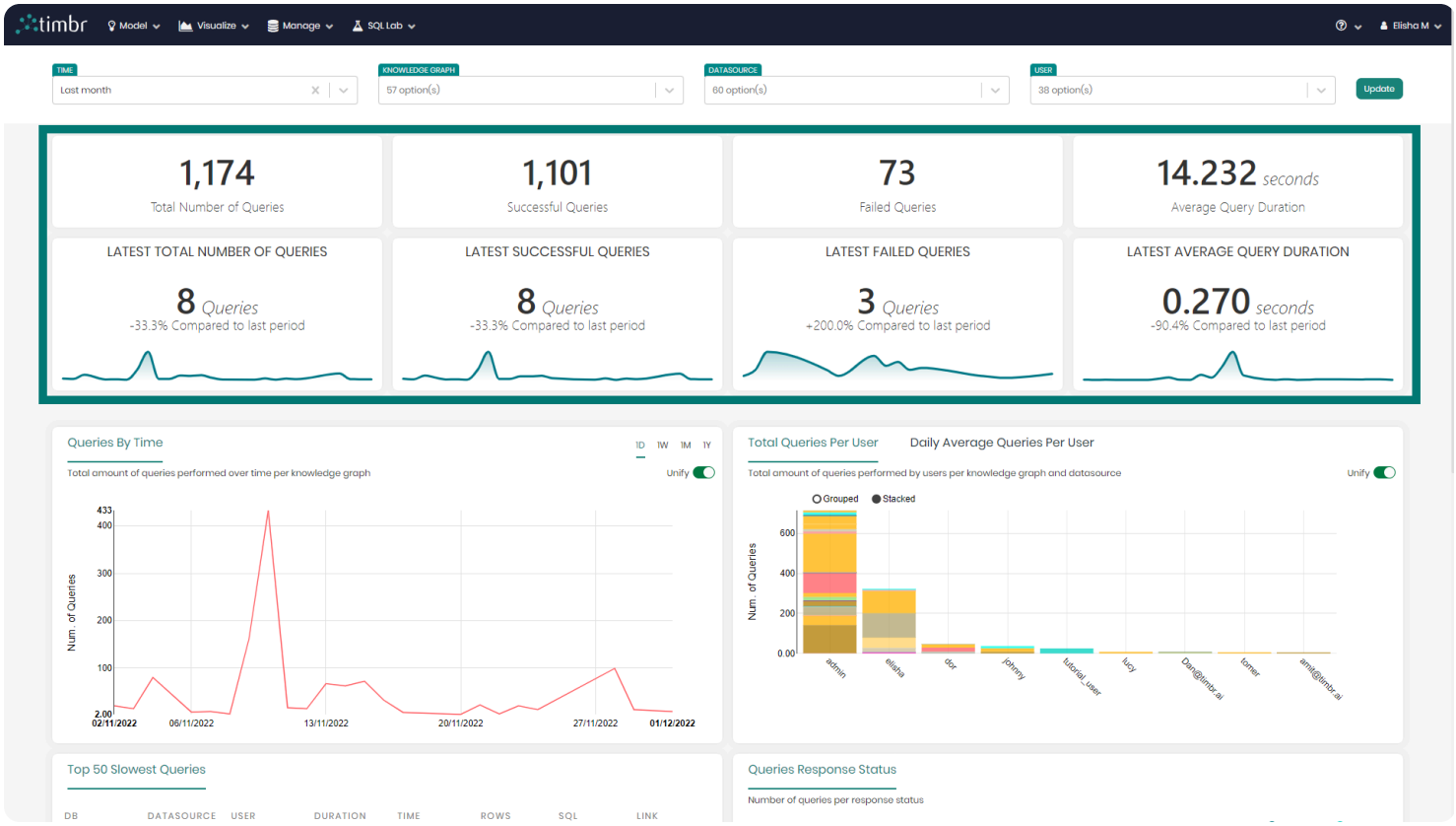
Datasources - Enables users to present data on the dashboard coming only from the selected datasources.

User - Presents data on the dashboard only for the chosen users.

> **IMPORTANT**
>
> Once changes are made in the filters, **Update** must be clicked in order for the filters to take effect, updating the dashboard metrics and charts.



Metrics



Below the filters are the eight metrics that include the following:

Total Number of Queries - Presents the total number of queries ran based on the chosen filters.

Successful Queries - States the number of successful queries ran out of the total.

Failed Queries - States the number of queries that failed while running.

Average Query Duration - Presents the average time it took to run the queries.

Latest Total Number of Queries - Shows the latest period (depending on the applied filters) total number of queries

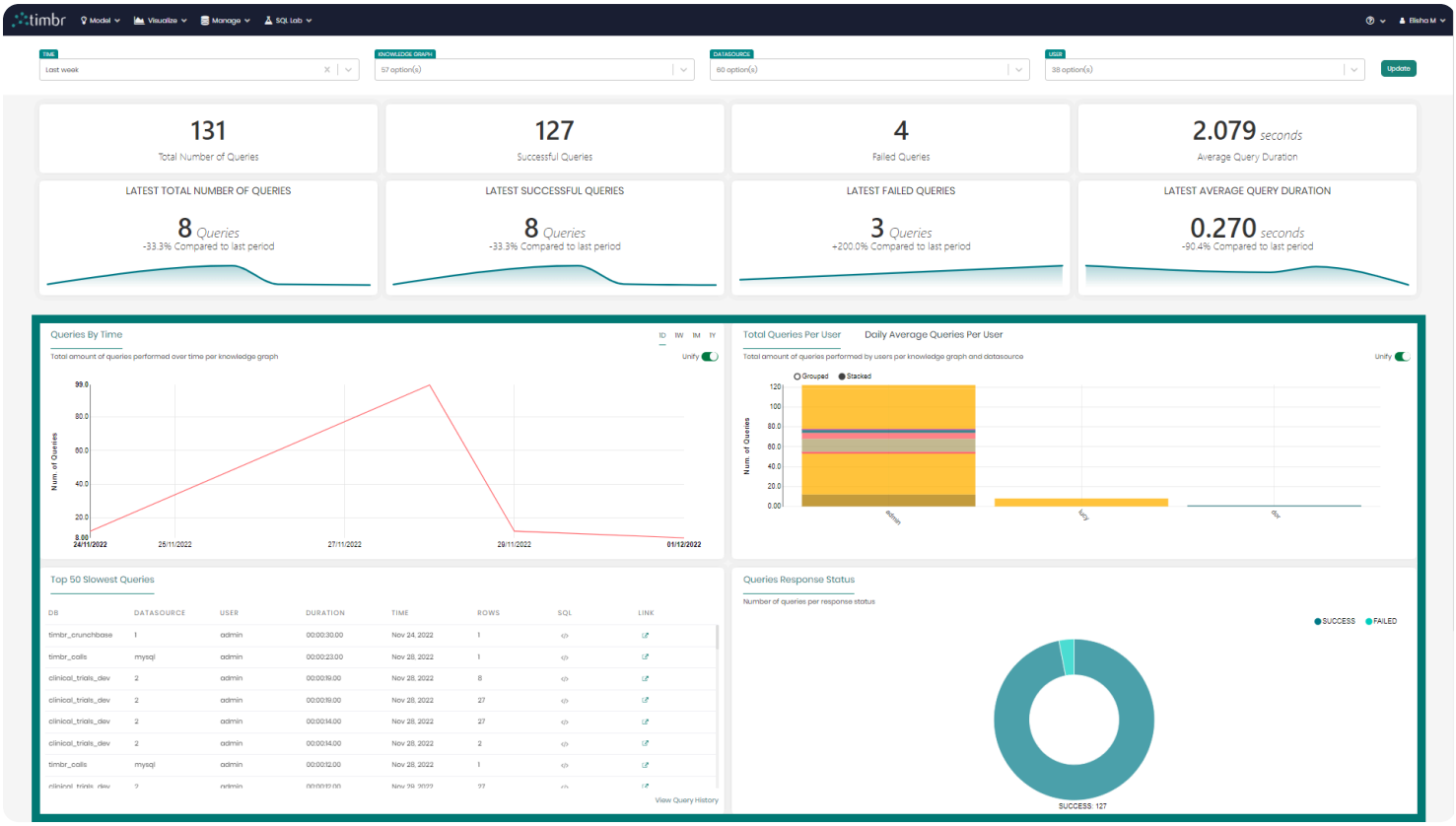
Latest Successful Queries - Shows the latest period (depending on the applied filters) of successful queries

Latest Failed Queries - Shows the latest period (depending on the applied filters) of failed queries

Latest Average Query Duration - Shows the average query duration of the latest period (depending on the applied filters)

Below the eight metrics are the main charts.

Main Charts



There are currently six main charts created using different visualizations, which include:

Queries By Time - A line chart presenting the number of queries over the selected time frame.

Total Queries Per User / Daily Average Queries Per User - A bar chart with the total and daily average number of queries per user.

Top 50 Slowest Queries - A table with various information regarding the top 50 slowest queries that were run.

Queries Response Status - A Donut chart presenting the query statuses by successful vs. failed.

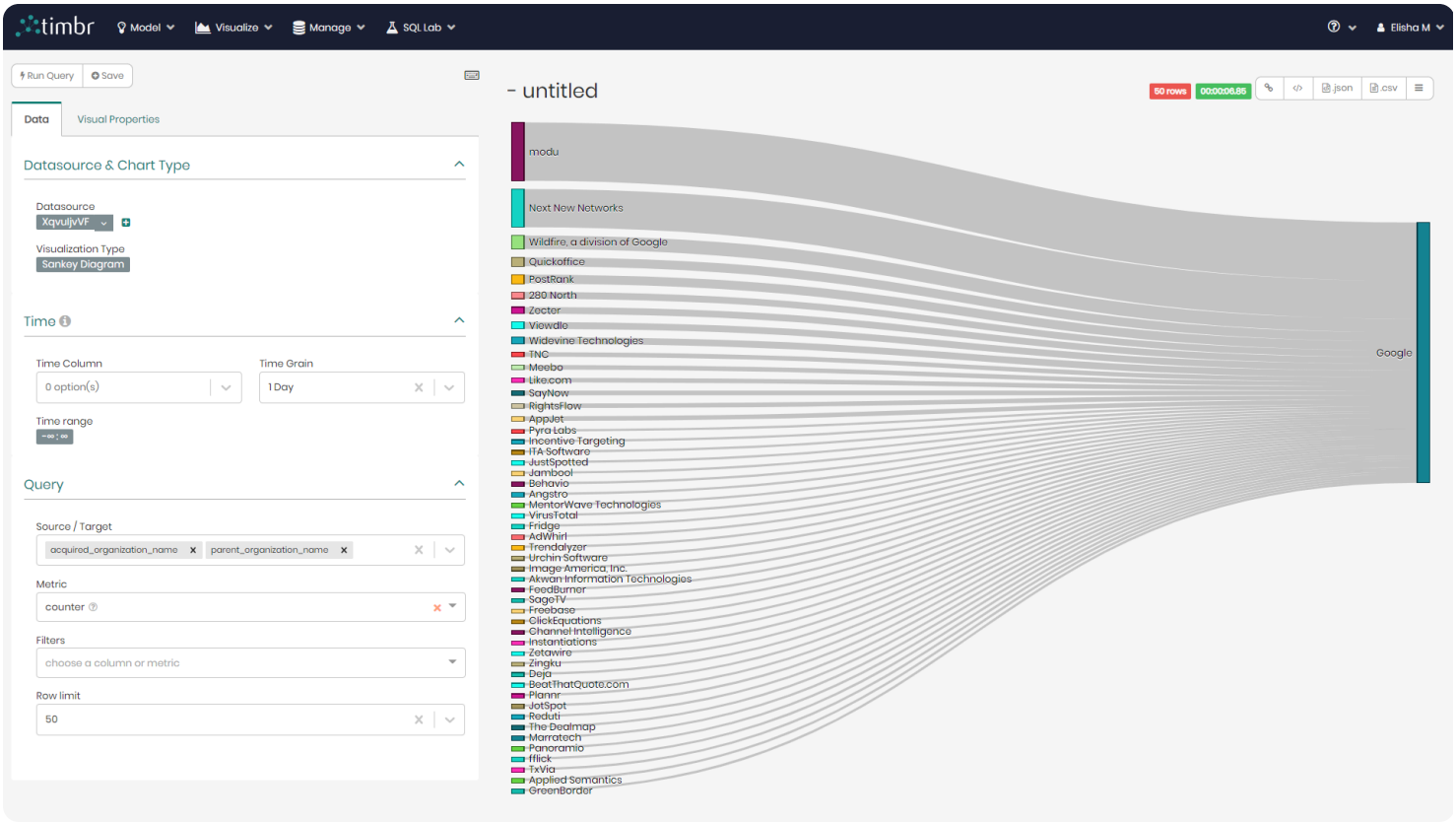
Queries Average Duration - A bar chart presenting the query's average running time based on the selected time frame.

Query Duration Statistics - A candlestick chart showing the distribution of query duration and skewness per knowledge graph and datasource.

> **IMPORTANT**
>
> The different metrics and charts can be interacted with by simply hovering over and clicking on the desired metric or chart.

BI Charts

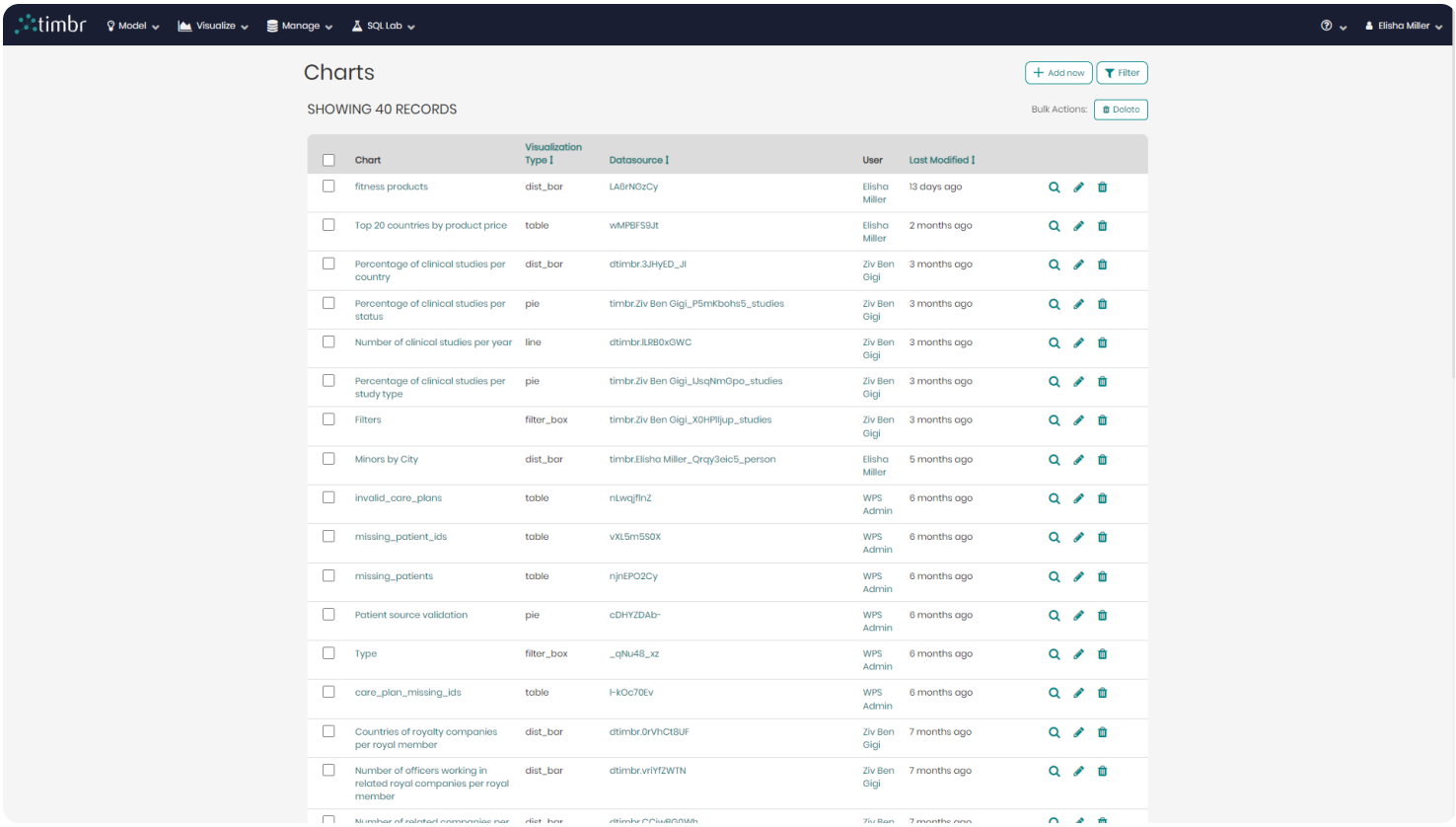
The Charts page in Timbr is part of an embedded BI component that allows users to create visualizations using various charts and share them with others. Charts can be based on the physical tables, semantic models, or Timbr system tables to analyze data or monitor the activity in the Timbr platform. The charts can also be added to interactive Dashboards.



Getting started

The charts component in charge of creating and editing knowledge graph charts, can be accessed through the **Visualize** tab as well as the *Ontology Explorer* and *SQL Editor* when wanting to visualize a specific concept or query.

When entering through the *Visualize* tab, a list of all the previously created charts will appear.

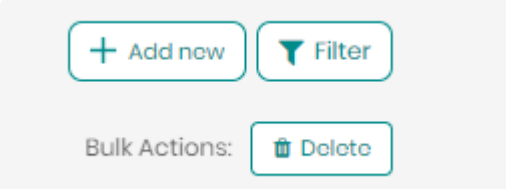


Each chart name will appear with the visualization type, datasource, user and last modified date appearing to its right. Further down the line users have the following three buttons:

- *Magnifying glass* - Opens additional information about the selected chart.
- *Pencil* - Opens an edit chart window to edit the different specifications and parameters of the selected chart.
- *Trash can* - Deletes the selected chart.

Above the three buttons are the following options:

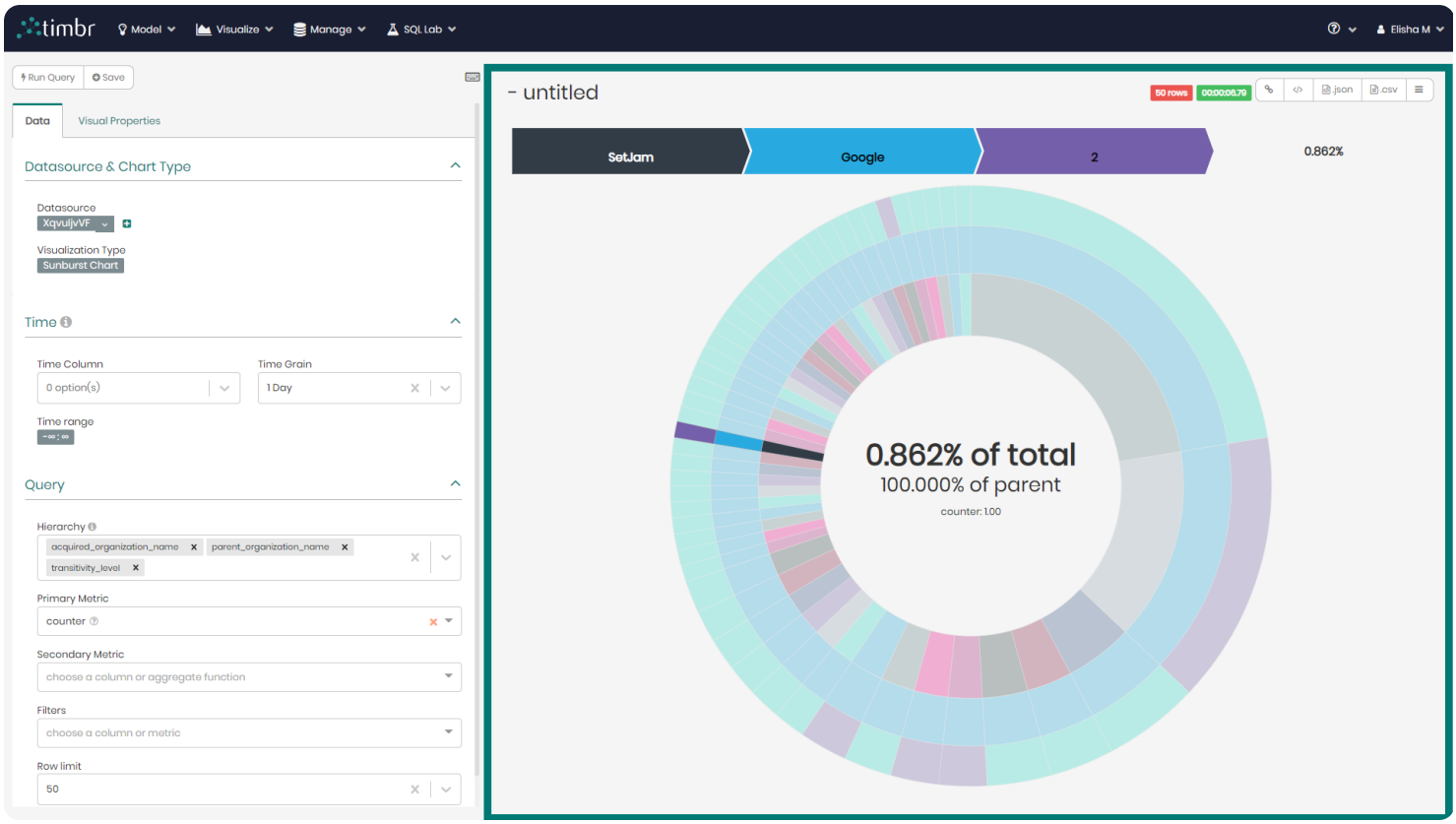
- *Add new* - Opens a window to create a new chart.
- *Filter* - Enables users to filter through the list of charts in order to locate them faster and more conveniently.
- *Delete* - A bulk action that deletes all the highlighted charts selected by the user.



BI Charts Components

Once a chart is selected or created from scratch the main chart creation and editing page will appear containing the following components:

Main Chart Box



The main chart box is the space where the created charts are represented and interacted with. Charts can be hovered over and clicked on to create desired viewing filters.

On the top left portion of the main chart box, users can give a name to their current chart.

On the top right portion of the main chart box, users can find the following options:

Row counter - Shows the number of rows used to present the current chart.

Query timer - A timer showing the amount of time it took to run the query that's representing the chart.

Share - Generates a URL link that can be copied or shared with anyone via email.

Iframe edit - Enables users to edit the iframe code as well as change the height and width representing the chart.

Export to JSON - Exports the current chart as a JSON file.

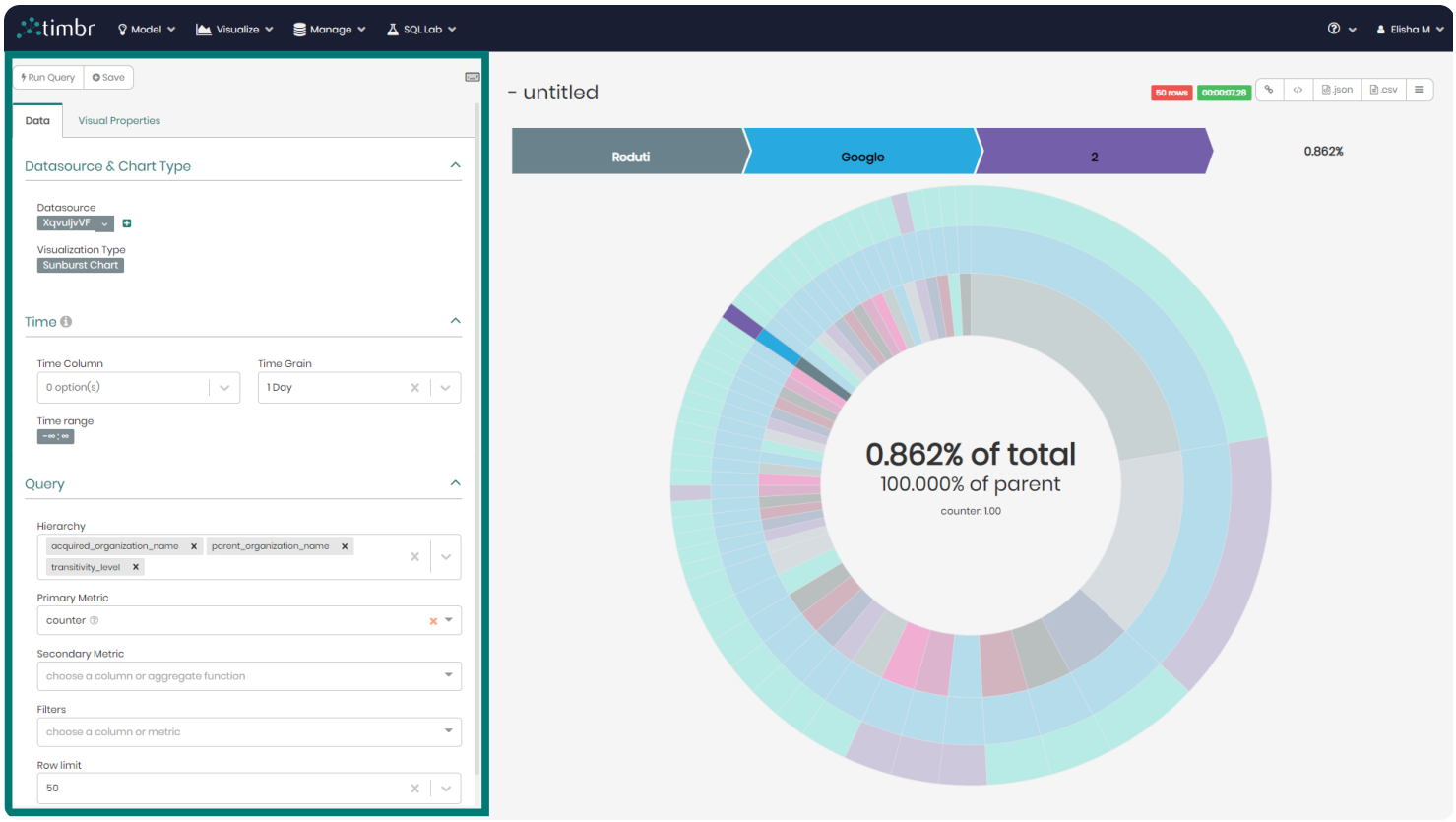
Export to CSV - Exports the current chart as a CSV file.

Export to CSV - Exports the current chart as a CSV file.

More options - additional options which include:

- **View query** - Presents the SQL query behind the current chart.
- **View results** - Presents the current charts results in a table format.
- **View samples** - Presents additional sample results in a table format related to the current chart.
- **Run in SQL Lab** - Runs the SQL behind the current chart in Timbr's SQL lab.

Chart Query Builder Panel



The chart query builder panel is where users edit the metrics and data behind what will be presented in the charts that will be viewed to the right in the main chart box.

On the top right of the panel, users can find keyboard shortcuts to help run and save the charts in a faster manner.

On the top left of the panel there are two buttons which are:

Run query - Runs the query that builds the chart once we've entered the desired metrics.

Save - Naming the chart and saving it with the following options:

- **Do not add to dashboard** - Users can save the chart which can later be found under the saved charts, without adding it to a dedicated dashboard.
- **Add chart to existing dashboard** - Saves the chart and adds it to an existing dashboard that can be selected from the dropdown menu.
- **Add to new dashboard** - Saves the chart and adds it to a new dashboard with a new given name.

Beneath Run query and save, there are two tabs which are **Data** and **Visual properties**.

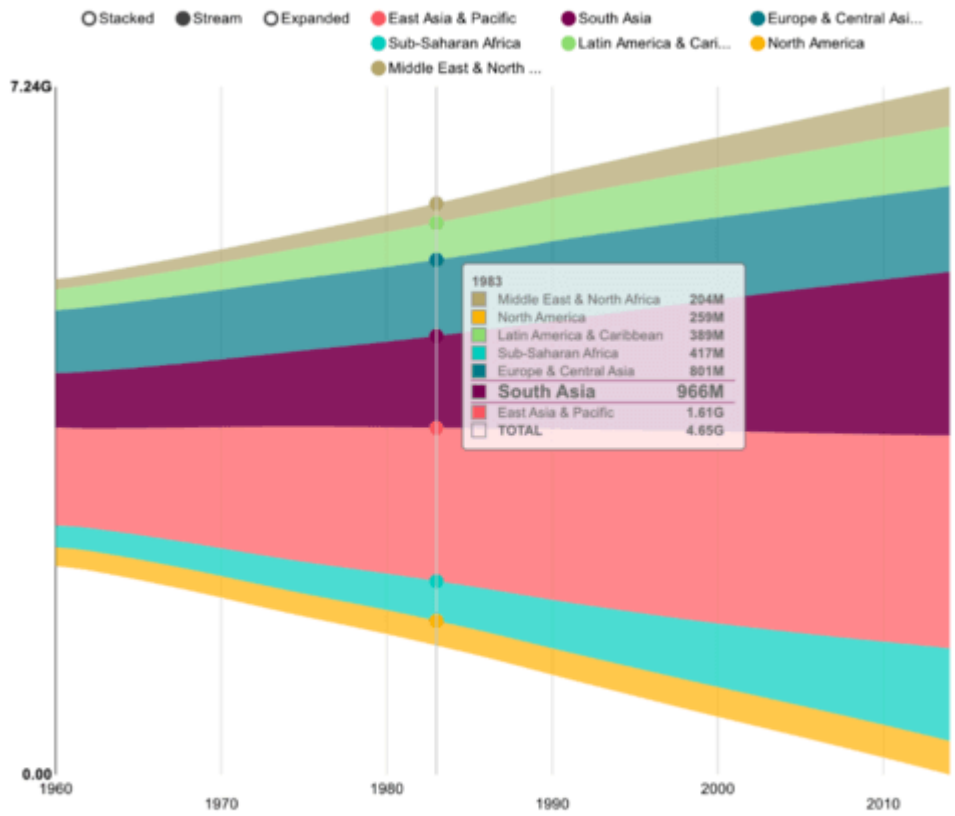
The Data tab is where all the metrics and edits are made on the data that create the charts and depending on the chart type can contain the following sections:

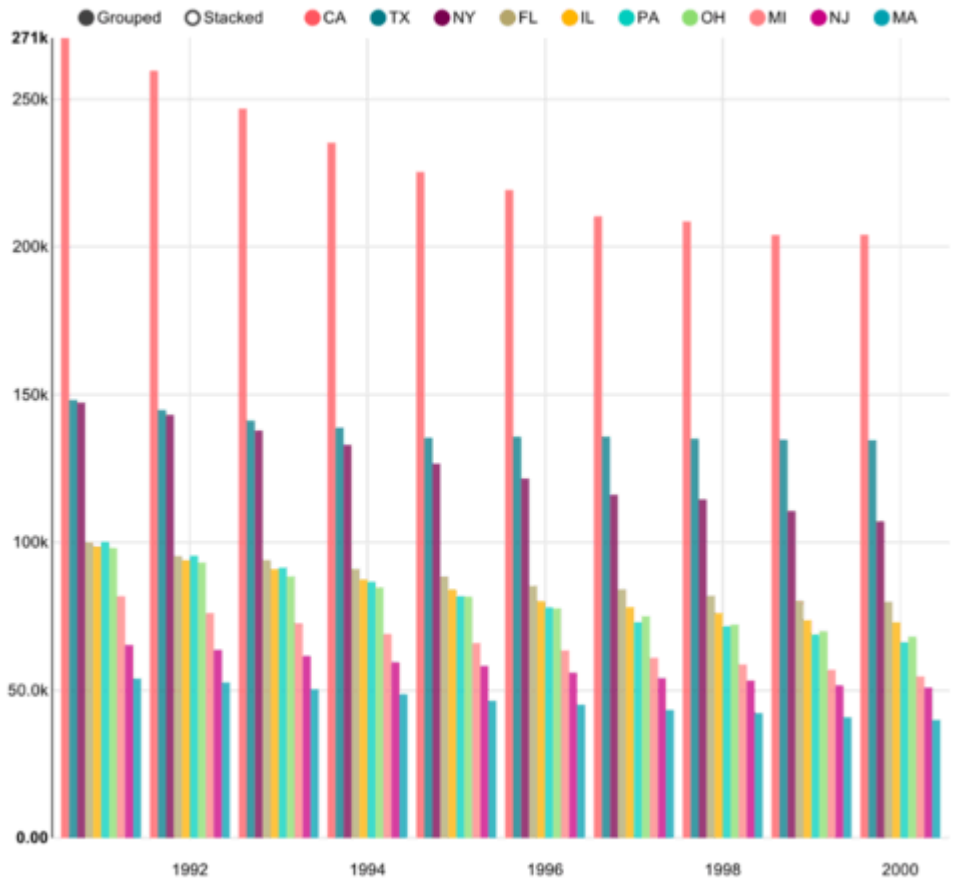
Datasource & Chart Type

This section contains the following two parts:

Data source - Here users can view and edit the current datasource chosen to build the chart. By clicking on the datasource users can enter the datasource editor and edit the different datasource settings as well as edit the data columns and metrics.

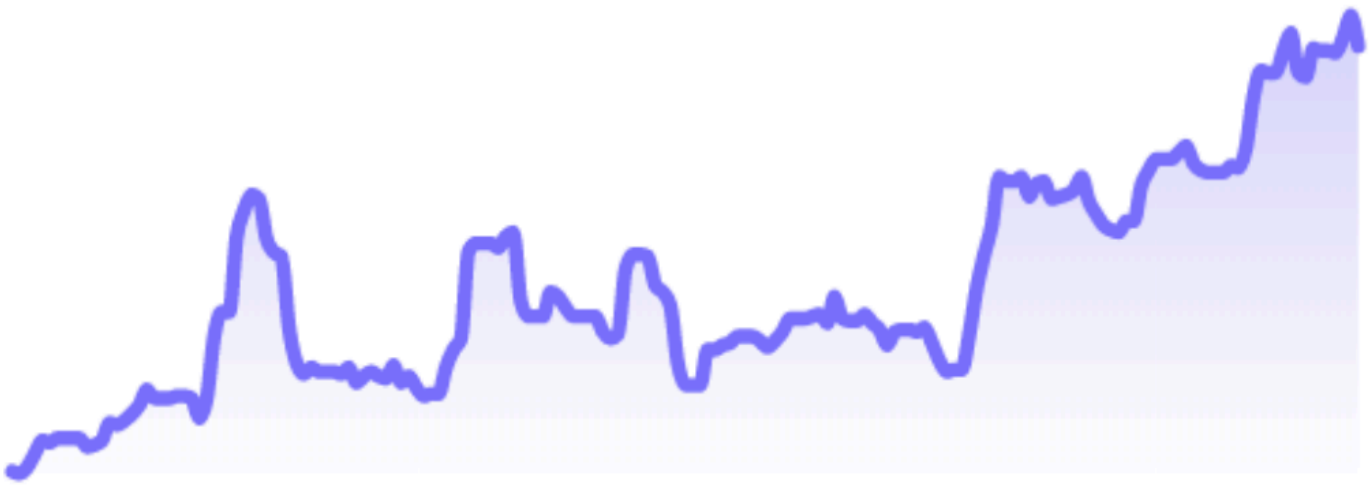
Visualization Type - Opens a window where users can choose the type of chart visualization to display. Timbr offers the following visualizations:



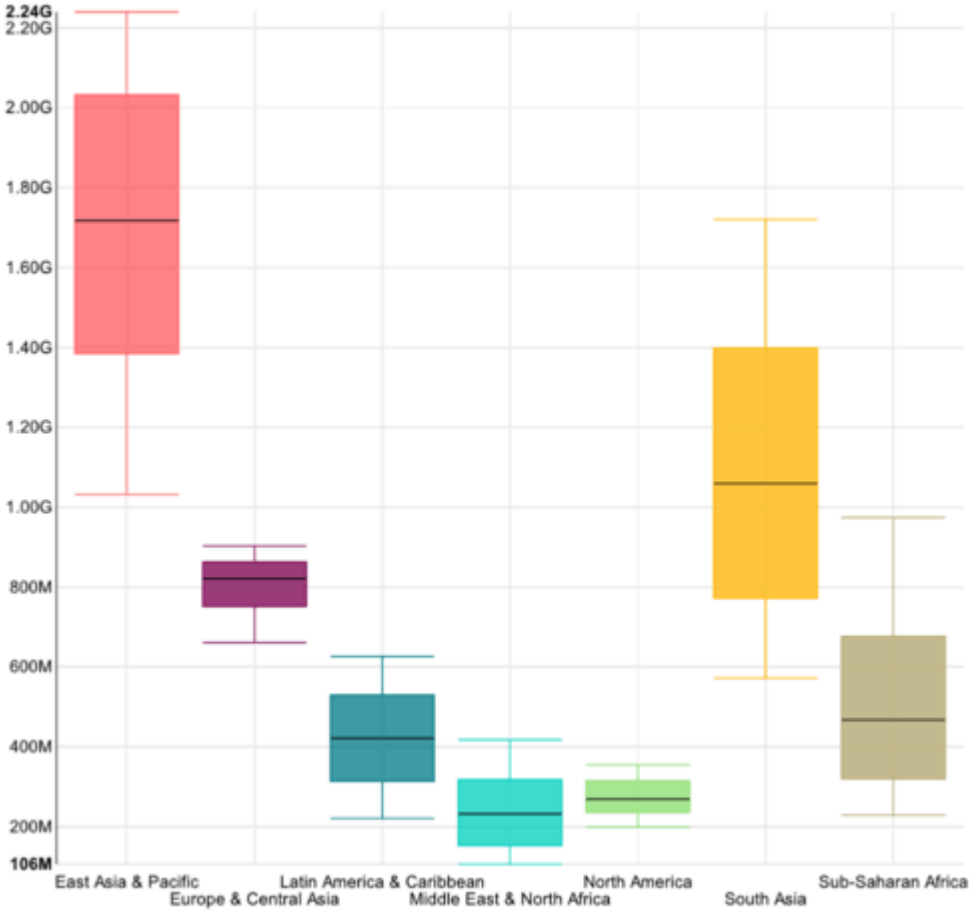

Grouped
Stacked
CA
TX
NY
FL
IL
PA
OH
MI
NJ
MA
271k
250k
200k
150k
100k
50.0k
0.00
1992
1994
1996
1998
2000

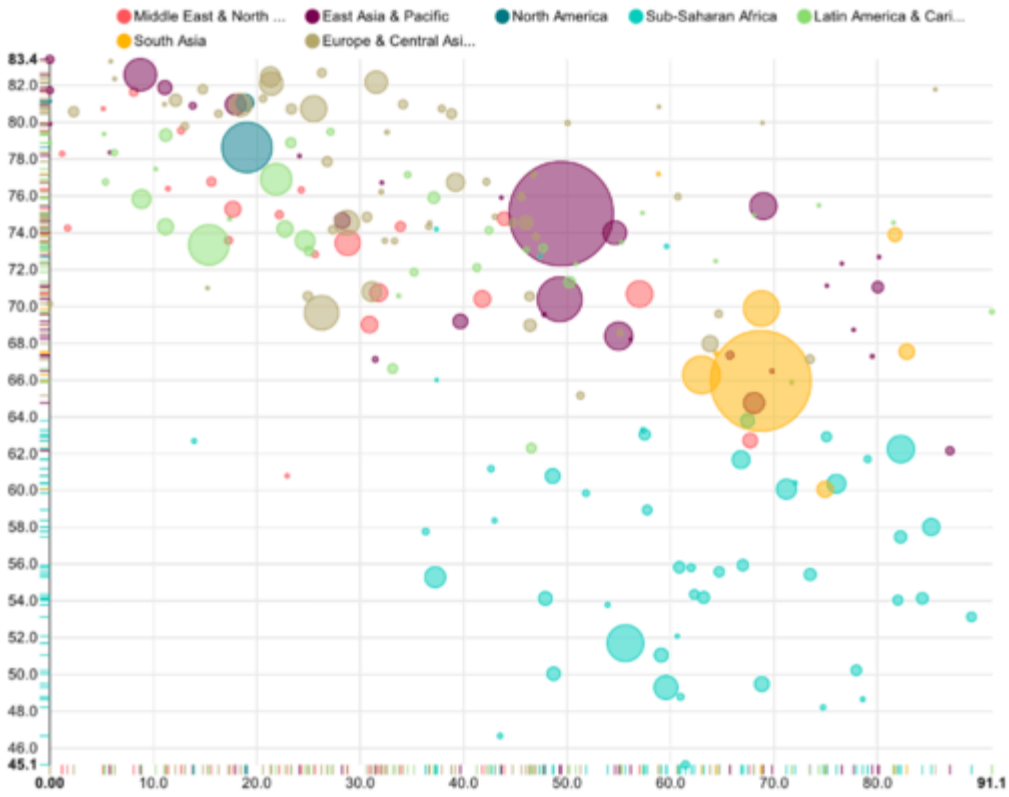
215

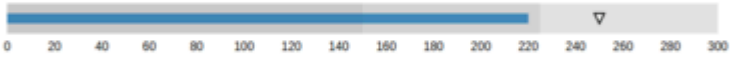
+7.0% WoW

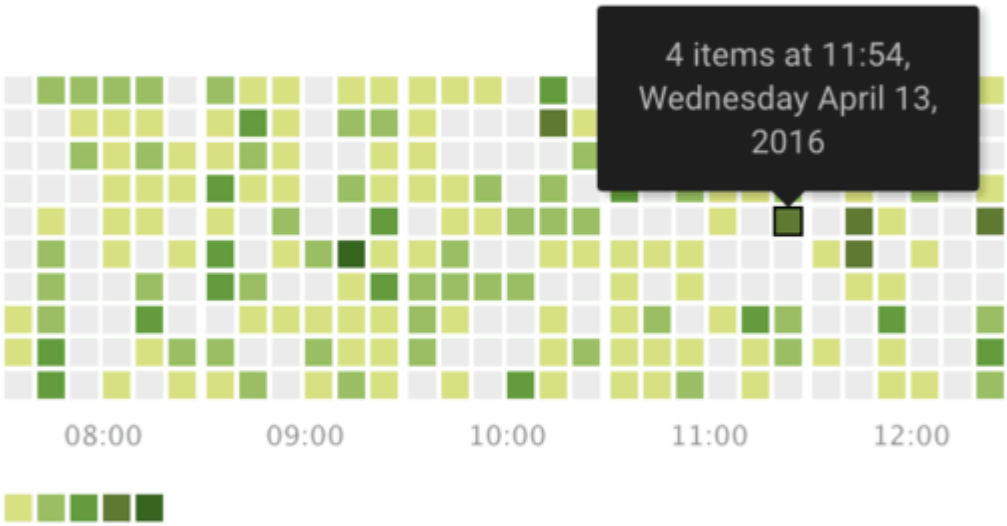


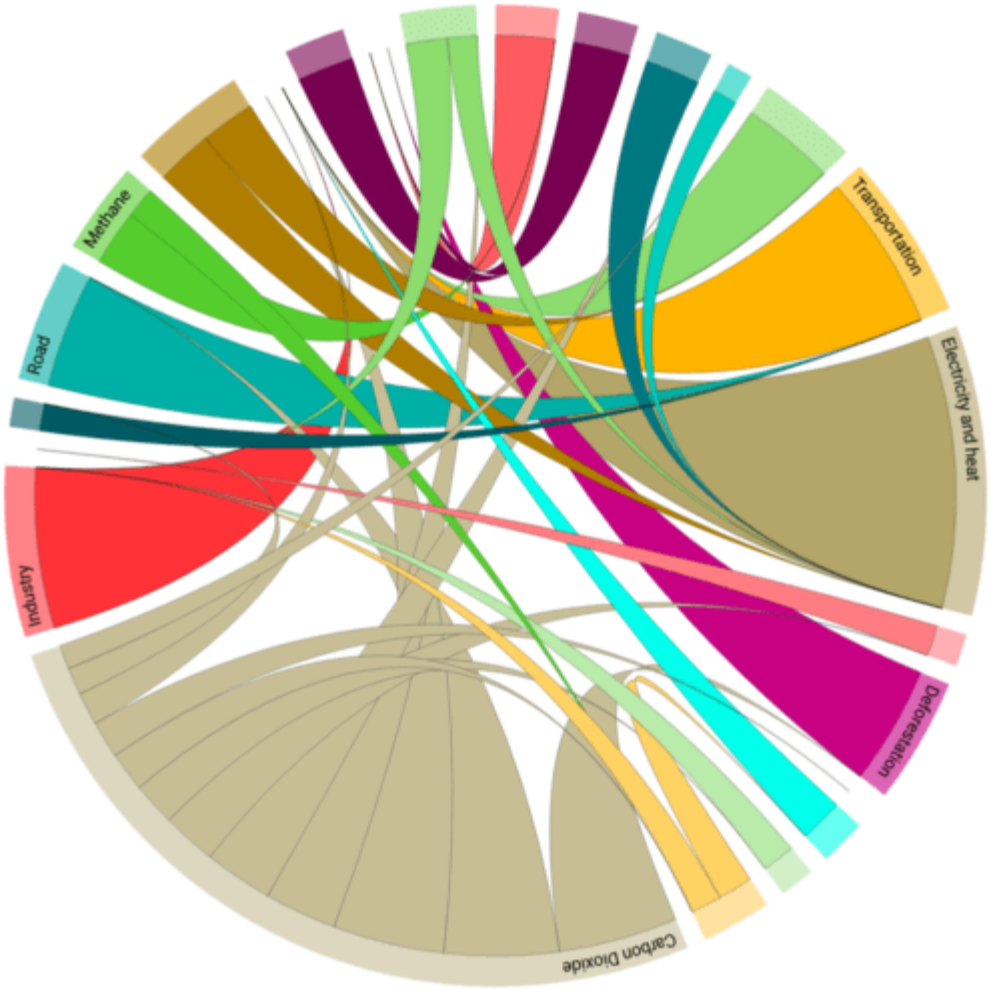

80.7M

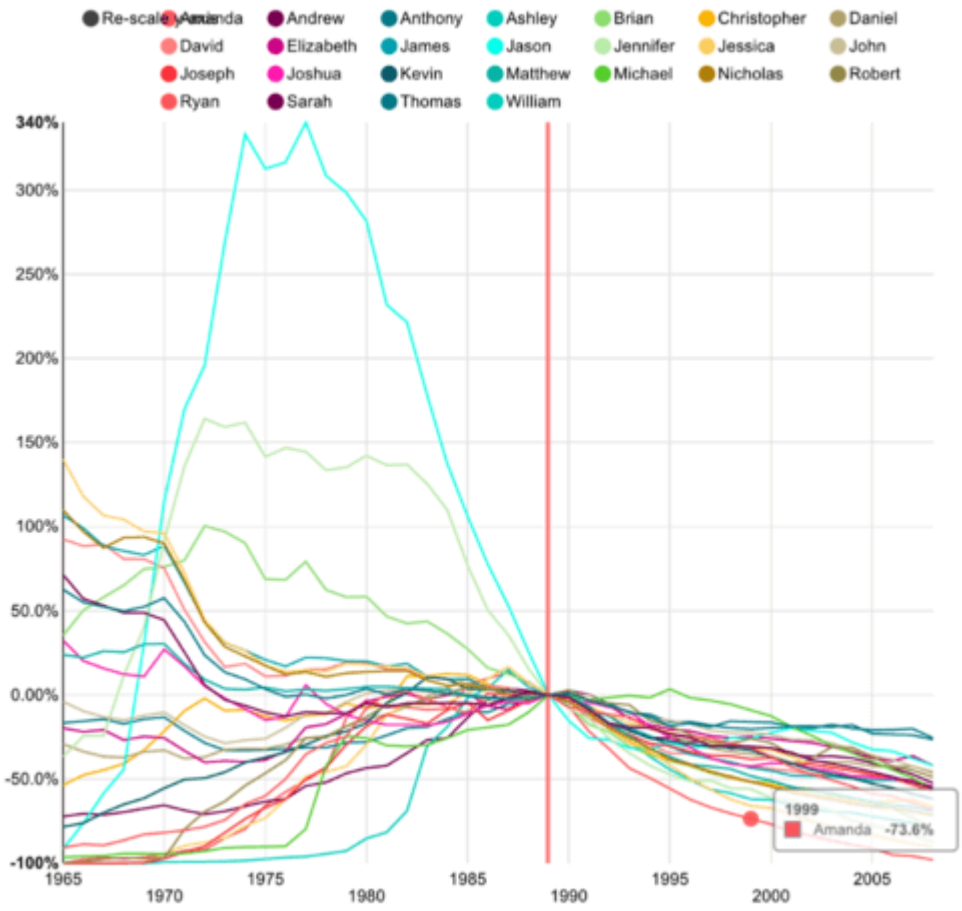

2.24G
2.20G
2.00G
1.80G
1.60G
1.40G
1.20G
1.00G
800M
600M
400M
200M
106M
East Asia & Pacific
Europe & Central Asia
Latin America & Caribbean
Middle East & North Africa
North America
South Asia
Sub-Saharan Africa

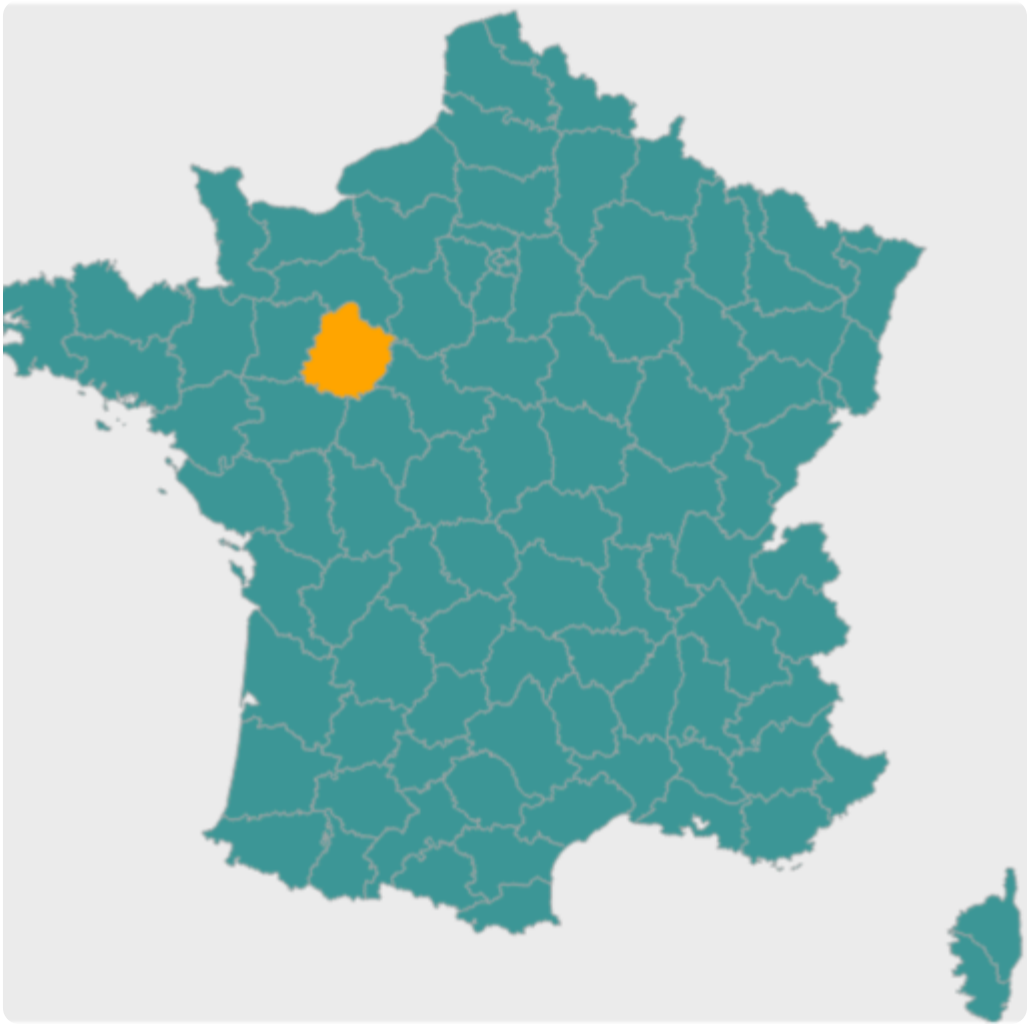

Middle East & North ...
East Asia & Pacific
North America
Sub-Saharan Africa
Latin America & Cari...
South Asia
Europe & Central Asi...
83.4
82.0
80.0
78.0
76.0
74.0
72.0
70.0
68.0
66.0
64.0
62.0
60.0
58.0
56.0
54.0
52.0
50.0
48.0
46.0
45.1
0.00
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
91.1

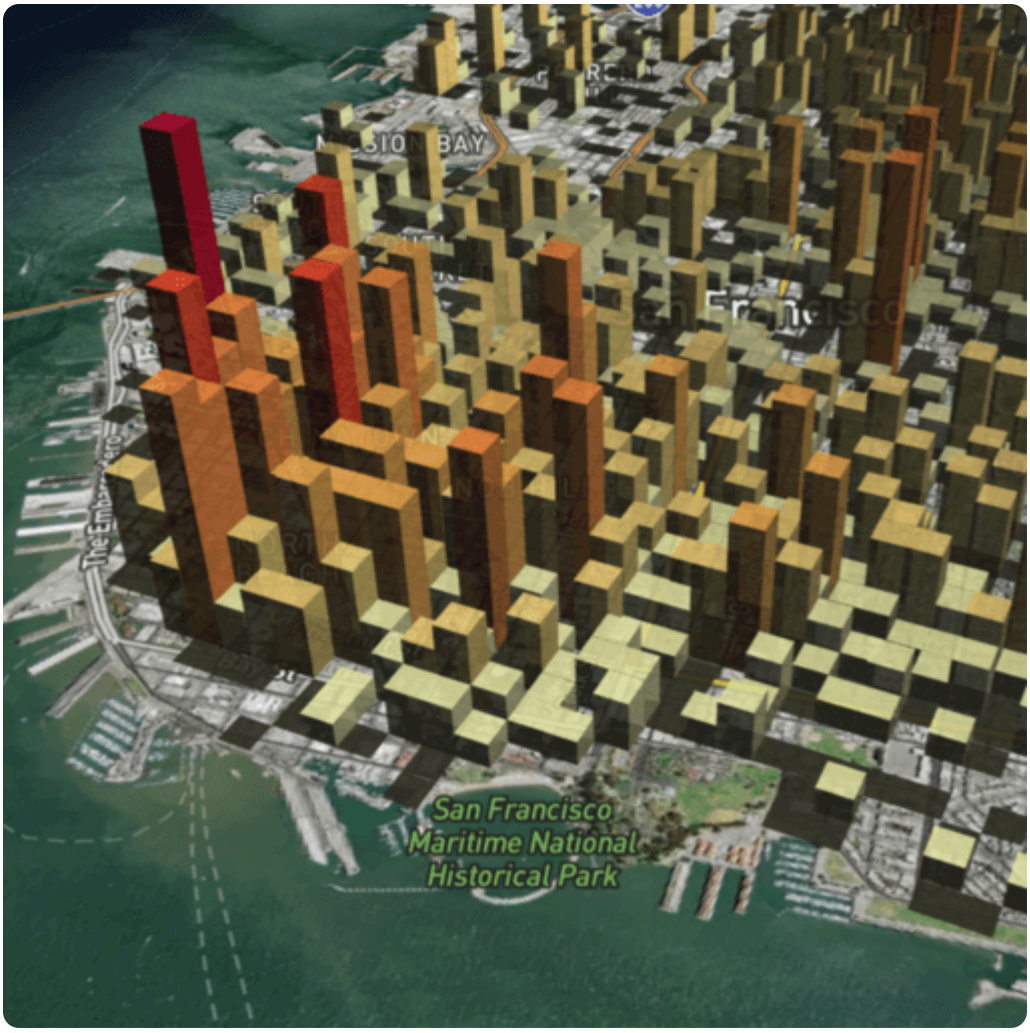

0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300

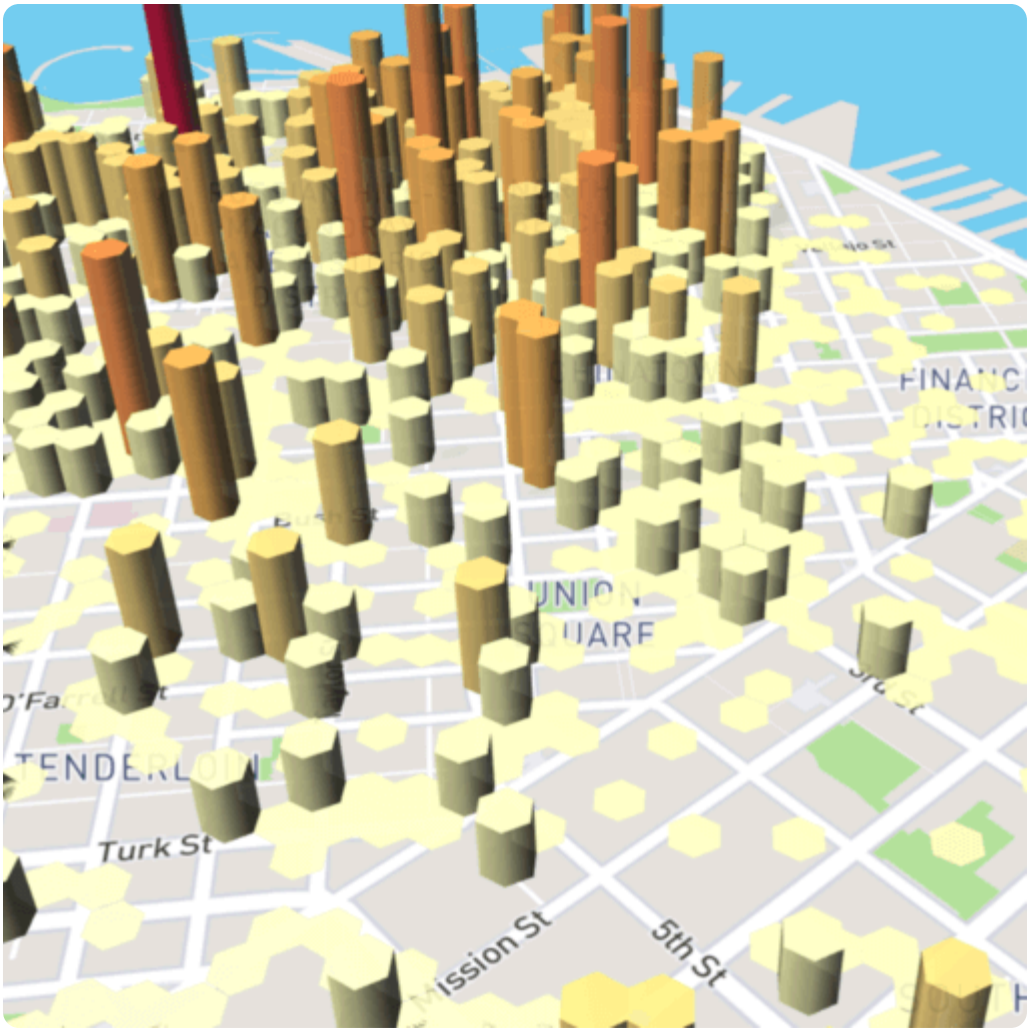

4 items at 11:54, Wednesday April 13, 2016
08:00
09:00
10:00
11:00
12:00


Transportation
Electricity and heat
Deforestation
Carbon Dioxide
Industry
Road
Methane


Re-scale y-axis
Amanda
Andrew
Anthony
Ashley
Brian
Christopher
Daniel
David
Elizabeth
James
Jason
Jennifer
Jessica
John
Joseph
Joshua
Kevin
Matthew
Michael
Nicholas
Robert
Ryan
Sarah
Thomas
William
340%
300%
250%
200%
150%
100%
50.0%
0.00%
-50.0%
-100%
1965
1970
1975
1980
1985
1990
1995
2000
2005
1999
Amanda -73.6%

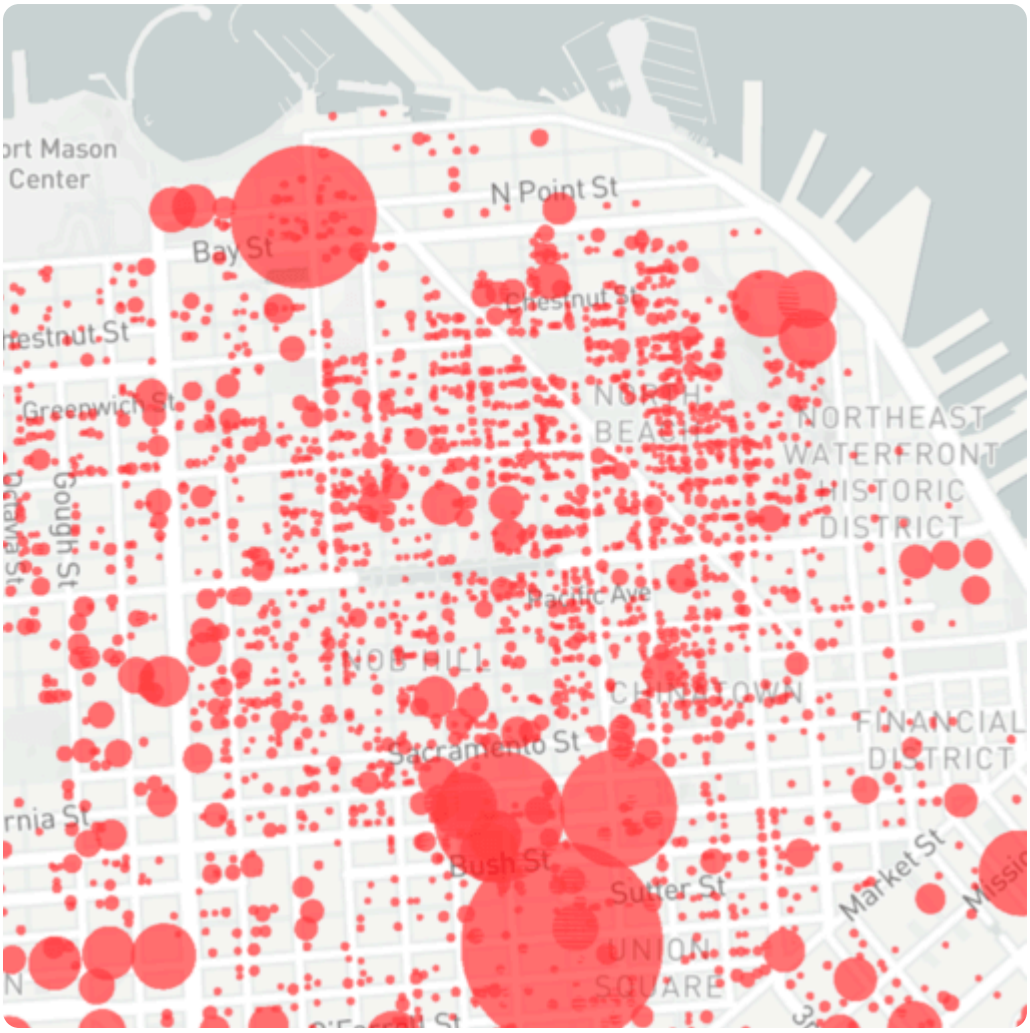


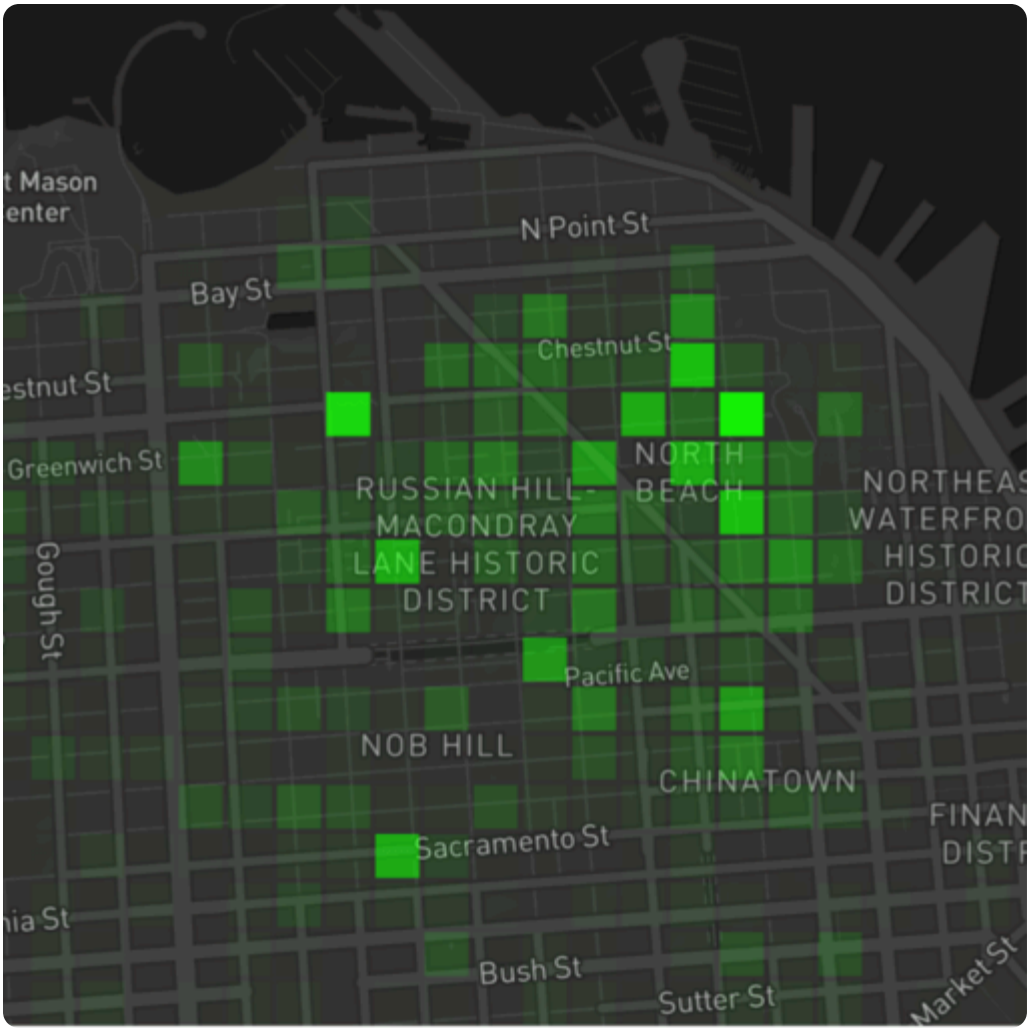

San Francisco
Maritime National
Historical Park


Turk St
5th St

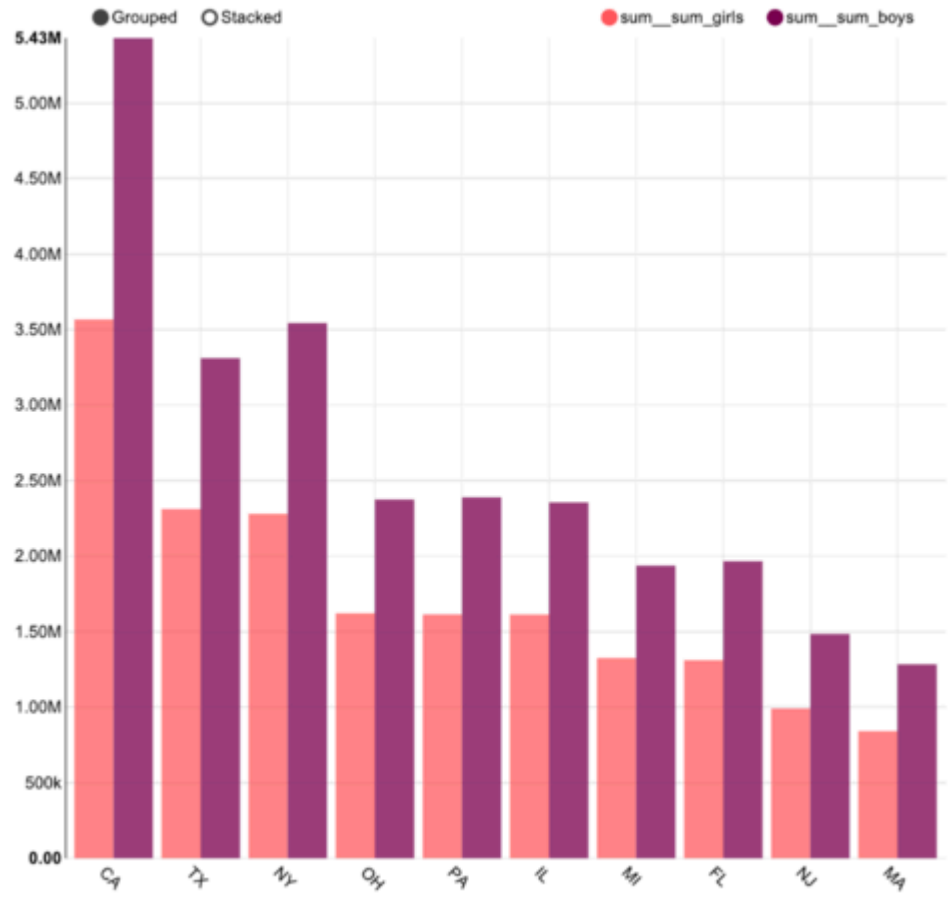

Fort Mason Center
N Point St
Bay St
Chestnut St
NORTHEAST WATERFRONT HISTORIC DISTRICT
NOB HILL
CHINATOWN
California St
Pine St
Sutter St
Ellis St
WESTERN ADDITION
San Franci
WEST SOMA
5th St
Brannan St
Townsend St
10th St
9th St
MISSION BAY
101
Spear St
Cadero

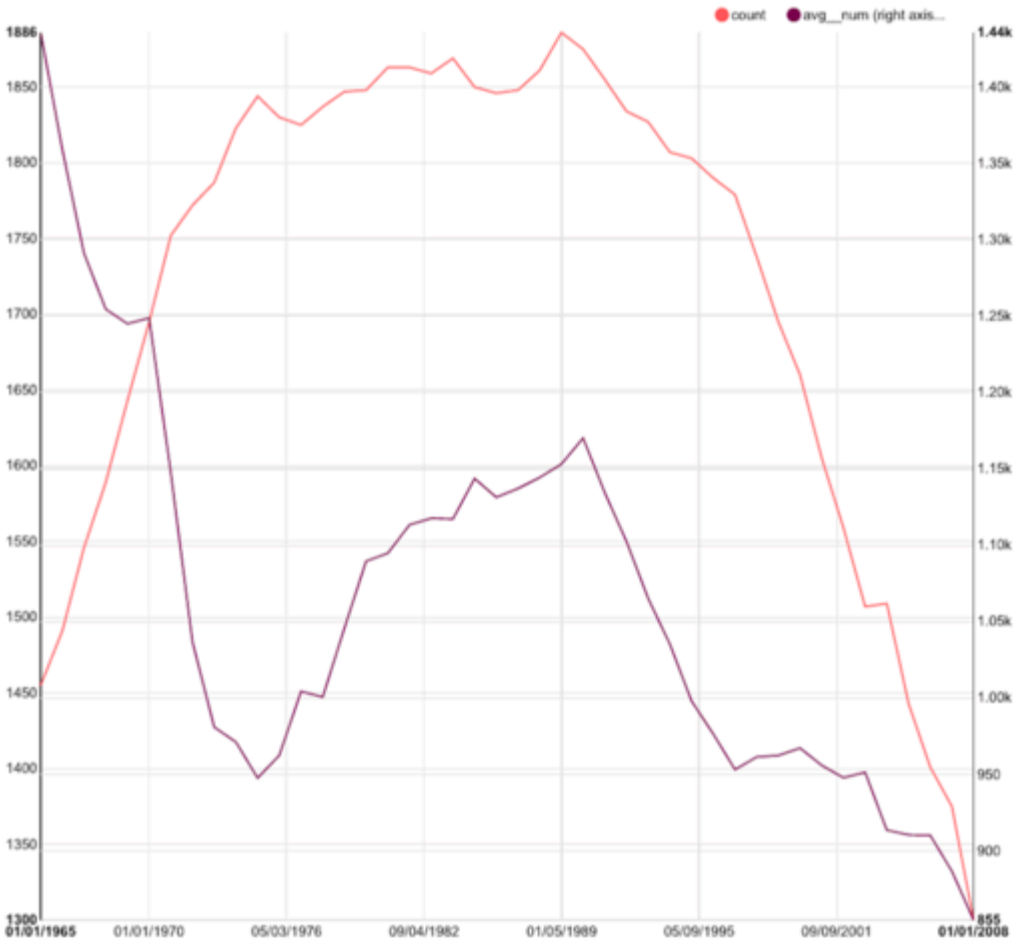

Presidio
San Francisco
Golden Gate Park
Daly City
Broadmoor
San Bruno
Mountain State
and County Park
Colma
Brisbane
South San
Francisco
Westborough
San Bruno
Pacifica

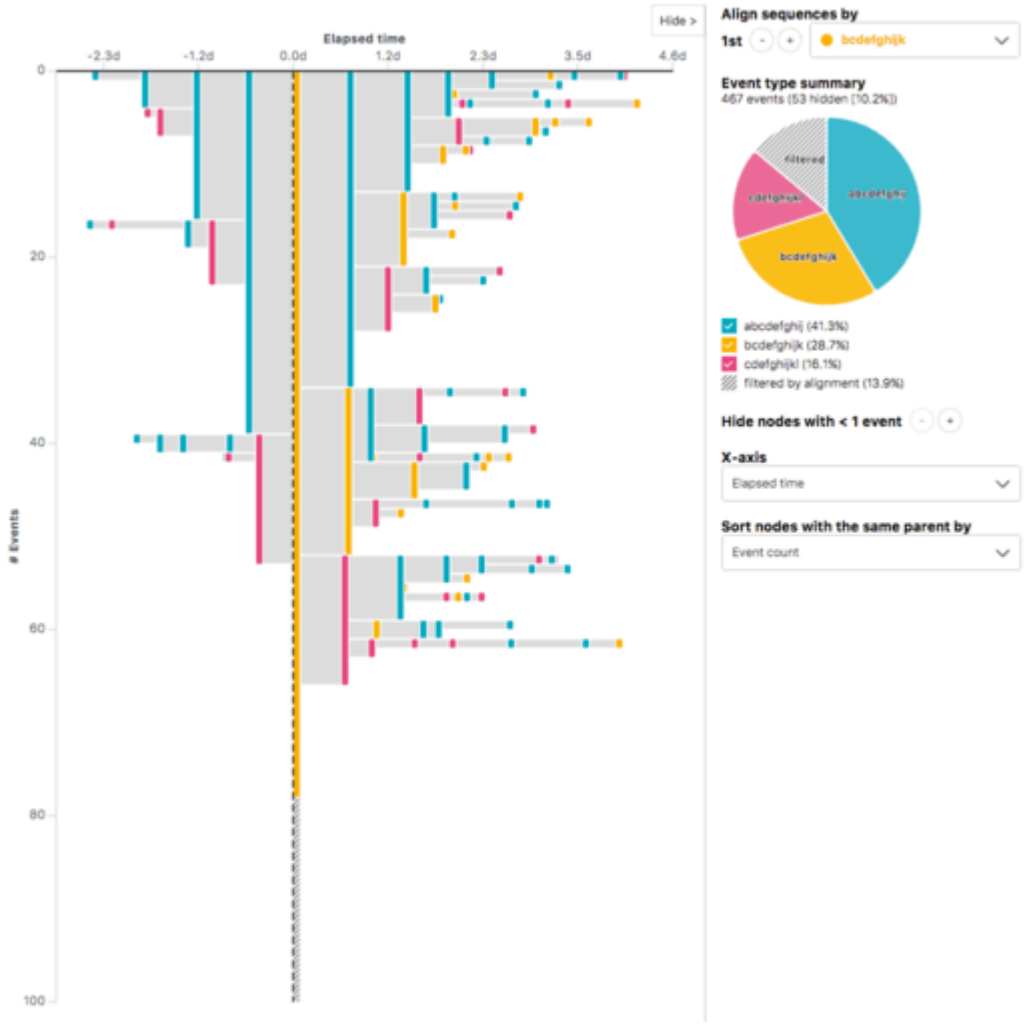

N Point St
Bay St
Chestnut St
Greenwich St
Gough St
Pacific Ave
Sacramento St
Bush St
Sutter St
Market St
NORTHEAST
WATERFRONT
HISTORIC
DISTRICT
FINANCIAL
DISTRICT
UNION
SQUARE

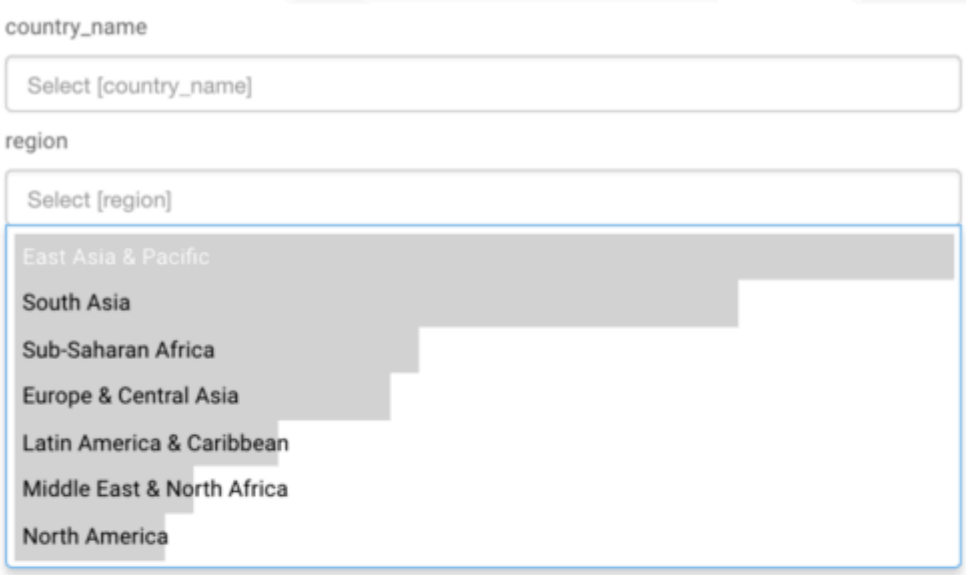

N Point St
Bay St
Chestnut St
Greenwich St
RUSSIAN HILL-
MACONDRAY
LANE HISTORIC
DISTRICT
NORTH
BEACH
Gough St
Pacific Ave
NOB HILL
CHINATOWN
Sacramento St
Bush St
Sutter St
Market St

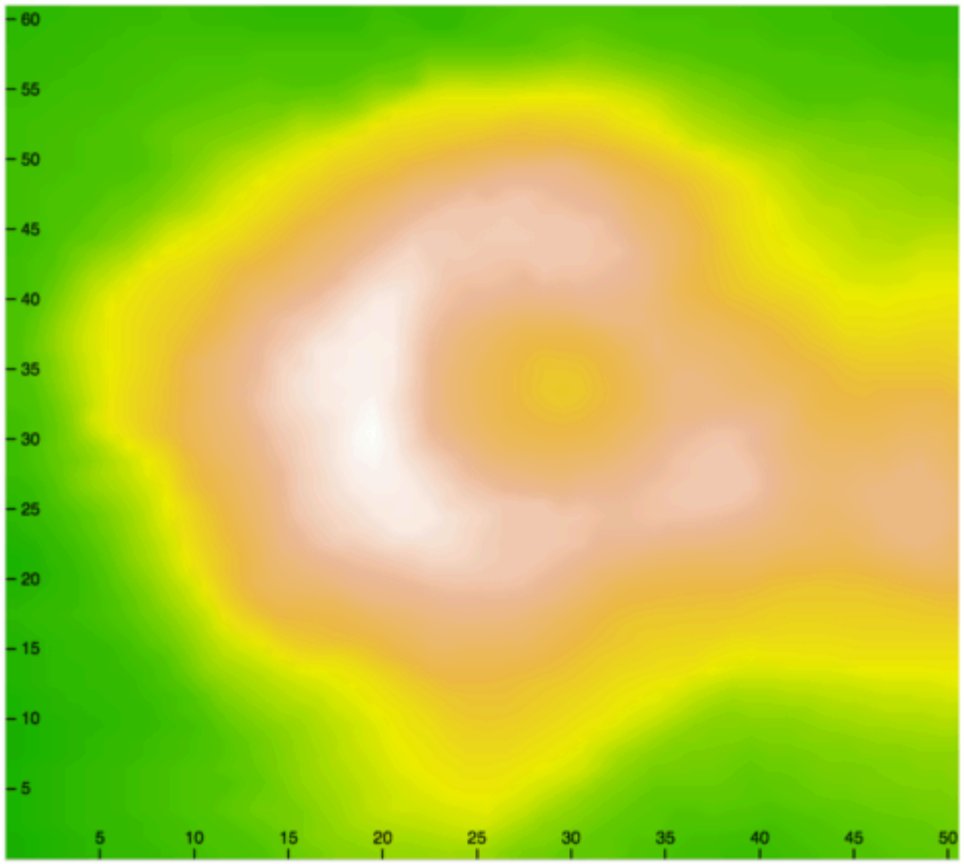

Cement
Agricultural Energy Use
Chemicals
Iron and Steel
Food and Tobacco
Unallocated Fuel Combustion
Industrial Processes
Rail - Ship and Other Transport
Residential Buildings
Other Fuel Combustion
Industry
Harvest / Management
Commercial Buildings
Carbon Dioxide
Transportation
Land Use C
HFCs - PFCs
Other Industry
Nitrous Oxide
Methane
Deforestation
Energy
Oil and Gas Processing
Waste water - Other Waste
Air
Landfills
Rice Cultivation
Coal Mining
Agriculture Soils
Other Agriculture
Fugitive Emissions
Waste
Agriculture

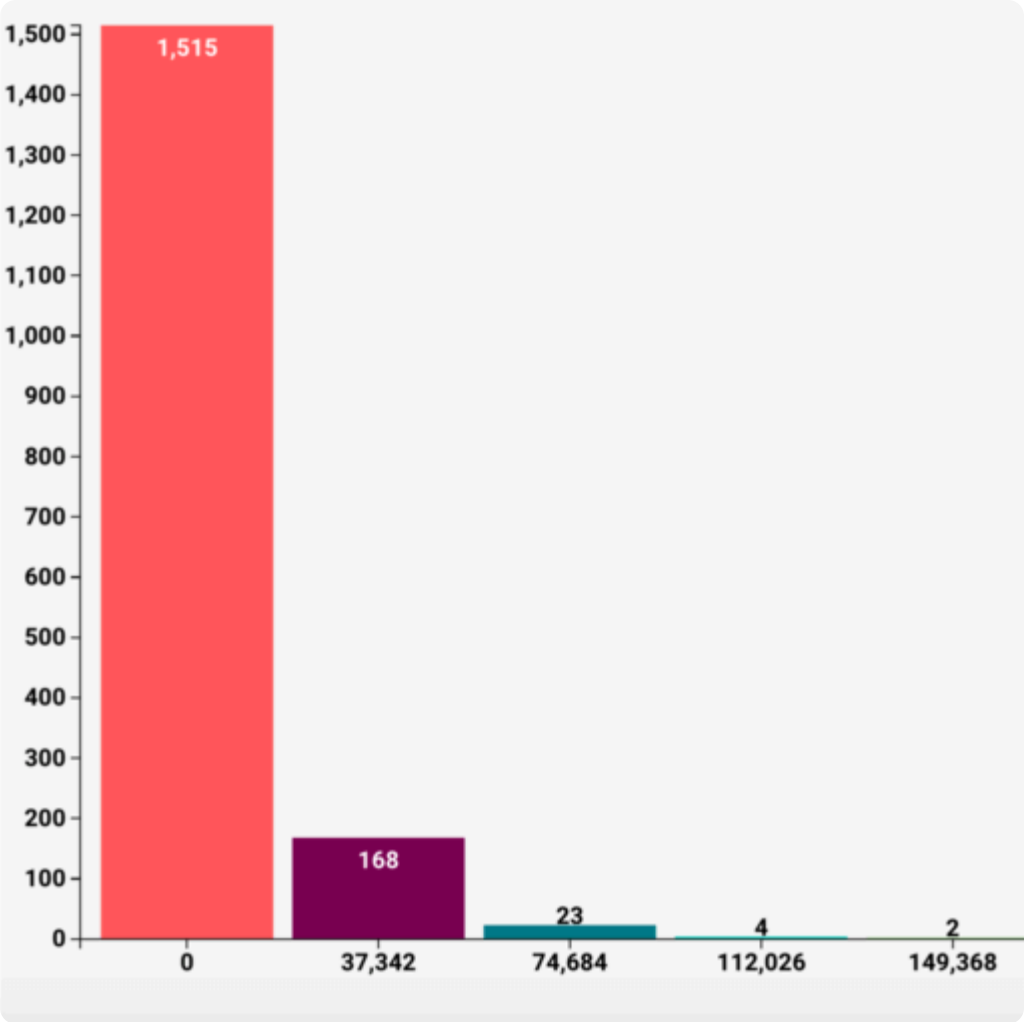

Grouped
Stacked
sum__sum_girls
sum__sum_boys
5.43M
5.00M
4.50M
4.00M
3.50M
3.00M
2.50M
2.00M
1.50M
1.00M
500k
0.00
CA
TX
NY
OH
PA
IL
MI
FL
NJ
MA

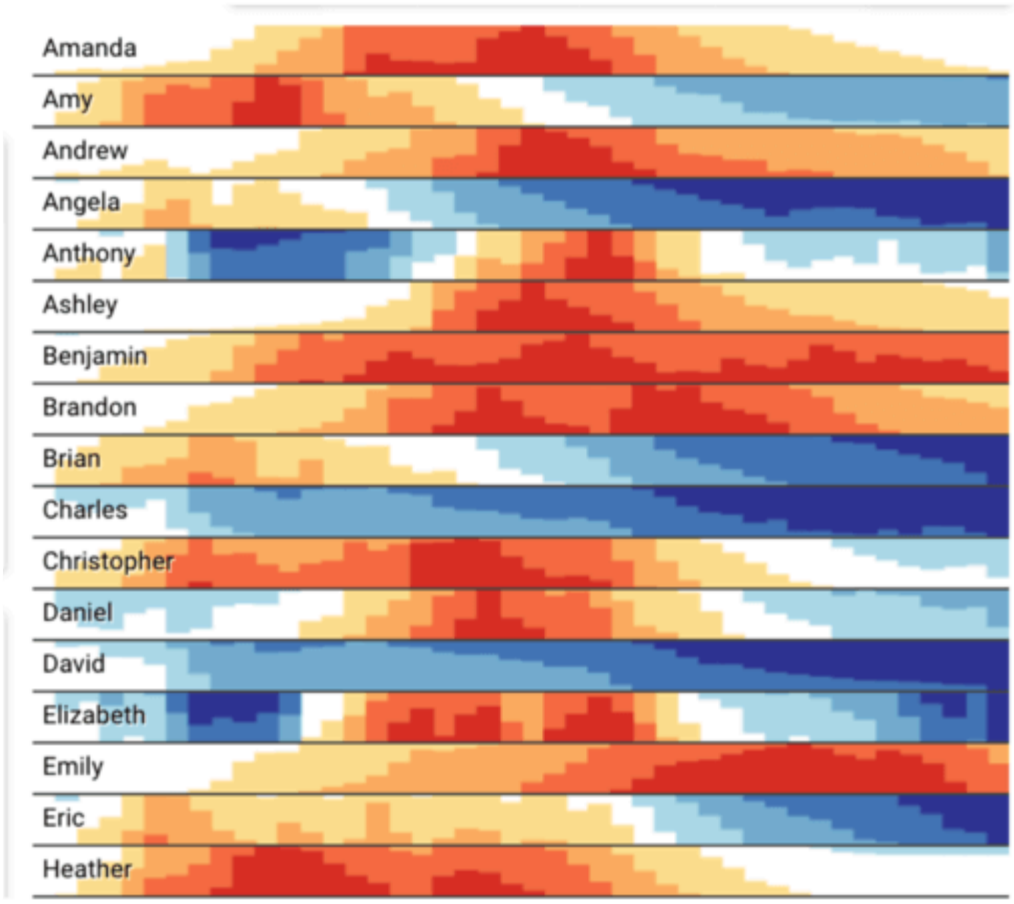

count
avg__num (right axis...
1886
1850
1800
1750
1700
1650
1600
1550
1500
1450
1400
1350
1300
1.44k
1.40k
1.35k
1.30k
1.25k
1.20k
1.15k
1.10k
1.05k
1.00k
950
900
855
01/01/1965
01/01/1970
05/03/1976
09/04/1982
01/05/1989
05/09/1995
09/09/2001
01/01/2008

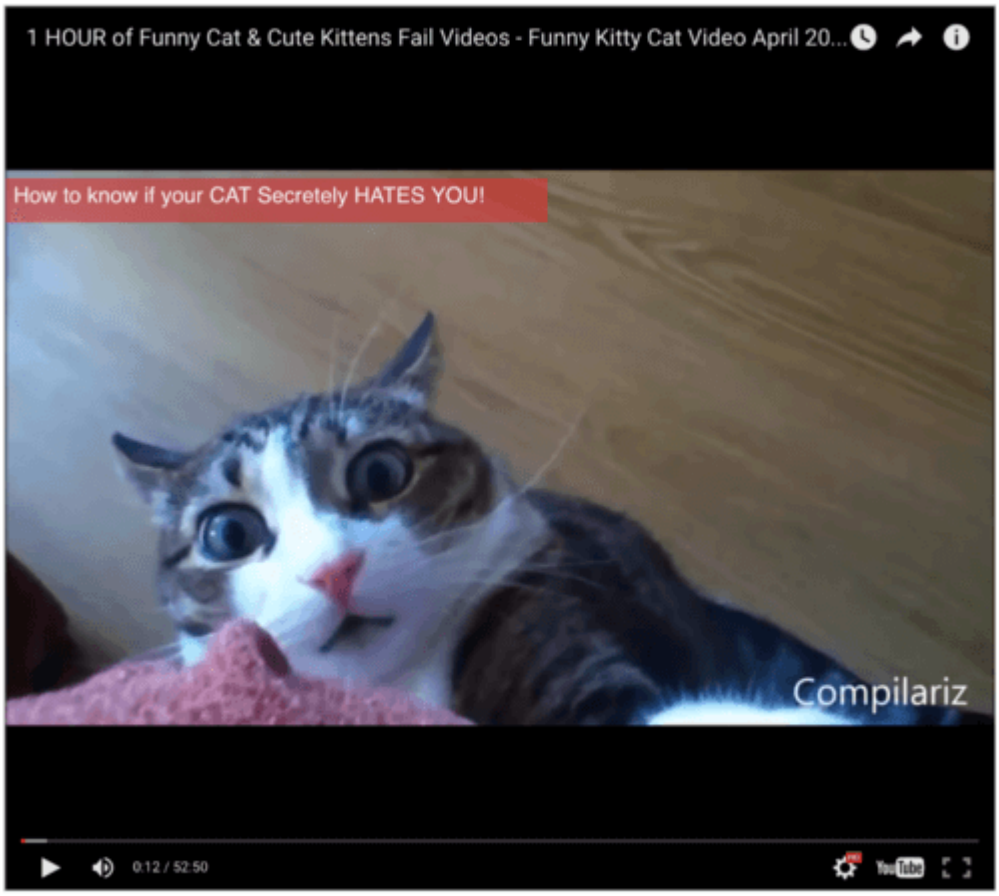

Hide >
Elapsed time
-2.3d
-1.2d
0.0d
1.2d
2.3d
3.5d
4.6d
0
20
40
60
80
100
Events
Align sequences by
1st
bcdefghijk
Event type summary
467 events (53 hidden (10.2%))
filtered
abcdefghij
cdefghijkl
bcdefghijk
abcdefghij (41.3%)
bcdefghijk (28.7%)
cdefghijkl (16.1%)
filtered by alignment (13.9%)
Hide nodes with < 1 event
X-axis
Elapsed time
Sort nodes with the same parent by
Event count

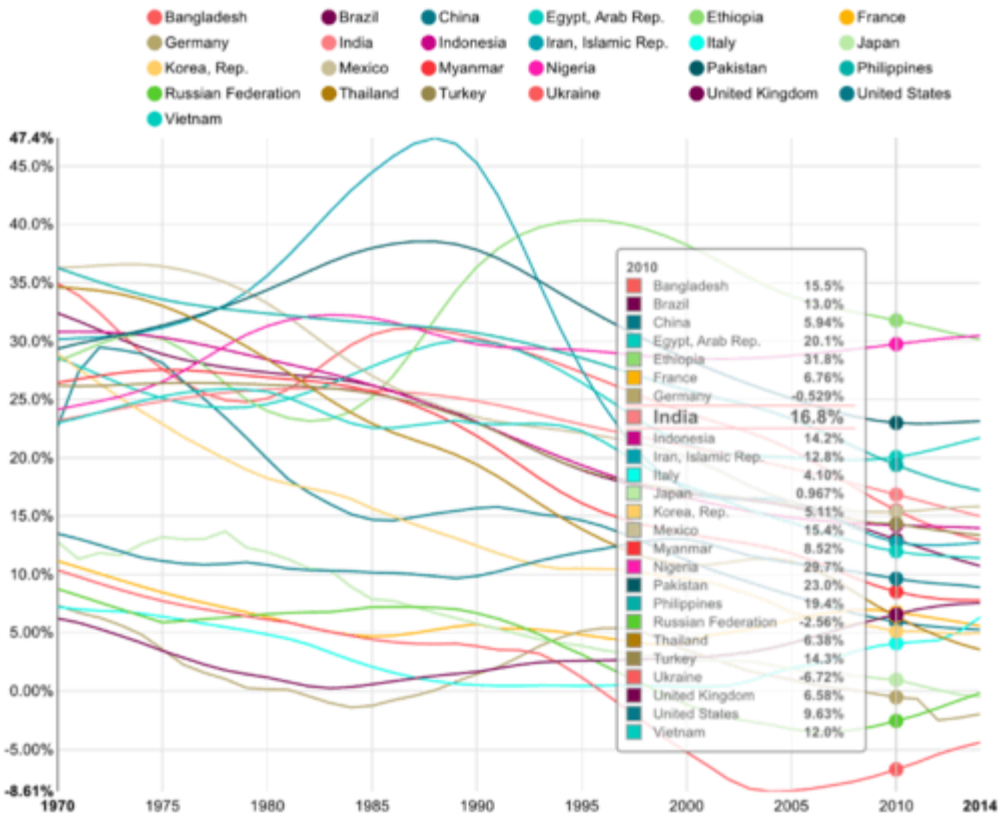

country_name
Select [country_name]
region
Select [region]
East Asia & Pacific
South Asia
Sub-Saharan Africa
Europe & Central Asia
Latin America & Caribbean
Middle East & North Africa
North America

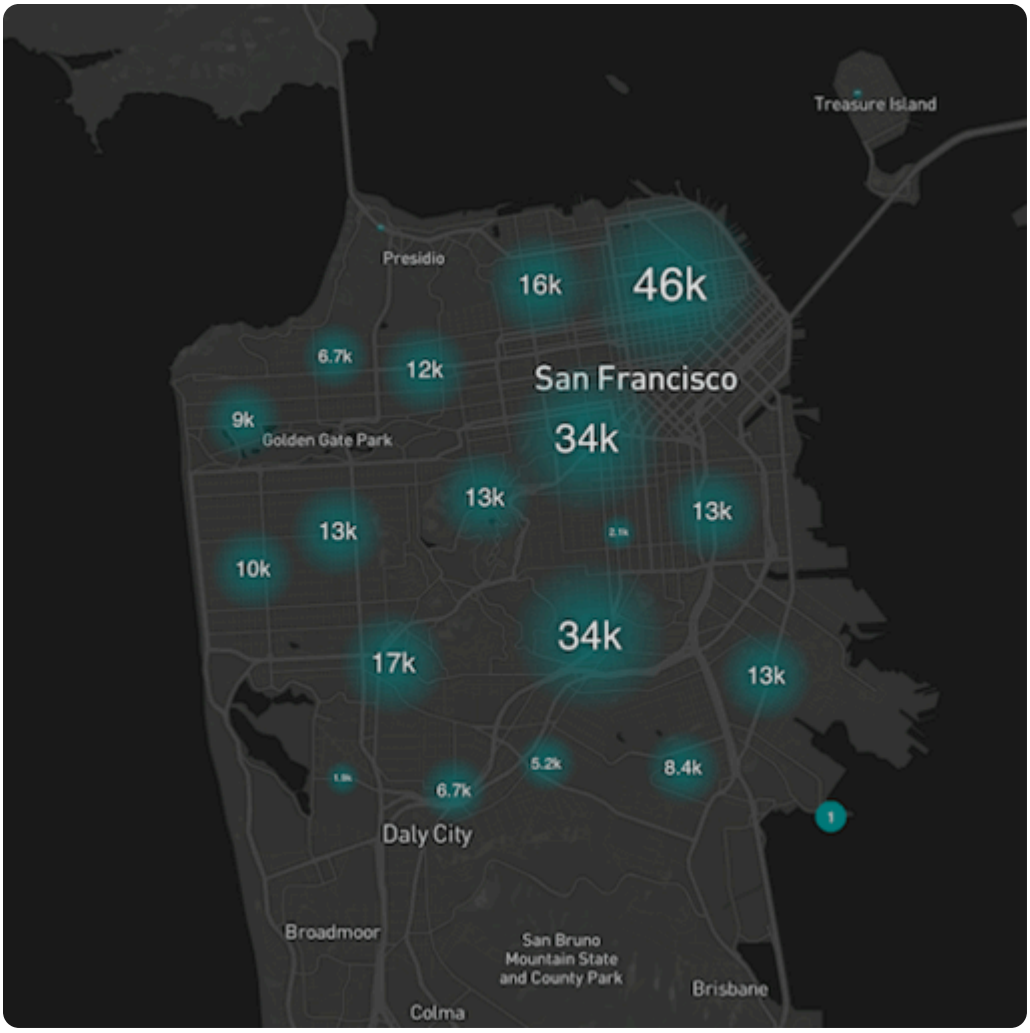

60
55
50
45
40
35
30
25
20
15
10
5
5
10
15
20
25
30
35
40
45
50


1,500
1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
1,515
168
23
4
2
0
37,342
74,684
112,026
149,368


Amanda
Amy
Andrew
Angela
Anthony
Ashley
Benjamin
Brandon
Brian
Charles
Christopher
Daniel
David
Elizabeth
Emily
Eric
Heather

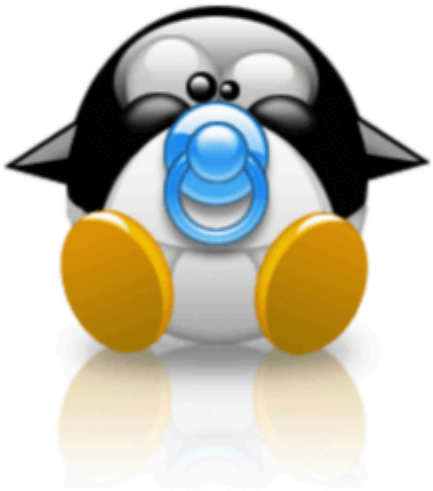

1 HOUR of Funny Cat & Cute Kittens Fail Videos - Funny Kitty Cat Video April 20...
How to know if your CAT Secretely HATES YOU!
Compilariz
0:12 / 52:50


Bangladesh
Brazil
China
Egypt, Arab Rep.
Ethiopia
France
Germany
India
Indonesia
Iran, Islamic Rep.
Italy
Japan
Korea, Rep.
Mexico
Myanmar
Nigeria
Pakistan
Philippines
Russian Federation
Thailand
Turkey
Ukraine
United Kingdom
United States
Vietnam
47.4%
45.0%
40.0%
35.0%
30.0%
25.0%
20.0%
15.0%
10.0%
5.00%
0.00%
-5.00%
-8.61%
1970
1975
1980
1985
1990
1995
2000
2005
2010
2014
2010
Bangladesh 15.5%
Brazil 13.0%
China 5.94%
Egypt, Arab Rep. 20.1%
Ethiopia 31.8%
France 6.76%
Germany -0.529%
India 16.8%
Indonesia 14.2%
Iran, Islamic Rep. 12.8%
Italy 4.10%
Japan 0.967%
Korea, Rep. 5.11%
Mexico 15.4%
Myanmar 8.52%
Nigeria 29.7%
Pakistan 23.0%
Philippines 19.4%
Russian Federation -2.56%
Thailand 6.38%
Turkey 14.3%
Ukraine -6.72%
United Kingdom 6.58%
United States 9.63%
Vietnam 12.0%


Treasure Island
Presidio
16k
46k
6.7k
12k
San Francisco
9k
Golden Gate Park
34k
13k
13k
13k
10k
34k
17k
13k
5.2k
8.4k
6.7k
Daly City
Broadmoor
San Bruno
Mountain State
and County Park
Brisbane
Colma

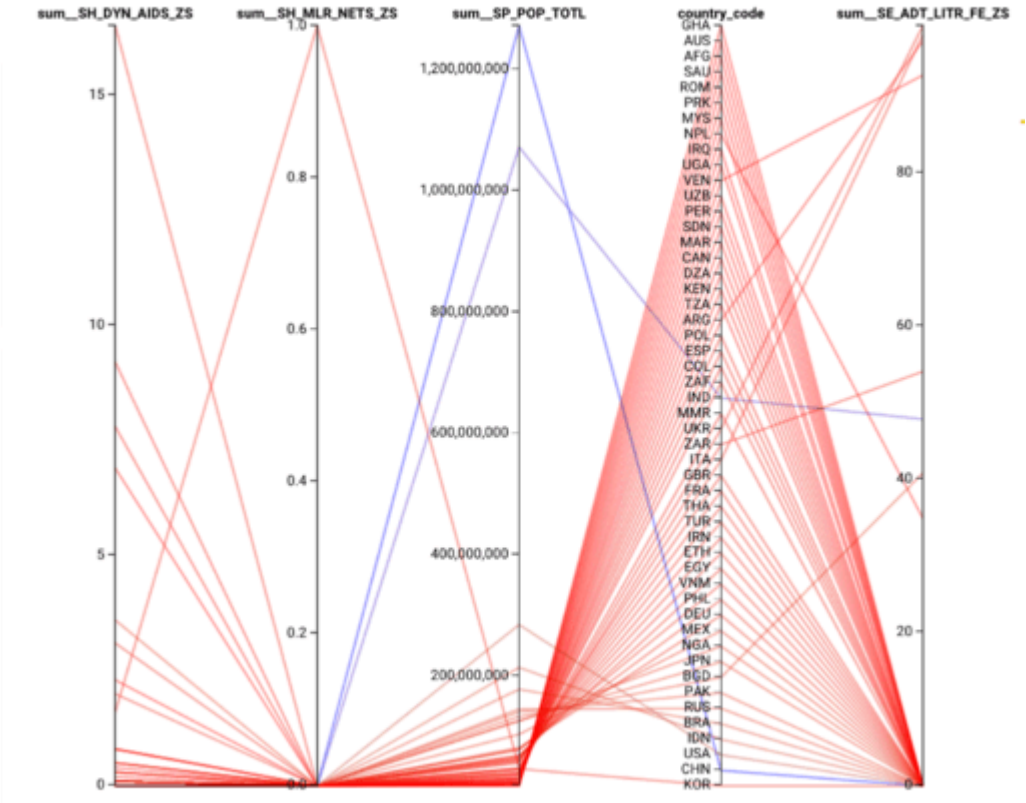
Birth Names Dashboard

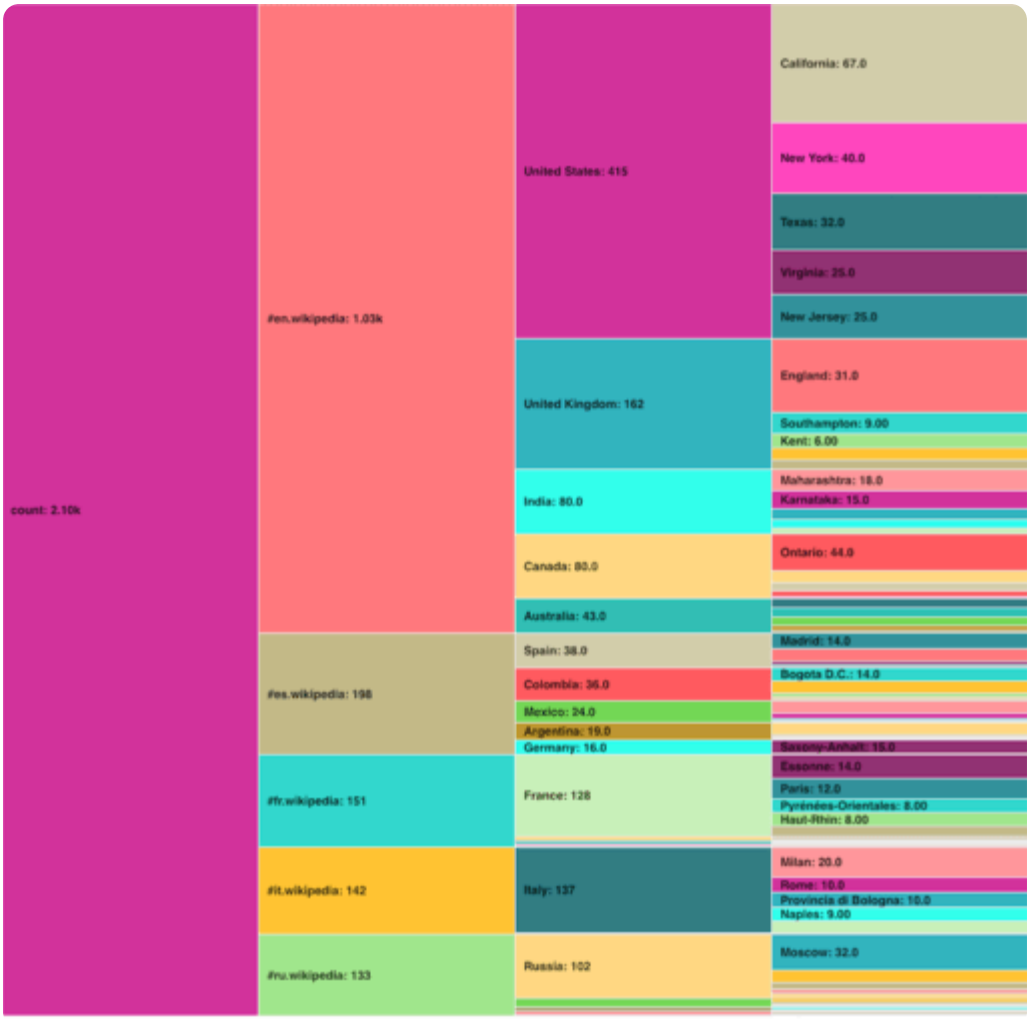
The source dataset came from [here]

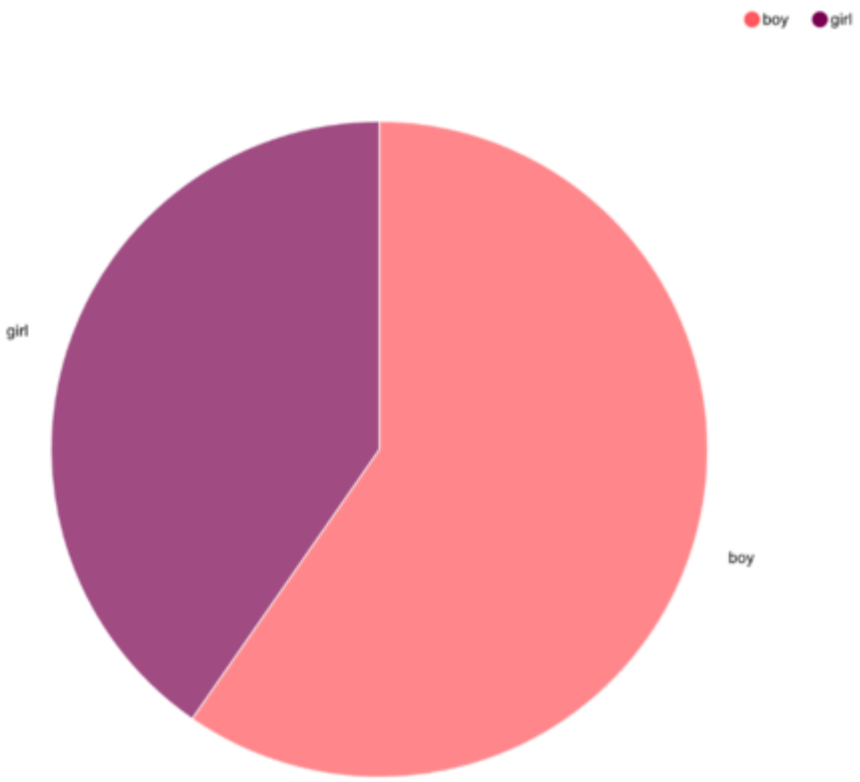


sum__SP_POP_TOTL

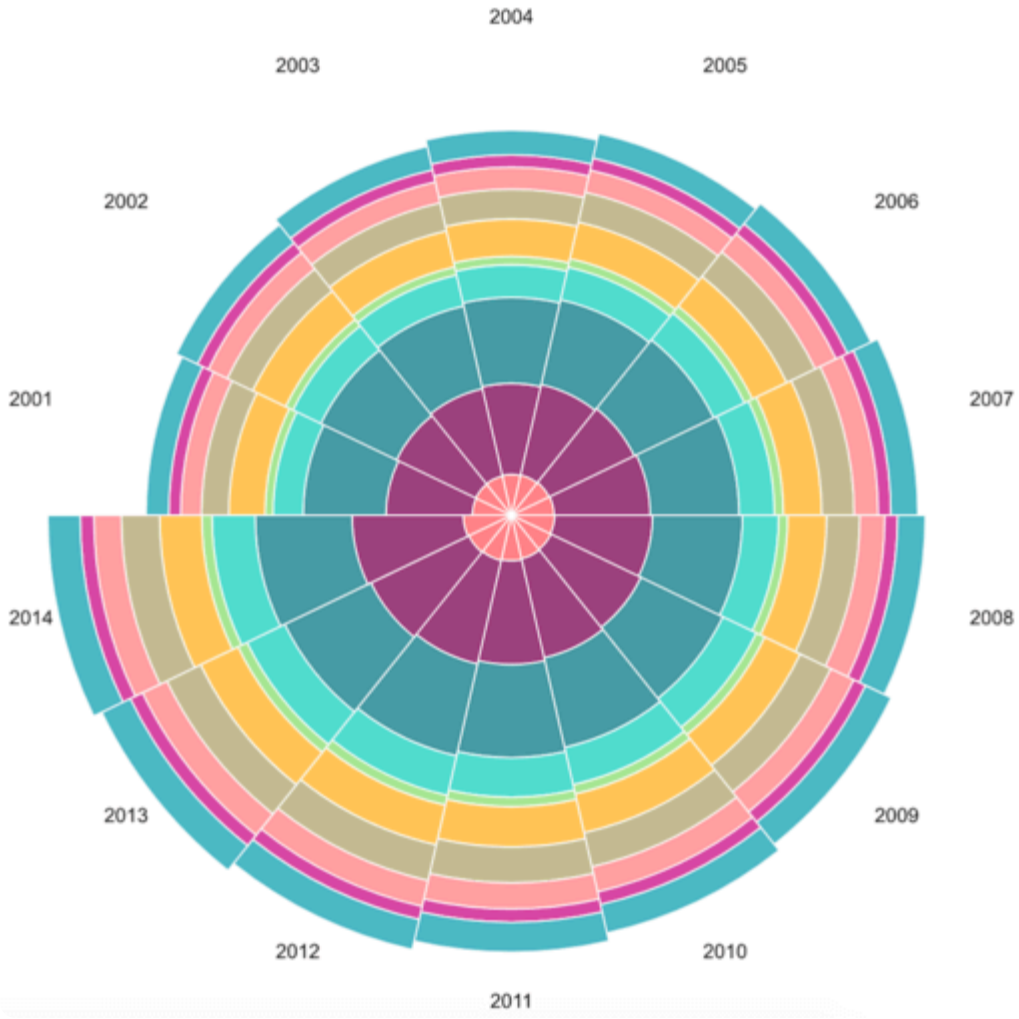
country_name	p-value	Lift %	Significant
Bangladesh	control	control	control
Brazil	0.000000	39.6953	true
China	0.000000	951.4094	true
Egypt, Arab Rep.	0.000000	-46.5174	true
Ethiopia	0.000000	-50.4267	true
France	0.000000	-43.2062	true
Germany	0.000009	-21.3987	true
India	0.000000	729.3543	true
Indonesia	0.000000	68.6326	true
Iran, Islamic Rep.	0.000000	-51.0290	true
Italy	0.000000	-44.4454	true
Japan	0.000011	16.3150	true
Korea, Rep.	0.000000	-60.0585	true
Mexico	0.000000	-19.9055	true
Myanmar	0.000000	-61.6733	true
Nigeria	0.000000	-5.2162	true
Pakistan	0.026927	2.6450	true
Philippines	0.000000	-41.0369	true
Russian Federation	0.000000	38.1659	true
Thailand	0.000000	-49.0534	true
Turkey	0.000000	-49.4488	true
Ukraine	0.000000	-52.1057	true
United Kingdom	0.000000	-42.8912	true

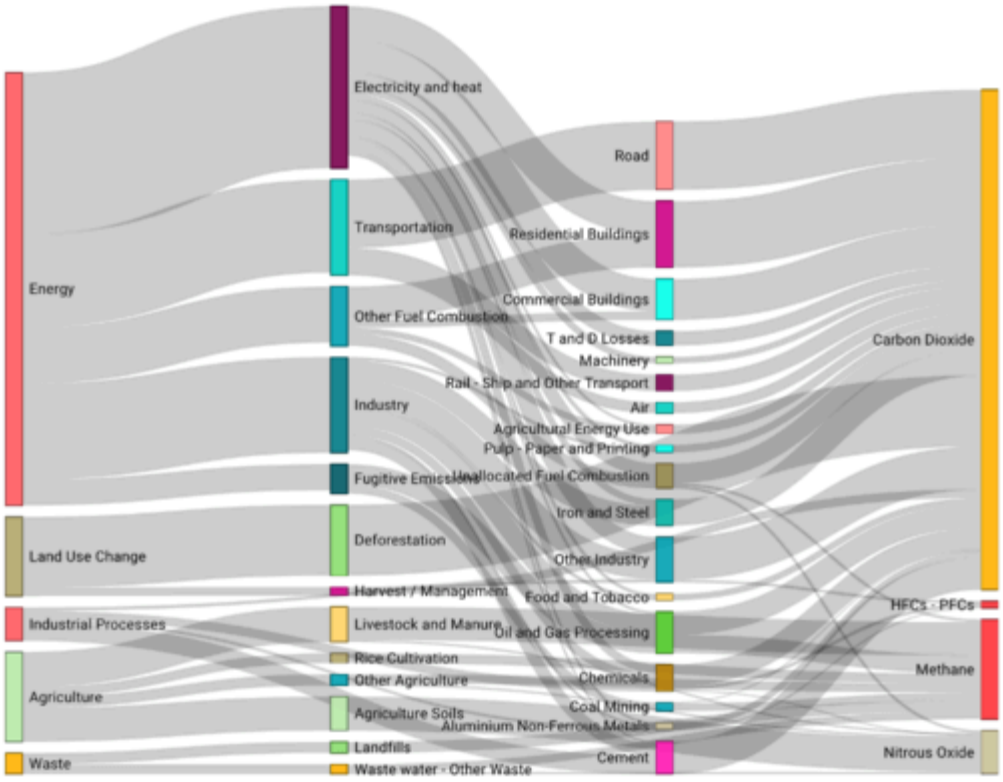

sum__SH_DYN_AIDS_ZS
sum__SH_MLR_NETS_ZS
sum__SP_POP_TOTL
country_code
sum__SE_ADT_LITR_FE_ZS
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
200,000,000
400,000,000
600,000,000
800,000,000
1,000,000,000
1,200,000,000
0
20
40
60
80
GHA
AUS
AFG
SAU
ROM
PRK
MYS
NPL
IRQ
UGA
VEN
UZB
PER
SDN
MAR
CAN
DZA
KEN
TZA
ARG
POL
ESP
COL
ZAF
IND
MMR
UKR
ZAR
ITA
GBR
FRA
THA
TUR
IRN
ETH
EGY
VNM
PHL
DEU
MEX
NGA
JPN
BGD
PAK
RUS
BRA
IDN
USA
CHN
KOR


count: 2.10k
#en.wikipedia: 1.03k
#es.wikipedia: 198
#fr.wikipedia: 151
#it.wikipedia: 142
#ru.wikipedia: 133
United States: 415
United Kingdom: 162
India: 80.0
Canada: 80.0
Australia: 43.0
Spain: 38.0
Colombia: 36.0
Mexico: 24.0
Argentina: 19.0
Germany: 16.0
France: 128
Italy: 137
Russia: 102
California: 67.0
New York: 40.0
Texas: 32.0
Virginia: 25.0
New Jersey: 25.0
England: 31.0
Southampton: 9.00
Kent: 6.00
Maharashtra: 18.0
Karnataka: 15.0
Ontario: 44.0
Madrid: 14.0
Bogota D.C.: 14.0
Saxony-Anhalt: 15.0
Essonne: 14.0
Paris: 12.0
Pyrénées-Orientales: 8.00
Haut-Rhin: 8.00
Milan: 20.0
Rome: 10.0
Provincia di Bologna: 10.0
Naples: 9.00
Moscow: 32.0


boy
girl
girl
boy

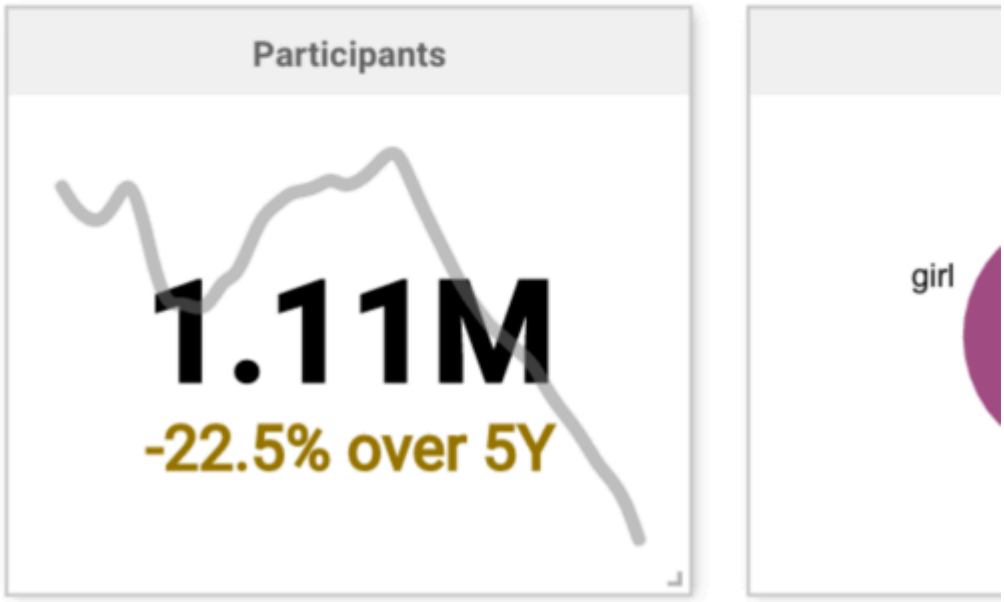
	sum__num							
state	CA	FL	IL	MA	MI	NJ	NY	OH
name								
All	8998550.0	3280653.0	3971838.0	2127272.0	3264550.0	2478828.0	5824694.0	3999199.0
Michael	259809.0	102070.0	134250.0	80370.0	102958.0	100214.0	226479.0	123660.0
Christopher	185674.0	79368.0	72462.0	49619.0	60170.0	57680.0	135892.0	75189.0
David	203216.0	62192.0	77826.0	44211.0	64717.0	49043.0	119754.0	78723.0
James	122151.0	64316.0	72096.0	39056.0	58361.0	44501.0	103237.0	78491.0
John	127360.0	56425.0	71680.0	47541.0	50189.0	55801.0	123228.0	68969.0
Matthew	141032.0	51145.0	69172.0	46238.0	59089.0	47300.0	104538.0	72477.0
Jennifer	159368.0	50954.0	70922.0	37870.0	55561.0	44390.0	103066.0	66141.0
Robert	139518.0	56420.0	65105.0	36917.0	55716.0	49421.0	105120.0	69607.0
Daniel	178189.0	48886.0	65886.0	35722.0	43133.0	42501.0	99315.0	49035.0
Joseph	115570.0	43743.0	57852.0	35094.0	44378.0	51143.0	118684.0	55193.0
William	84381.0	48452.0	48397.0	29333.0	36910.0	35864.0	76417.0	52937.0
Joshua	116469.0	50667.0	43319.0	21328.0	40412.0	18942.0	55333.0	52355.0
Jessica	128294.0	45278.0	44265.0	24653.0	36460.0	30891.0	70665.0	43756.0
Jason	104688.0	35621.0	44467.0	23439.0	43971.0	25268.0	69017.0	50371.0
Andrew	116597.0	32300.0	44569.0	27449.0	41568.0	28062.0	68847.0	49187.0
Anthony	131214.0	41169.0	49394.0	20216.0	33355.0	37995.0	91305.0	40732.0
Brian	96750.0	29628.0	51739.0	28022.0	38858.0	34490.0	70418.0	52584.0
Ryan	98980.0	32934.0	41729.0	24345.0	37161.0	27486.0	51100.0	41863.0
Kevin	99965.0	31145.0	43310.0	24911.0	32541.0	31248.0	69478.0	36228.0
Ashley	86398.0	41656.0	32856.0	15299.0	27777.0	19556.0	43831.0	35668.0
Thomas	59525.0	27653.0	40866.0	25355.0	31834.0	33657.0	73013.0	39072.0
Sarah	75972.0	28108.0	35391.0	22556.0	33316.0	16650.0	45705.0	39001.0
Nicholas	78214.0	31547.0	40117.0	24991.0	33032.0	28587.0	63446.0	39284.0

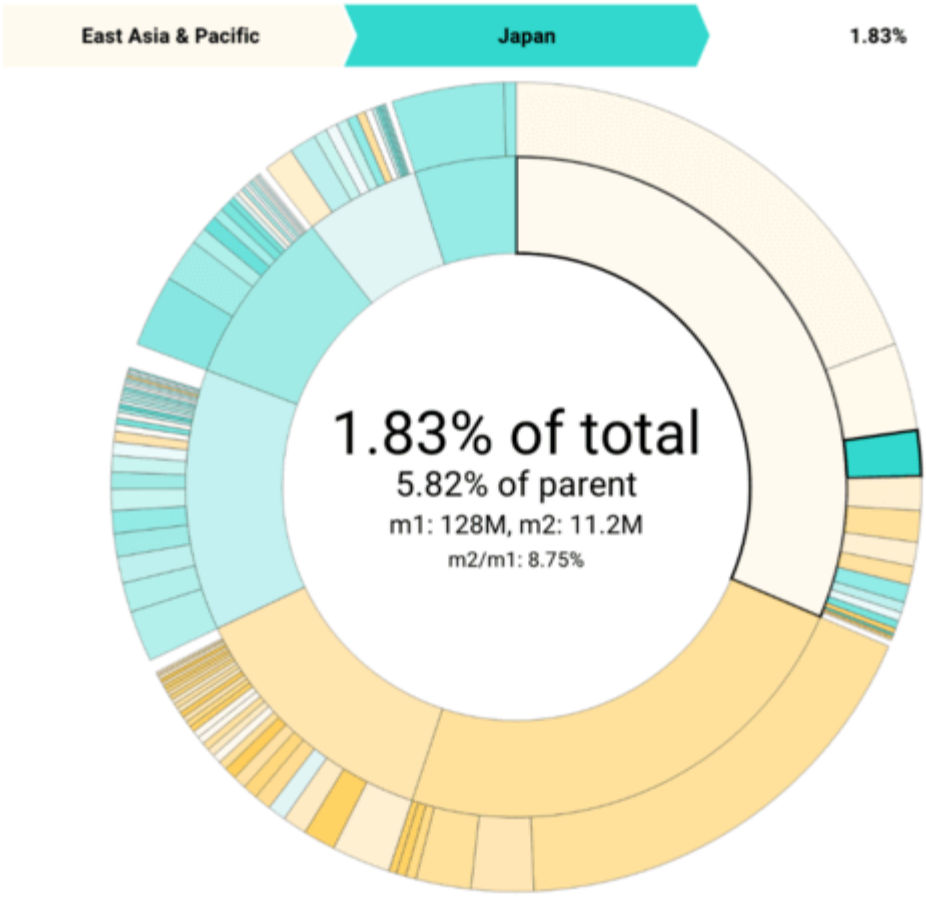

2004
2003
2005
2002
2006
2001
2007
2014
2008
2013
2009
2012
2010
2011


Energy
Land Use Change
Industrial Processes
Agriculture
Waste
Electricity and heat
Transportation
Other Fuel Combustion
Industry
Fugitive Emissions
Deforestation
Harvest / Management
Livestock and Manure
Rice Cultivation
Other Agriculture
Agriculture Soils
Landfills
Waste water - Other Waste
Road
Residential Buildings
Commercial Buildings
T and D Losses
Machinery
Rail - Ship and Other Transport
Air
Agricultural Energy Use
Pulp - Paper and Printing
Unallocated Fuel Combustion
Iron and Steel
Other Industry
Food and Tobacco
Oil and Gas Processing
Chemicals
Coal Mining
Aluminium Non-Ferrous Metals
Cement
Carbon Dioxide
HFCs - PFCs
Methane
Nitrous Oxide

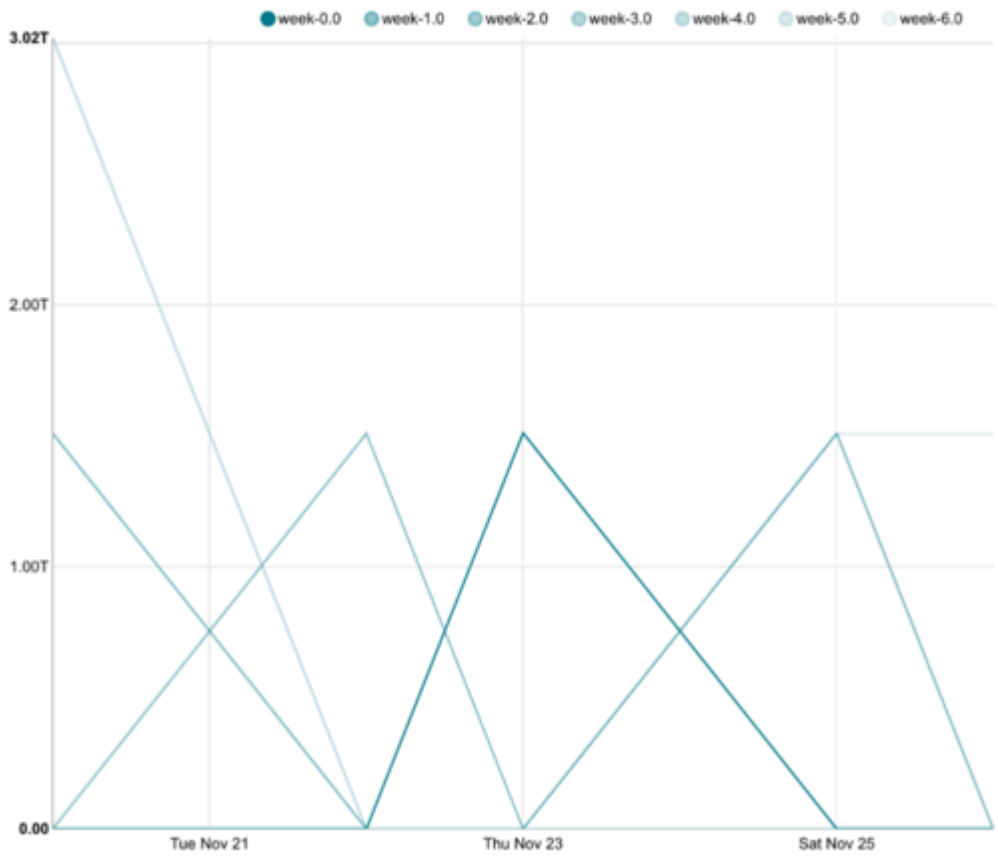
First Section

This is an paragraph that explains what you should expect to fir

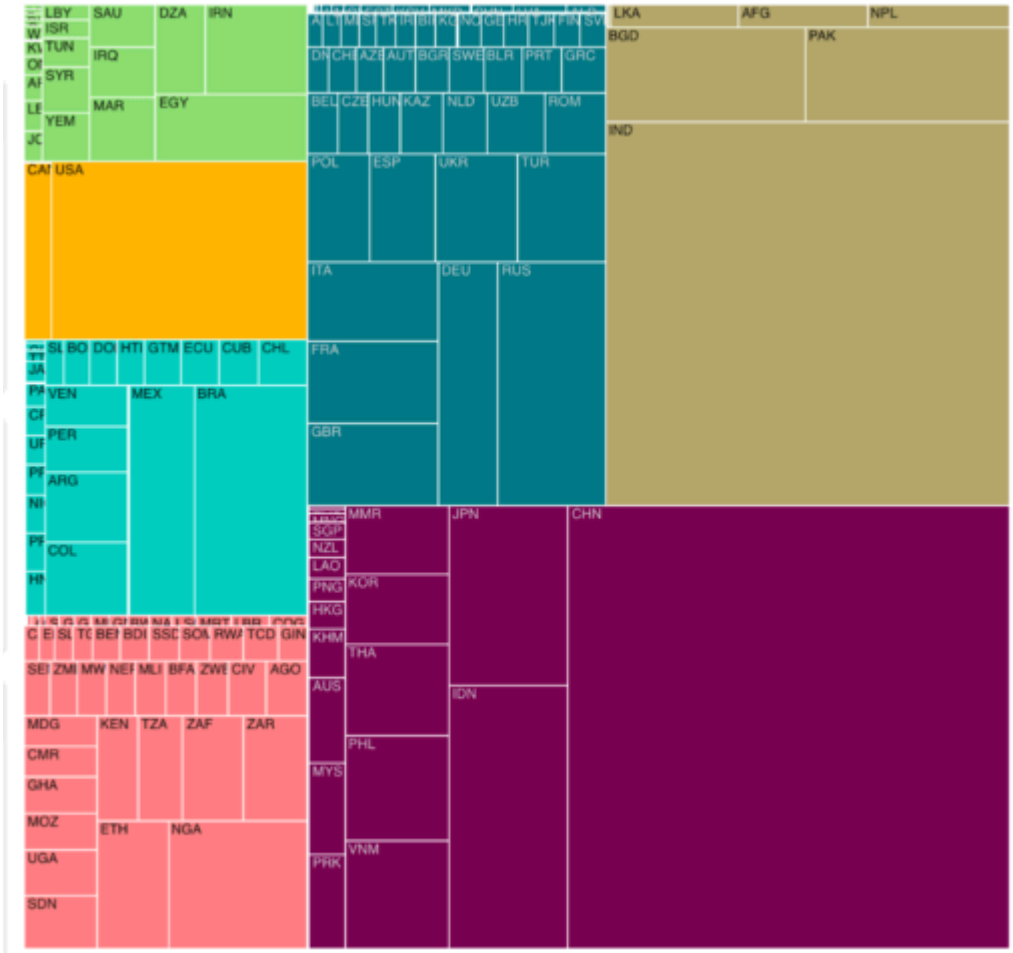



East Asia & Pacific
Japan
1.83%
1.83% of total
5.82% of parent
m1: 128M, m2: 11.2M
m2/m1: 8.75%

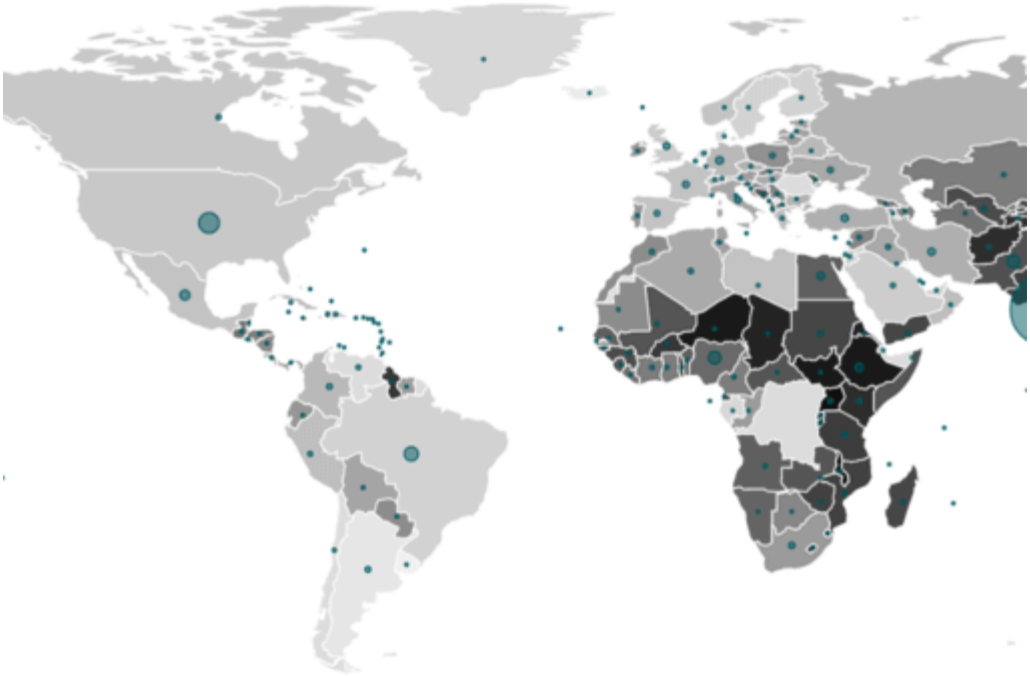
country_name	sum__SH_DYN_AIDS	sum__SP_POP_TOTL	sum__SP_RUR_TOTL
China	0.00	1.36G	622M
India	0.00	1.30G	876M
United States	0.00	319M	59.2M
Indonesia	640k	254M	120M
Brazil	0.00	206M	30.0M
Pakistan	92.0k	185M	114M
Nigeria	3.00M	177M	94.2M
Bangladesh	8.60k	159M	106M
Russian Federation	0.00	144M	37.5M
Japan	0.00	127M	8.87M
Mexico	190k	125M	26.4M
Philippines	35.0k	99.1M	55.0M
Ethiopia	620k	97.0M	78.5M
Vietnam	240k	90.7M	60.8M
Egypt, Arab Rep.	8.60k	89.6M	51.0M
Germany	0.00	80.9M	20.1M
Iran, Islamic Rep.	73.0k	78.1M	21.2M
Turkey	0.00	75.9M	20.6M
Congo, Dem. Rep.	390k	74.9M	43.4M
Thailand	440k	67.7M	34.4M
France	0.00	66.2M	13.7M
United Kingdom	0.00	64.5M	11.4M
Italy	0.00	61.3M	19.1M
South Africa	6.50M	54.0M	19.3M
Myanmar	200k	53.4M	35.5M
Tanzania	1.40M	51.8M	35.8M
Korea, Rep.	0.00	50.4M	8.89M


week-0.0
week-1.0
week-2.0
week-3.0
week-4.0
week-5.0
week-6.0
3.02T
2.00T
1.00T
0.00
Tue Nov 21
Thu Nov 23
Sat Nov 25

Metric	Current	YoY	10 y ago	10 AVG	SPARK
other	491k	+10.39%	757k	627k	
CA	161k	+8.07%	209k	187k	
TX	110k	+4.82%	135k	125k	
NY	72.0k	+10.03%	115k	92.0k	
FL	63.3k	+10.61%	81.9k	74.0k	
IL	44.6k	+14.33%	76.1k	61.4k	
PA	42.1k	+9.55%	71.5k	55.8k	
OH	38.8k	+9.52%	72.2k	54.7k	


LBY
SAU
DZA
IRN
ISR
TUN
IRQ
SYR
MAR
EGY
YEM
USA
LKA
AFG
NPL
BGD
PAK
IND
POL
ESP
UKR
TUR
ITA
DEU
RUS
FRA
GBR
VEN
MEX
BRA
PER
ARG
COL
MMR
JPN
CHN
KOR
THA
IDN
PHL
VNM
PRK
MYS
AUS
KHM
HKG
PNG
LAO
NZL
MDG
KEN
TZA
ZAF
ZAR
CMR
GHA
MOZ
UGA
SDN
ETH
NGA


Michael
Christopher
David
James
John
Robert
Jennifer
William
Joseph
Matthew
Daniel
Joshua
Andrew
Jason
Thomas
Jessica
Brian
Kevin
Amanda
Justin
Brandon
Stephanie
Ryan
Eric
Nicholas
Jacob
Michelle
Emily
Sarah
Nicole
Kimberly
Elizabeth
Melissa
Jeffrey
Tyler
Lisa
Richard
Samantha
Ashley
Anthony
Jonathan
Mark
Steven
Scott
Amy
Timothy
Charles
Heather
Angela
Benjamin



Time

In this section, users can enter the metrics related to charts that contain time parameters.

Query

The Query section changes based on the visualization type chosen, though this is generally where users will enter metrics and parameters such as group by, sort, filter, series, etc.

Chart Options

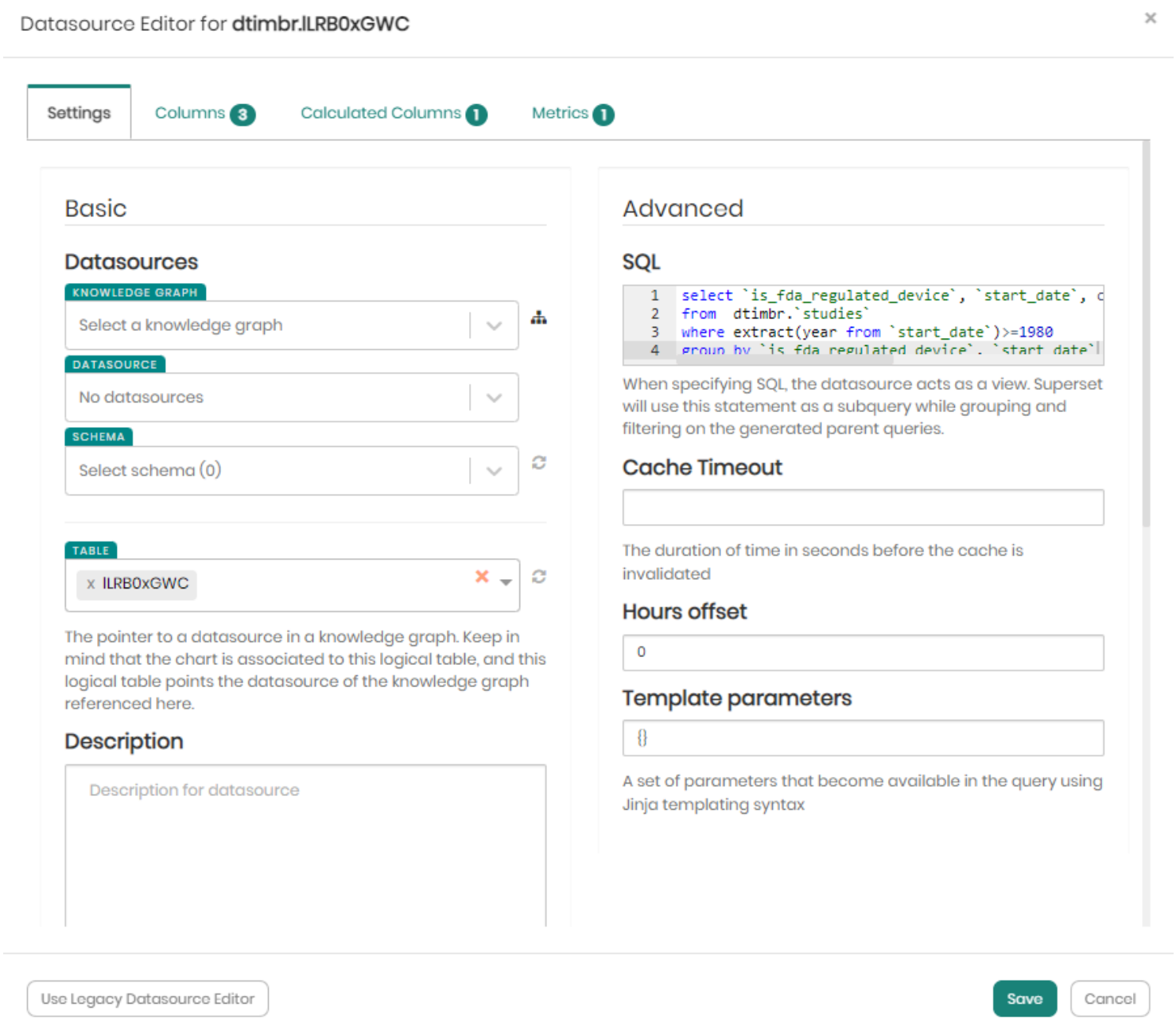
Some visualization types will include this section which will add some additional editing options to the chart.

The second tab in the chart Query builder panel is **Visual properties** which includes the following section:

Chart Options

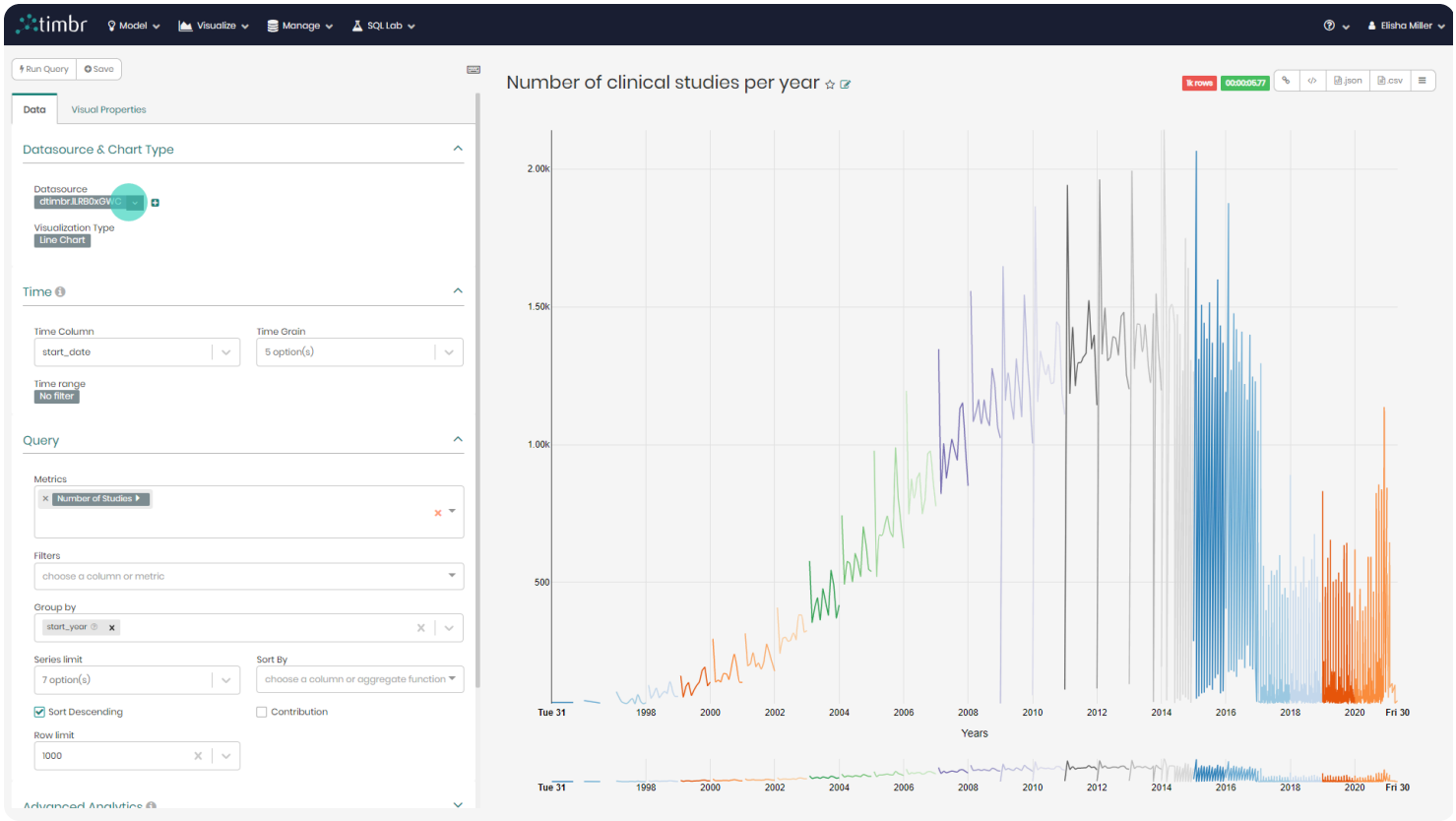
Some visualization types will include this section in the first tab while others will add it here in the second tab. This section will add additional editing options to the chart such as the ability to edit the chart colors, add legends, bar values, rename X and Y axis and more.

Modifying Chart Datasource



Users can perform different modifications to the datasources used to create their charts, this includes access control and additional settings.

Opening the charts edit datasource window can be done by choosing the desired chart and clicking on the datasource that appears in the *Data* tab, under *Datasources & Chart Type*.

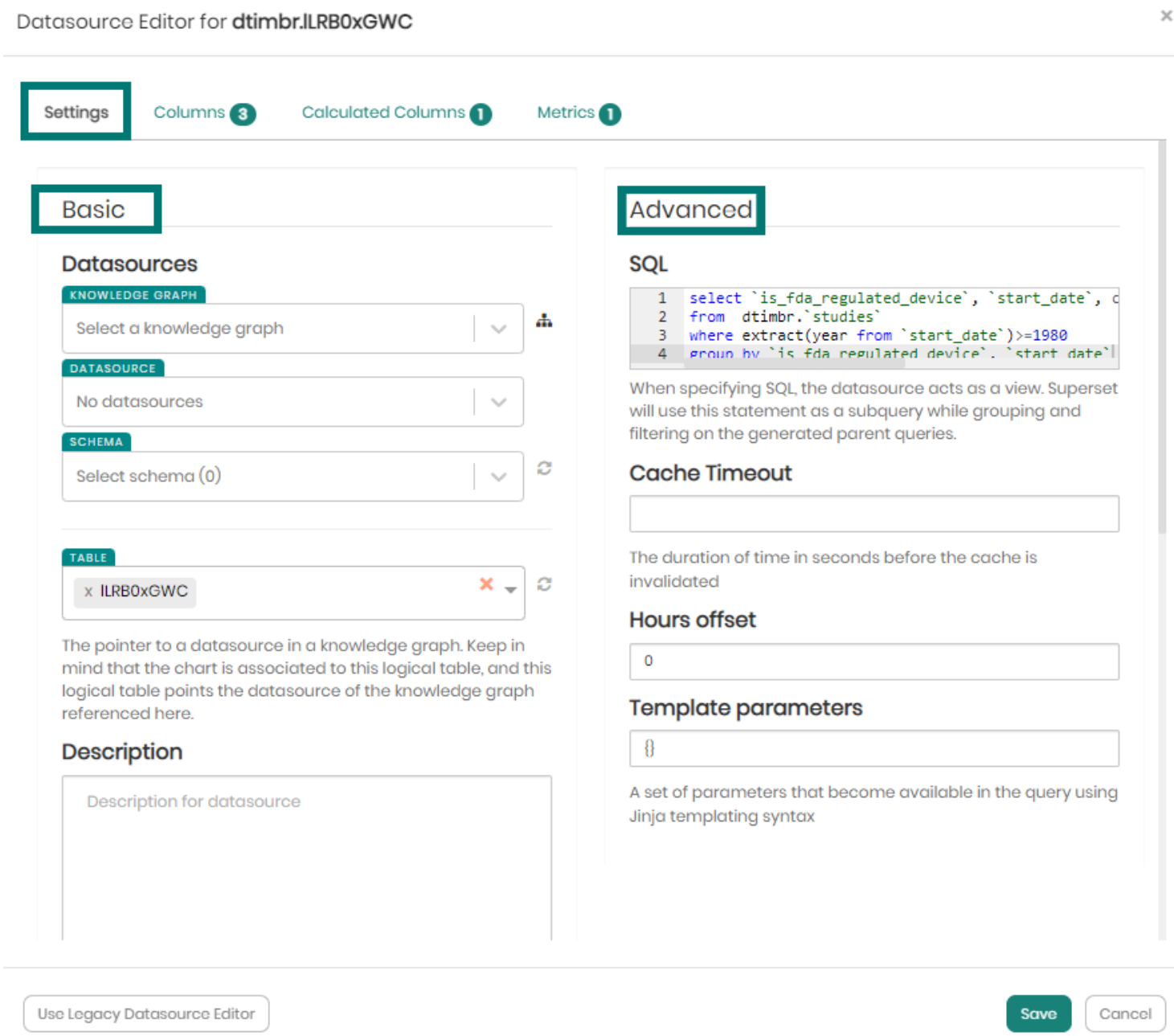


In the popup window that appears, there are four tabs which are:

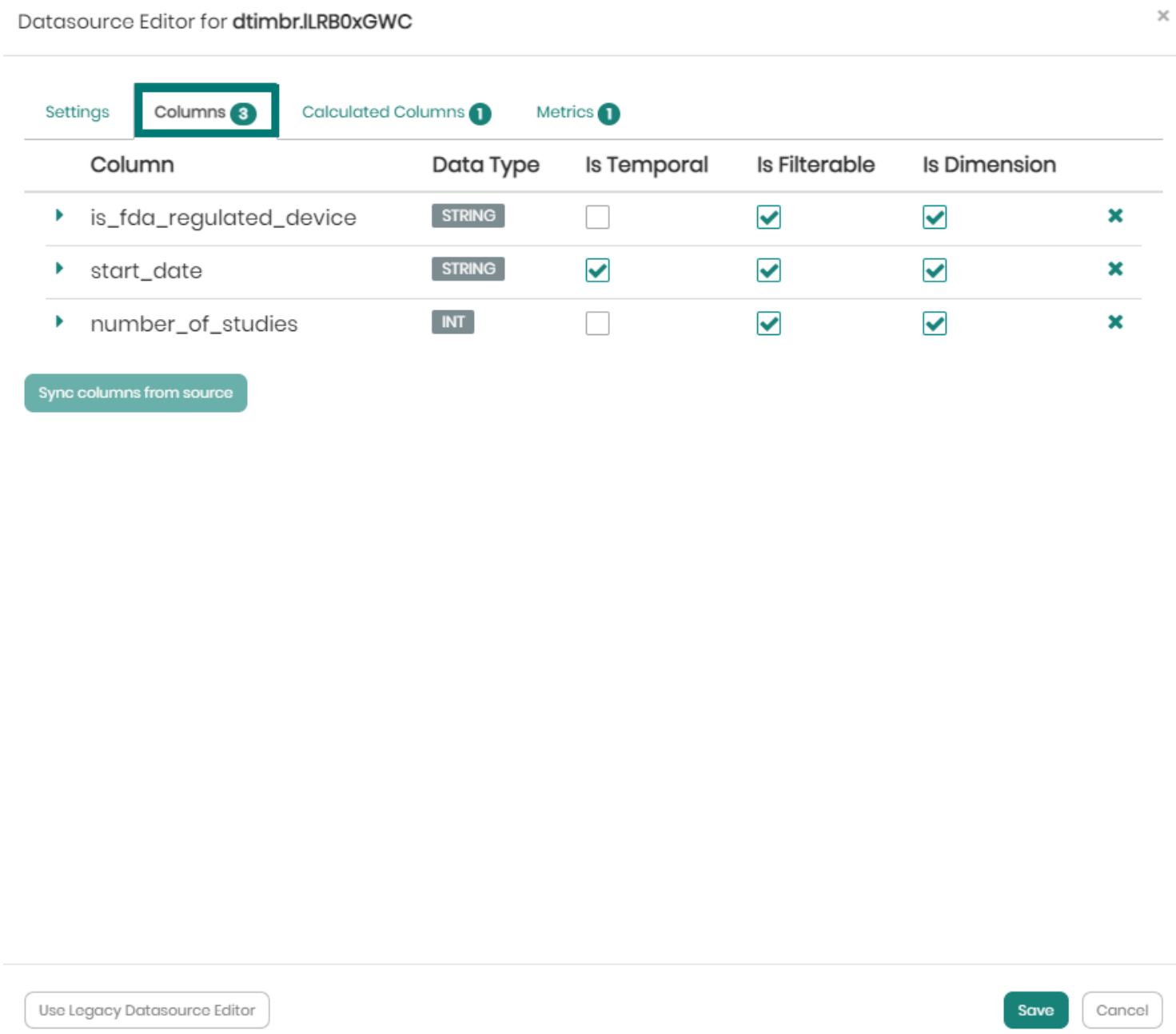
- **Settings** - In the Settings tab, users can find the main datasource settings, both the basic settings and the advanced settings.

  Basic Settings - In the Basic Settings on the left, users can study their datasource and other data sources more in-depth. When scrolling down, users can also add a default URL to their datasource, they can add autocomplete filters or autocomplete query predicates, and they can choose who the owners of the datasource will be.

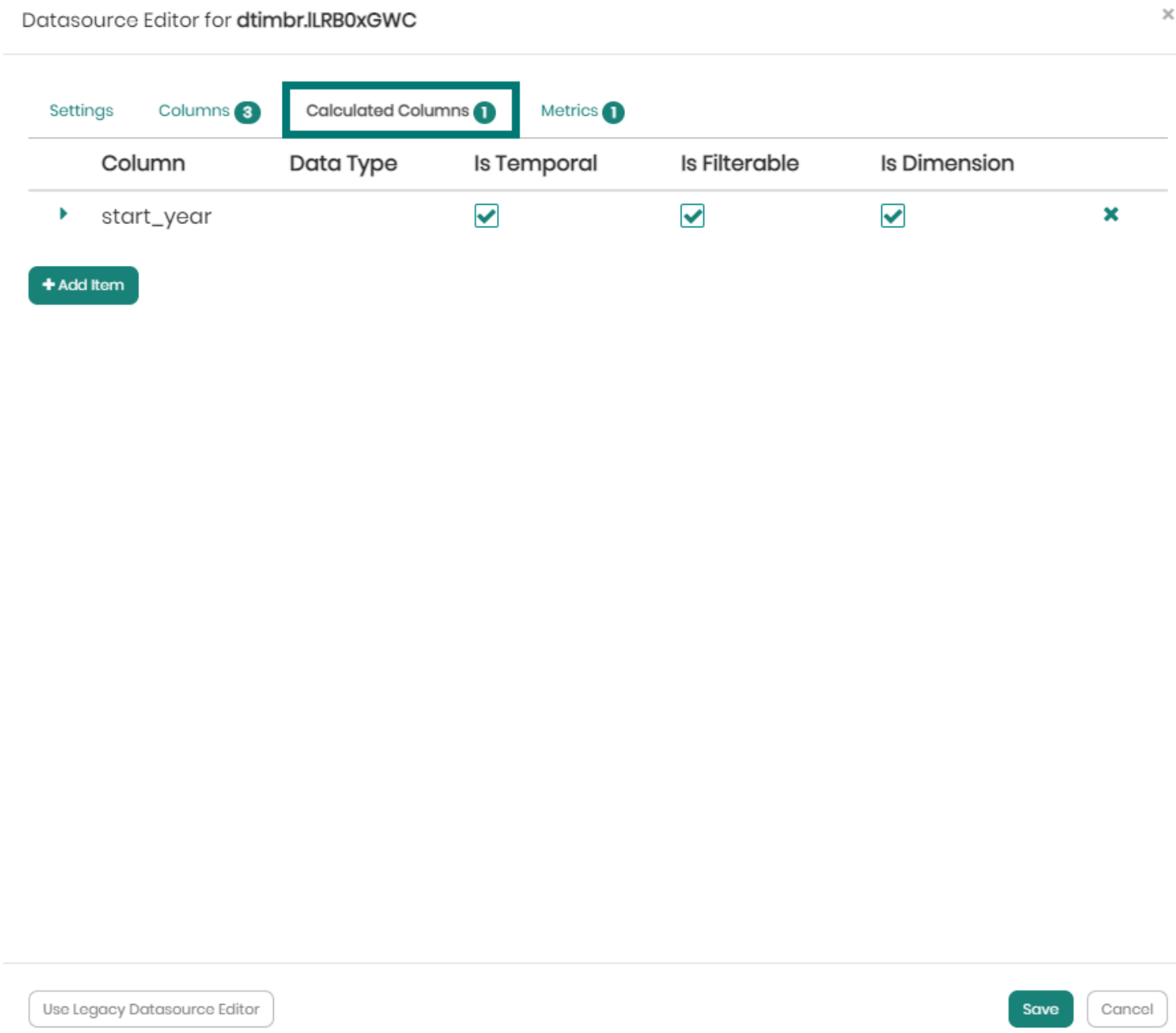
  Advanced Settings - On the right side in Advanced Settings, users can edit the SQL query behind the chart, creating a view out of the datasource, users can also edit the duration of time before the cache times out, they can enter the offset in hours, and finally, users can set parameters using Jinja templating syntax.



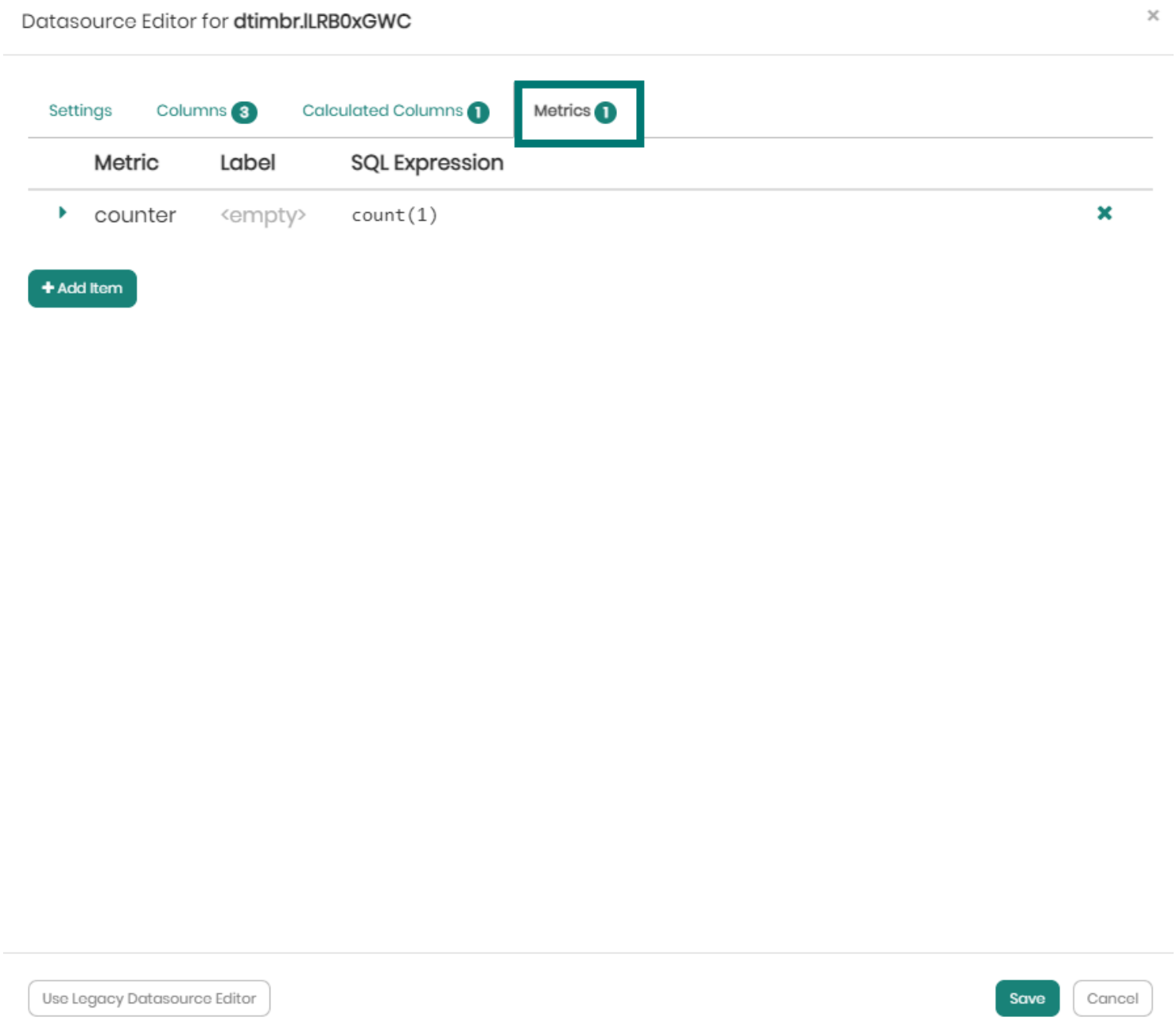
- **Columns** - In the Columns tab, users have the ability to edit the different characteristics of the datasource columns including their label, description, format, and their expression in the database.



- **Calculated Columns** - In the Calculated Columns tab, users can click to add new calculated columns to be represented in their charts.



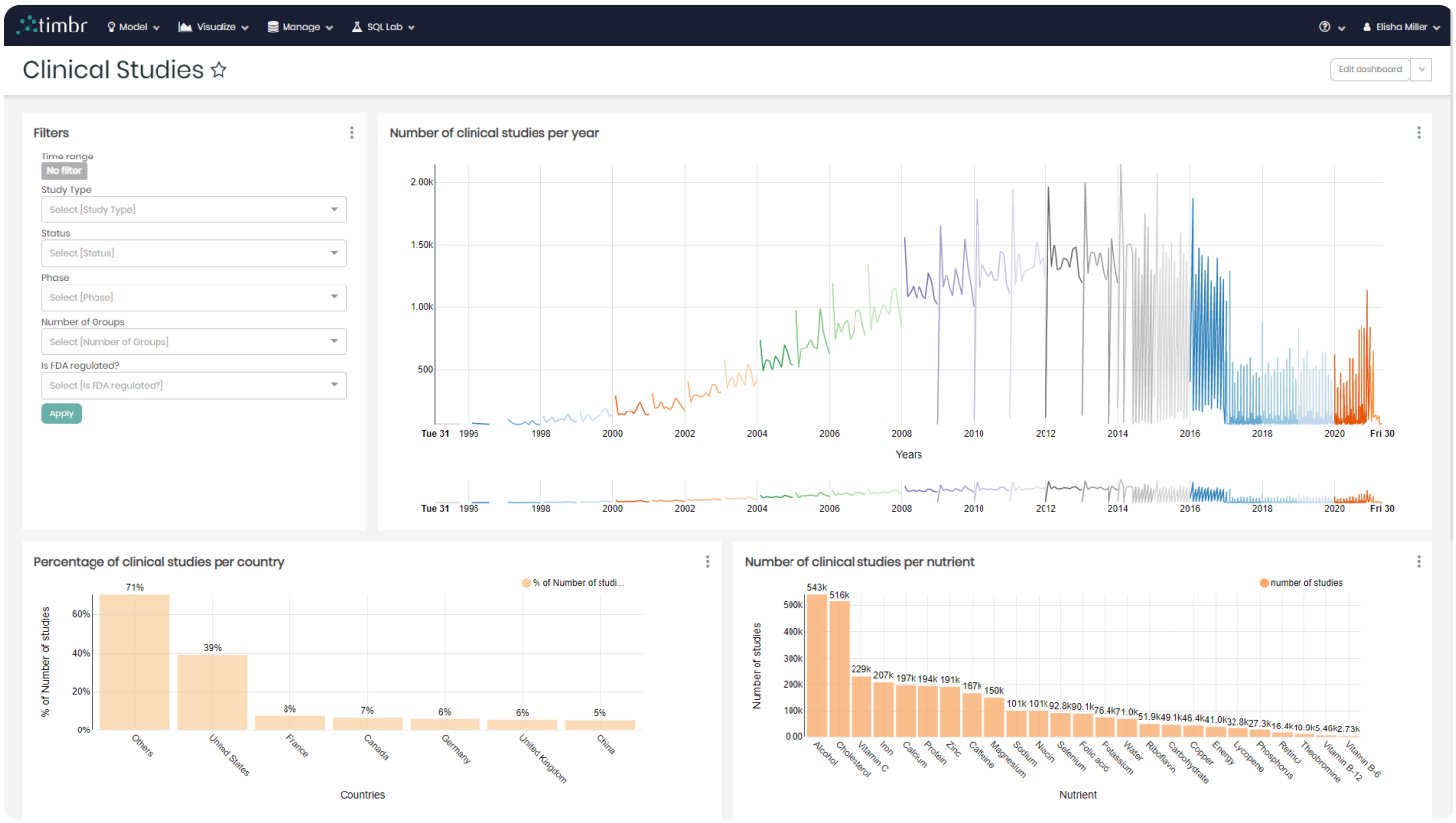
Metrics - In the Metrics tab, users will be presented with all the metrics they currently have available to them, and will be able to click and create new metrics to be used in their charts.



Once changes are made, Save must be clicked on the bottom right of the window in order to save all the changes.

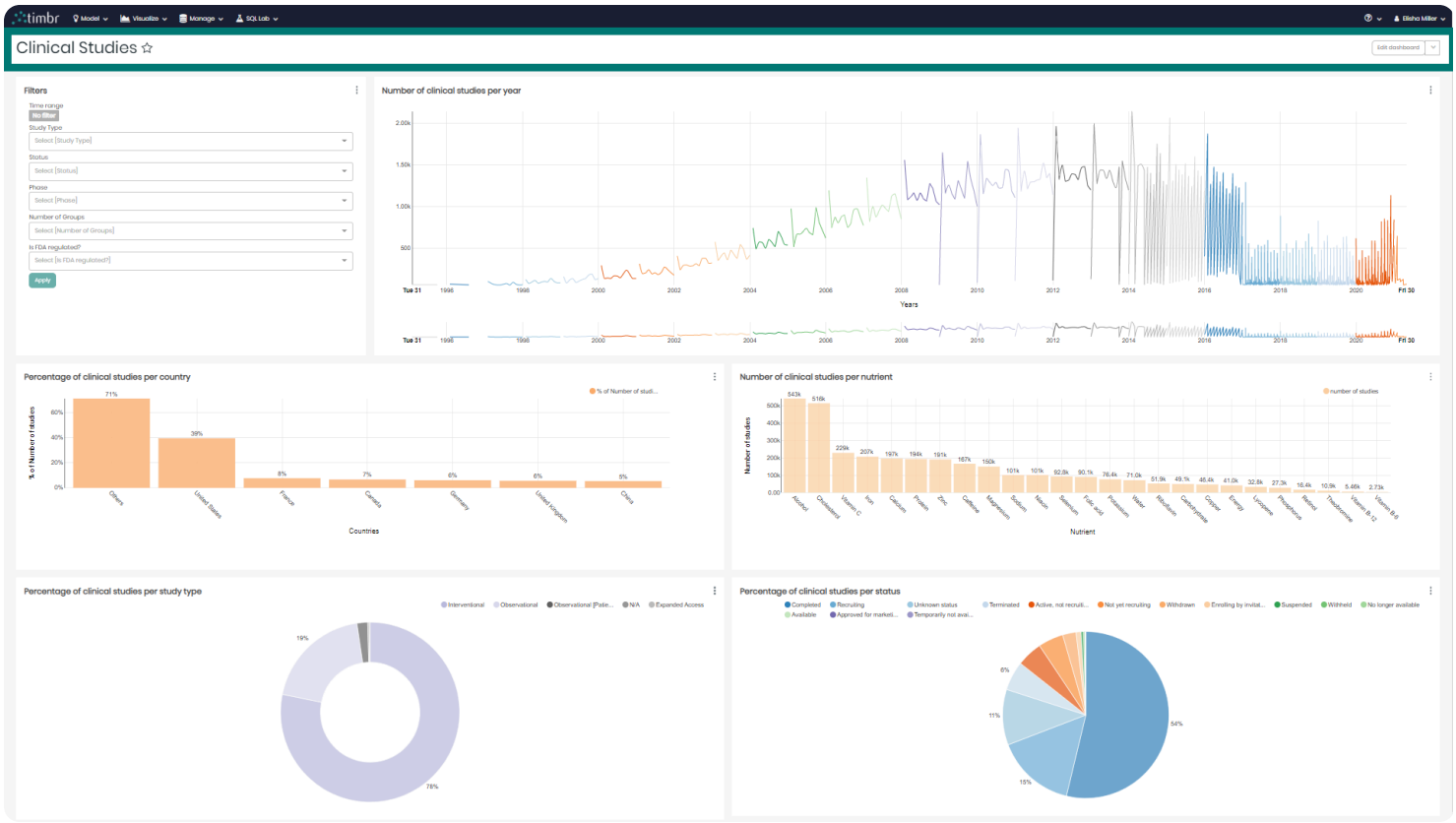
BI Dashboards

The dashboard page in Timbr enables users to create interactive dashboards and expose them to any user. The dashboard can be defined with automated refresh intervals, designed with customized CSS templates, and can include multiple tabs, charts, and filters.



The dashboard component in charge of editing and presenting knowledge graph charts can be found in the **visualize tab** and contains the following components:

Top Menu



On the top left, users can see the name of the selected dashboard and can choose to add or remove the dashboard from the favorites by clicking on the star.

On the top right, is **Edit dashboard** that when clicked on offers the following editing options:

Your charts & filters - Enables users to add previously created charts and filters to the current dashboard.

Tabs - Allows users to add new tabs to the current dashboard.

Row - Allows users to add new rows to the current dashboard.

Column - Allows users to add new columns to the current dashboard.

Header - Allows users to add new headers to the current dashboard.

Markdown - Enables users to edit their dashboard using markdown syntax.

Divider - Allows users to add dividers to the current dashboard.

The stated options can be hidden by clicking on **Hide components** located on the top right.

All charts in the dashboard can be dragged, resized as desired, and even deleted by clicking on the small trash can on the top right of each individual chart.

To the left of hide components, users can find the **Undo/Redo** buttons in order to undo/redo any edits done on the dashboard.

To the right of hide components is **Switch to view mode**, which will close the editing components and return to the dashboard.

Connected to Switch to view mode is a downwards pointing arrow which when clicked on offers the following options:

Save as - Allows users to save the current dashboard.

Force refresh dashboard - Forces a refresh of the entire dashboard in case there have been any changes to the data reflected in the dashboard charts.

Set auto-refresh interval - Enables users to set an automated refresh to the dashboard after a specified time frame.

Edit dashboard metadata - Enables users to edit the metadata behind the dashboard. This includes changes to the dashboard name, owners, JSON, CSS and more.

Share dashboard - Generates a shareable URL that can be sent to colleagues via a link or by Email in order to share the current dashboard.

Edit CSS - Enables users to edit and style the dashboard using CSS and CSS templates.

Once finished editing and clicking on Switch to view mode, users will find another downward pointing arrow next to Edit dashboard that when clicked on offers some of the same options being:

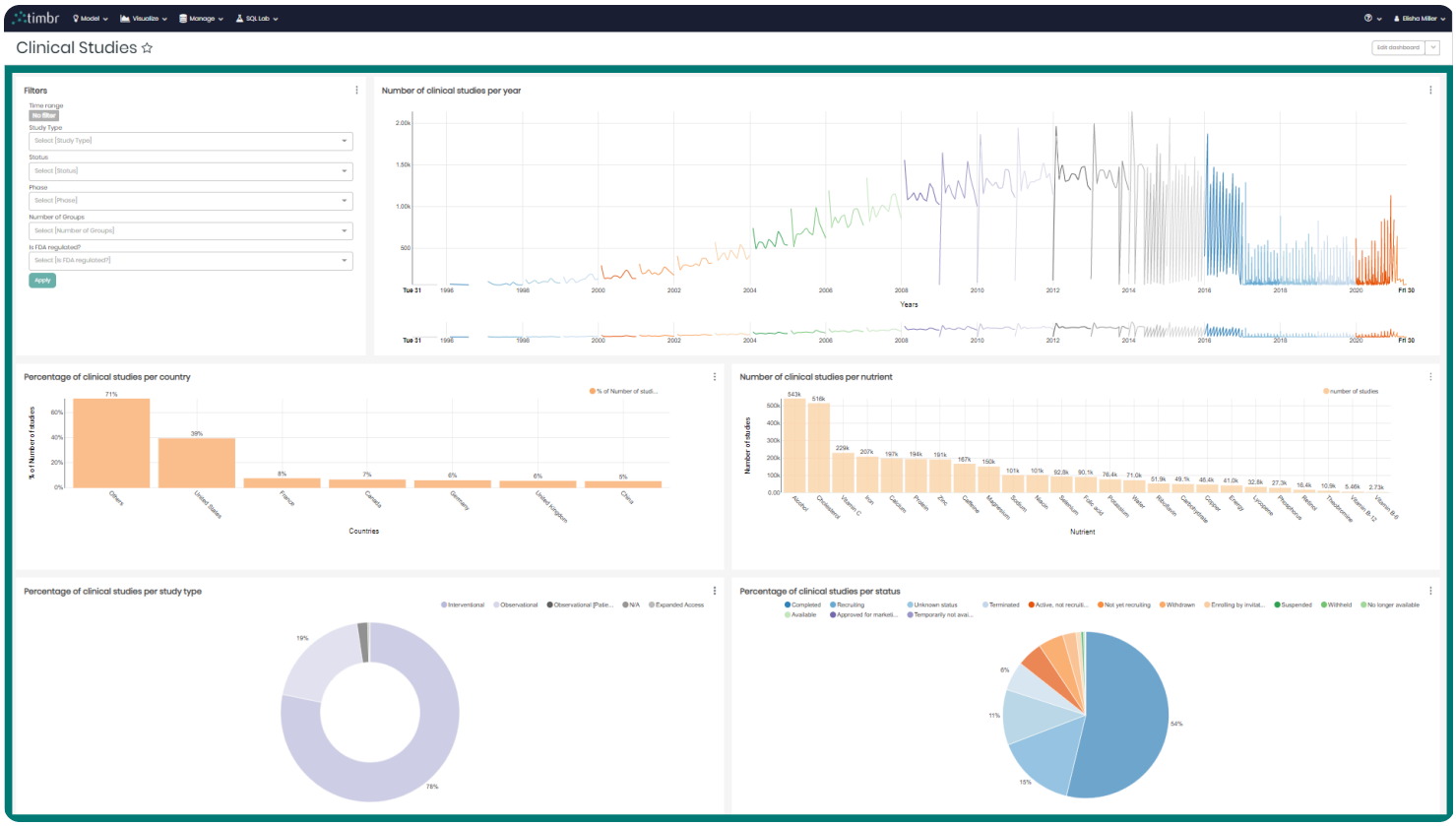
Save as - Allows users to save the current dashboard.

Force refresh dashboard - Forces a refresh of the entire dashboard in case there have been any changes to the data reflected in the dashboard charts.

Set auto-refresh interval - Enables users to set an automated refresh to the dashboard after a specified time frame.

Share dashboard - Generates a shareable URL that can be sent to colleagues via a link or by Email in order to share the current dashboard.

Dashboard Charts



The main bulk of the dashboard component space is dedicated to the dashboard charts that can be seen below the top menu. Users can scroll up and down to view all the charts in the dashboard as the dashboard expands.

Charts in the dashboard can be interacted with by hovering over the different charts as well as clicking on the charts.

In addition, each chart contains 3 vertical dots on the top right of the chart that when clicked on offer the following:

Force refresh - Forces a refresh on the specific chart selected, refreshing in case any of the chart's underlying data has been altered.

Edit chart metadata - Enables users to edit the metadata behind the selected chart. This includes changes to the chart name, owners, JSON, CSS and more.

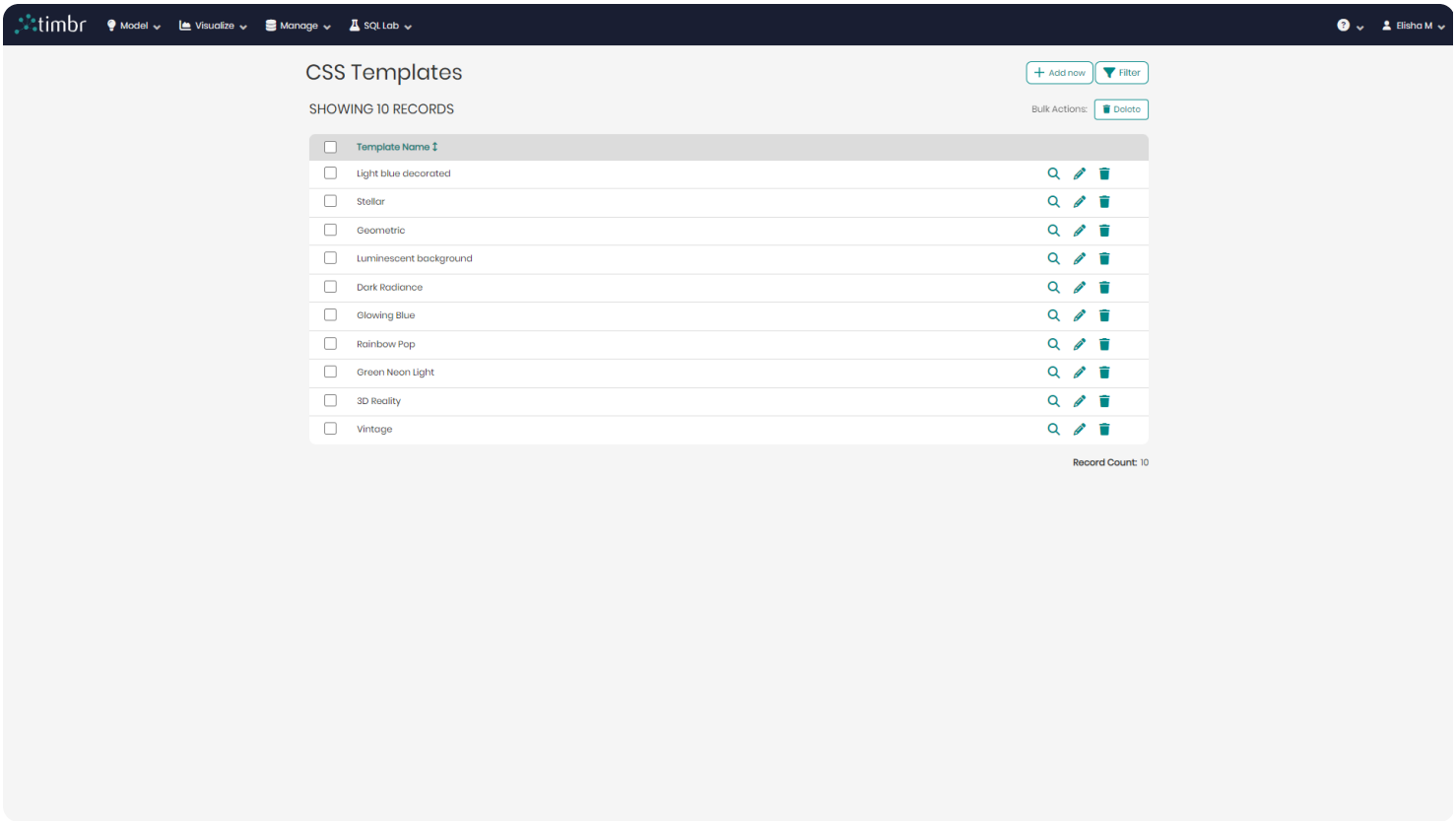
Export CSV - Enables users to export the data representing the selected chart into a CSV file.

Explore chart - When clicked on, the selected chart will open in Timbr's built-in charts component in order for users to explore and edit the chart further if necessary.

Share chart - Generates a shareable URL that can be sent to colleagues via a link or by Email in order to share the selected chart.

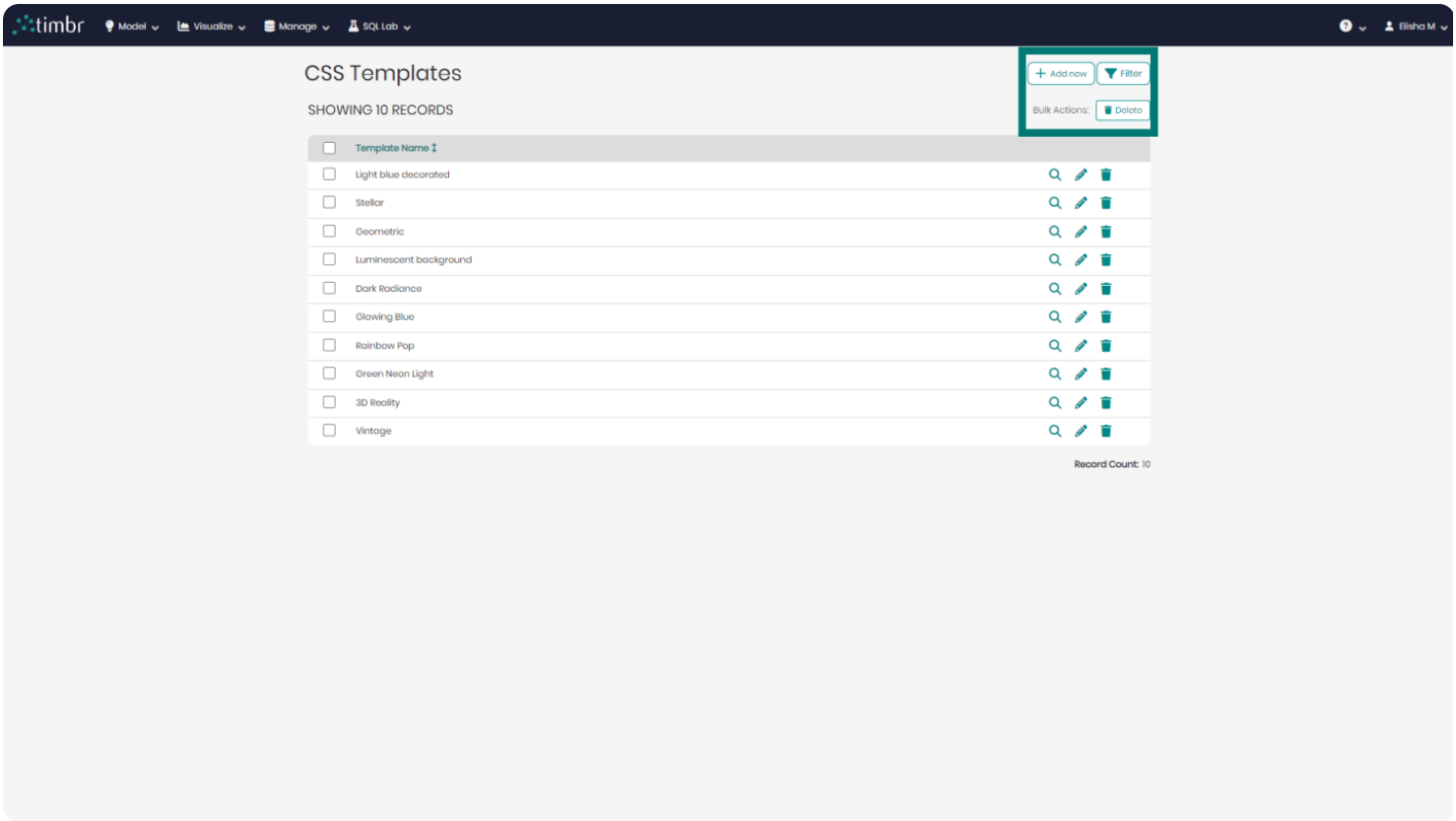
CSS Templates

CSS Templates page enable users to define or upload their own CSS designs and apply them on customized dashboards.



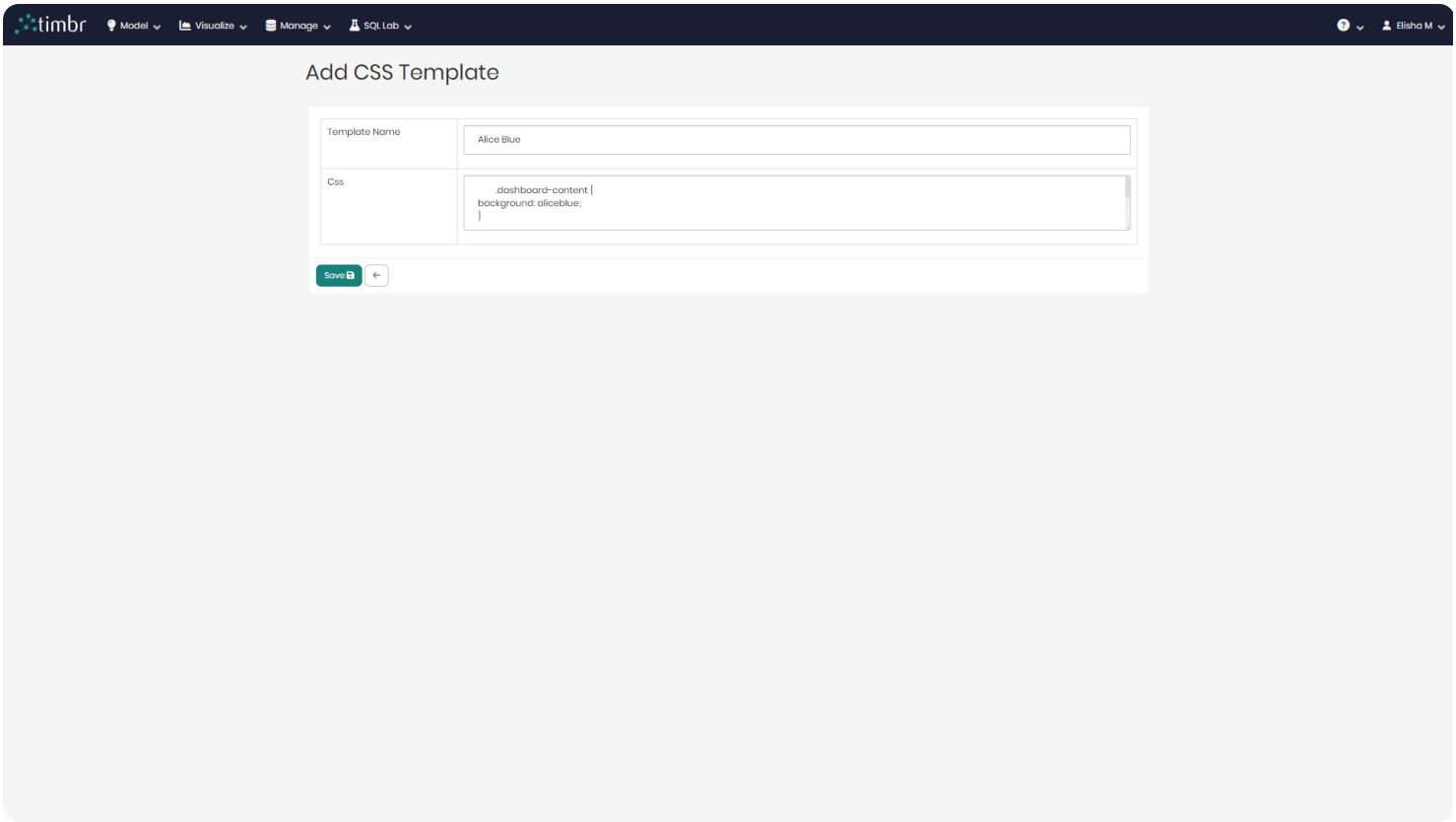
The **CSS Templates** can be accessed through the **Visualize** tab and contains the following components:

Top right actions

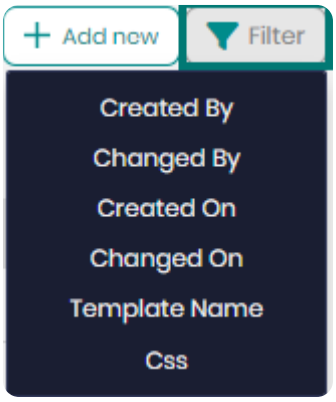


When entering the CSS Templates page, on the top right you will see the following 3 options:

Add new - When clicked on, a new screen will open up in order to create a new CSS Template by entering a name for the CSS template and entering the CSS syntax.

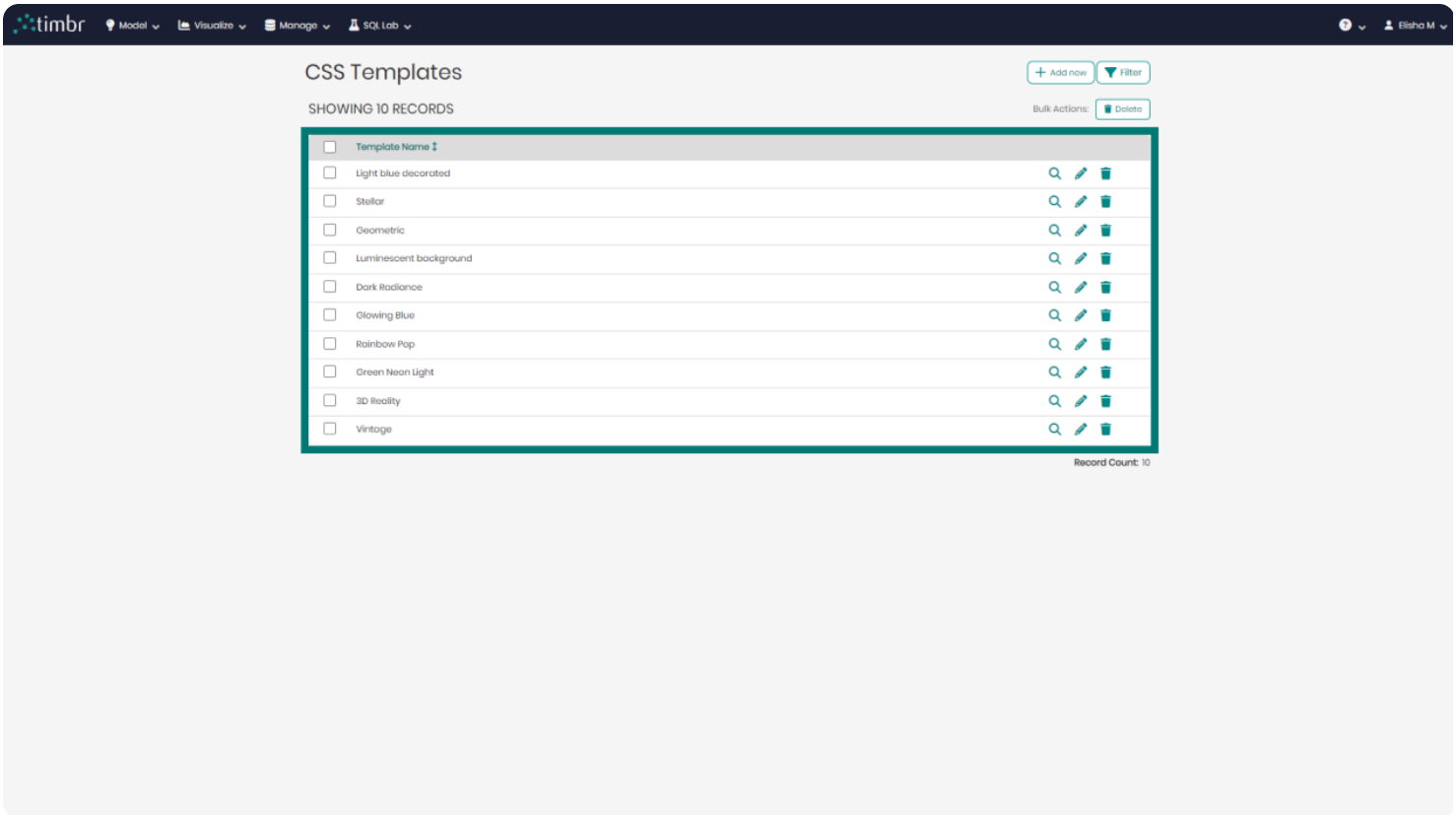


Filter - When clicking on *Filter* a small pop-up will appear with different filtering options including filtering by *Created By*, *Changed By*, *Created On*, *Changed On*, *Template name*, and *CSS*.



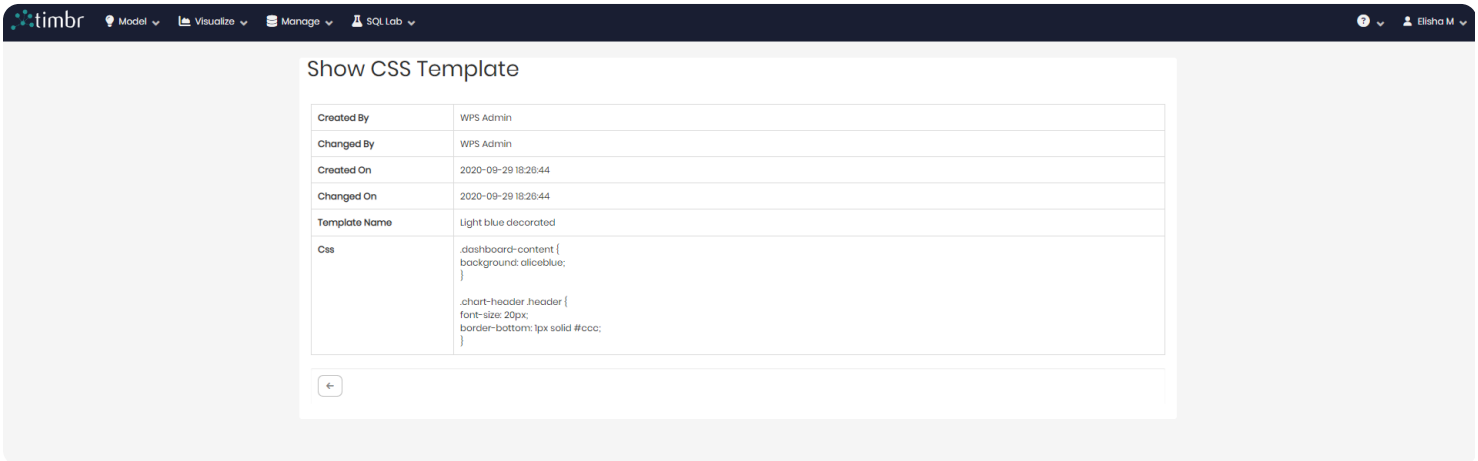
Bulk Actions - Bulk actions contain the delete button which enables users to choose multiple CSS Templates to delete at once.

CSS Template list

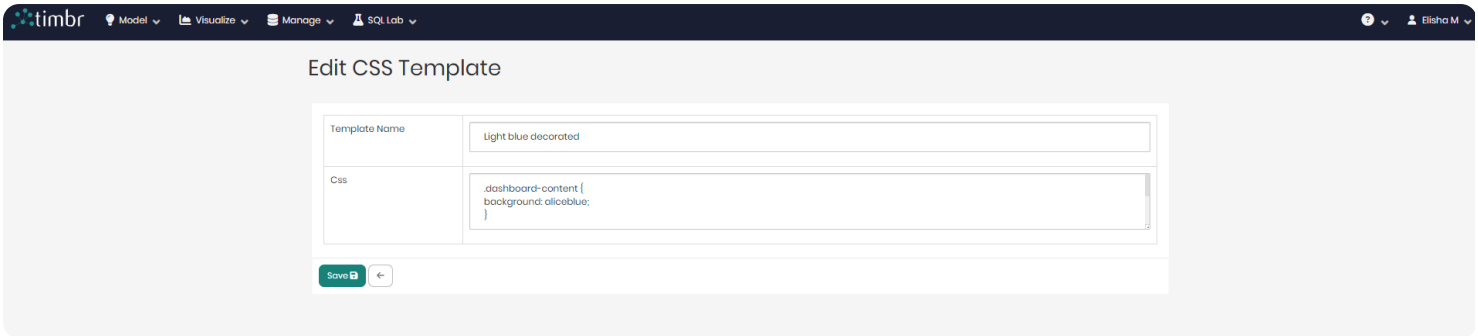


The CSS Template list in the center of the screen contains all the saved CSS Templates and their additional information. To the right of each CSS Template name are the following options:

Show CSS Template - Show CSS Template is presented by a *Magnifying glass* icon that when clicked on opens a window with additional information about the CSS Template.



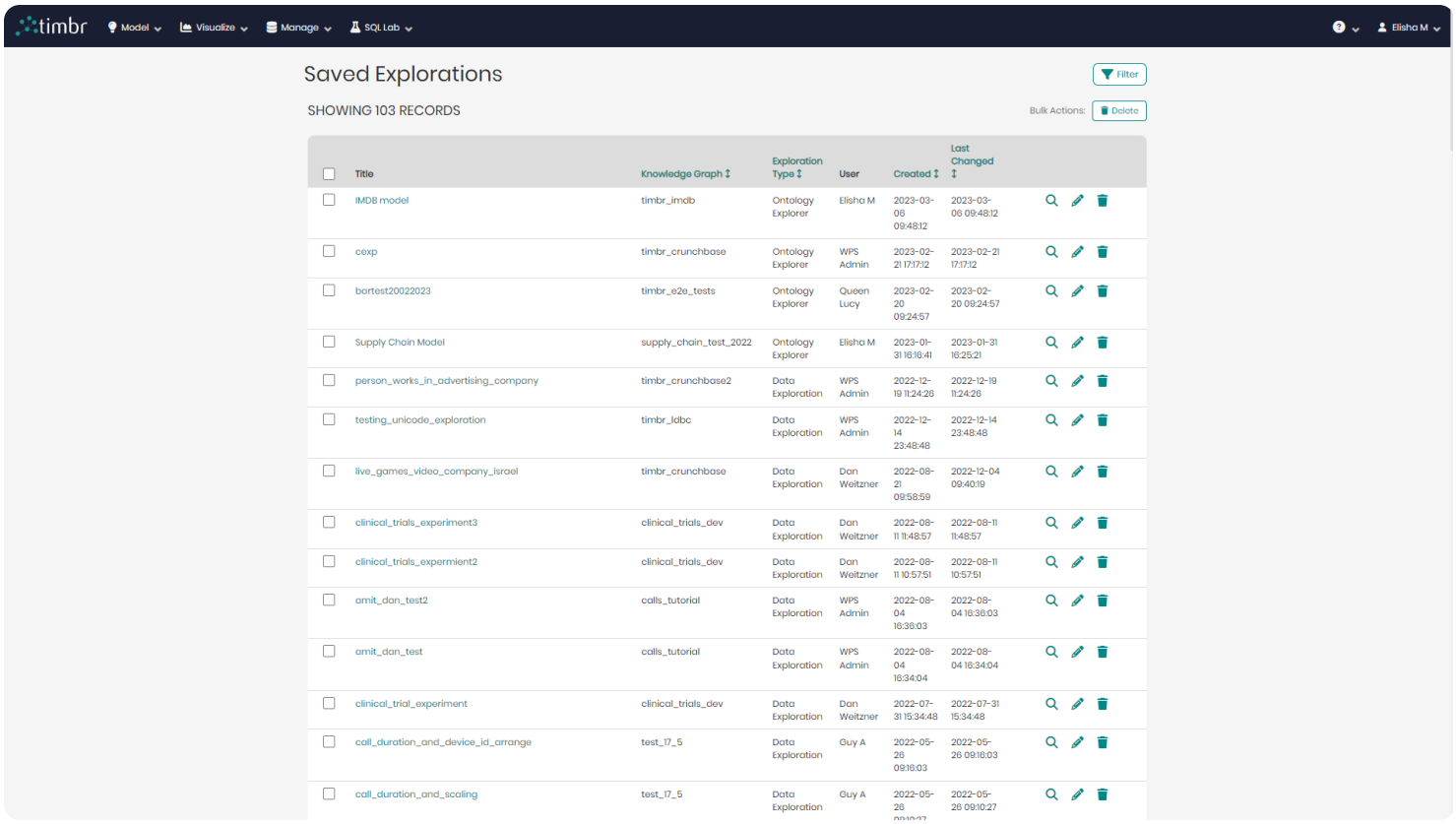
Edit CSS Template - Edit CSS Template is presented by a *Pencil* icon that when clicked on opens a window that enables editing the CSS behind the saved CSS Template.



Delete CSS Template - Delete CSS Template is presented by a *Trash can* icon that when clicked on deletes the selected CSS Template.

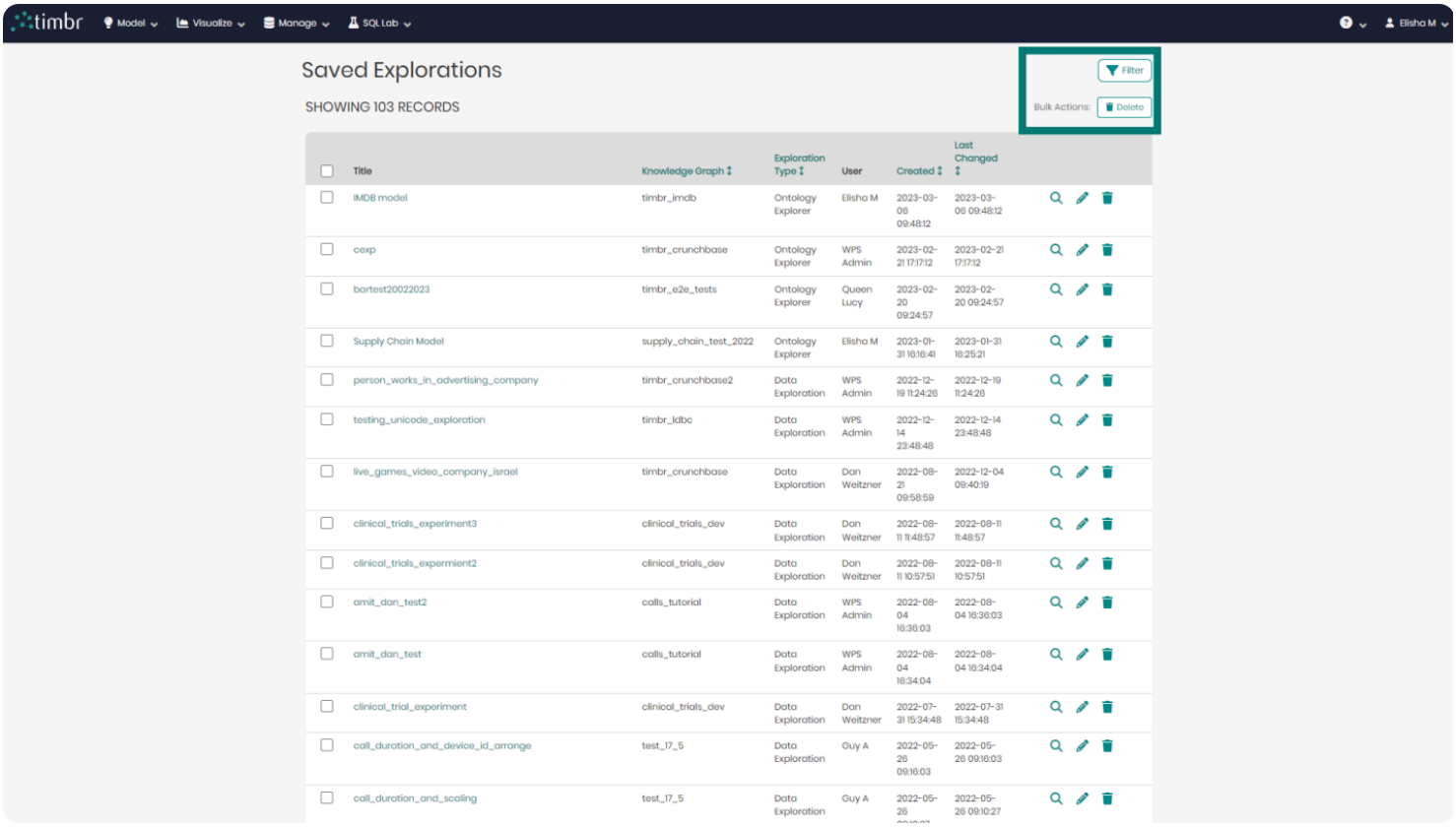
Saved Explorations

Saved Explorations page enables users to load saved ad-hoc explorations of the ontology graph in the **Ontology Explorer** page, or an exploration of a graph saved in the **Graph Explorer** page. Saved Explorations can be shared among users when having the proper permissions.



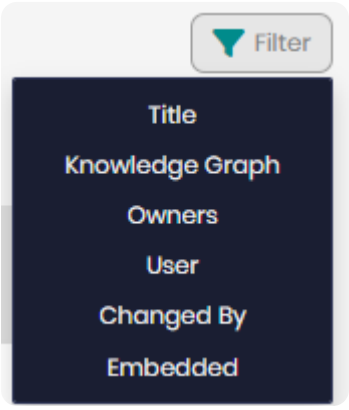
The **Saved Explorations** can be accessed through the **Visualize** tab and contains the following components:

Top right actions



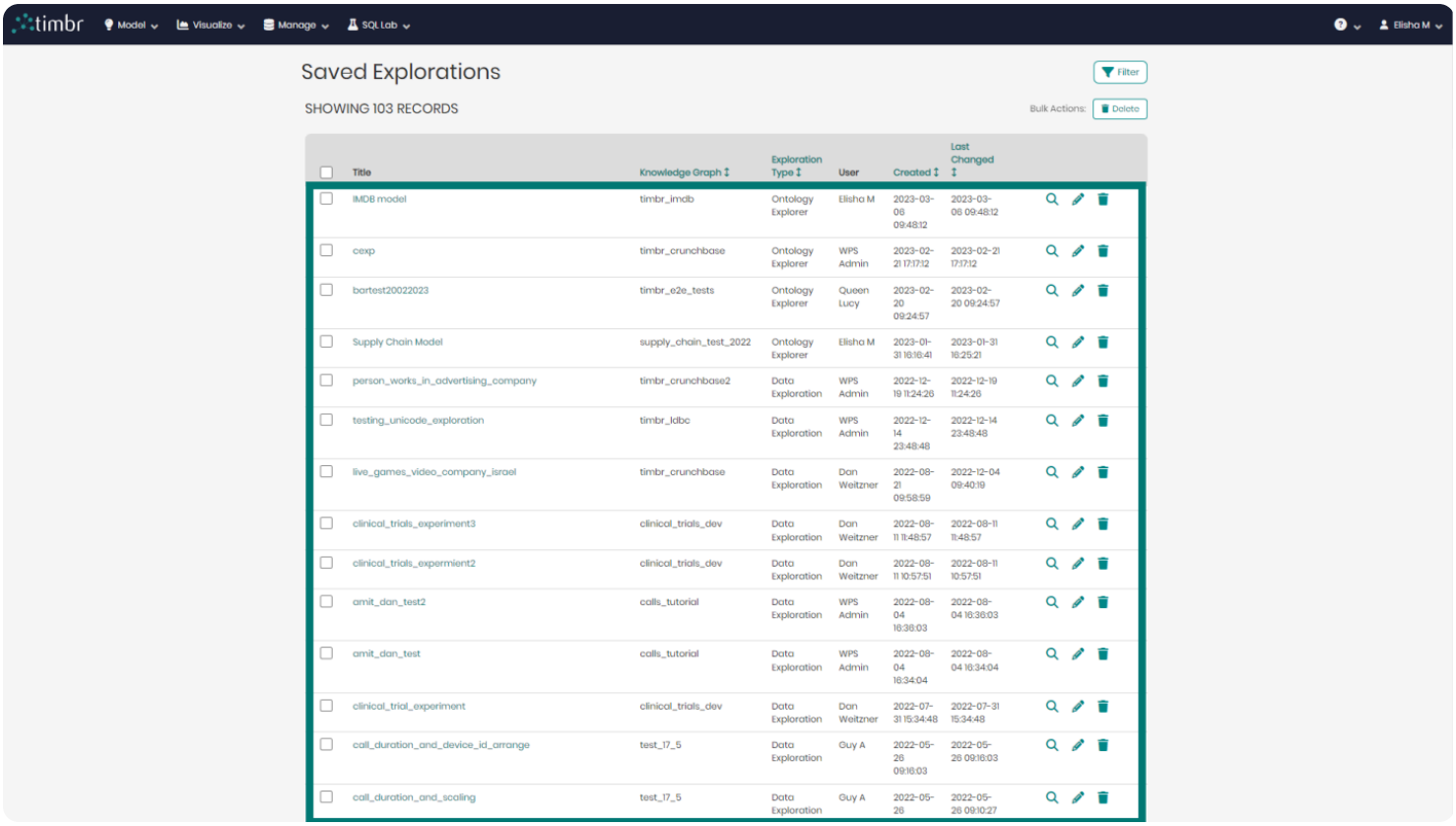
When entering the Saved Explorations page, on the top right you will see the following 2 options:

Filter - When clicking on *Filter* a small pop-up will appear with different filtering options including filtering by *Title*, *Knowledge Graph*, *Owners*, *User*, *Changed By*, and *Embedded*.



Bulk Actions - Bulk actions contain the delete button which enables users to choose multiple saved queries to delete at once.

Saved Exploration list



The saved exploration list in the center of the screen contains all the saved explorations and their additional information under the following columns:

Checkbox - Allows to click and choose specific saved explorations to perform bulk actions on.

Title - Presents the title that was given to each saved exploration. By clicking on a title, the exploration with the selected title will open in the platform for further exploration or editing.

Knowledge Graph - Presents the name of the knowledge graph behind the saved Exploration.

Exploration Type - Presents the type of saved exploration, specifying whether it is a saved *Ontology Exploration* or saved *Data Exploration*.

User - Presents the user behind each saved exploration.

Created - Presents the date on which the saved exploration was created.

Last Changed - Presents the last time the saved exploration was changed or modified.

Show Saved Exploration - Show saved exploration is presented by a *Magnifying glass* icon that when clicked on opens a window with additional information about the saved exploration.

Edit Saved Exploration - Edit saved exploration is presented by a *Pencil* icon that when clicked on opens a window that enables editing the JSON behind the saved exploration.

Delete Saved Exploration - Delete saved Exploration is presented by a *Trash can* icon that when clicked on deletes the selected saved query.

Knowledge Graphs

In the Knowledge Graphs page you can create, manage, edit, tag, and delete Knowledge Graphs. A Knowledge Graph in Timbr is a construct of a semantic data model (the ontology) and the data sources associated with it. The knowledge graph represents concepts and their relationships with one another. We use the data from the data sources to map to different concepts and relationships in the knowledge graph.

In a specific Knowledge Graph page, users can connect a specific data source or multiple data sources, define a virtualization engine, set an active data source, and change the data model settings according to user requirements.

DIFFERENCE BETWEEN ONTOLOGY AND KNOWLEDGE GRAPH

The difference between an ontology and a knowledge graph is that the ontology corresponds to the model of the knowledge graph.

Supported Back-ends:

Timbr supports full back-end integration to any relational database that is SQL / ANSI SQL compliant, or can be queried in SQL. The connection can be established either by a JDBC or ODBC connector.

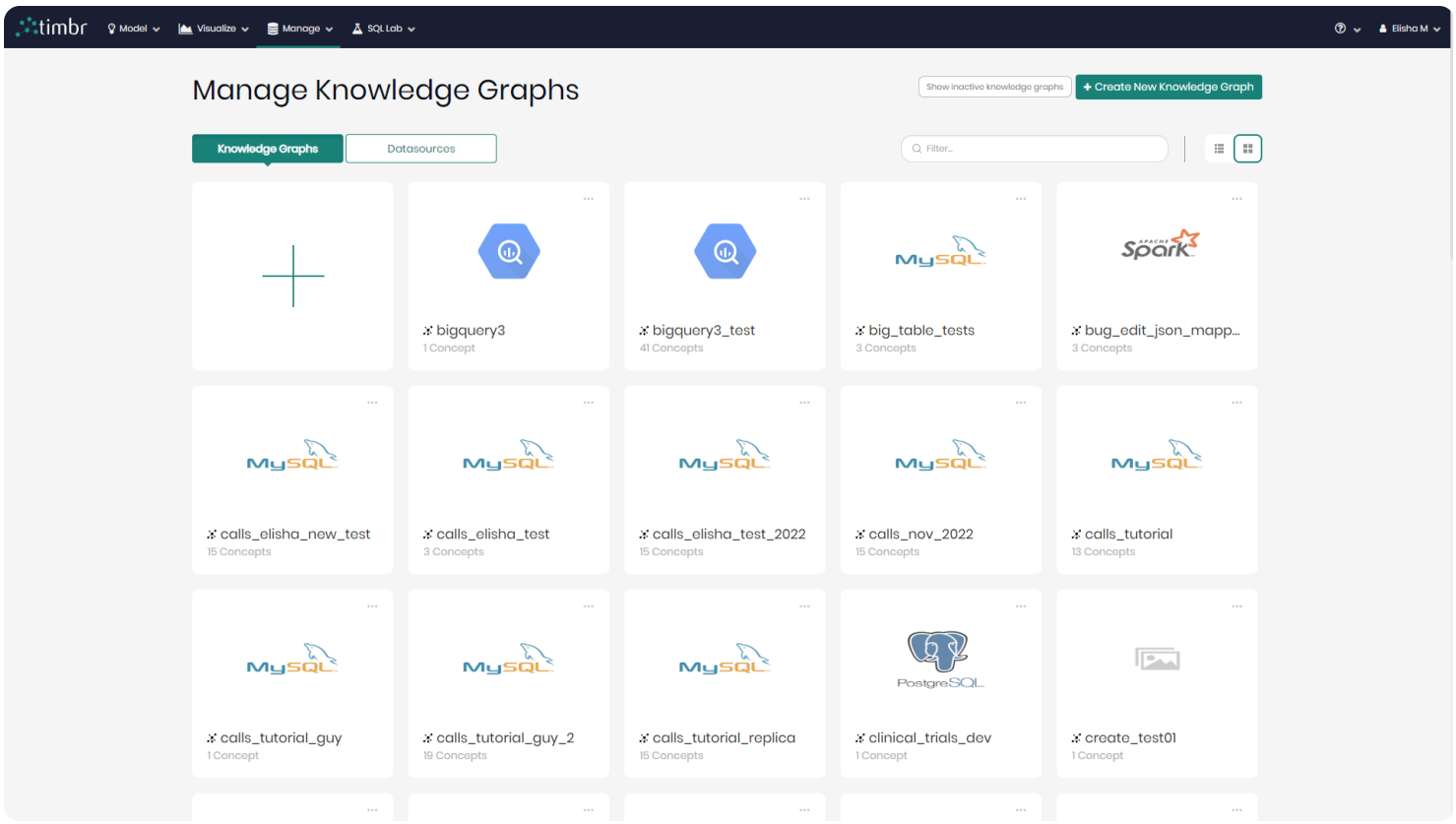
The Timbr platform currently supports the following back-ends:

- Amazon Athena
- Amazon Redshift
- Amazon S3
- Snowflake
- Databricks
- SAP Hana
- Google BigQuery
- Google Cloud Storage
- MySQL
- Apache Spark
- Apache Drill
- Apache Hive
- Azure Blob Storage
- Azure Datalake Storage
- Oracle

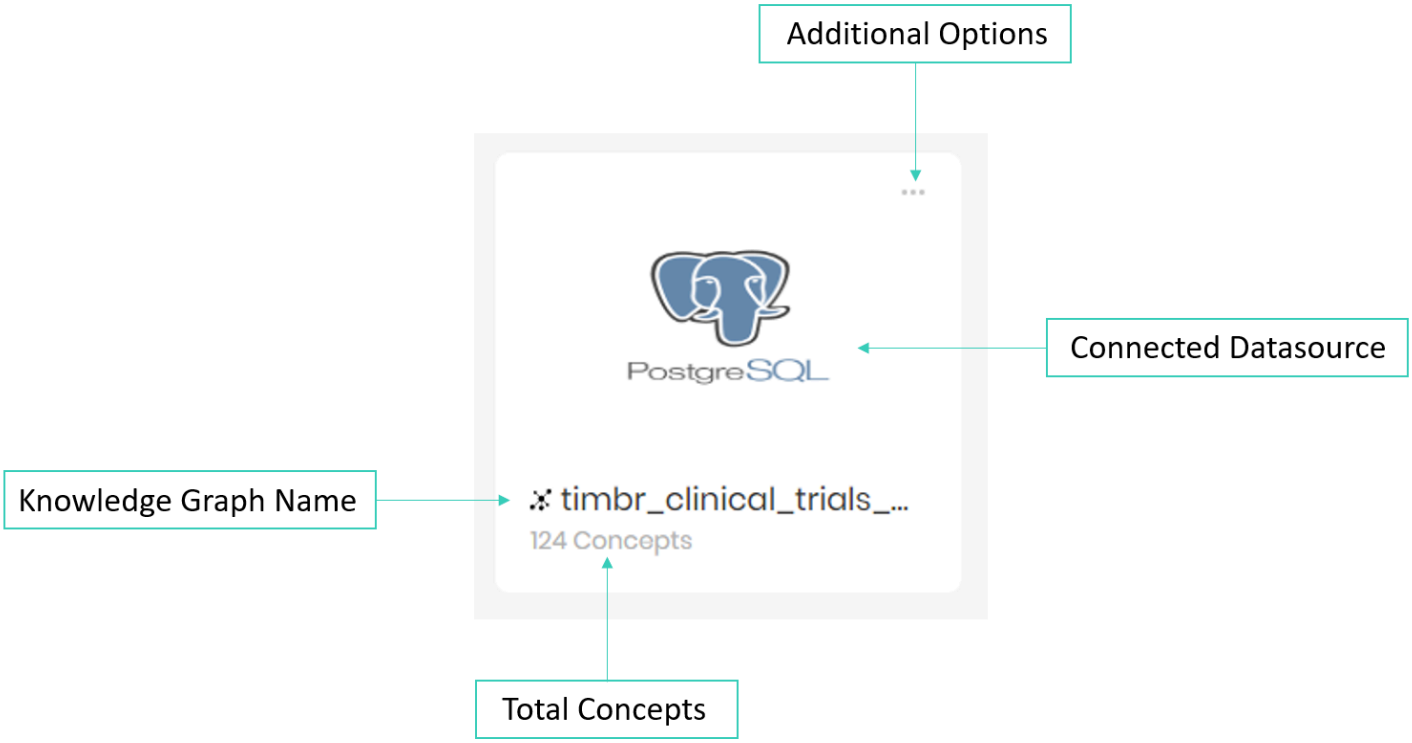
- Impala
- Microsoft SQL Server
- PostgreSQL
- Trino
- Presto
- Vertica Analytics Platform
- More can be added by request

Manage Knowledge Graphs Page

All the available Knowledge Graphs and connected datasources can be found in the Knowledge Graph Manager Page, which can be accessed through the **Manage** tab by clicking on **Knowledge Graphs**.



Each box represents a knowledge graph.



Each box contains 3 horizontal dots that when clicked on offer the following additional options on the selected Knowledge Graph:

Edit - Opens a window to edit the datasources and the different configurations of the selected Knowledge Graph.

Set description - Enabling to add a description to the knowledge to further explain what it represents.

Change active datasource - Enabling to choose the active datasource that will appear when querying the data.

Manage Tags - Opens a pop-up window to manage the existing tags given to the knowledge graph, as well as the option to add new tags.

Duplicate - Duplicates the selected knowledge graph.

Backup - Creates a backup of the knowledge graph.

Export - When hovering over *export* the 2 following export options become available:

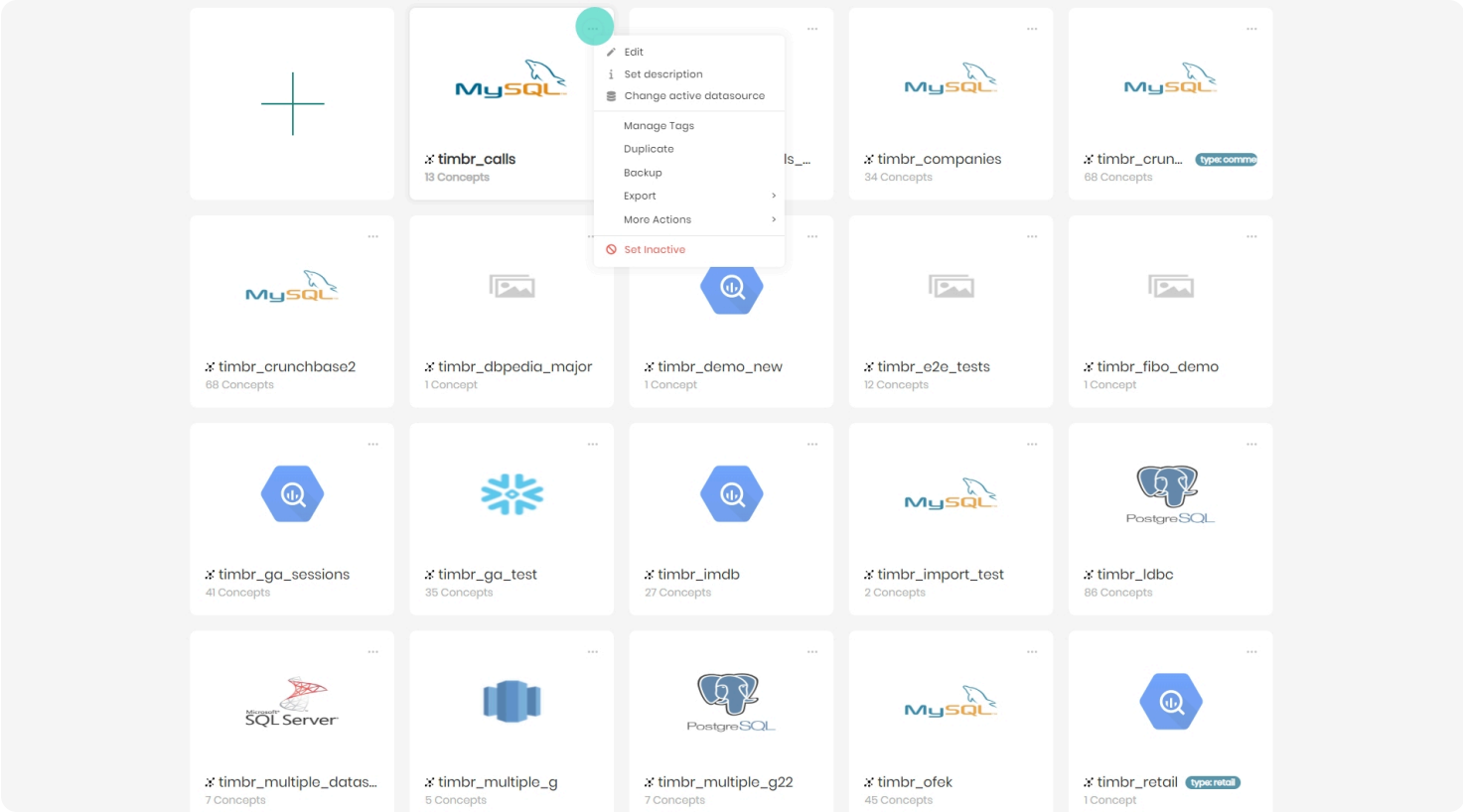
- **Export to SQL** - Exports the SQL syntax representing the knowledge graph.
- **Export to OWL** - Exports the OWL syntax representing the knowledge graph.

More Actions - When hovering over *More Actions* the following options appear:

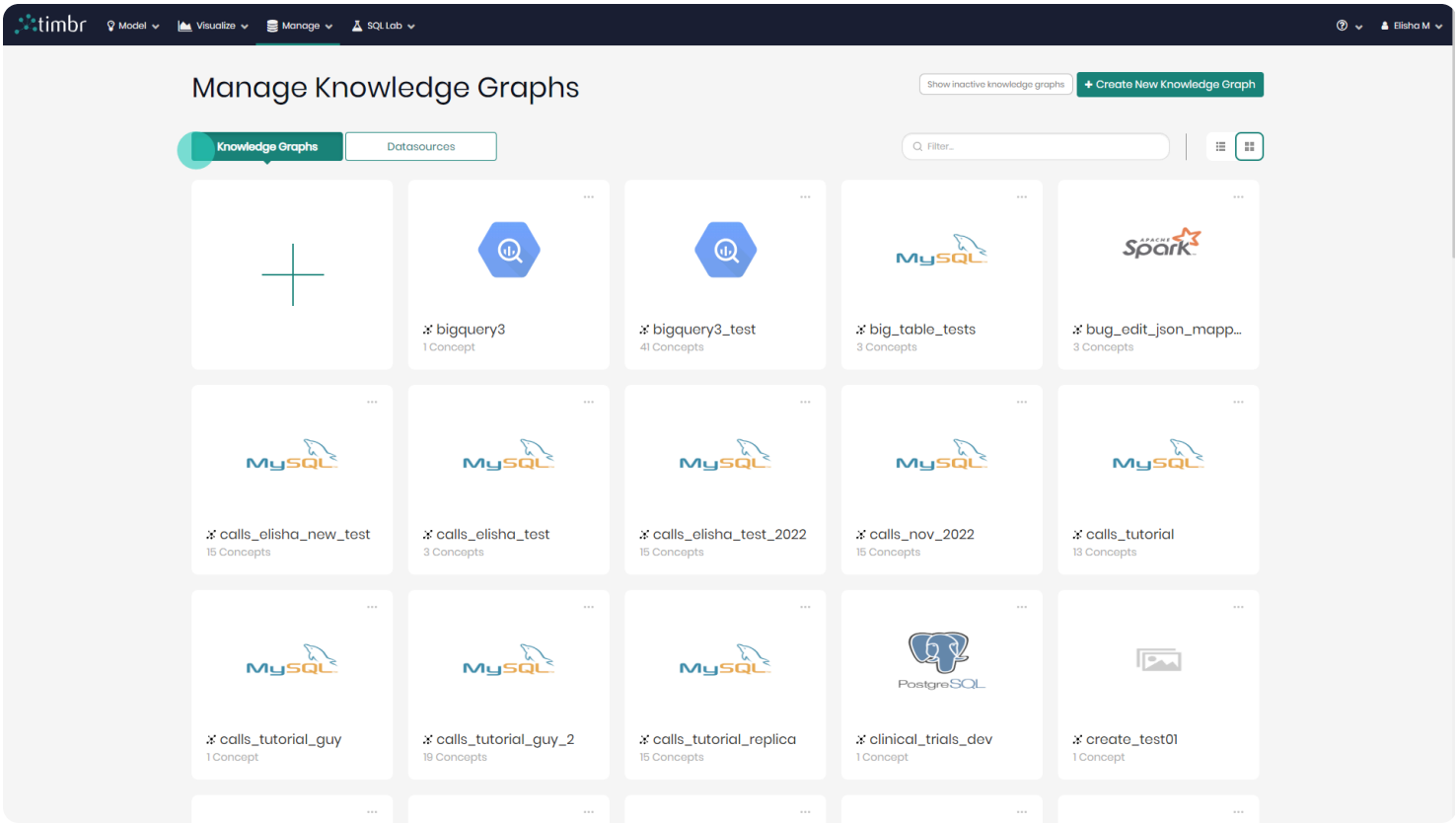
- **Model** - Opens Timbr's *Ontology Explorer* to model and edit the selected Knowledge Graph Model.
- **Map** - Opens Timbr's *Data Mapper* to create, edit and delete data mappings of the selected Knowledge Graph.
- **Explore** - Opens Timbr's *Graph Explorer* in order to explore the selected Knowledge Graphs data on a graph interface.
- **Ontology Views** - Opens Timbr's *Ontology Views* component to create, edit and delete data views of the selected Knowledge Graph.

- **Ontology Lineage** - Opens Timbr's *Data Lineage* component to fully understand the selected Knowledge Graphs layout and hierarchy of its data and its sources behind the scenes.

Set Inactive - Enabling to categorize the Knowledge Graph based on active or inactive.



Above the different Knowledge Graphs are two tabs giving the ability to switch between viewing the different Knowledge Graphs and Datasources in the environment by choosing the desired tab.



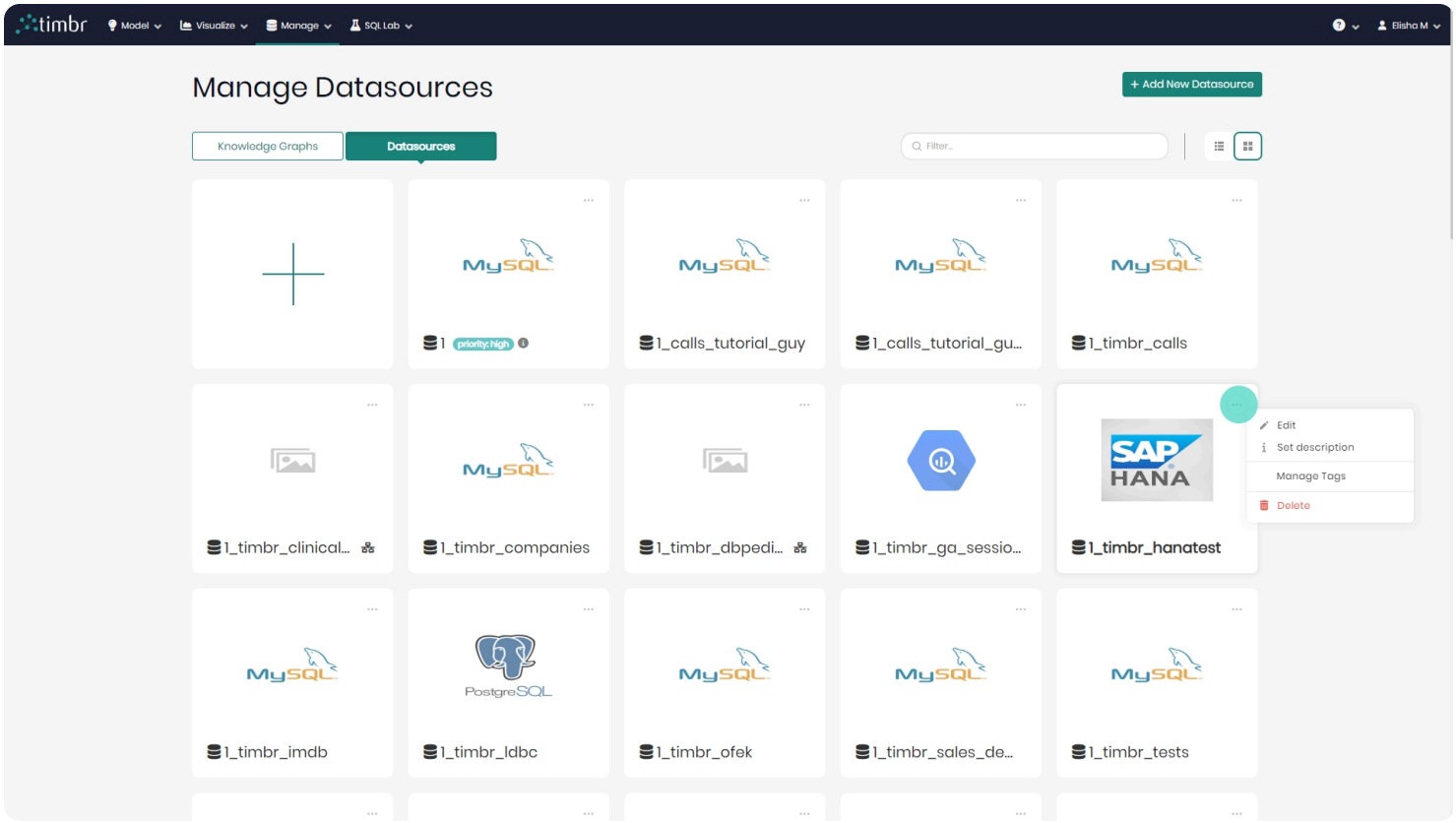
When switching to the Datasources tab, each box will represent a datasource that has been added to the environment. Here too, each box contains 3 horizontal dots that when clicked on offer the following additional options on the selected Datasource:

Edit - Opens a window to edit the selected datasource and its different configurations.

Set description - Enabling to add a description to the datasource to further explain what it represents.

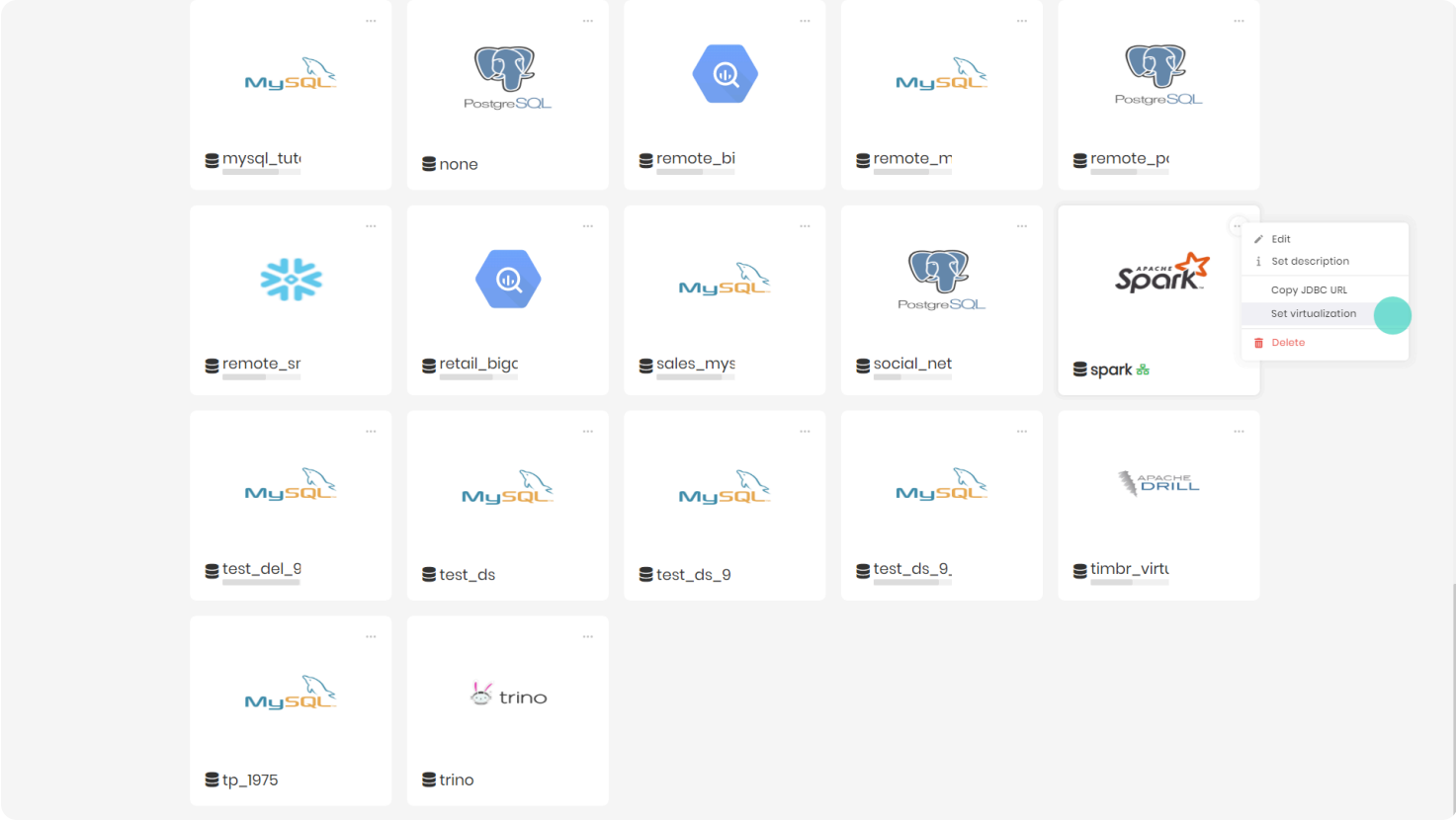
Manage Tags - Opens a pop-up window to manage the existing tags given to the knowledge graph, as well as the option to add new tags.

Delete - Deletes the selected datasource.

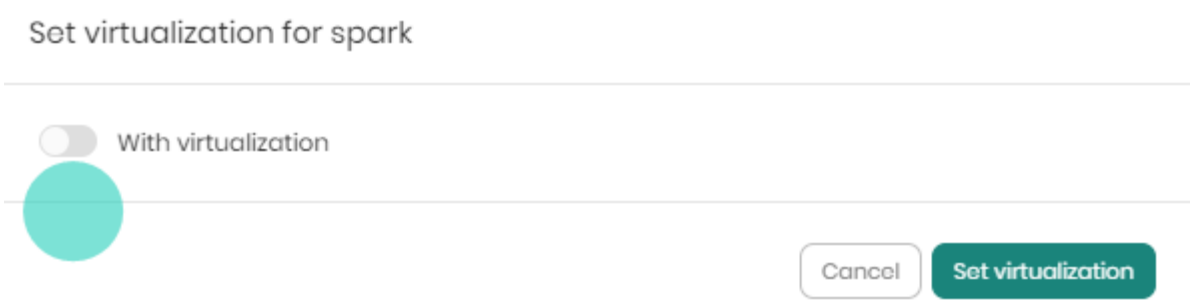


Timbr offers **Virtualization** with **Apache Spark** and **Databricks**, so in addition to the name given to the datasource, a small icon will appear on all **Apache Spark** or **Databricks** datasources signifying that those datasources have the ability of virtualization. When the virtualization is on the icon will be green, when it is turned off the icon will be black.

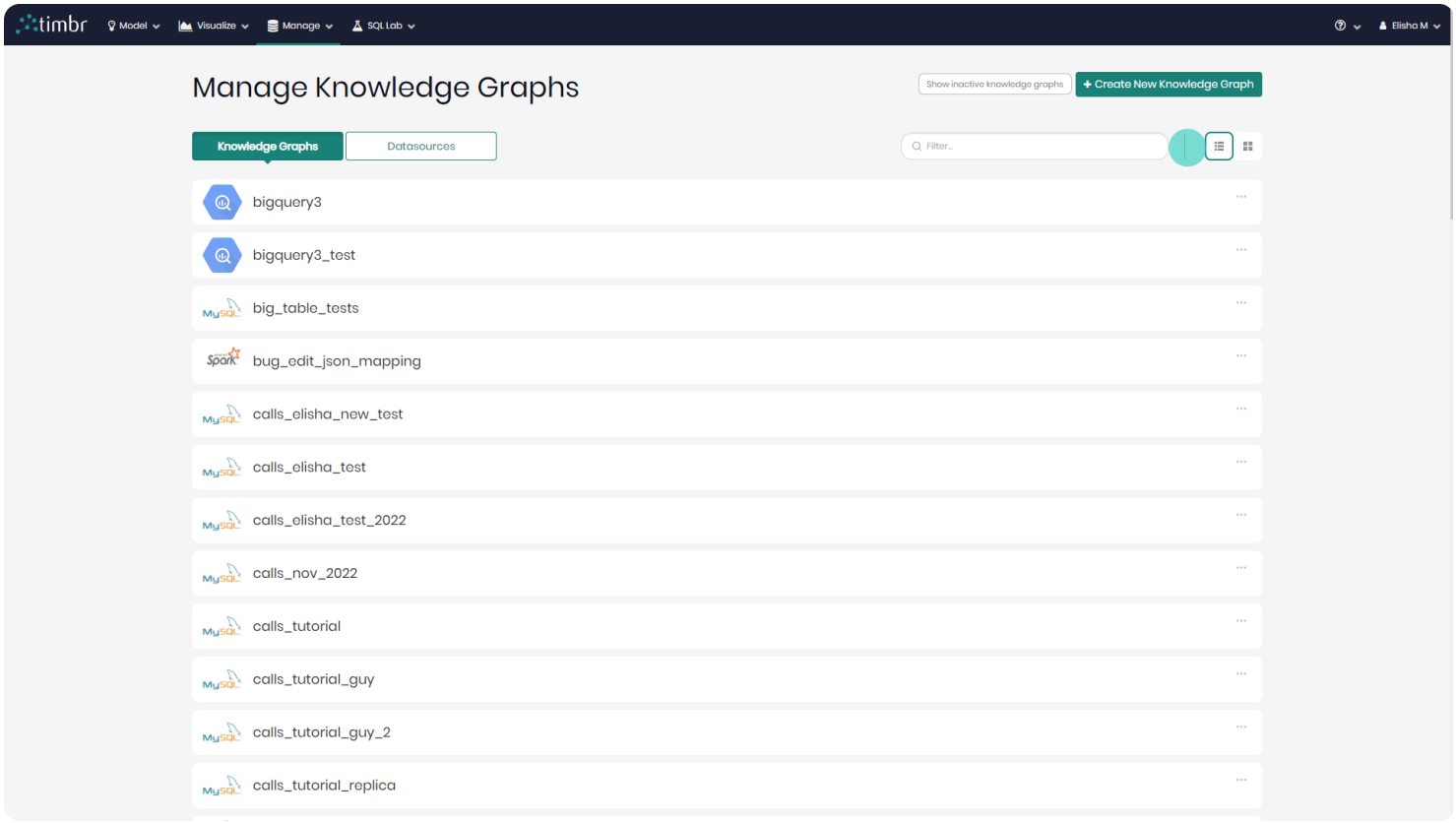
To turn the *Virtualization* on and off the 3 horizontal dots need to be clicked. Here you will find **Set Virtualization** In addition to the 4 options every datasource has when the dots are clicked, which are: *Edit*, *Set description*, *Manage Tags* and *Delete*.



After *Set Virtualization* is clicked, in the window that appears, when the **With virtualization** toggle is turned on and *Set virtualization* is clicked on the bottom right, virtualization will be turned on and the icon will show green. When the **With virtualization** toggle is turned off and *Set virtualization* is clicked on the bottom right, virtualization will be turned off and the icon will show black.



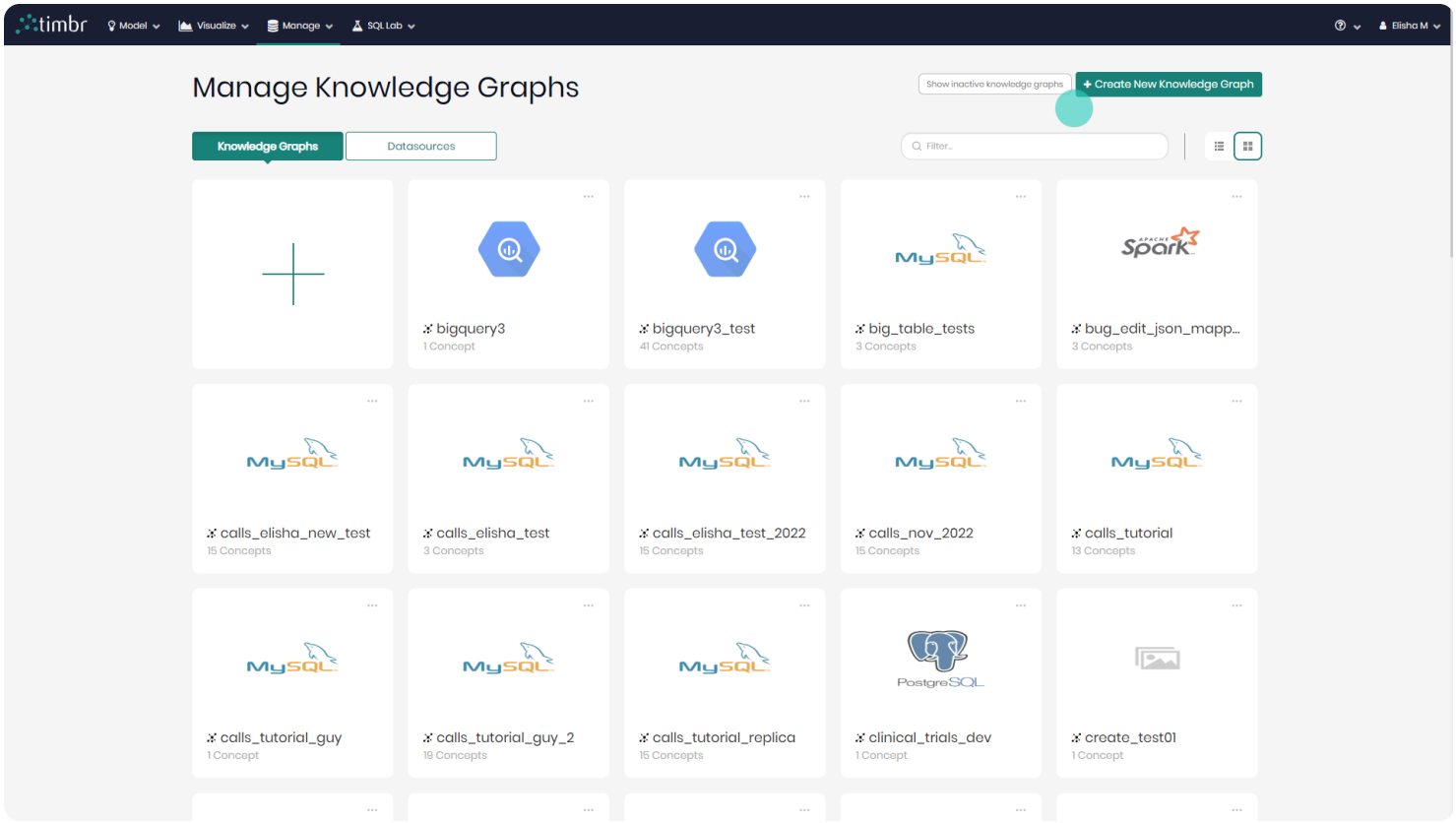
To the right of both the knowledge graph and datasource tabs is the *search filter* enabling one to search through the knowledge graphs and datasources using the search bar, as well as the option to decide whether to view the knowledge graphs and datasources as boxes or in a list.



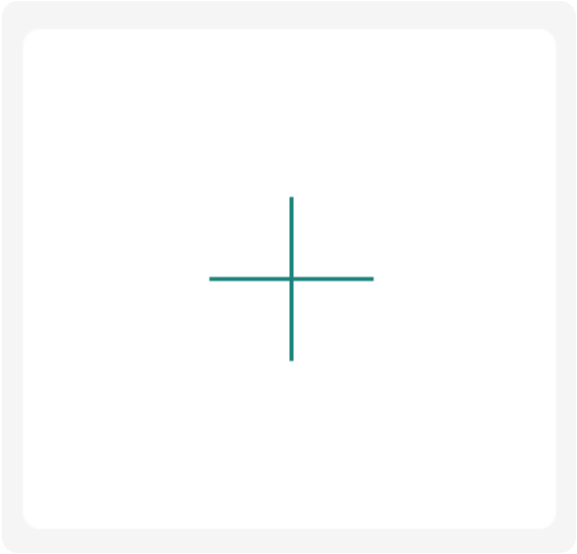
When the *Knowledge Graphs* tab is chosen, above the filter and different viewing options, are the following two buttons:

Show inactive knowledge graphs - When clicked on, only inactive knowledge graphs will appear. To return to the active knowledge graphs, the *Show knowledge graphs* must be clicked.

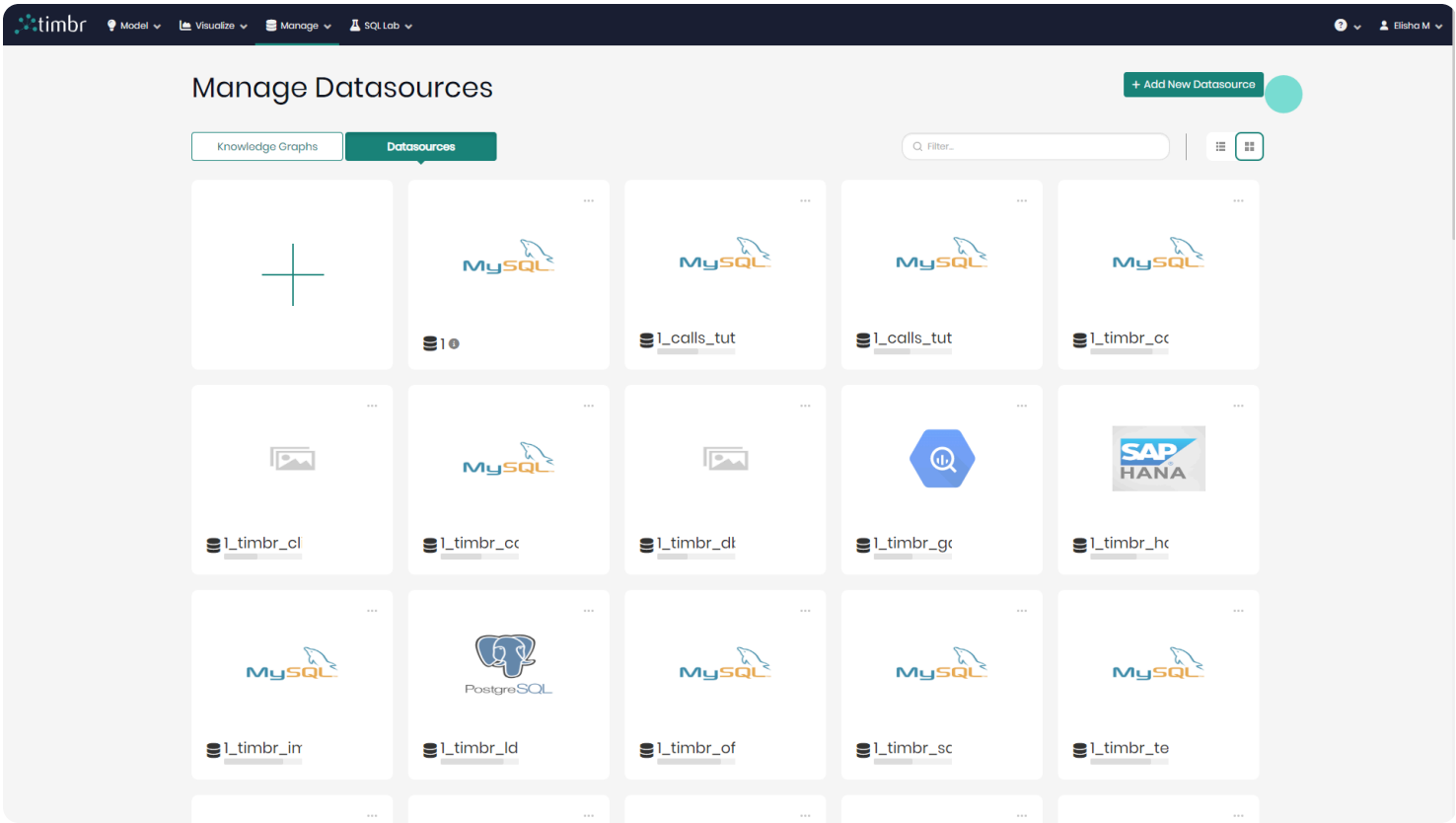
Create New Knowledge Graph - When clicked on, a pop-up will appear in order to begin creating a new knowledge graph.



Another way to add a new Knowledge Graph is by clicking the big add button beneath the knowledge graphs and datasources tabs.

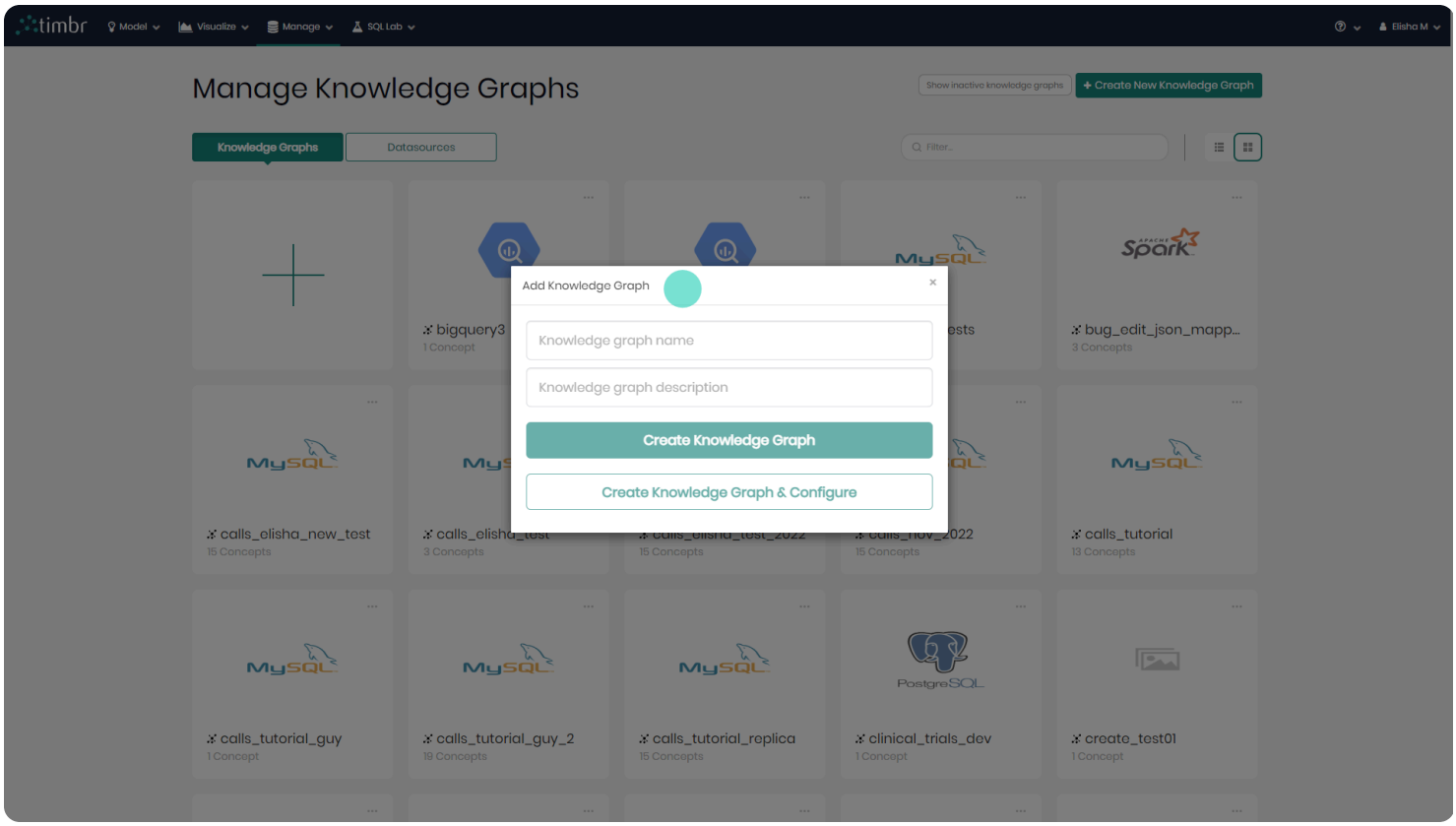


The same goes for *Datasources*. When the *Datasource* tab is chosen, on the top right is **Add New Datasource** that When clicked on will open a pop-up in order to begin adding a new datasource to the environment.



Another way to add a new datasource is by clicking the big add button beneath the knowledge graphs and datasources tabs.

Knowledge Graph - Create/Connect to a new Knowledge Graph

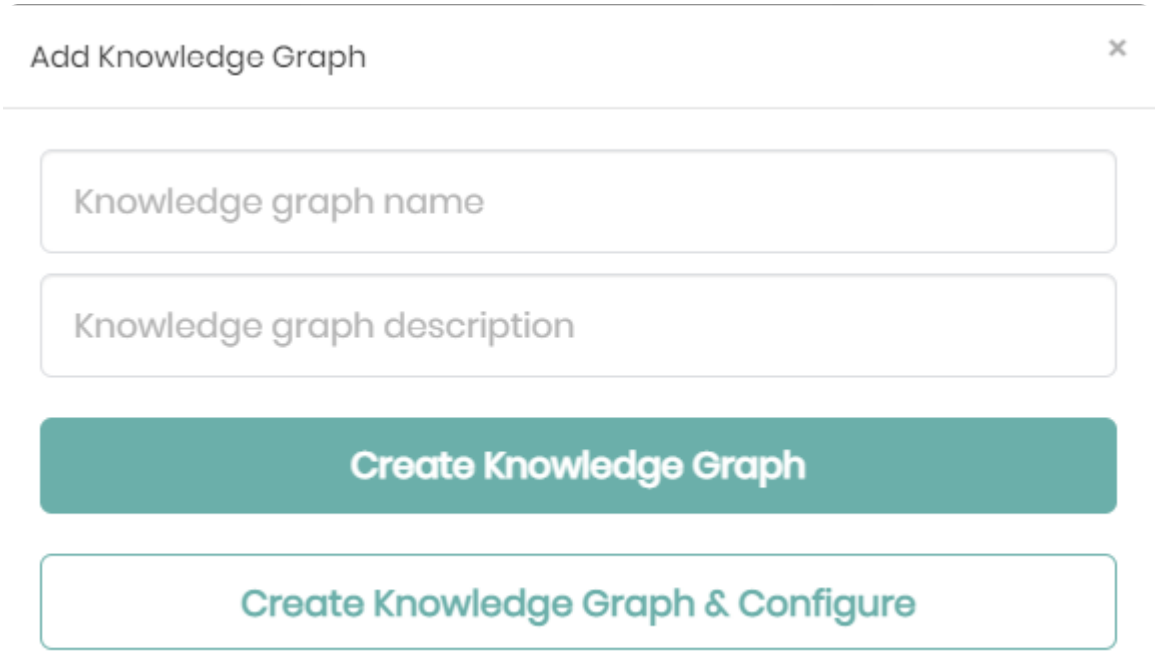


Every Knowledge Graph is created through the **timbr thrift-server** where the Knowledge Graphs are stored and managed.

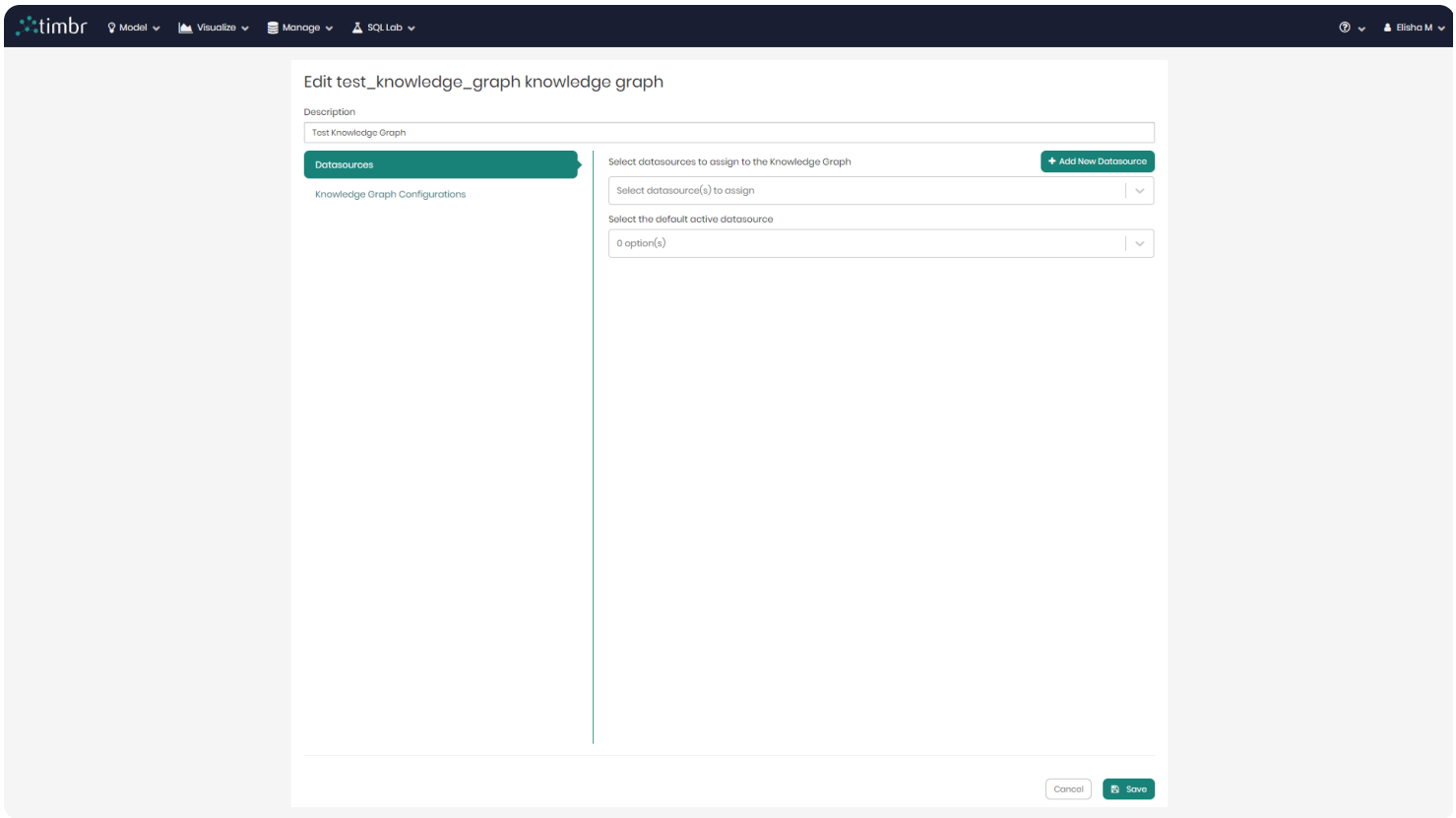
When a Knowledge Graph is being created or edited, all the information about the Knowledge Graph and back-end datasources are presented except for **passwords** and/or **private keys**.

Please note that not all users have access to create knowledge graphs and/or data sources, this is determined by the platform admin.

The creation begins with giving the new Knowledge Graph a **Name** and an optional **Description**.



The knowledge graph can then be created and saved by clicking on **Create Knowledge Graph** saving the knowledge graph with only its name and no connected datasources, or instead, the knowledge graph can be created by clicking on **Create Knowledge Graph & Configure**, which will open a new window in order to connect the desired datasources to the knowledge graph, as well as edit its different configurations.

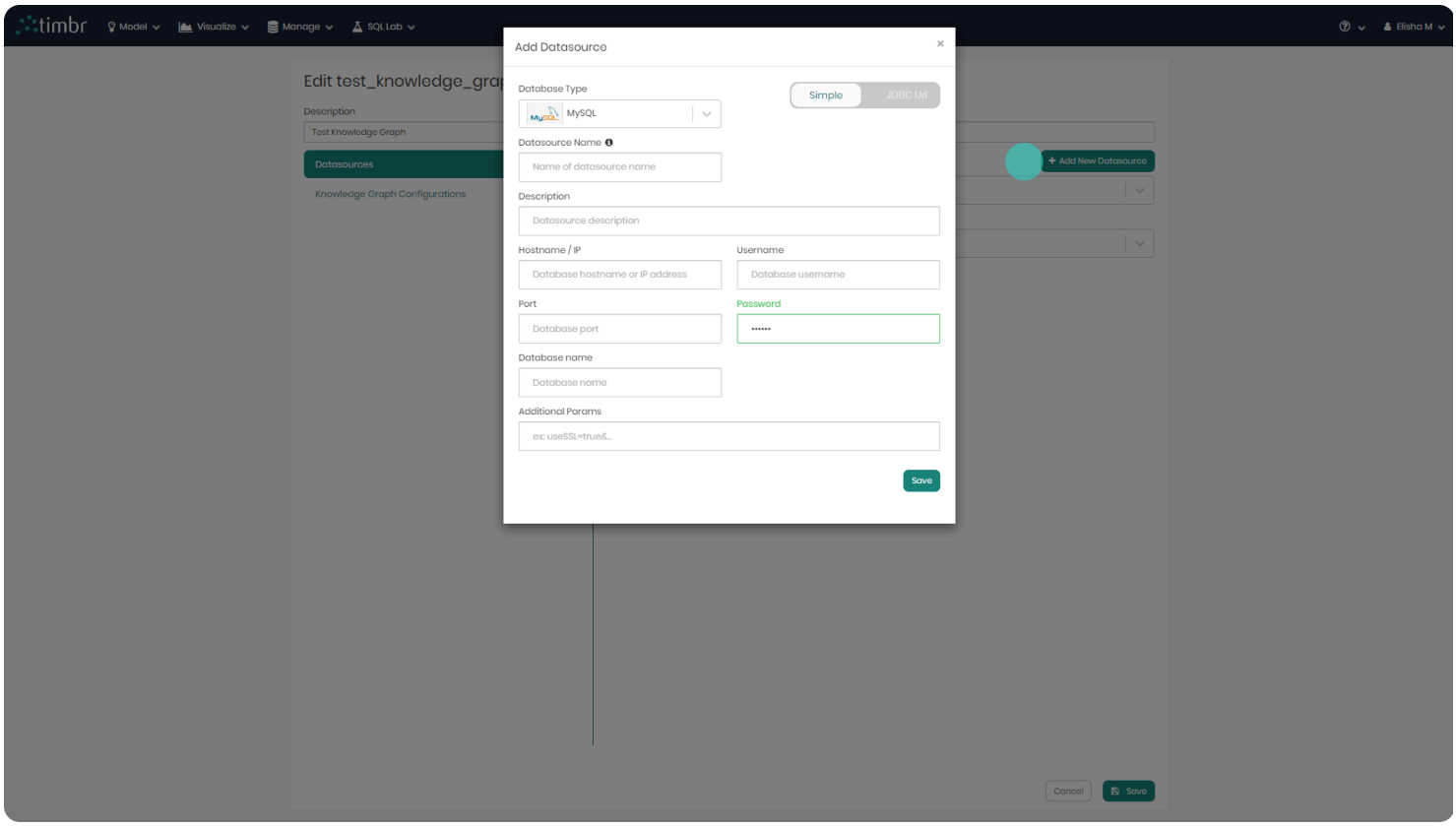


In the new window, beneath the name and description are two tabs, **Datasources** and **Knowledge Graph Configurations**.

Datasources

In the **Datasources** tab there are two ways to connect a datasource or multiple datasources to the knowledge graph:

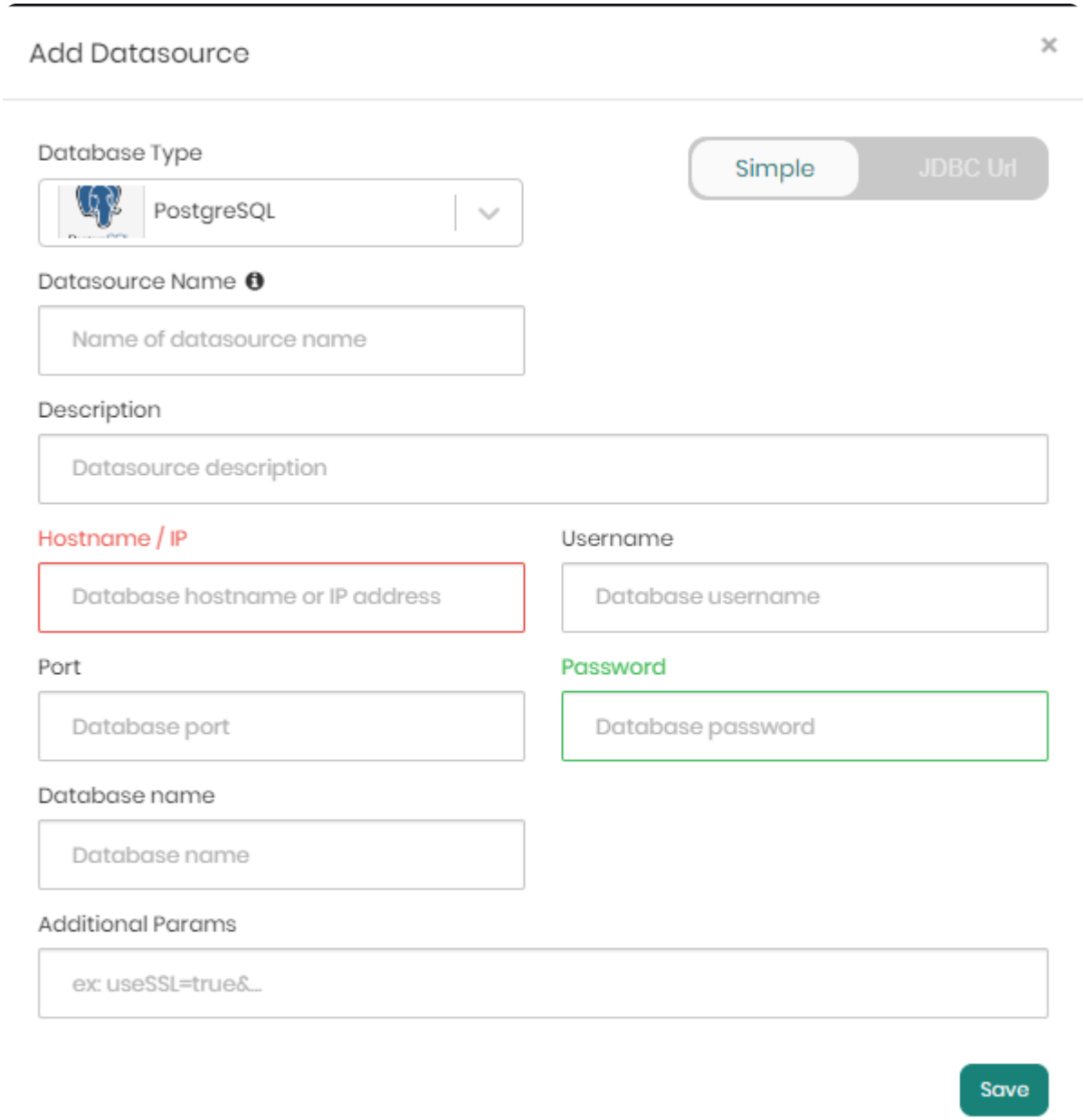
1. **Add New Datasource** - On the top right when *Add New Datasource* is clicked on, a pop-up window will appear where all the relevant connection details must be provided in order to connect the datasource to the knowledge graph.



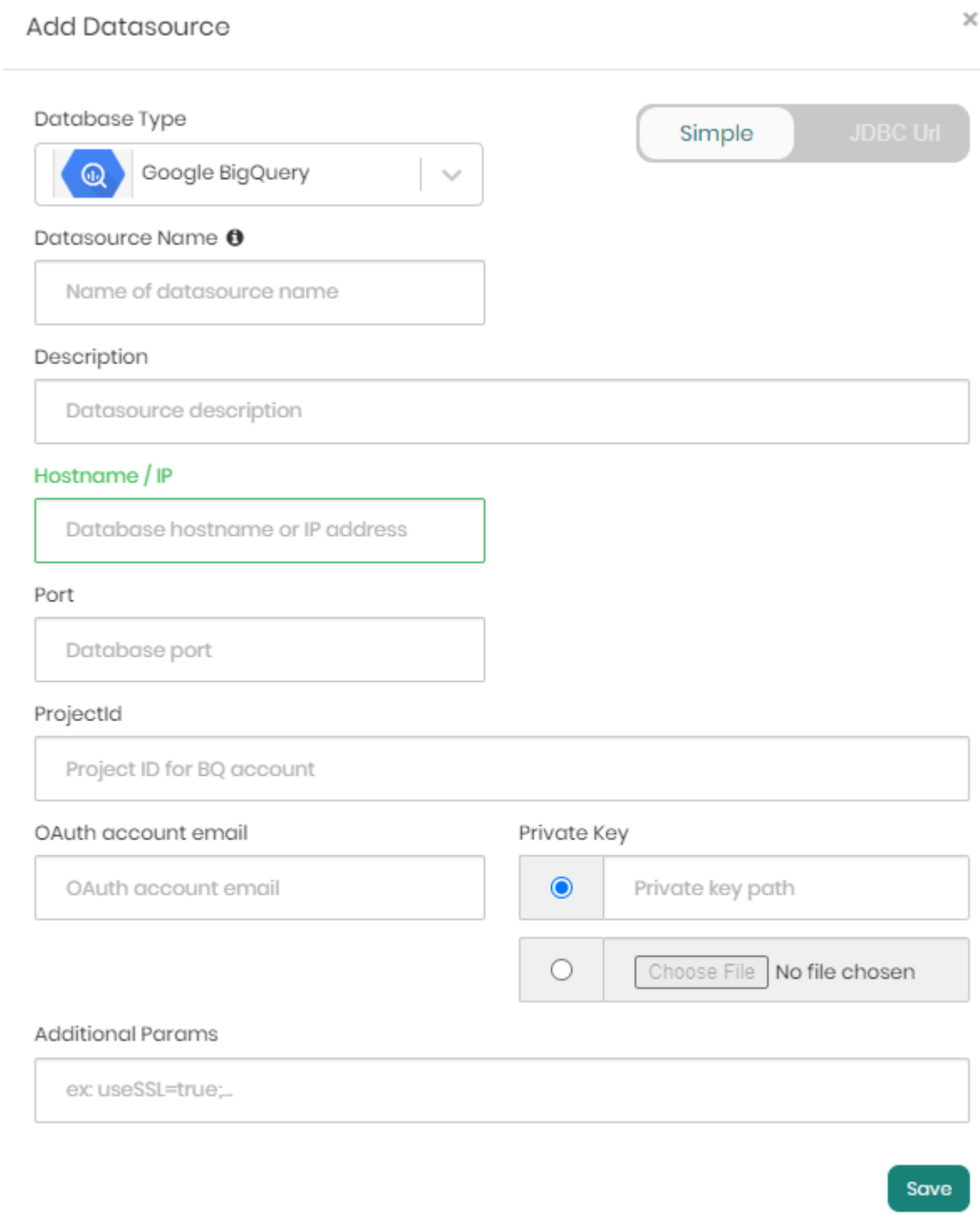
Each **back-end datasource** has slightly different requirements in order to connect to it, but the following information is required for the vast majority of the **back-end datasources:**

- The **hostname** for accessing the back-end datasource server
- The **port** for accessing the back-end datasource server
- The **username** for the back-end datasource
- The **password** for the back-end datasource

Some **back-end datasources** like **PostgreSQL**, **Athena**, and **SAP Hana** require to specify a **database name** in order to create a connection

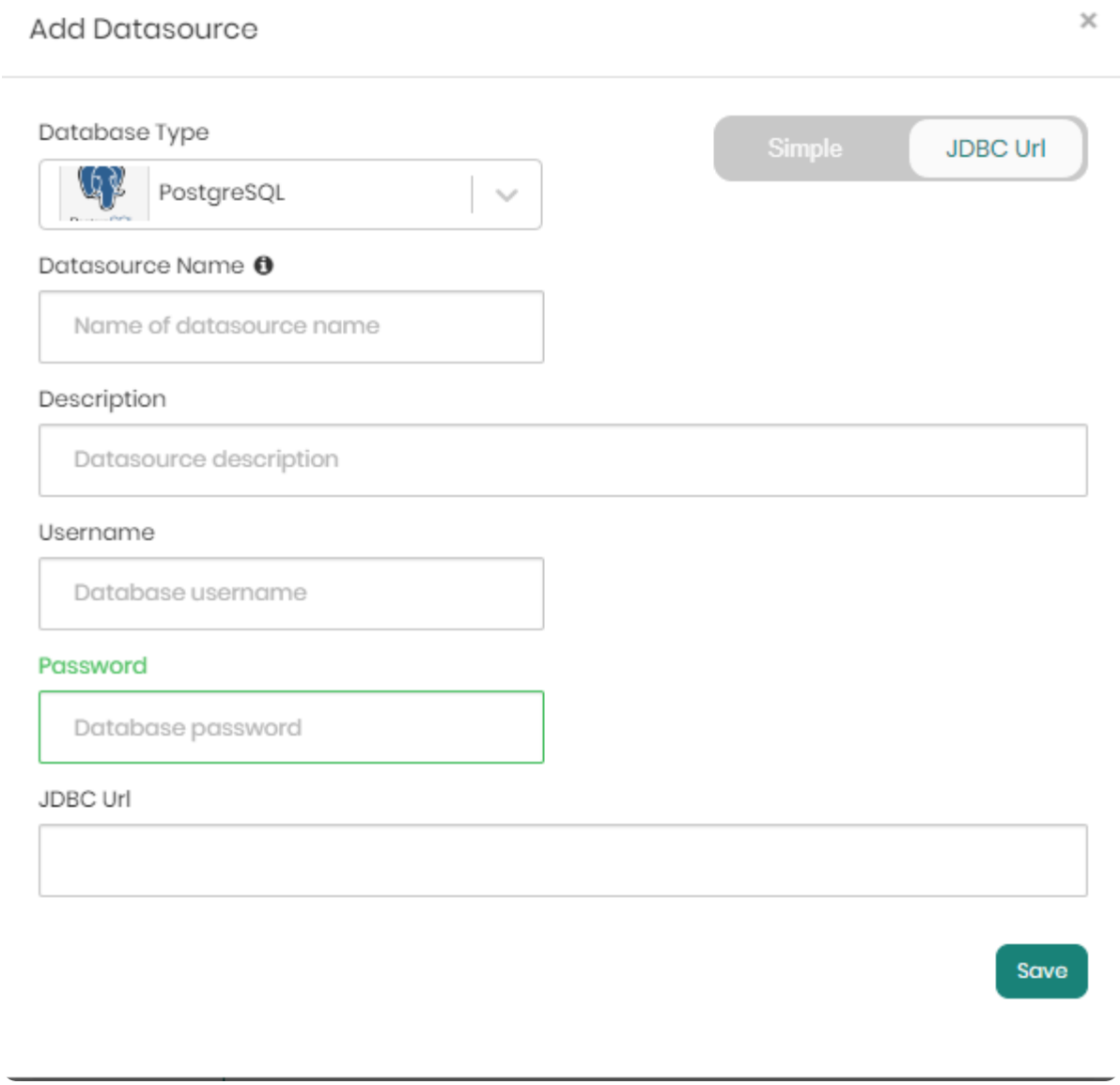


Other **back-end datasources** like **Big Query** require different information in order to create a connection. Parameters such as: - The **project id** as specified in the Google BigQuery project - The **associated email** with access to the Google BigQuery project specified in the **project id** field - A **private key** file in JSON format as provided by Google BigQuery for authentication



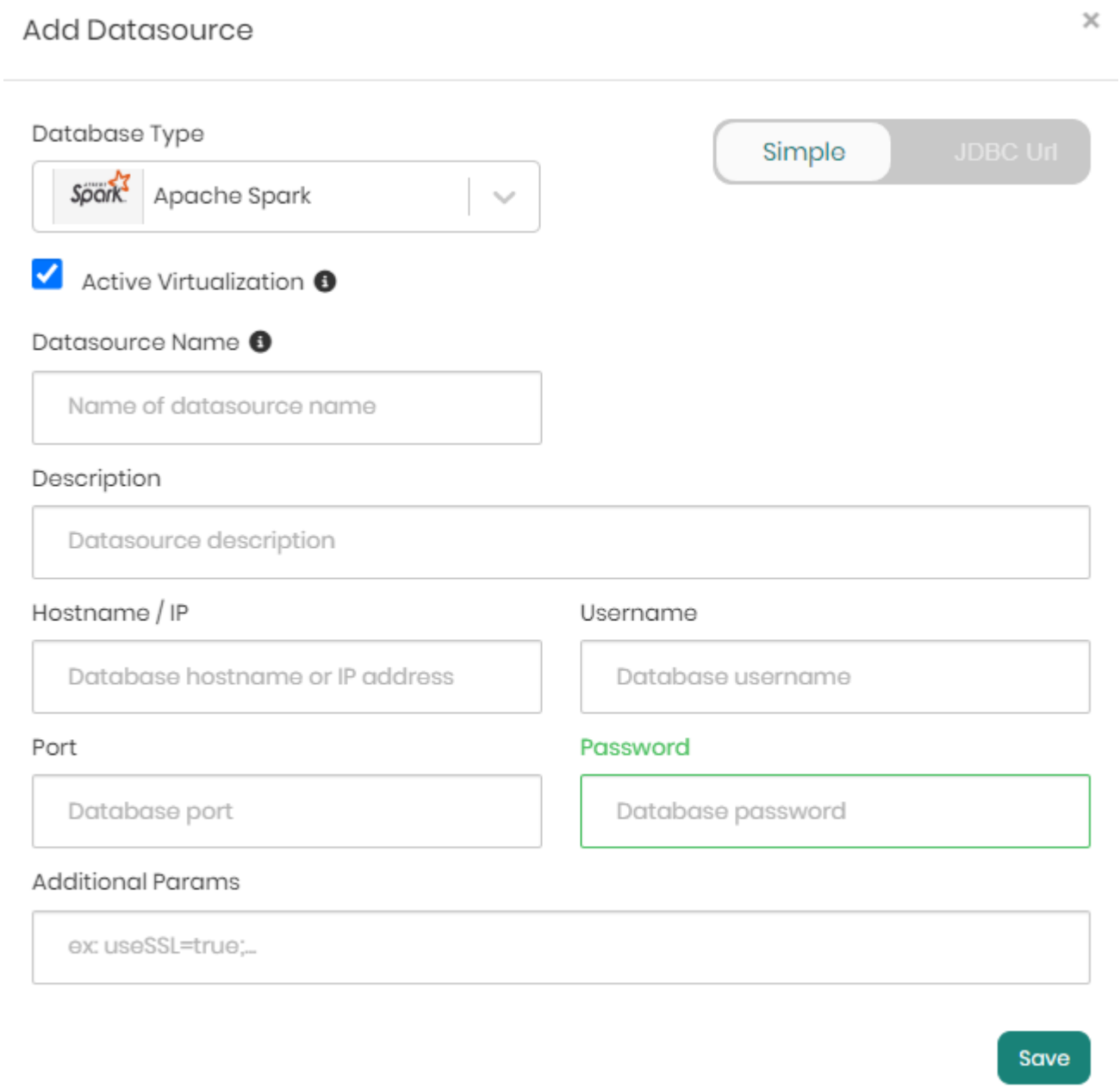
If supported by the **datasource back-end**, optional **additional parameters** (Additional Params) can be integrated into the connection between the **Knowledge Graph** and the **back-end datasource**

On the top right, the toggle can be switched from **Simple** to **JDBC Url** in order to connect the datasource using the relevant JDBC URL.

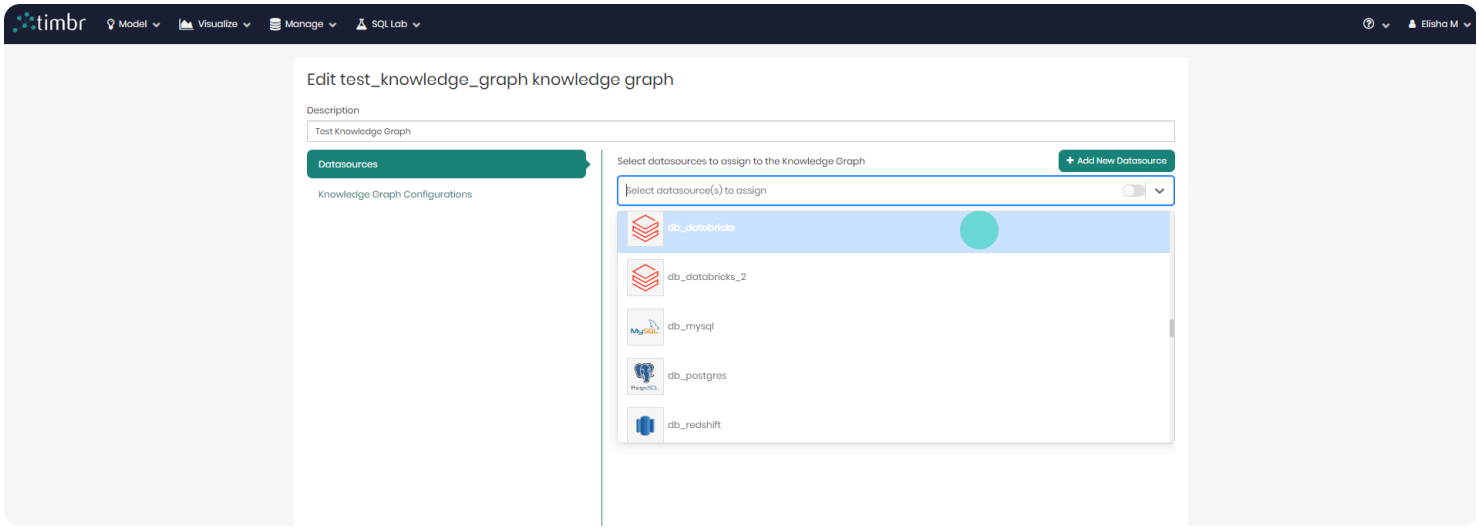


When the details are entered either with a username and password or via the JDBC URL, **Save** needs to be clicked in order to save the information and connect the datasource to the knowledge graph.

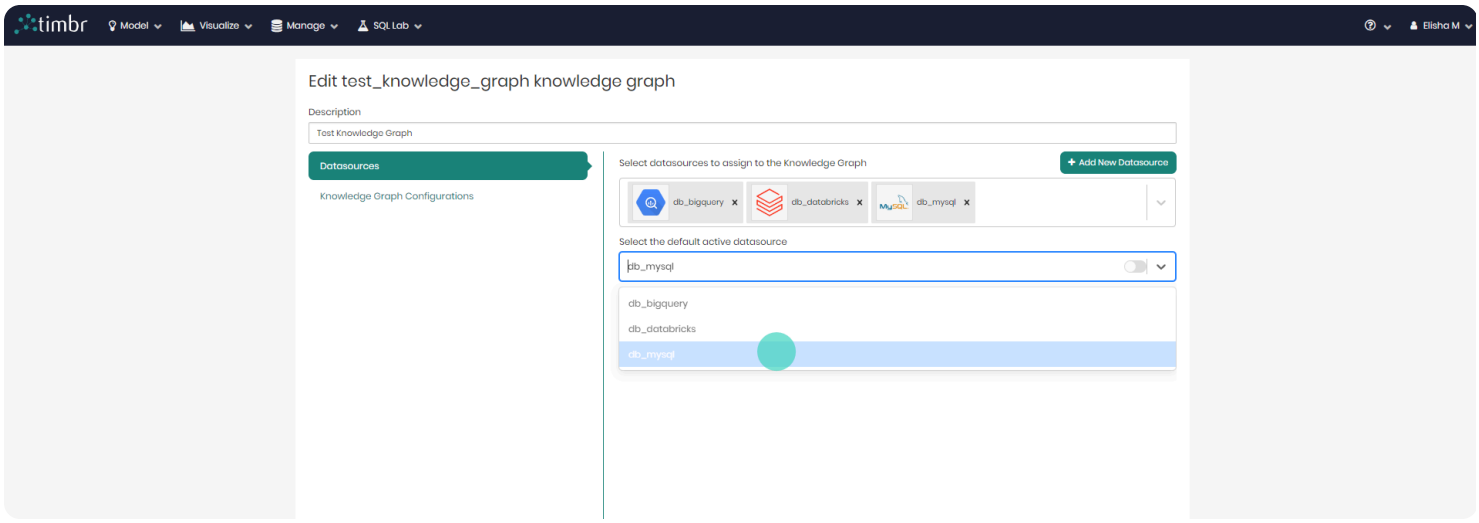
Timbr enables to add **Virtualization** for the **Apache Spark** and **Databricks** datasources. When these datasources are selected, a checkbox with an option for *Active Virtualization* will appear which indicates whether virtualization is on or off for the specific datasource. This option can be switched on and off at a later time as well.



2. **Select datasource to assign to the Knowledge Graph** - The second option for connecting a datasource to the knowledge graph, is by choosing an existing datasource that has already been connected from the list in the dropdown.



In cases where more than one datasource is chosen, we will be asked to select the **Default active datasource** which will be the default datasource when we query the knowledge graph for data.

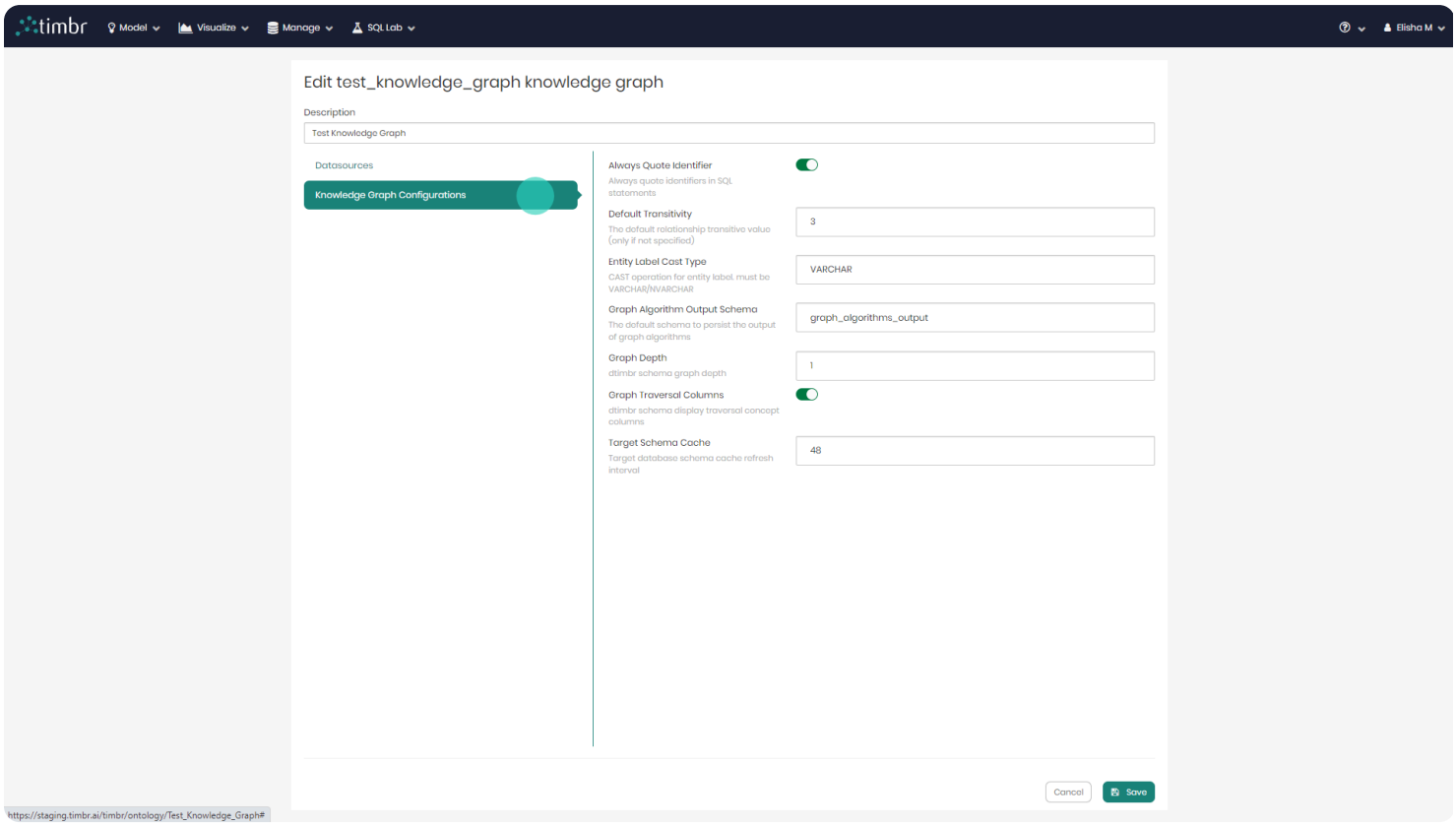


When adding **Virtualization** with **Apache Spark** or **Databricks** with the intention to query concepts that have data mapped from multiple datasources, it's important to choose **Apache Spark** or **Databricks** as the **Default active datasource**, which will enable you to query data that was mapped to the knowledge graph concepts from multiple datasources. So, if for example there are 3 datasources connected to a knowledge graph and you define that the **Default active datasource** does not contain virtualization, you will only be able to query the knowledge graph concepts that contain data mapped from the **Default active datasource** you chose.

Once again, when the details are entered **Save** needs to be clicked on the bottom right in order to save the information and connect the datasource or multiple datasources to the knowledge graph.

Knowledge Graph Configurations

The second tab is *Knowledge Graph Configurations* which contains different configurations that apply to a certain Knowledge Graph **even outside** of the Timbr-platform.



The following configuration flags are available:

"**Setting name**", "**Type**", "**Description**" "**Quote Identifier**", "*Boolean*", "When true, Timbr shows quote identifiers in SQL statements" "**Default Transitivity**", "*Integer*", "The default transitive relationship value" "**Entity Label Cast Type**", "*Varchar/NVarchar*", "Default CAST operation for the entity label" "**Graph Algorithm Output Schema**", "*Varchar/NVarchar*", "The default schema to persist the output of graph algorithms" "**Meta-data Graph Depth**", "*Integer*", "he depth of graph traversals presented when requesting the meta-data of a concept in the **dtimbr** schema" "**Show Traversal Columns**", "*Boolean*", "Display the properties relationships in the meta-data of a concept in the **dtimbr** schema" "**Target Schema Cache**", "*Integer*", "Datasource schema cache refresh interval in hours"

Once the desired configurations are set, **Save** must be clicked on the bottom right in order to save the *Knowledge Graph Configurations*.

After the **Save** button is clicked in the main window, assuming all the information provided is valid, a new box representing the new Knowledge Graph will appear in the **Knowledge Graph Manager Page**.

Once the Knowledge Graph appears in the **Knowledge Graph Manager** it can be used in all other sections of the Timbr platform.

Datasources

The Datasources page manages the different datasources that are associated with knowledge graphs. Users can add, edit, change, or remove a database and define the settings of each data source. A user can tag datasources, set supported datasources as virtualization and toggle them *on* or *off*, and set their description. Timbr supports granting granular access to different datasources based on *users* or *roles*.

> **RUNNING QUERIES COMBINED FROM TWO OR MORE DATASOURCES**
>
> If the datasource is **Apache Spark** or **Databricks** then the datasource can also be used as a **virtualization engine** and run queries from two different datasources.

Supported Datasources:

Timbr supports full back-end integration to any relational database that is SQL / ANSI SQL compliant or can be queried in SQL. The connection can be established either by a JDBC or ODBC connector.

The Timbr platform currently supports the following back-ends:

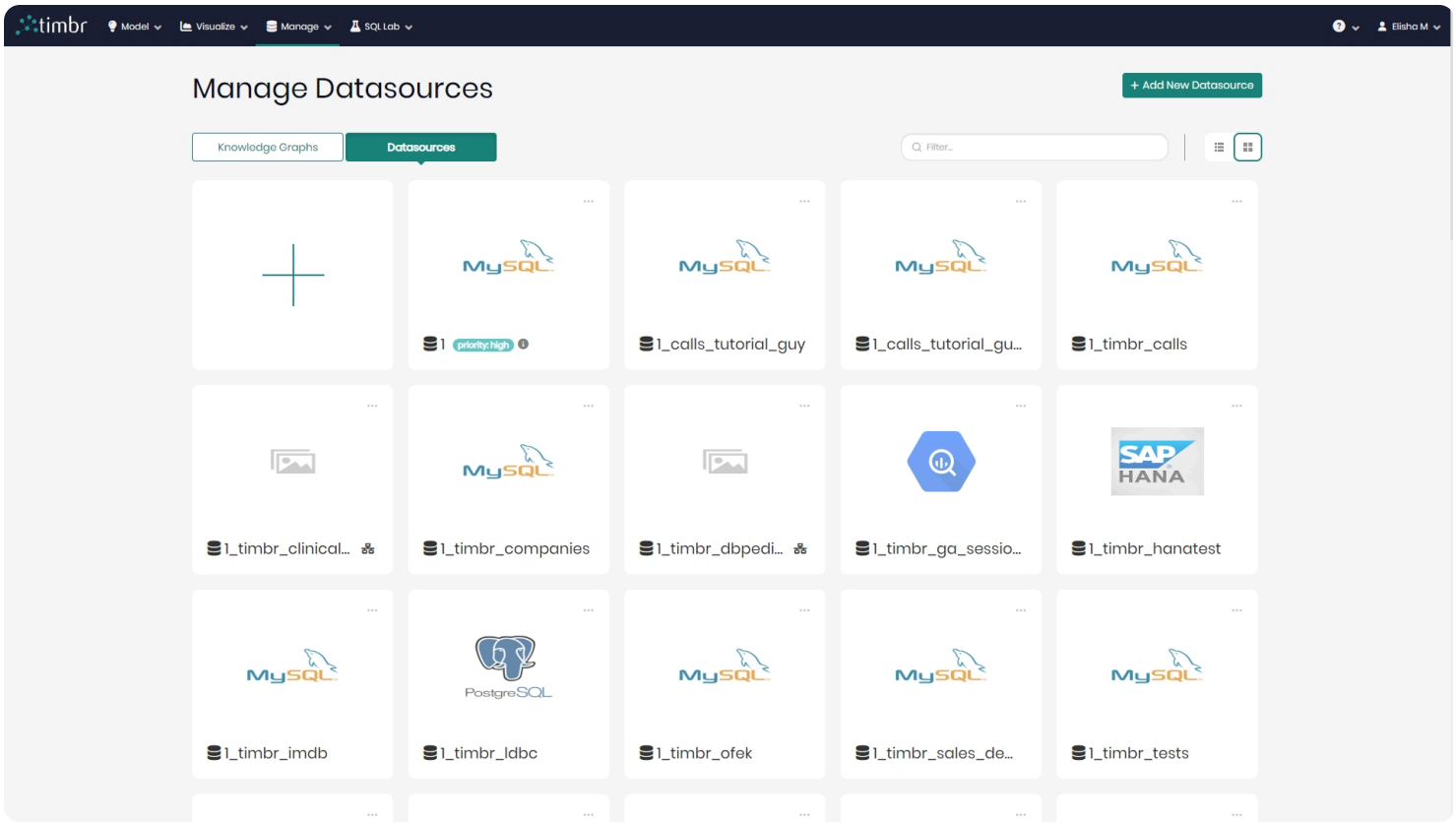
- Amazon Athena
- Amazon Redshift
- Amazon S3
- Snowflake
- Databricks
- SAP Hana
- Google BigQuery
- Google Cloud Storage
- MySQL
- Apache Spark
- Apache Drill
- Apache Hive
- Azure Blob Storage
- Azure Datalake Storage
- Oracle
- Impala
- Microsoft SQL Server

- PostgreSQL
- Trino
- Presto
- Vertica Analytics Platform
- IBM DB2

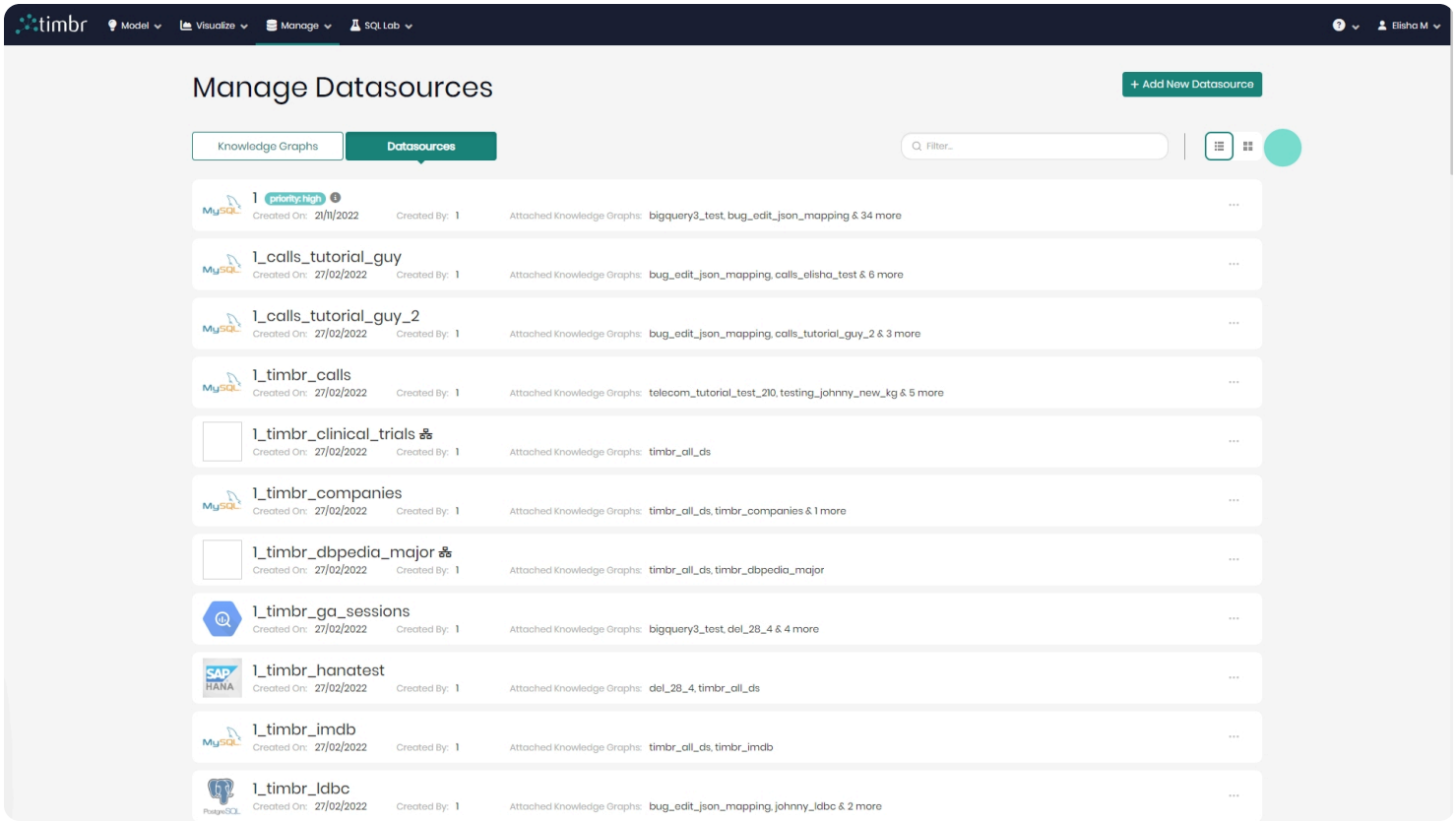
More can be added upon request

Manage Datasources Page

All the available datasources can be found in the **Manage Datasources** page, which can be accessed through the **Manage** tab by clicking on **Datasources**.

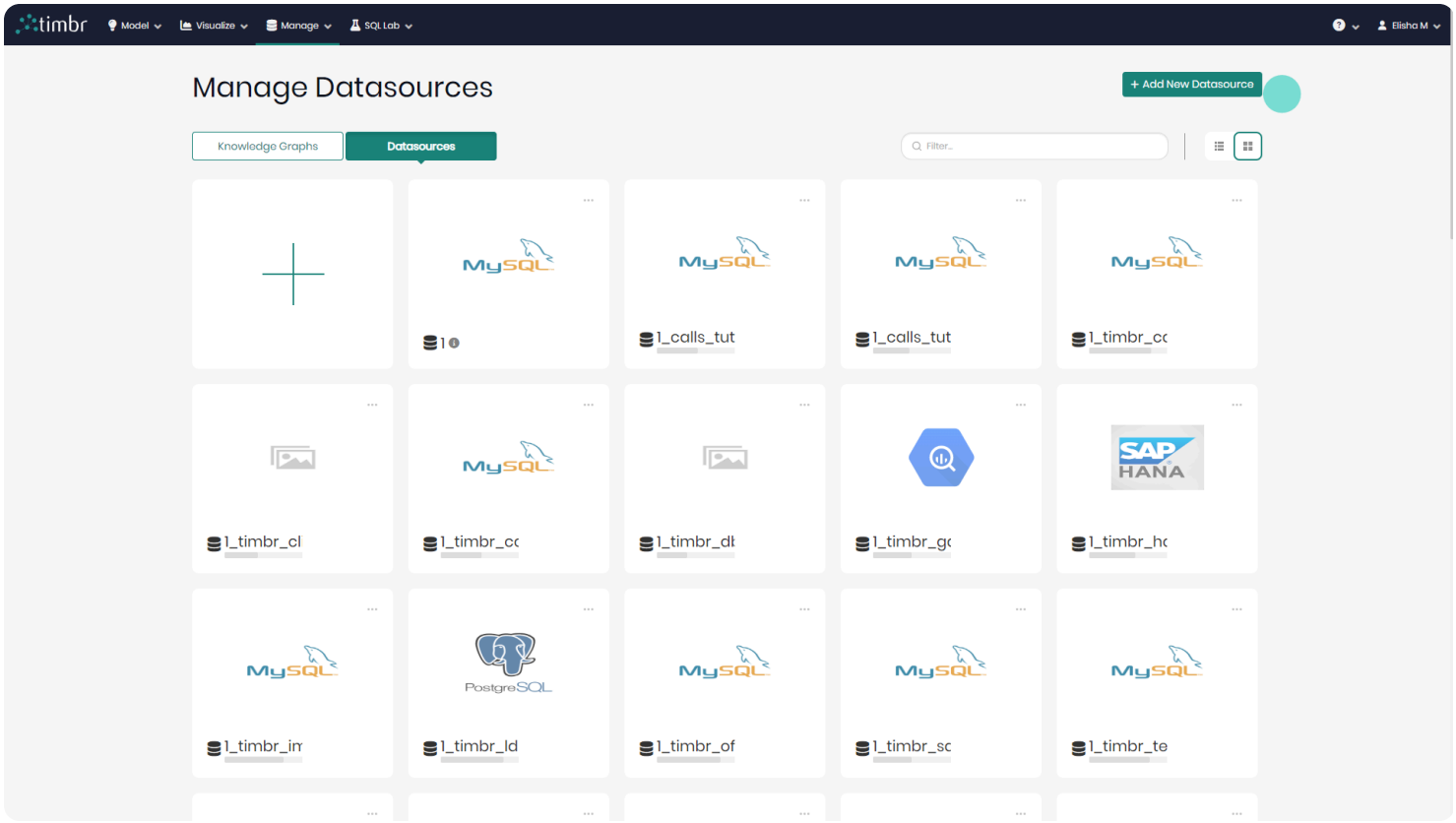


On the right of the screen is the *search filter* enabling you to search through the datasources using the search bar, as well as the option to decide whether to view the datasources as boxes or in a list.



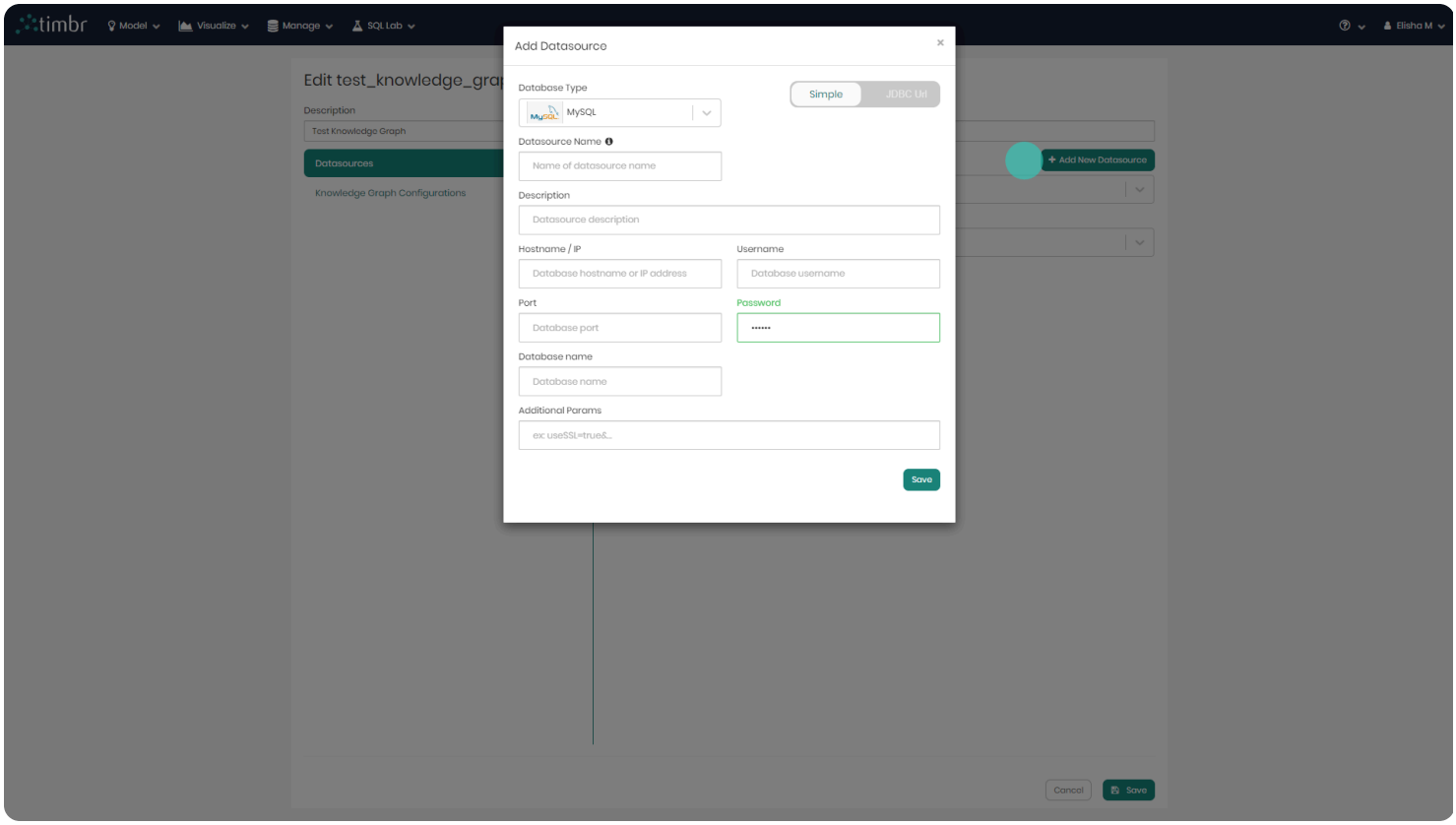
Adding a new Datasource

On the top right is **Add New Datasource** that When clicked on will open a pop-up in order to begin adding a new datasource to the environment.



Another way to add a new datasource is by clicking the big add button beneath the knowledge graphs and datasources tabs.

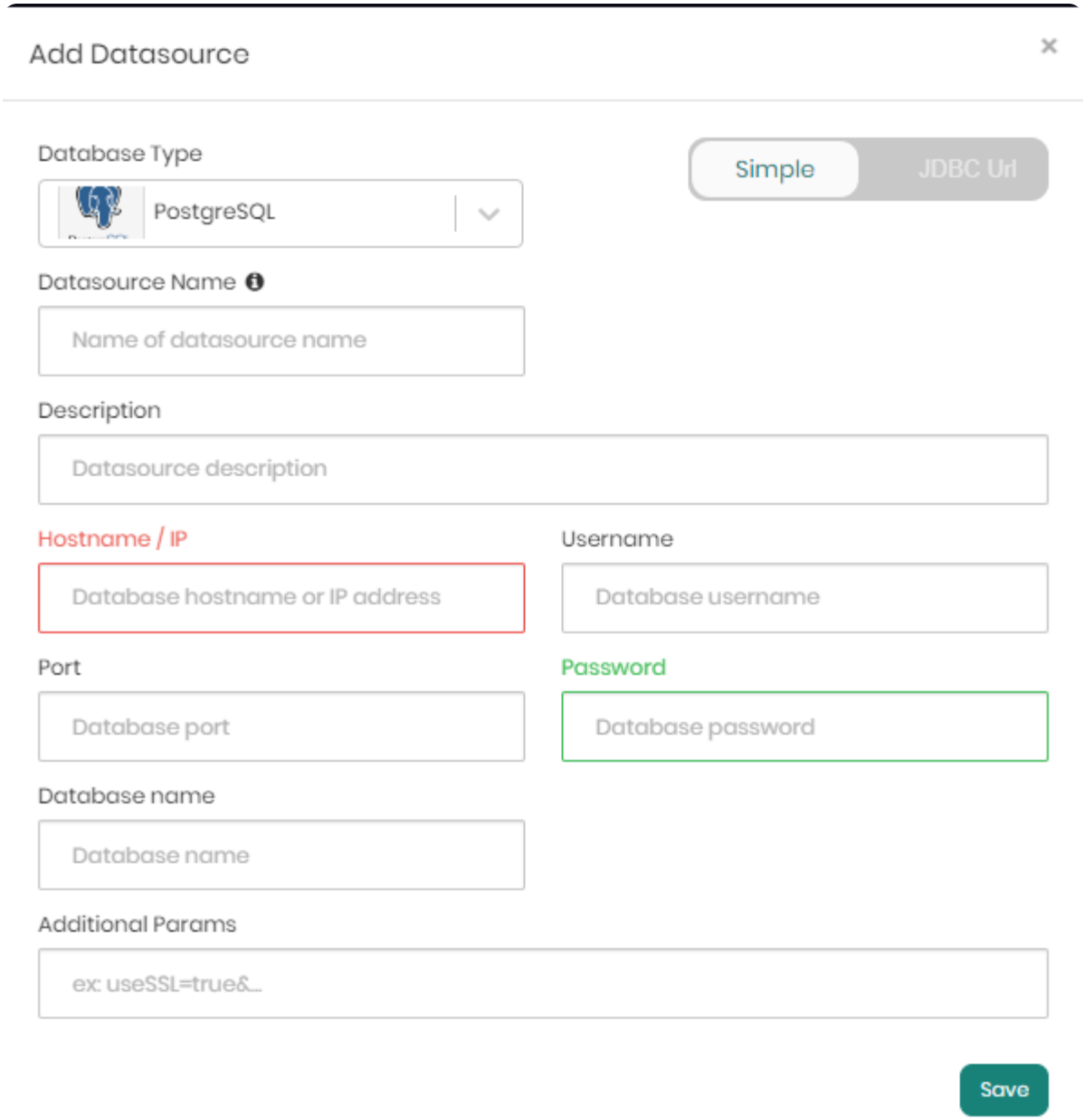
When **Add New Datasource** or the **Add button** are clicked on, a pop-up window will appear where all the relevant connection details must be provided in order to connect the datasource to the environment.



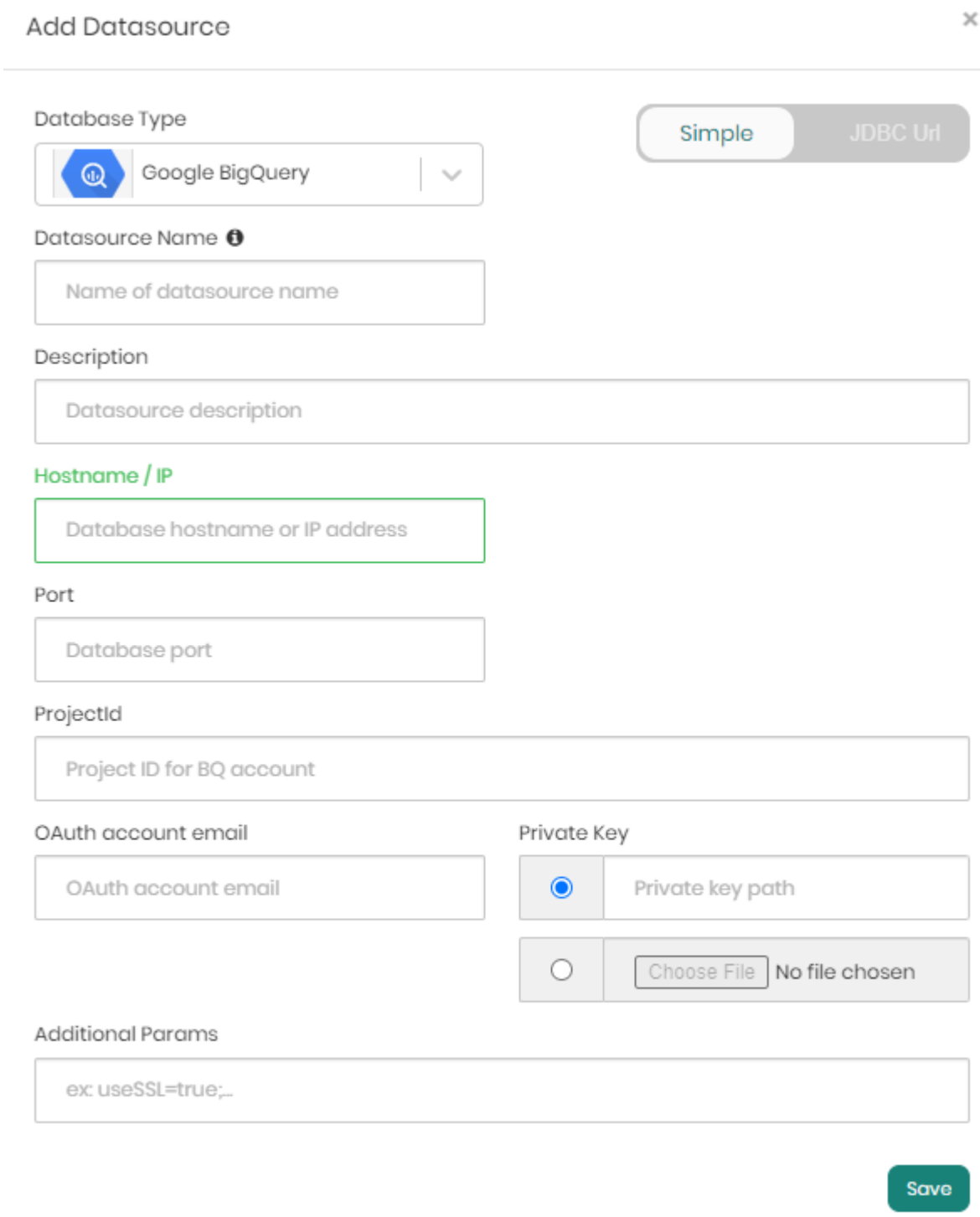
Each **back-end datasource** has slightly different requirements in order to connect to it, but the following information is required for the vast majority of the **back-end datasources:**

- The **hostname** for accessing the back-end datasource server
- The **port** for accessing the back-end datasource server
- The **username** for the back-end datasource
- The **password** for the back-end datasource

Some **back-end datasources** like **PostgreSQL**, **Athena**, and **SAP Hana** require to specify a **database name** in order to create a connection

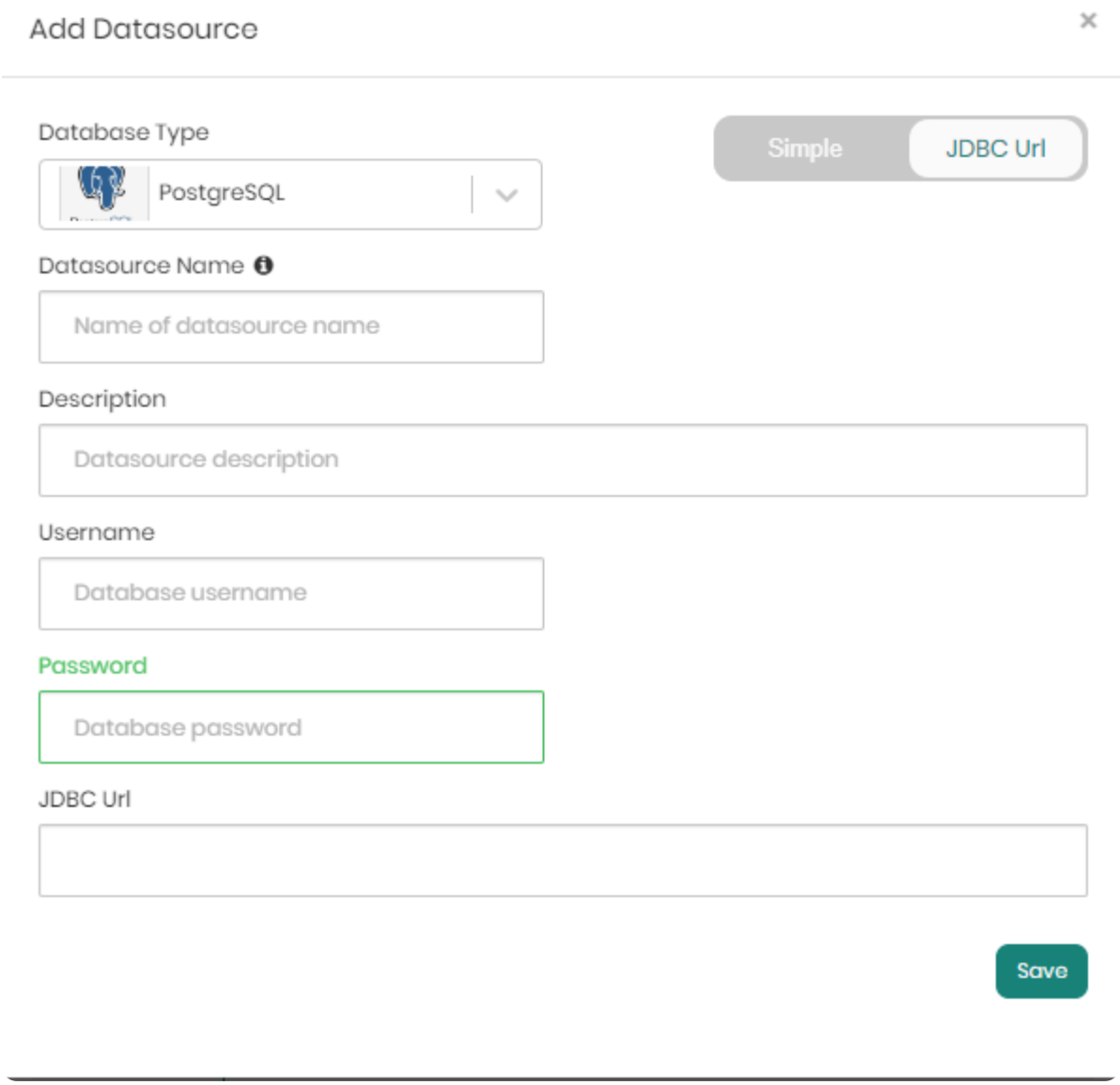


Other **back-end datasources** like **Big Query** require different information in order to create a connection. Parameters such as: - The **project id** as specified in the Google BigQuery project - The **associated email** with access to the Google BigQuery project specified in the **project id** field - A **private key** file in JSON format as provided by Google BigQuery for authentication



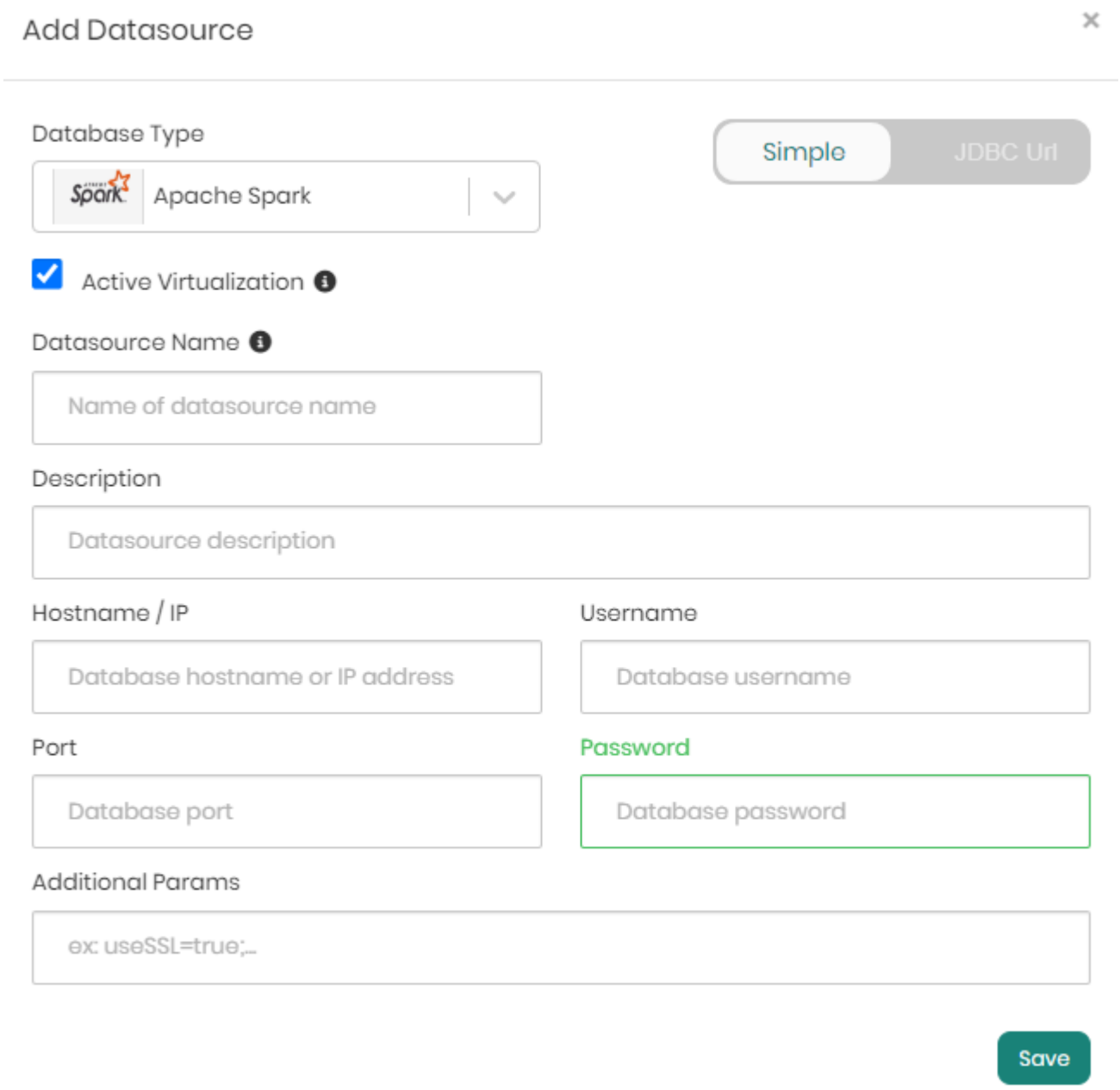
If supported by the **datasource back-end**, optional **additional parameters** (Additional Params) can be integrated into the connection between the **Knowledge Graph** and the **back-end datasource**

On the top right, the toggle can be switched from **Simple** to **JDBC Url** in order to connect the datasource using the relevant JDBC URL.



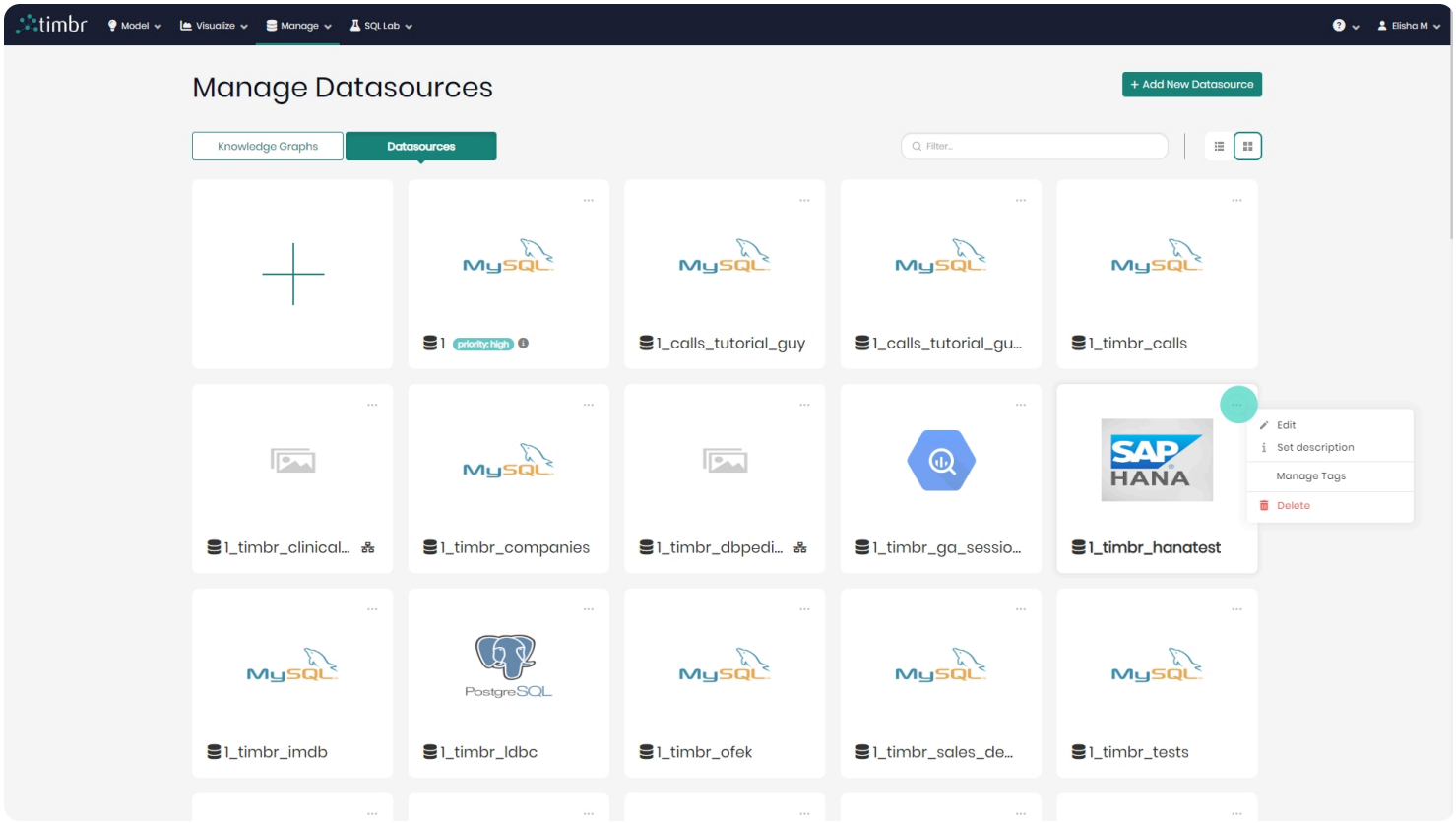
When the details are entered either with a username and password or via the JDBC URL, **Save** needs to be clicked in order to save the information and connect the datasource to the environment.

Timbr enables to add **Virtualization** for the **Apache Spark** and **Databricks** datasources. When these datasources are selected, a checkbox with an option for *Active Virtualization* will appear, indicating whether virtualization is on or off for the specific datasource. This option can be switched on and off at a later time as well.

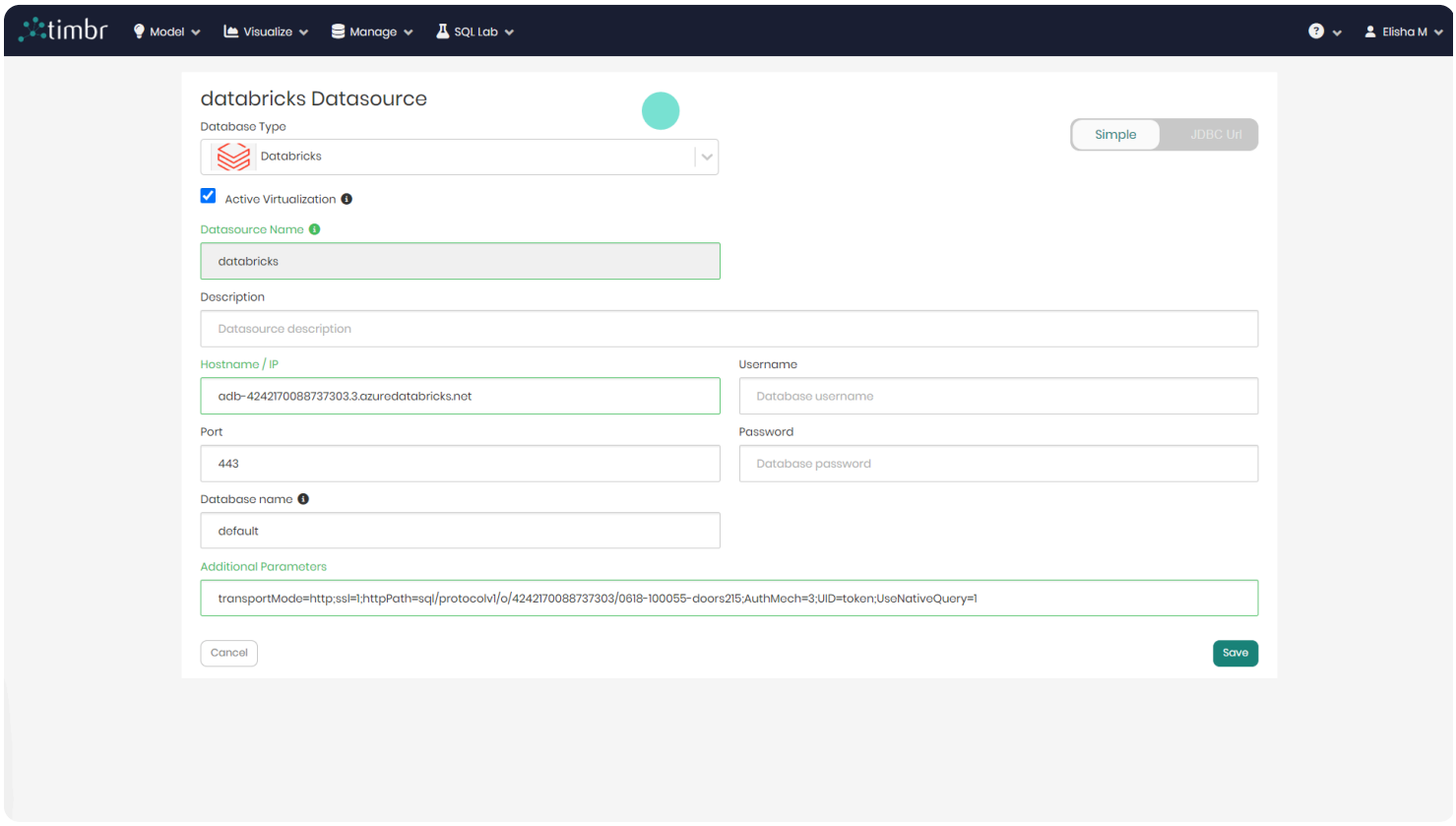


Editing an existing Datasource

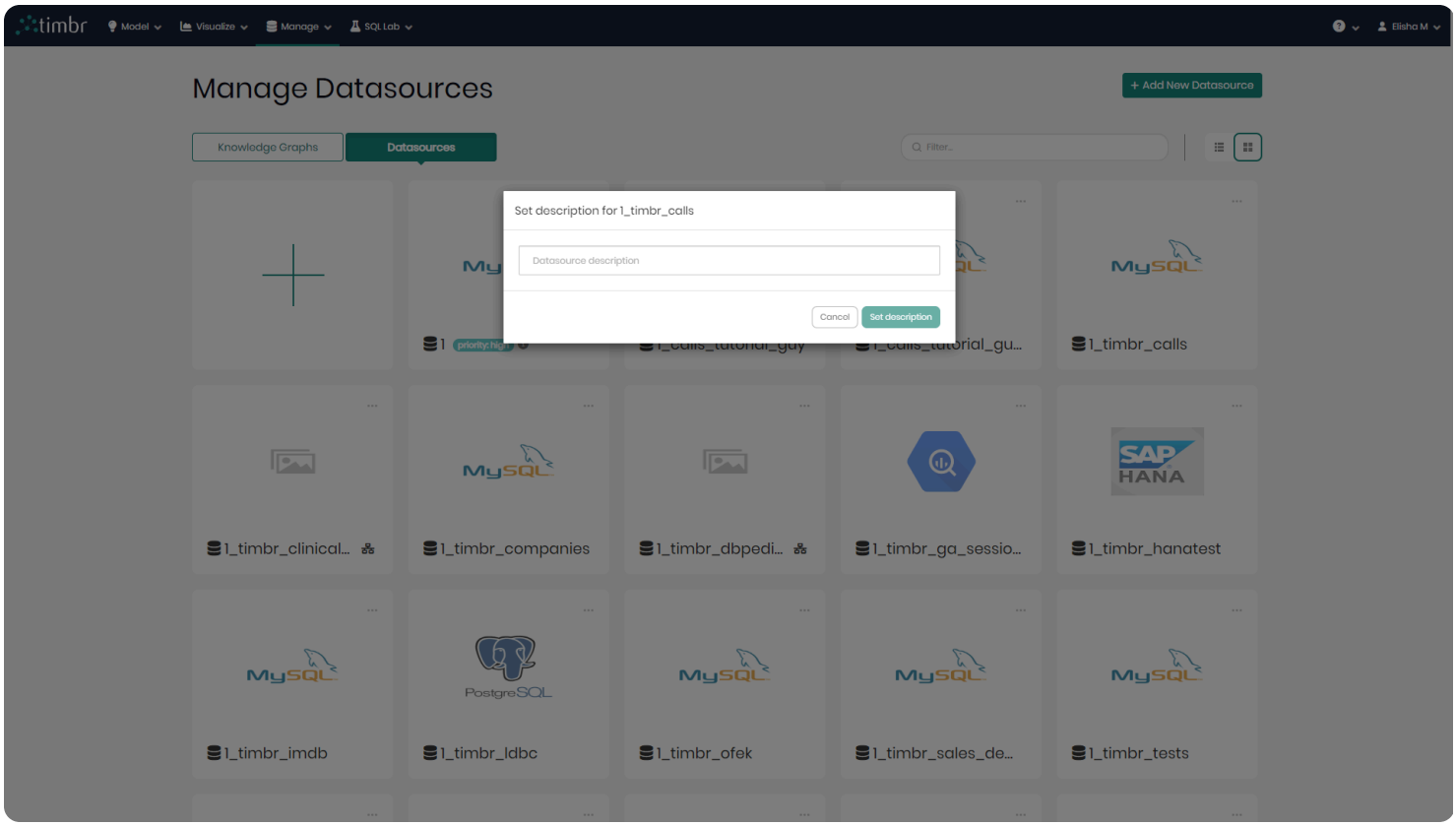
Each datasource both in box or list view contains 3 horizontal dots that when clicked on offer the following additional options on the selected Datasource:



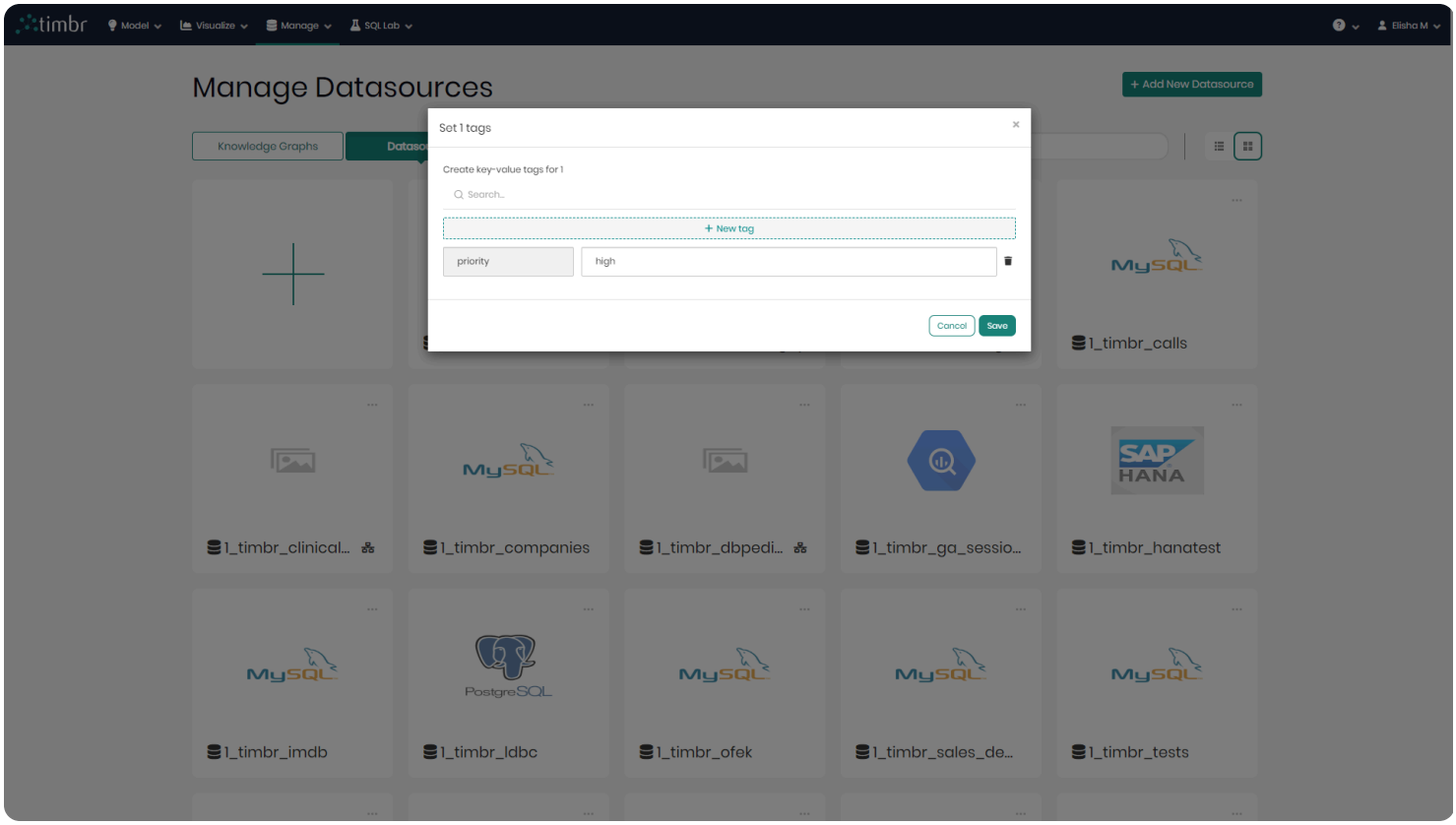
Edit - Opens a window to edit the selected datasource and its different configurations.



Set description - Enabling you to add a description to the datasource to further explain what it represents.



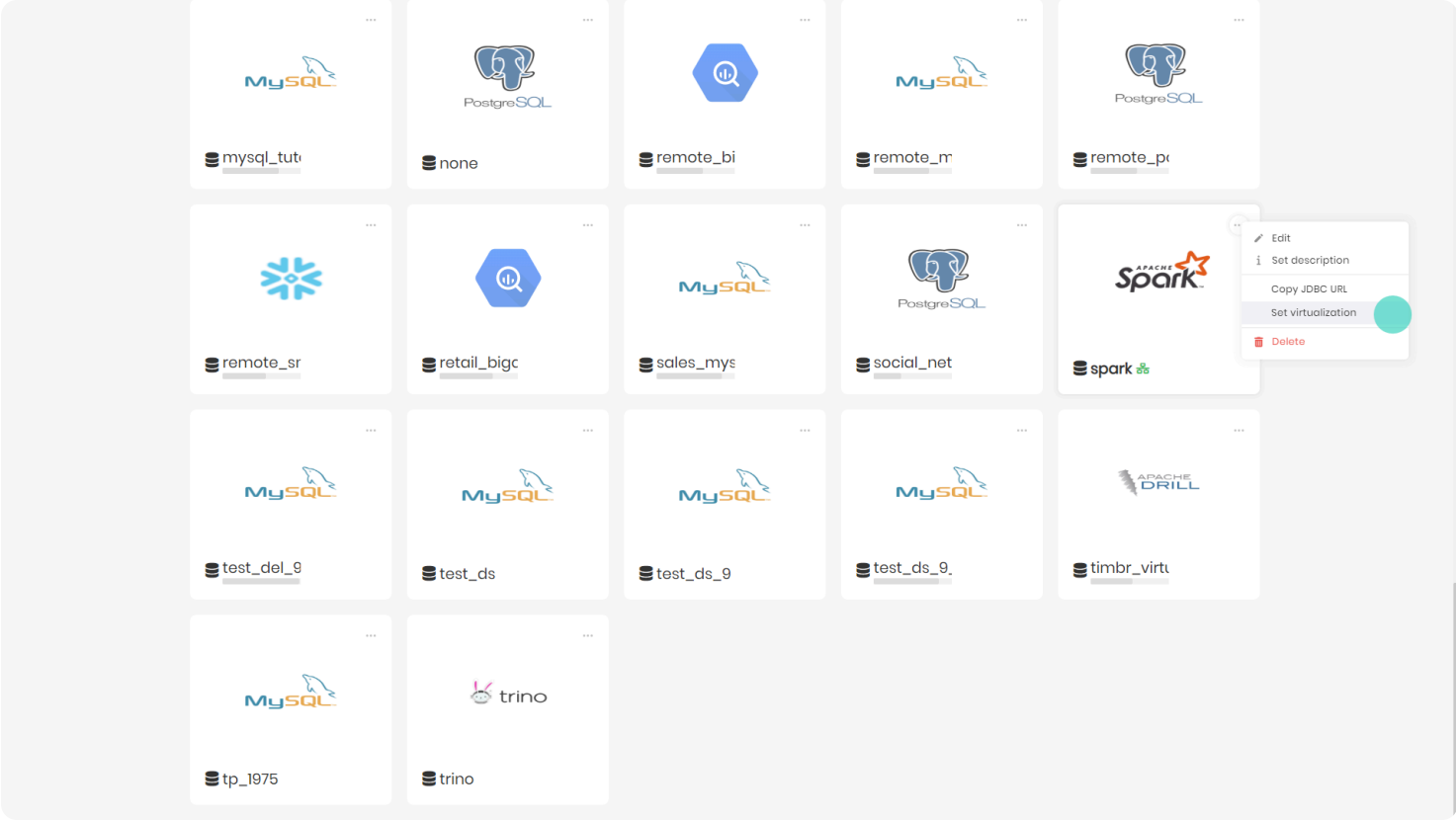
Manage Tags - Opens a pop-up window to manage the existing tags given to the knowledge graph, as well as the option to add new tags.



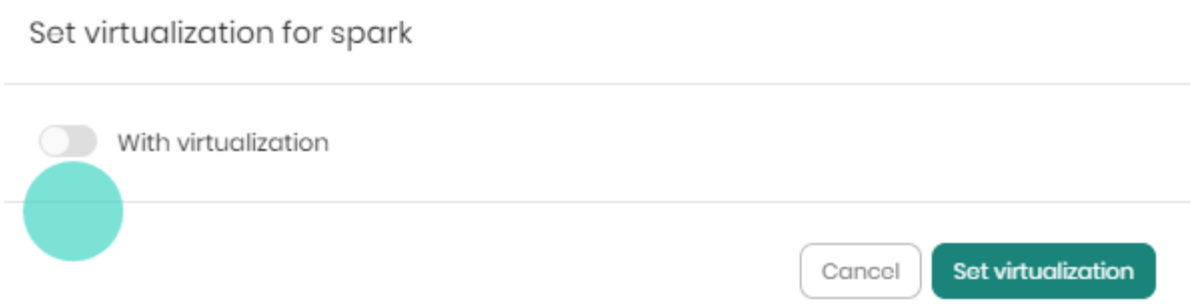
Delete - Deletes the selected datasource.

Timbr offers **Virtualization** with **Apache Spark** and **Databricks** so in addition to the name given to the datasource, a small icon will appear on all **Apache Spark** or **Databricks** datasources signifying that those datasources have the ability of virtualization. When the virtualization is on the icon will be green, when it is turned off the icon will be black.

To turn the *Virtualization* on and off the 3 horizontal dots must be clicked. Here you will find **Set Virtualization** In addition to the 4 options every datasource has when the dots are clicked, which are: *Edit*, *Set description*, *Manage Tags*, and *Delete*.



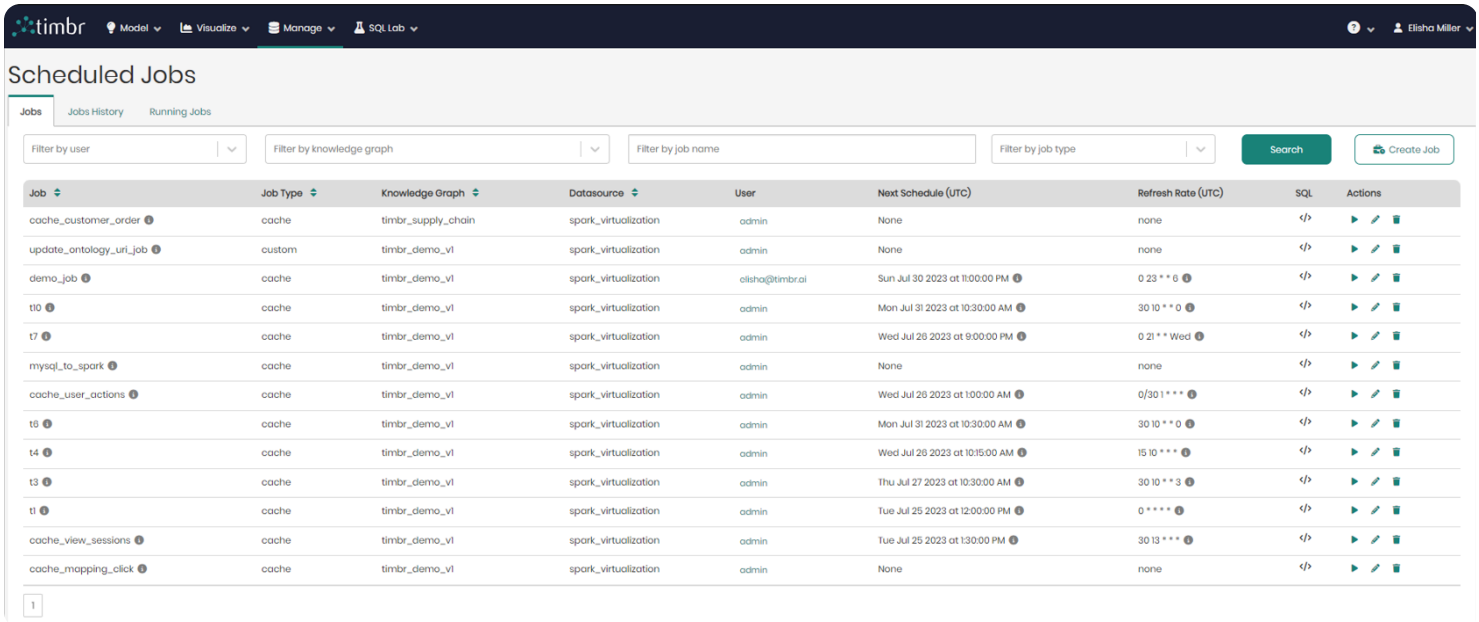
After *Set Virtualization* is clicked, in the window that appears, when the **With virtualization** toggle is turned on and *Set virtualization* is clicked on the bottom right, virtualization will be turned on and the icon will show green. When the **With virtualization** toggle is turned off and *Set virtualization* is clicked on the bottom right, virtualization will be turned off and the icon will show black.



Scheduled Jobs

The Scheduled Jobs component in the Timbr platform is for users that have enabled the cache mechanism in Timbr and set scheduled jobs to cache either mappings or views. Users can manage and edit existing jobs, running jobs, create future jobs, or review past jobs.

The Scheduled Jobs component centralizes every cache done in the Timbr platform, regardless of if it was made from the Data Mapper, Ontology Views, SQL Editor, or even if it was triggered outside of Timbr using an SQL endpoint connected to the platform.



Jobs

When entering the Scheduled Jobs page, the default tab that appears is the Jobs tab, which presents all the information on current Jobs that are scheduled by the users.

In the center of the screen is the list of all the current jobs that are scheduled. The list contains the following columns:

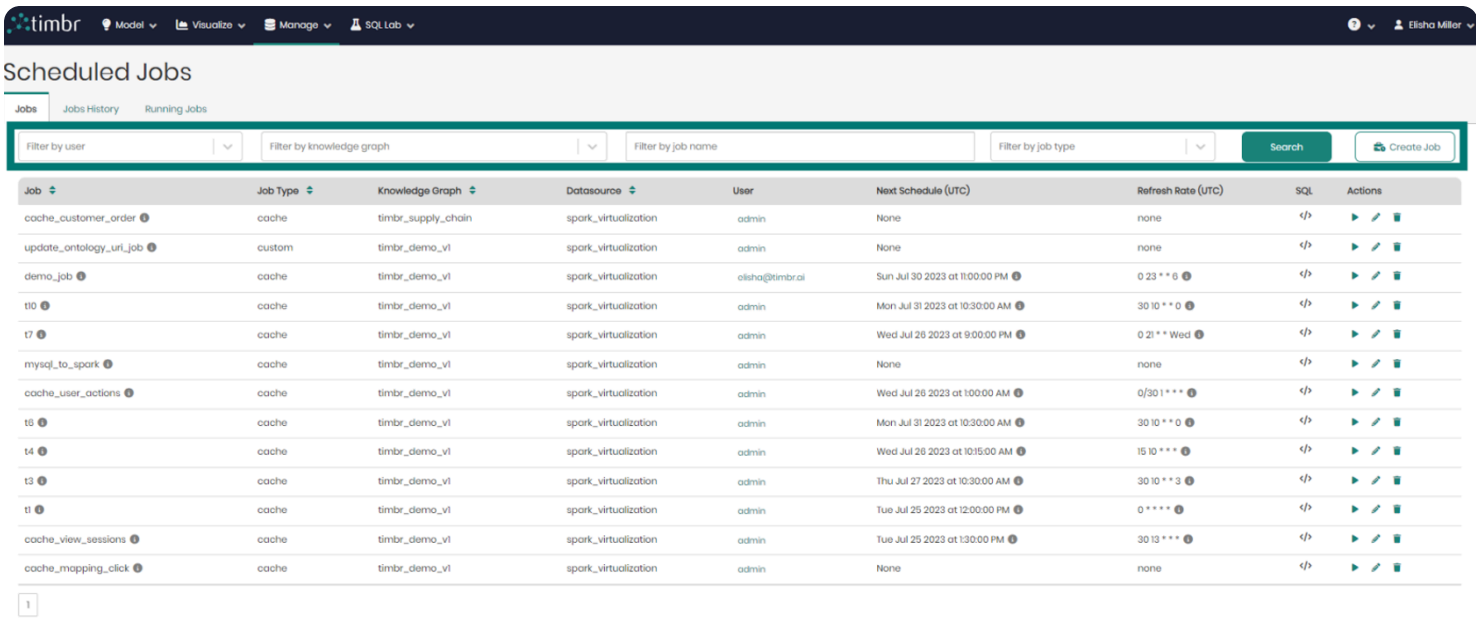
- **Job** - Presents the title of the jobs. When hovering over the tooltip, you can see the resource name and the user who scheduled the job.
- **Job Type** - Presents the type of each job.
- **Knowledge Graph** - Presents the name of the Knowledge Graph that the job is taking place on.

- **Datasource** - Presents the Datasource where the job will run.
- **User** - Shows the user who scheduled the job.
- **Next Schedule (UTC)** - States the exact time and date on which Timbr will run or refresh the job.

> **FYI**
>
> It's important to mention that by default Timbr will show times in the UTC format, but when hovering over the tooltip, users can see the next schedule according to their local time.

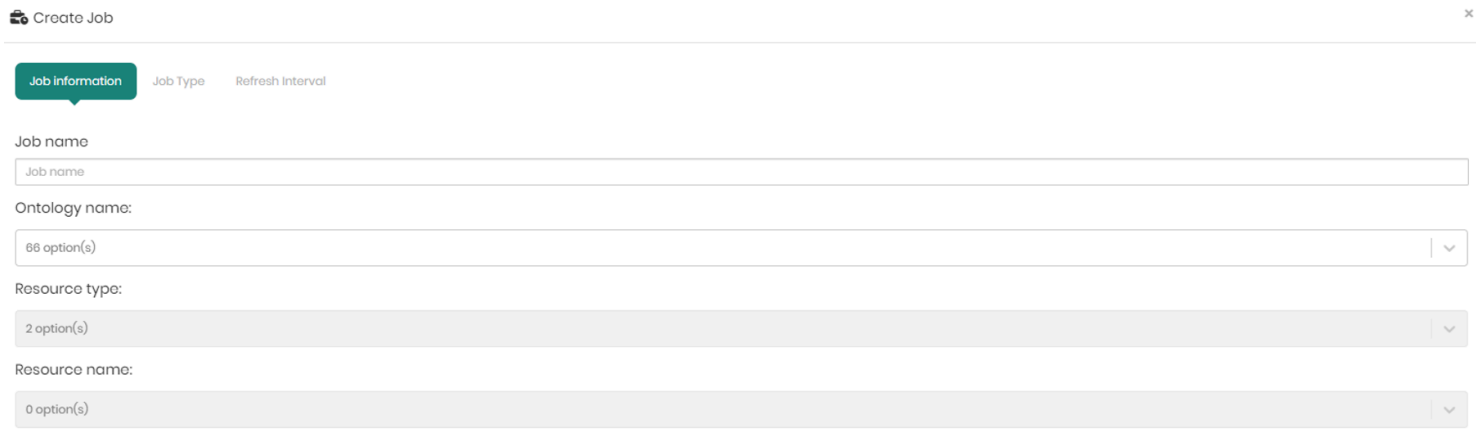
- **Refresh Rate (UTC)** - States the refresh interval rate for the job if any was defined.
- **SQL** - When hovered on, shows a preview of each cache query and its SQL syntax.
- **Actions** – There are 3 options under actions which are:
  - **Run job** – When the triangle play button is clicked on the selected job will begin running.
  - **Edit job** – When the pencil icon is clicked on an edit job window will appear.
  - **Remove job** – When the trash can logo is clicked on the selected job will be removed from the list of current jobs.

Above the list of jobs and beneath the 3 tabs is the upper filter pane which contains the following options:



- **Filter by user** - Enables searching through and filtering the jobs by specific users.
- **Filter by knowledge graph** - Enables searching through and filtering the jobs performed on any specific knowledge graph.
- **Filter by job name** - Enables searching through and filtering the jobs based on the job name in free text search.
- **Filter by job type** - Enables searching through and filtering the jobs based on the job type, being either a cached, custom, or cache incremental job.

- **Search** – Clicking on search performs the filtering based on the parameters chosen in the dropdowns on the left.
- **Create Job** – Opens a window in order to create a new job.

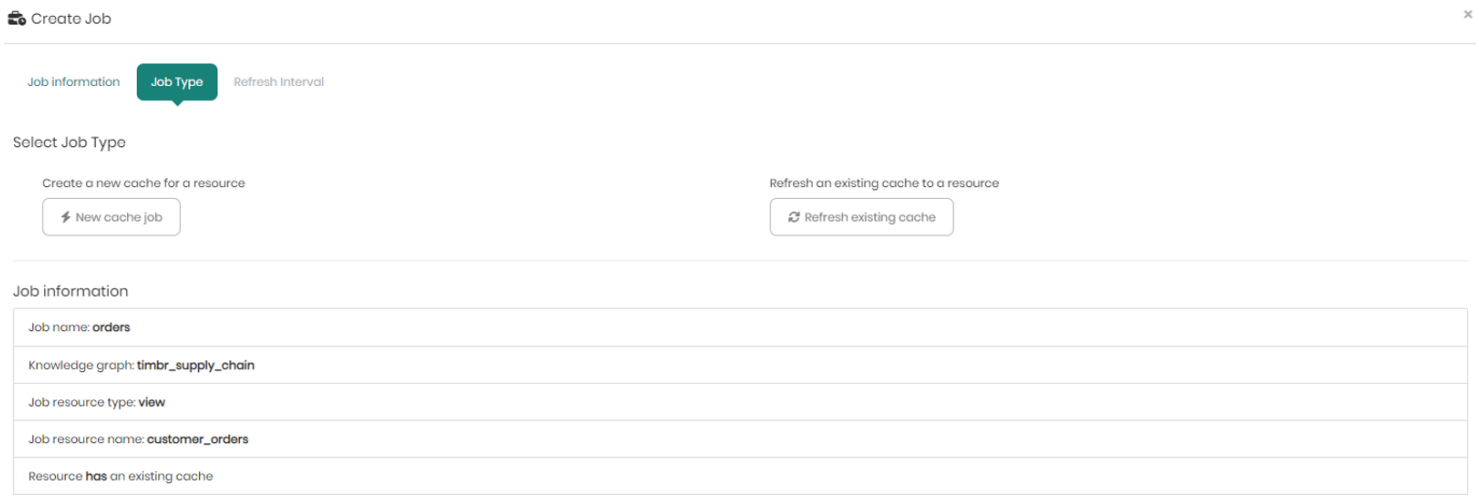


The create job window that appears contains 3 tabs which are **Job Information**, **Job Type**, and **Refresh Interval** which are in charge of the following:

Job Information – In the job information tab there are 4 dropdowns to choose from:

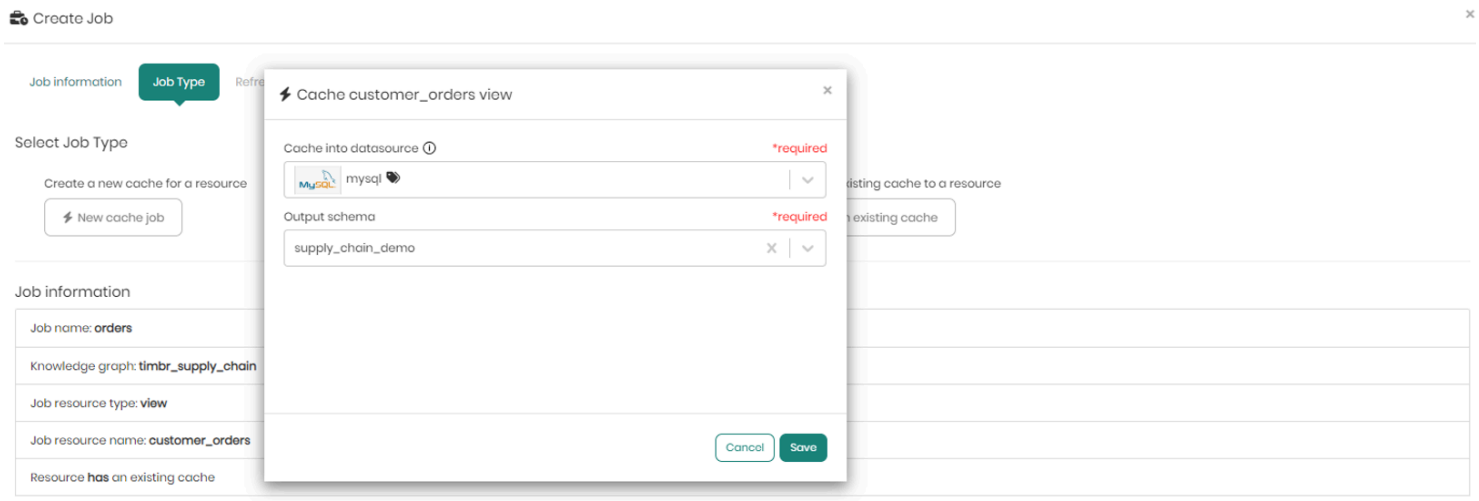
- **Job name** – To name the job.
- **Ontology name** - To choose the relevant ontology.
- **Resource type**- To choose the resource type to cache, either a mapping or a view.
- **Resource name** - According to the resource type, Timbr will present the mappings or views of the selected ontology to choose the resource you would like to cache.

Once the information is filled in, click Next to move to the next tab – **Job Type**.

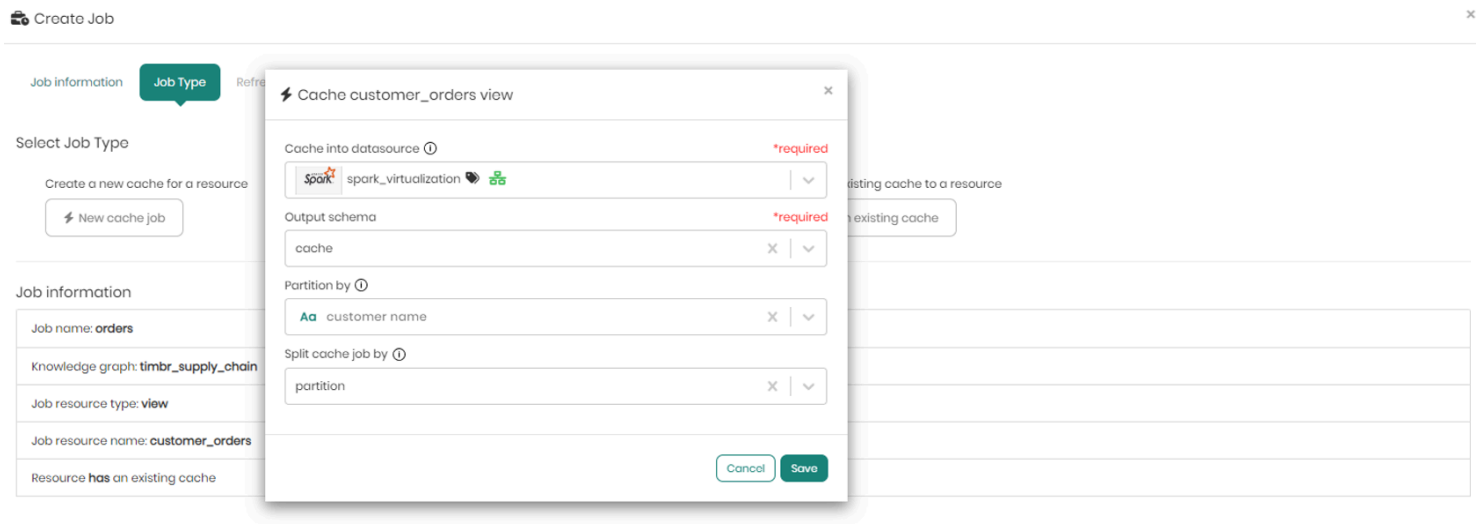


In the **Job Type** tab, you choose whether it's an existing cache job that you want to refresh, or if this is a new job for the chosen resource.

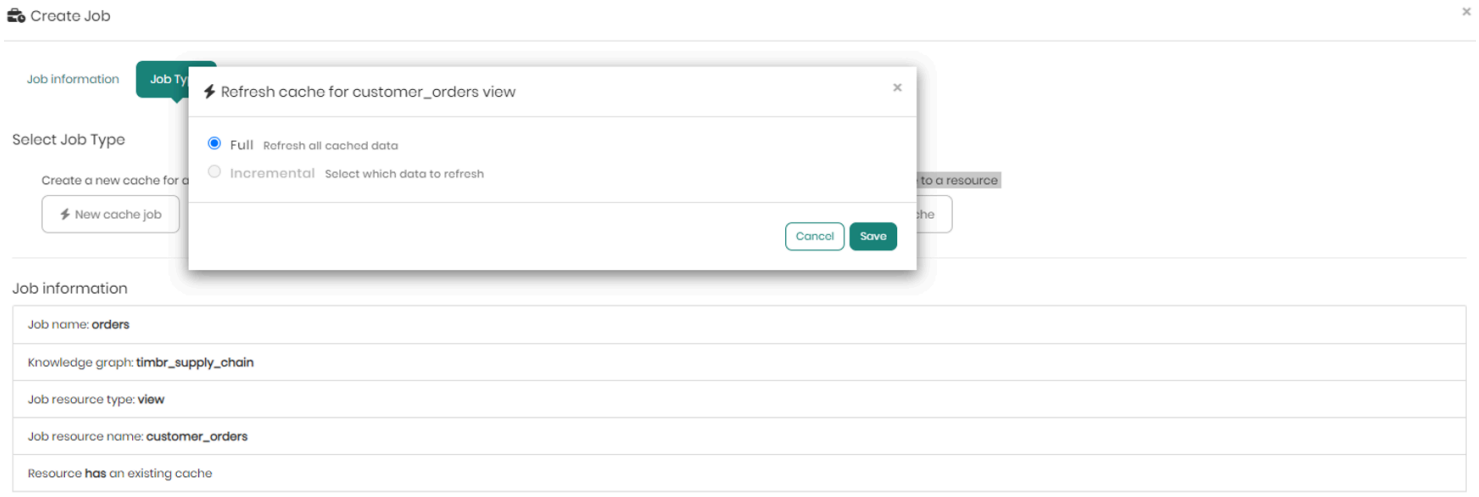
When clicking on **New cache job**, a pop-up window appears asking to choose the datasource and schema to store the cache in. Once chosen click on **Save** to move on to the third and final tab.



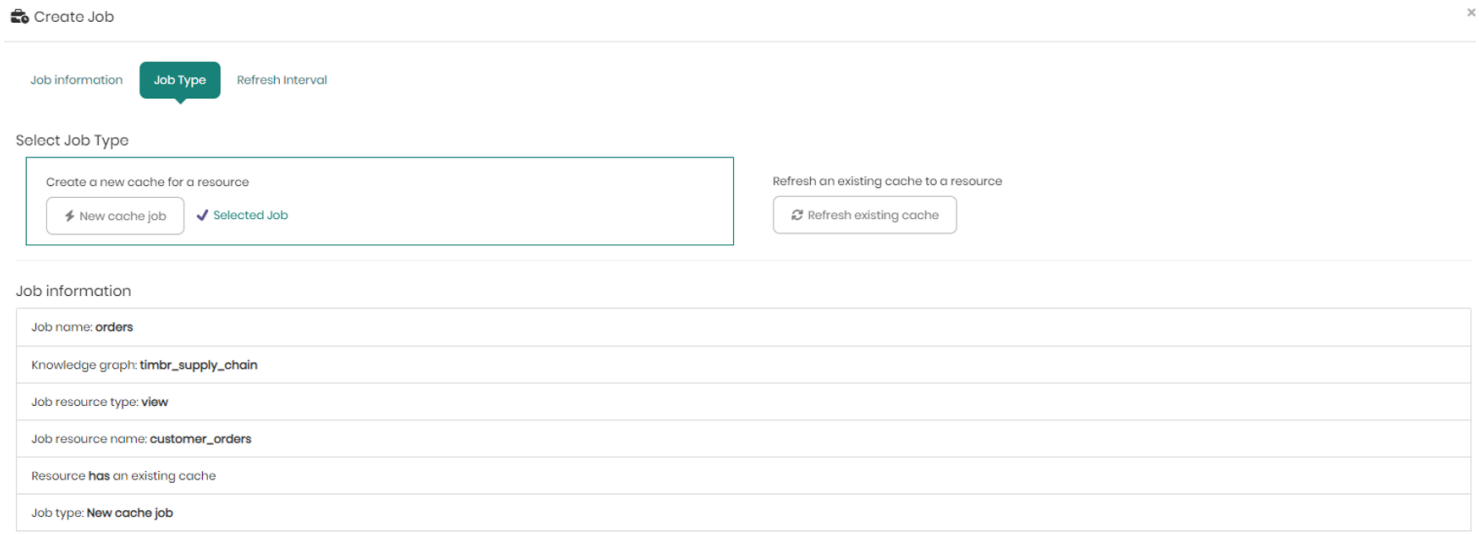
> **NOTE**
>
> When choosing a virtualization engine such as *Spark* or *Databricks*, there will be an additional option to partition, in case you want to partition the cache by a certain column. This is a very useful feature to have when dealing with big tables and large amounts of data to apply incremental refresh of the cache instead of caching the entire data each time.



When choosing to refresh an existing cache to a resource instead of creating a new one, a pop-up will appear enabling us to choose whether to perform a **full refresh** on all the cached data, or instead perform an **incremental refresh** selecting which specific data to refresh (valid only for data lakes or virtualized data sources with partition).



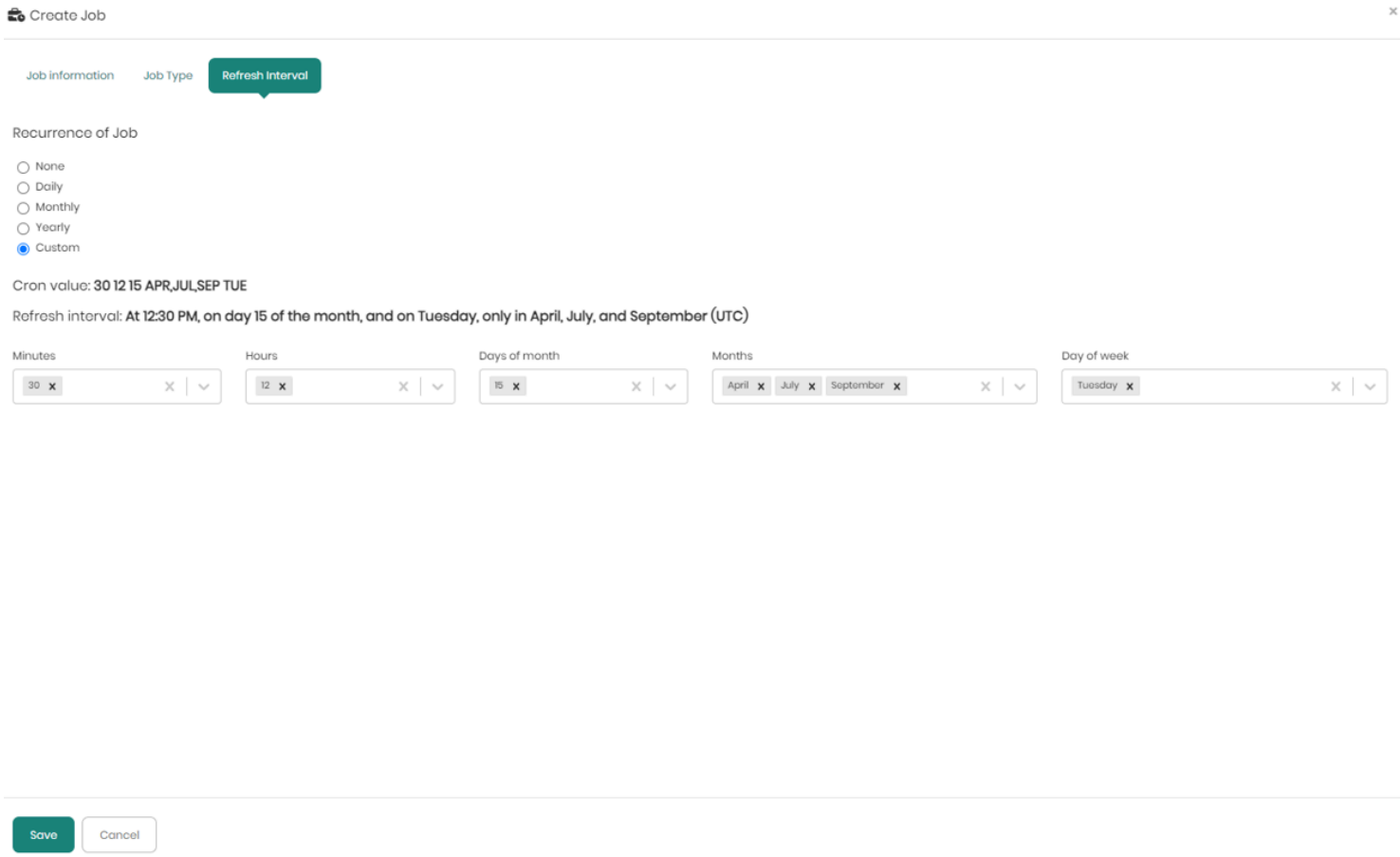
On the bottom half of the screen in the Job type tab, you can see the job information including the **Job name**, **Knowledge graph**, **Job resource type**, **Job resource name**, whether or not the resource has an existing cache or not, and finally the **Job Type** if a new job was selected.



Once selecting the new job with its data source and schema, and a partition column when relevant, move on to the third and final tab - **Refresh Interval**.

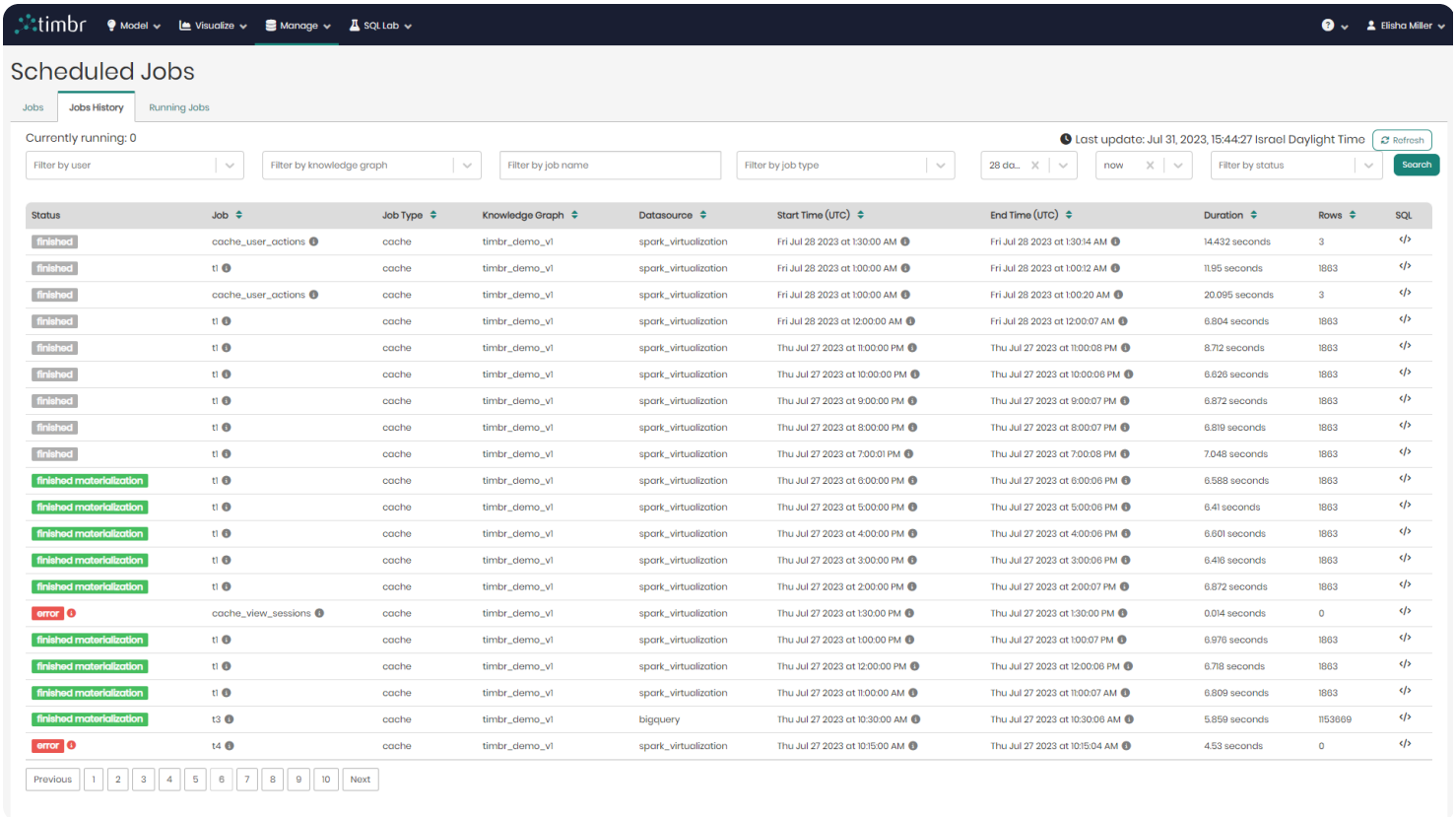
In the **Refresh Interval** tab you can define the cache refresh Interval with the following options:

- **None** - Without a refresh, for tables that doesn't require a refresh interval
- **Daily** - Executing the current job on a daily interval
- **Monthly** - Executing the current job on a monthly interval
- **Yearly** - Executing the current job on a yearly interval
- **Custom** - When choosing custom users use the standard CRON options as if you ran a cron job before with the following options:
  - **Minutes** – Choosing any minute within the range of 0-59 representing the minutes in each hour.
  - **Hours** - Choosing any hour within the range of 0-23 representing the 24 hours in each day.
  - **Days of month** – Choosing any day of the month within the range of 1-31 representing the maximum 31 days of a month.
  - **Months** - Choosing any of the months within the 12 months of the year.
  - **Day of the week** - Choosing a specific day within the 7 days of the week, regardless of the numeric date on that day.



Once the values are chosen in the dropdowns the **Cron value** and **Refresh interval** will be presented above the dropdowns. To save the job and the chosen refresh interval **Save** must be clicked on the bottom left.

Jobs History



The second tab in the Scheduled Jobs page is the **Jobs History** tab, which presents all the historical cached jobs performed by users.

beneath the tabs is the *upper filter pane* which contains the following options:

Currently running - Presents the number of cached jobs that are currently running.

Filter by user - Enables searching through and filtering the cached jobs by the specific user who performed the caching.

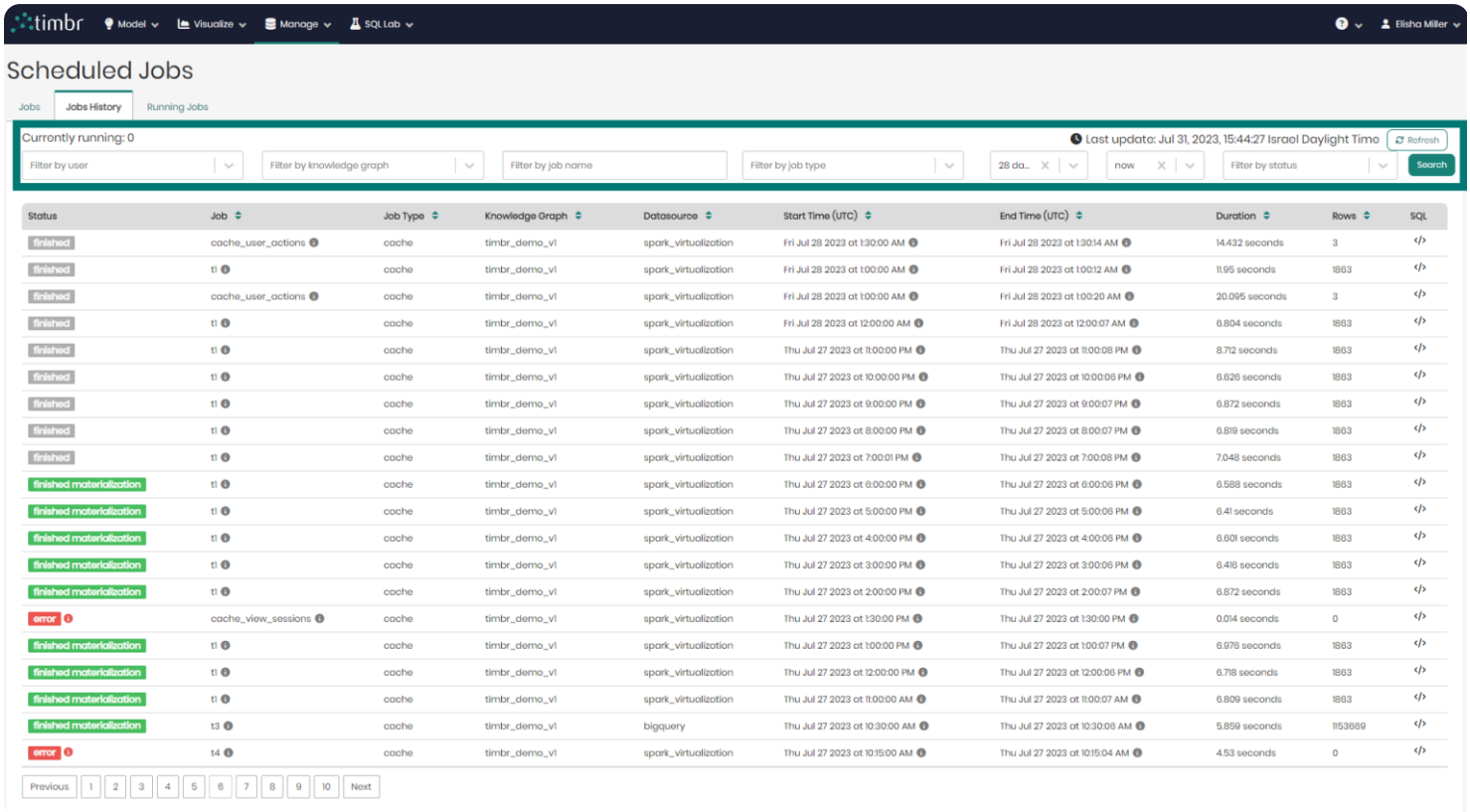
Filter by knowledge graph - Enables searching through and filtering the cached jobs performed on specific knowledge graphs.

From & To - Enables searching through and filtering the cached jobs based on a specific time frame that they were performed at.

Filter by status - Enables searching through and filtering the cached jobs based on a specific cached status such as *finished materialization* or *error*.

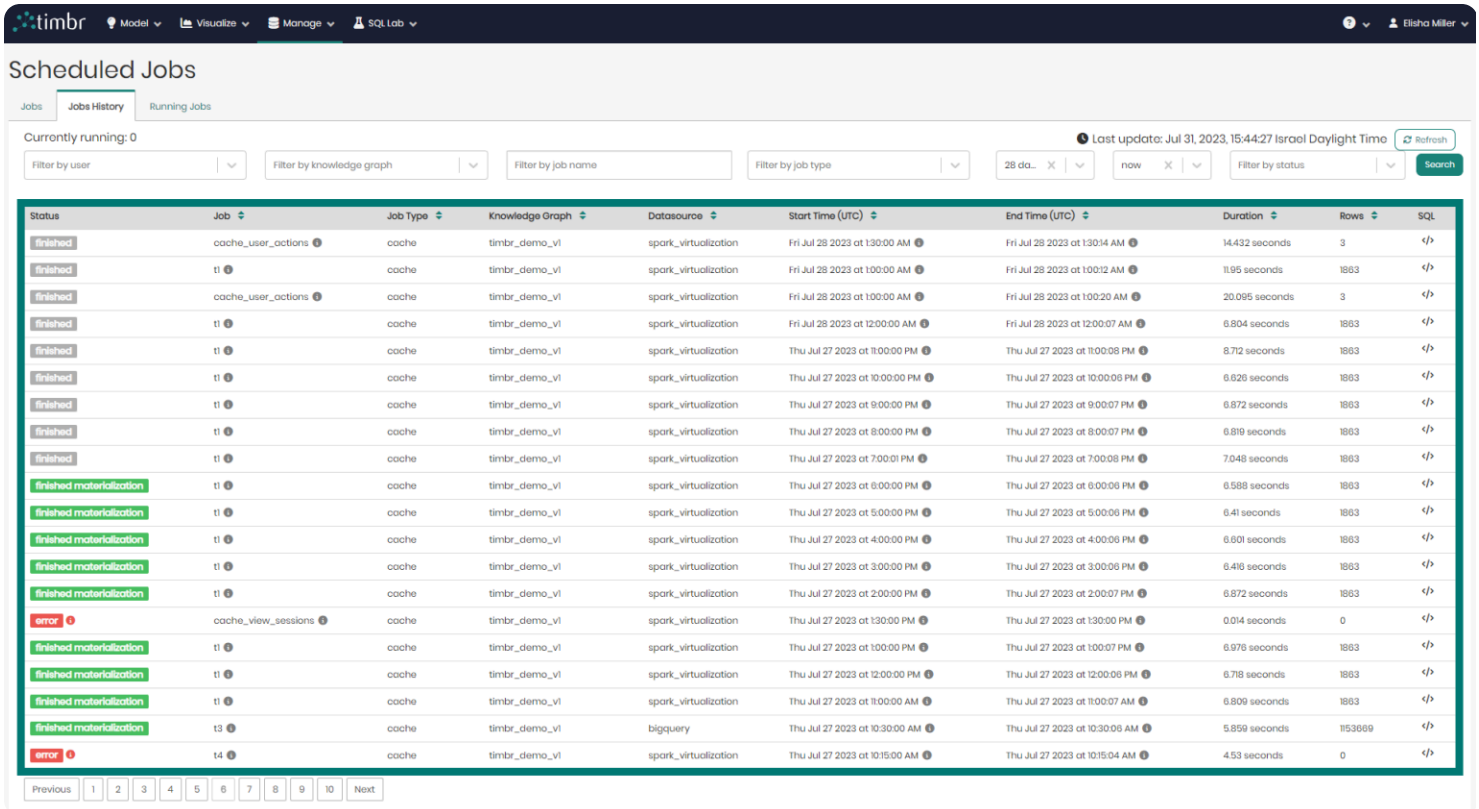
Refresh - Performs a refresh on the list of historical cached jobs.

Search - Performs a search on the cached jobs based on the parameters chosen in the various filters.

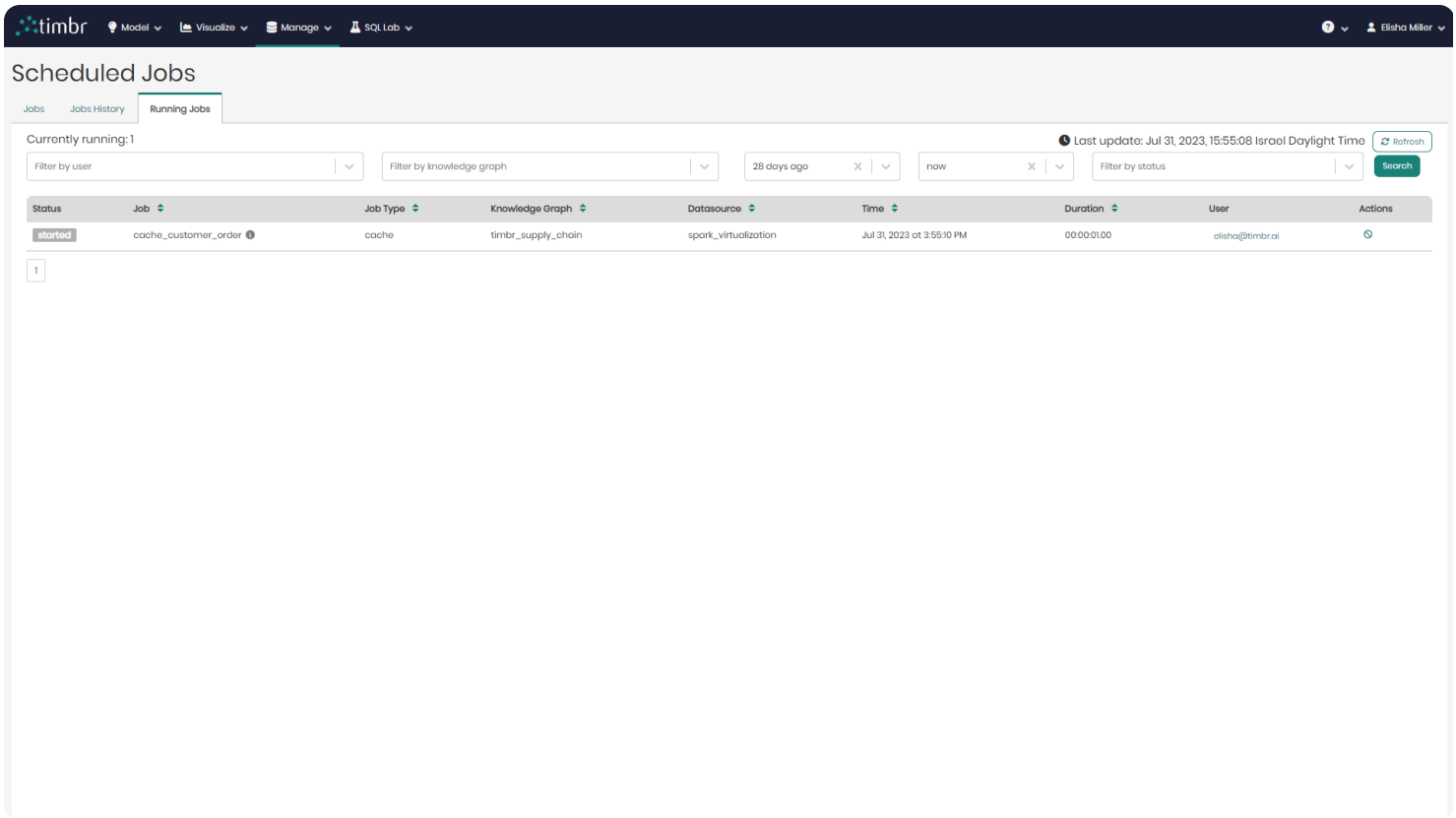


Beneath the upper filter pane is the entire list of historical cached jobs. The list contains the following columns:

- **Status** - Presents the status of the cached jobs.
- **Job** - Presents the title of the cached jobs.
- **Job Type** - Presents the type of each cached job.
- **Knowledge Graph** - Presents the name of the Knowledge Graph that the cached job is taking place on.
- **Datasource** - Presents the Datasource that is being used in each cached job.
- **Start Time** - States the exact time and date each cached job began on.
- **End Time** - States the exact time and date each cached job finished running at.
- **Duration** - States the amount of time each cached job ran for.
- **Rows** - Shows the number of result rows that were present in each cached job.
- **SQL** - When hovered on, shows a preview of each cached query and its SQL syntax.



Running Jobs



The third tab that appears on the Scheduled Jobs page is the **Running Jobs** tab, which presents all the information about currently running Cached Jobs.

Similarly to the other 2 tabs, here too there's an **upper query pane** that contains the following options:

Currently running - Presents the number of cached jobs that are currently running.

Filter by user - Enables searching through and filtering the currently running cached jobs by the specific user who performed the cached job.

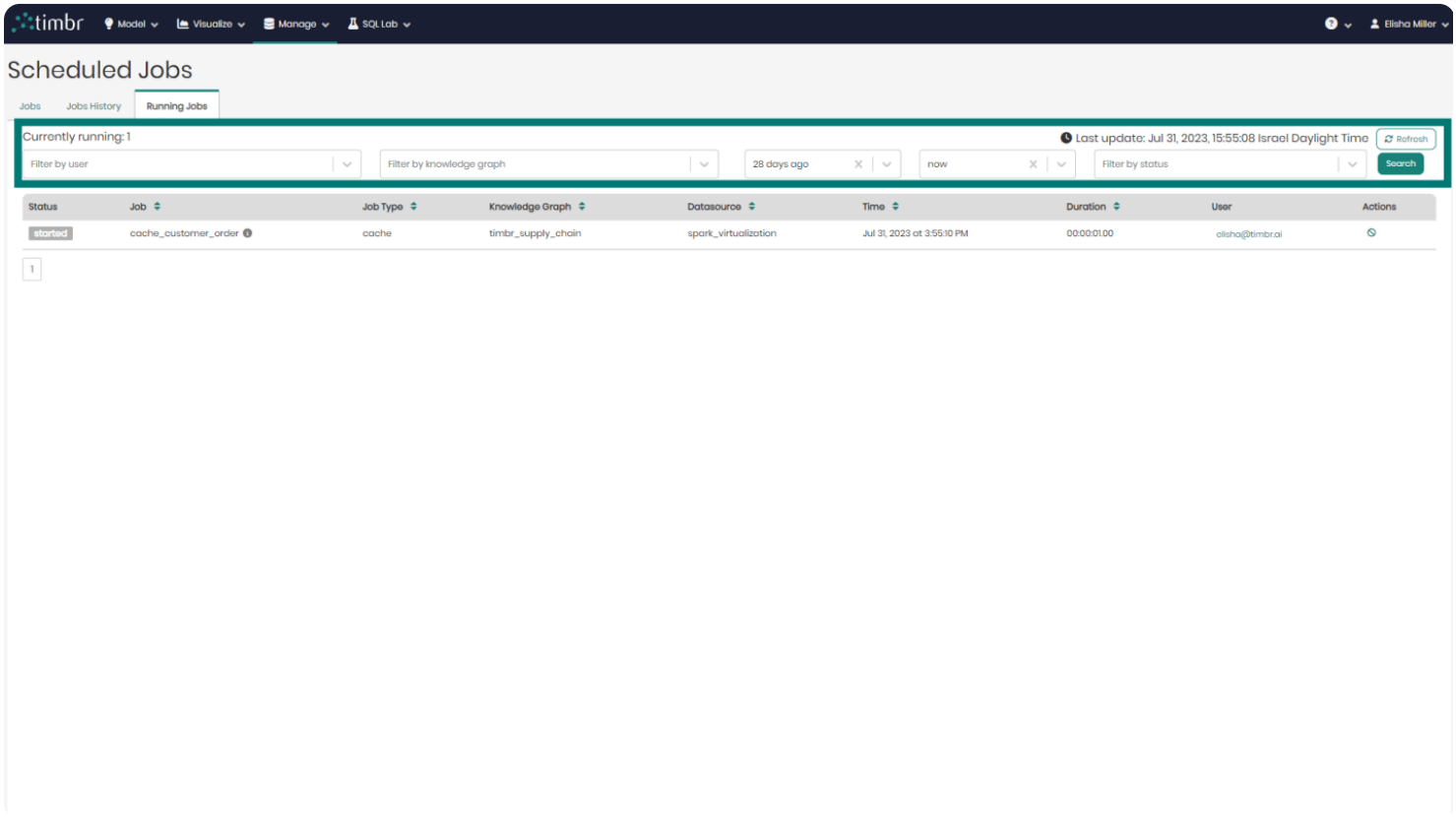
Filter by knowledge graph - Enables searching through and filtering the cached jobs that are currently running on specific knowledge graphs.

From & To - Enables searching through and filtering cached jobs that are currently running based on a specific time frame that they were performed at.

Filter by status - Enables searching through and filtering the cached jobs that are currently running based on a specific cached status such as *finished materialization* or *error*.

Refresh - Performs a refresh on the list of cached jobs that are currently running.

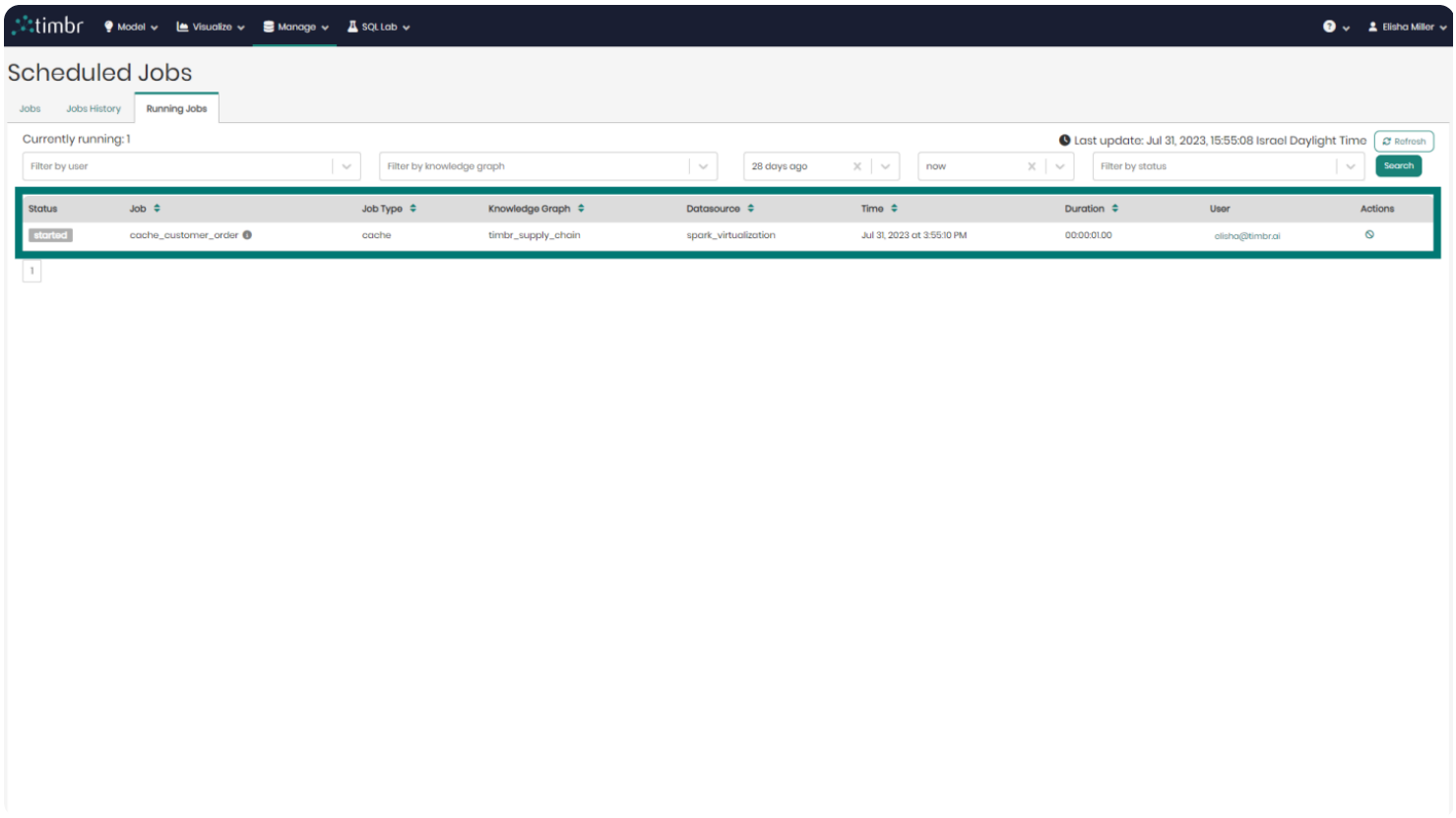
Search - Performs a search on the cached Jobs that are currently running based on the parameters chosen in the various filters.



Beneath the upper filter pane is the entire list of cached Jobs that are currently running. The list contains the following columns:

- **Status** - Presents the status of cached Jobs currently running.

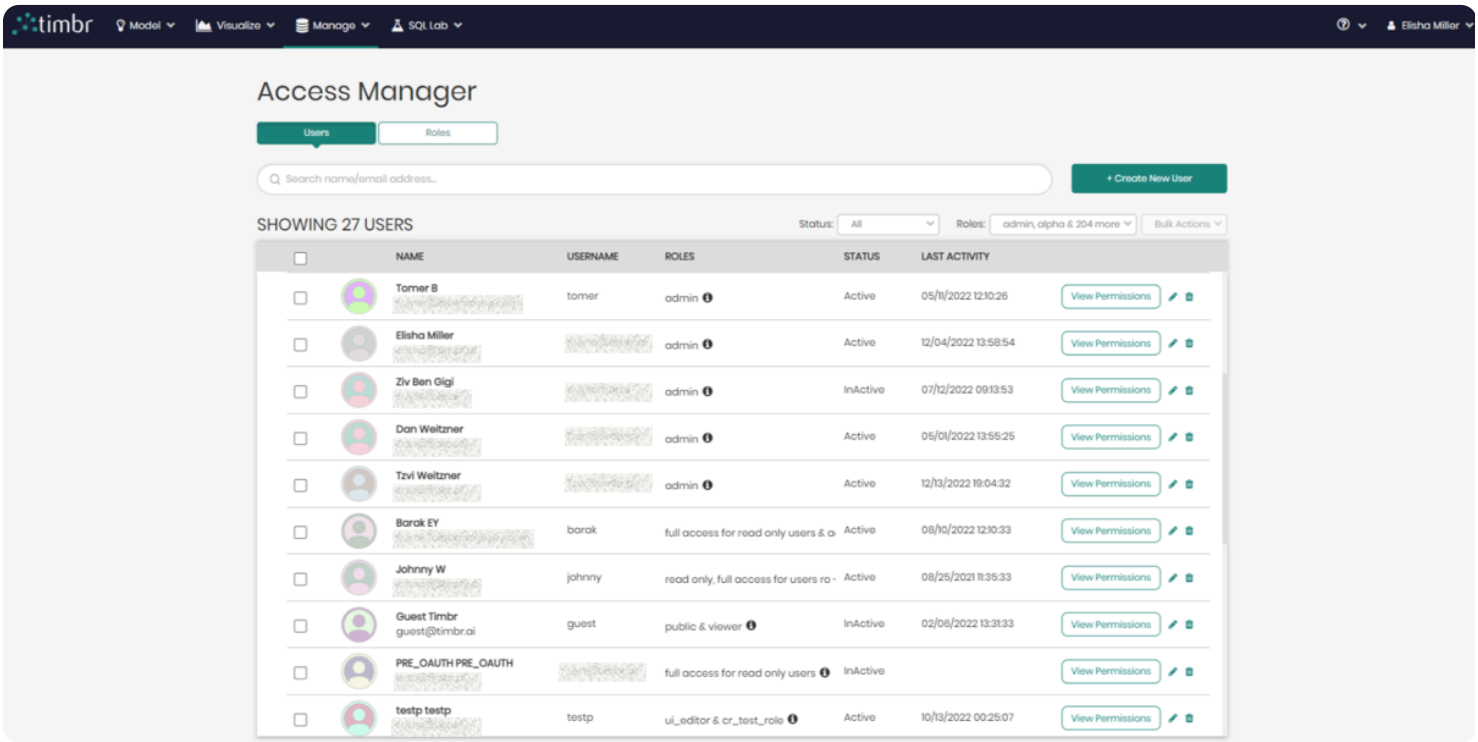
- **Job** - Presents the title of the cached Jobs currently running.
- **Job Type** - Presents the type of each cached job currently running.
- **Knowledge Graph** - Presents the name of the Knowledge Graph that the cached job currently running is taking place on.
- **Datasource** - Presents the Datasource that is being used in each cached job currently running.
- **Time** - States the exact time and date each cached job currently running began at.
- **Duration** - States the amount of time each currently running cached job has been running for.
- **User** - Shows the user who performed each currently running cached job.
- **Actions** - When clicked on, the currently running cache selected will be cancelled and will stop running.



Access Manager

The Access Manager component in Timbr is where users can create users and roles in the platform. Users can define roles based on projects, business functions, and data assets, with granular permissions to support any type of access. Admins can assign unique permissions to users without using a role.

The Access Manager is equipped with enterprise-grade access control and can integrate with **Azure AD**, **AD Roles**, and **SSO** mechanisms. Everything supported in the Access Manager can also be done using `GRANT` statements in SQL for those who prefer to code. This allows users to generate automated scripts based on input from other applications and always maintain controlled and governed access to data.



The **Access Manager** can be accessed through the **Manage** tab by clicking on **Access Manager**.

Introduction

Access control in Timbr is divided into two main aspects

1. **What a user can see** - **Access Permissions** - The user's access to the knowledge graph, datasources (tables), concepts, mappings, views, and all other data elements that a user is exposed to and can view, query, create and edit. This also includes the ability to grant or revoke the creation or modification of users, and roles.
2. **What a user can do** - **Platform Permissions** - The user's access to the different pages and components in the platform. This includes which components or pages a user has access to. For example the *Ontology Explorer*, *Data Mapper*, *Graph Explorer*, *Knowledge Lineage*, *Access Manager*, etc.

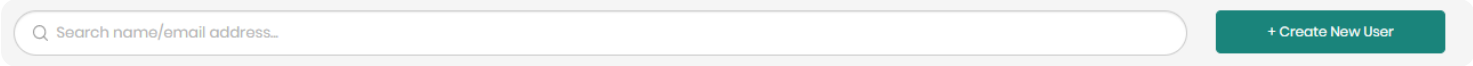
Getting Started

The Access Manager component can be accessed by clicking on the **Manage** tab and choosing **Access Manager**.

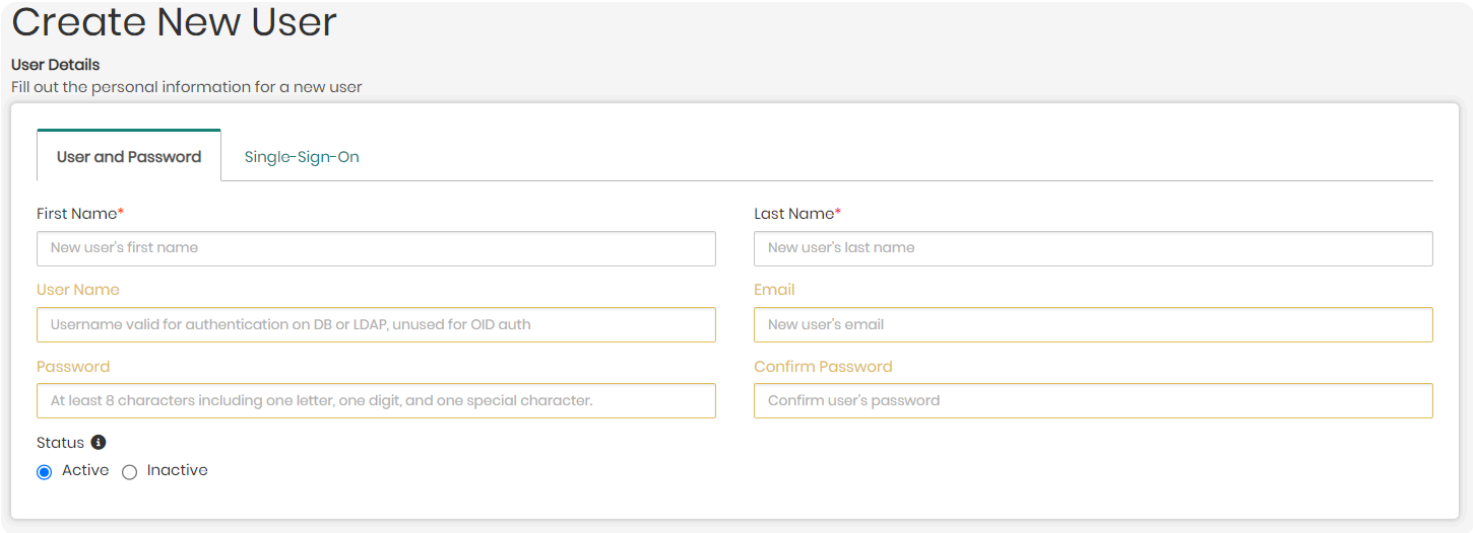
Once selected a list of all users will appear including their: *Name*, *Username*, *Roles*, *Status*, and *Last Activity*.

Above the list of users are the two tabs that enable you to switch between viewing the *Users* and viewing the different *Roles*.

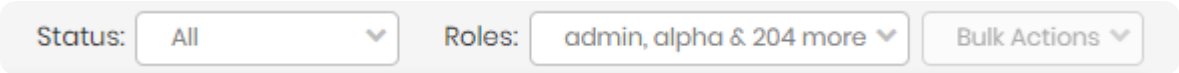
Beneath the two tabs is a search bar that can assist in finding specific users or roles from the list.



To the right of the search bar is **+ Create New User** which when clicked on will open a new box to create a new user.



Beneath **+ Create New User**, are different filtering options as well as bulk actions that can be done. This includes:

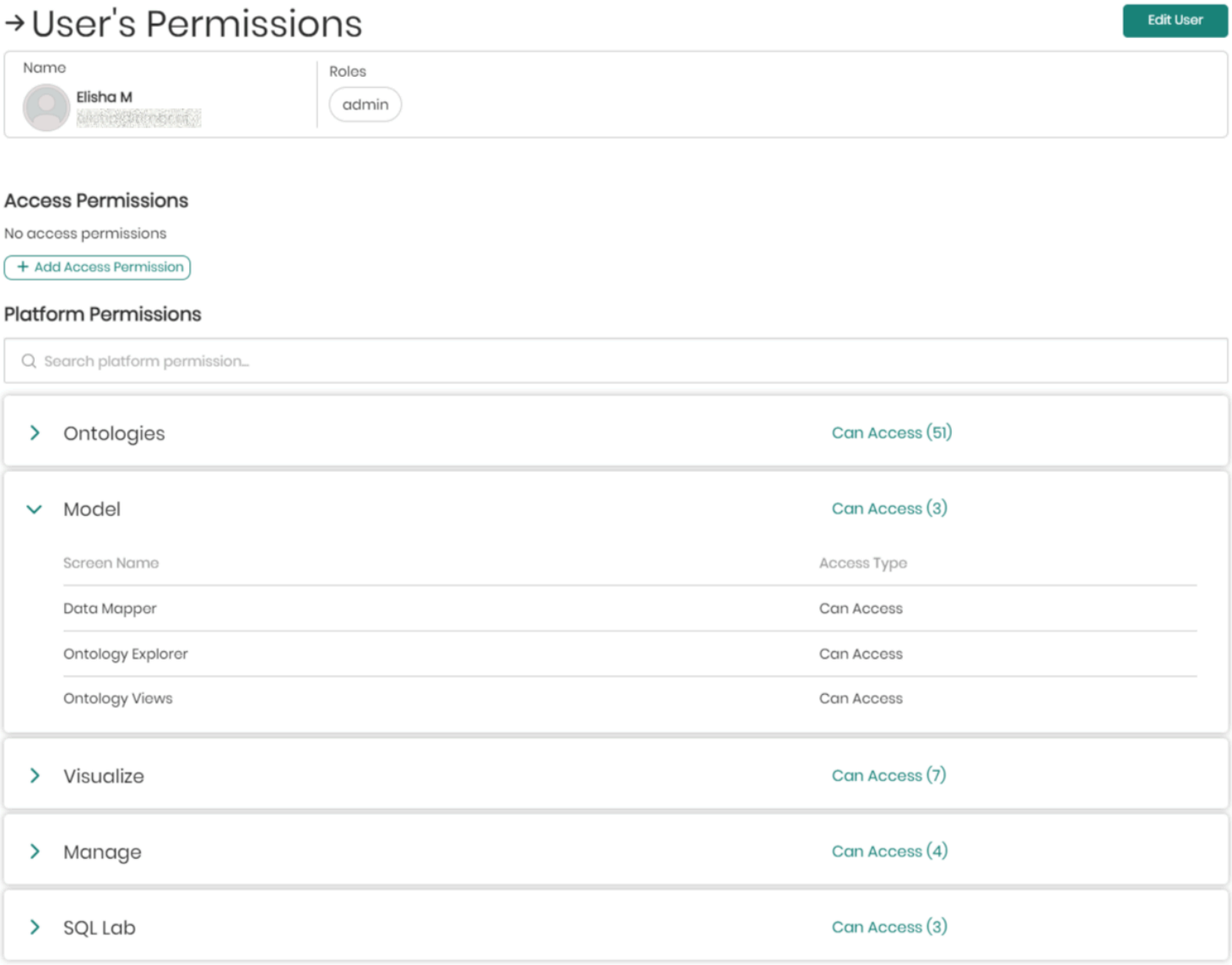


- **Status** - Filtering the graph based on Active/Inactive users.
- **Roles** - Filtering the graph based on specifically created roles.
- **Bulk Actions** - Enabling users to select multiple users to perform the following bulk actions on.

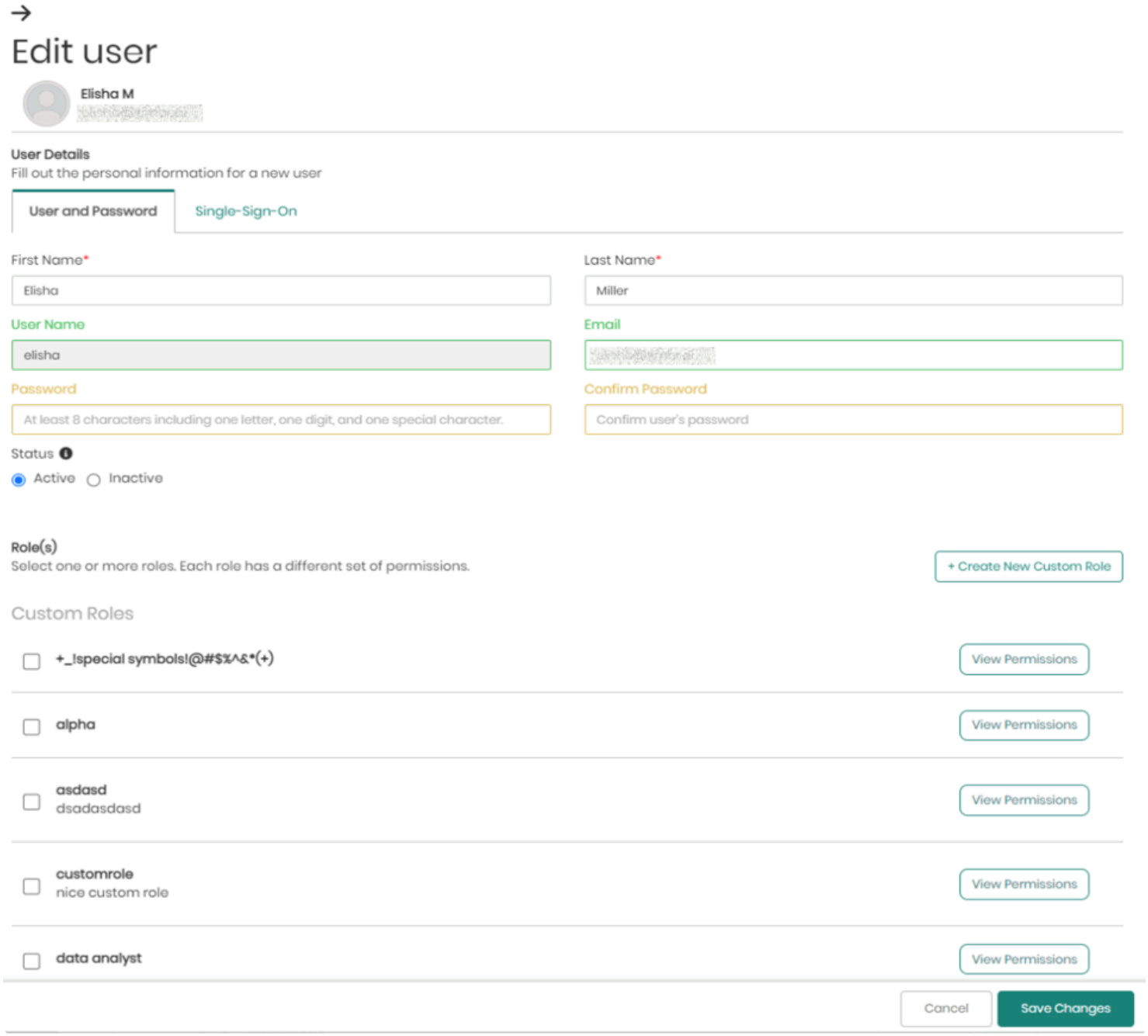
> - **Set role to users** - The ability to add a specific role to multiple users chosen from the list all in one click.
> - **Set active** - The ability to add the active status to multiple users chosen from the list all in one click.
> - **Set inactive** - The ability to add the inactive status to multiple users chosen from the list all in one click.
> - **Delete users** - The ability to delete multiple users chosen from the list all in one click.

To the right of each user, there are the three following buttons:

- **View Permissions** - This button will open a window with a detailed view of what roles the selected user has, as well as what permissions the selected user has in each aspect of the platform. In addition, here you will also be able to add permissions to the user by clicking on *+ Add Access Permissions*.



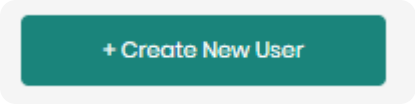
- **Edit User** - This button which appears as a pencil will open a window in order to edit the chosen user. This includes the user's login information, the user's status, and the different roles the user has in the platform.



- **Delete User** - The third button appearing as a trash can, when clicked on will delete the selected user.

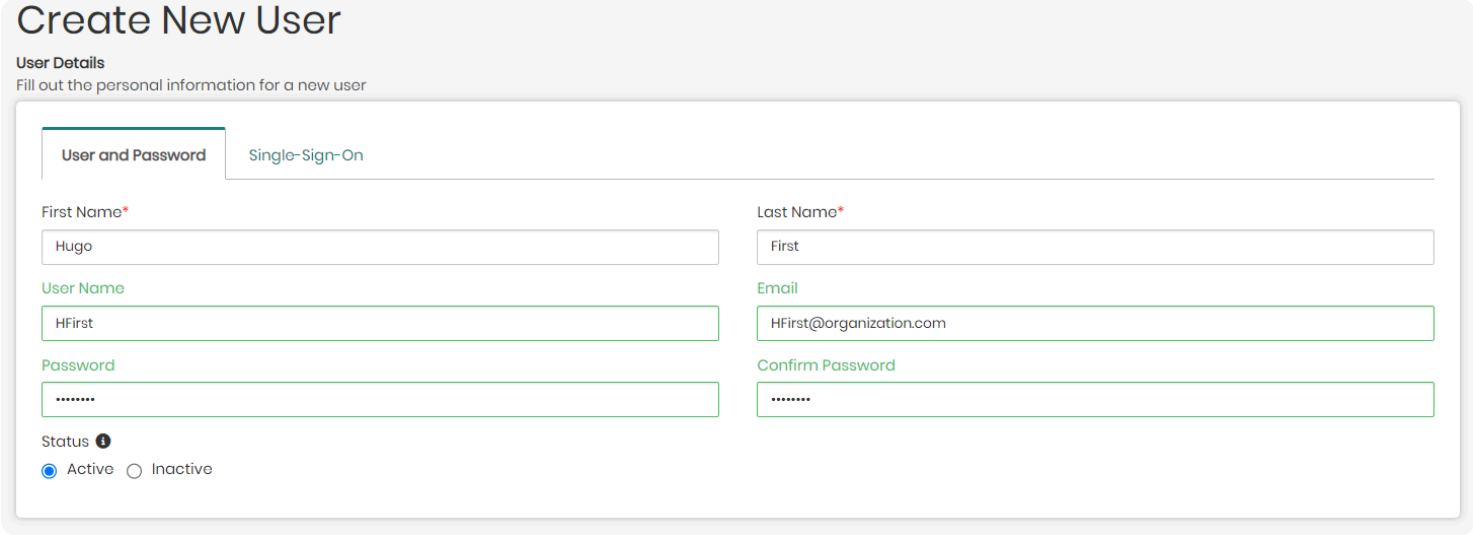
Creating a New User

1. In the main page of the access manager, click on **+ Create New User** on the upper right side.

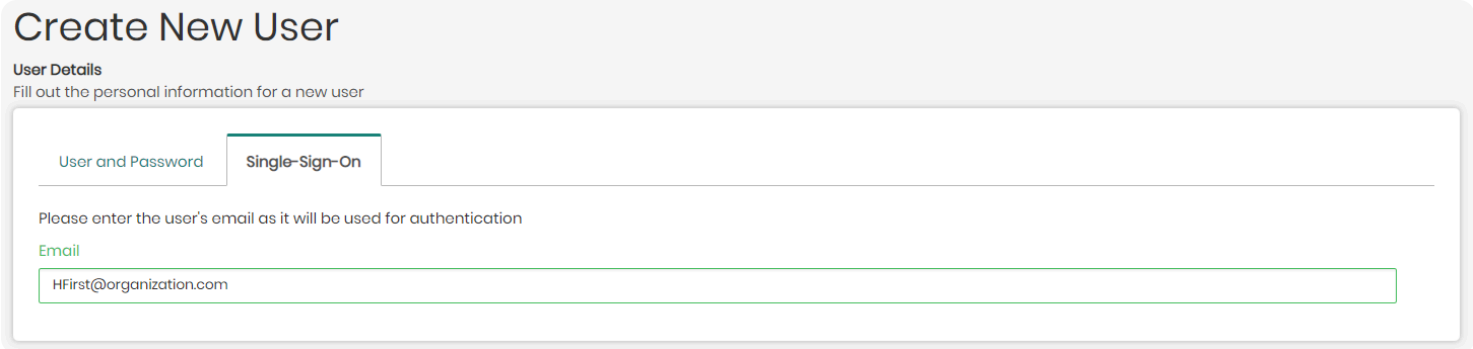


2. In the new window, you will be asked to fill in the new user's information that will be used when signing in to the platform using one of the two following tabs:

- **User and Password** - To create a user using this option, a *First* and *Last Name* of the user will be required, as well as a *Username*, *Email*, and *Password* for the new user, which is connected to the relevant server of the user's organization.



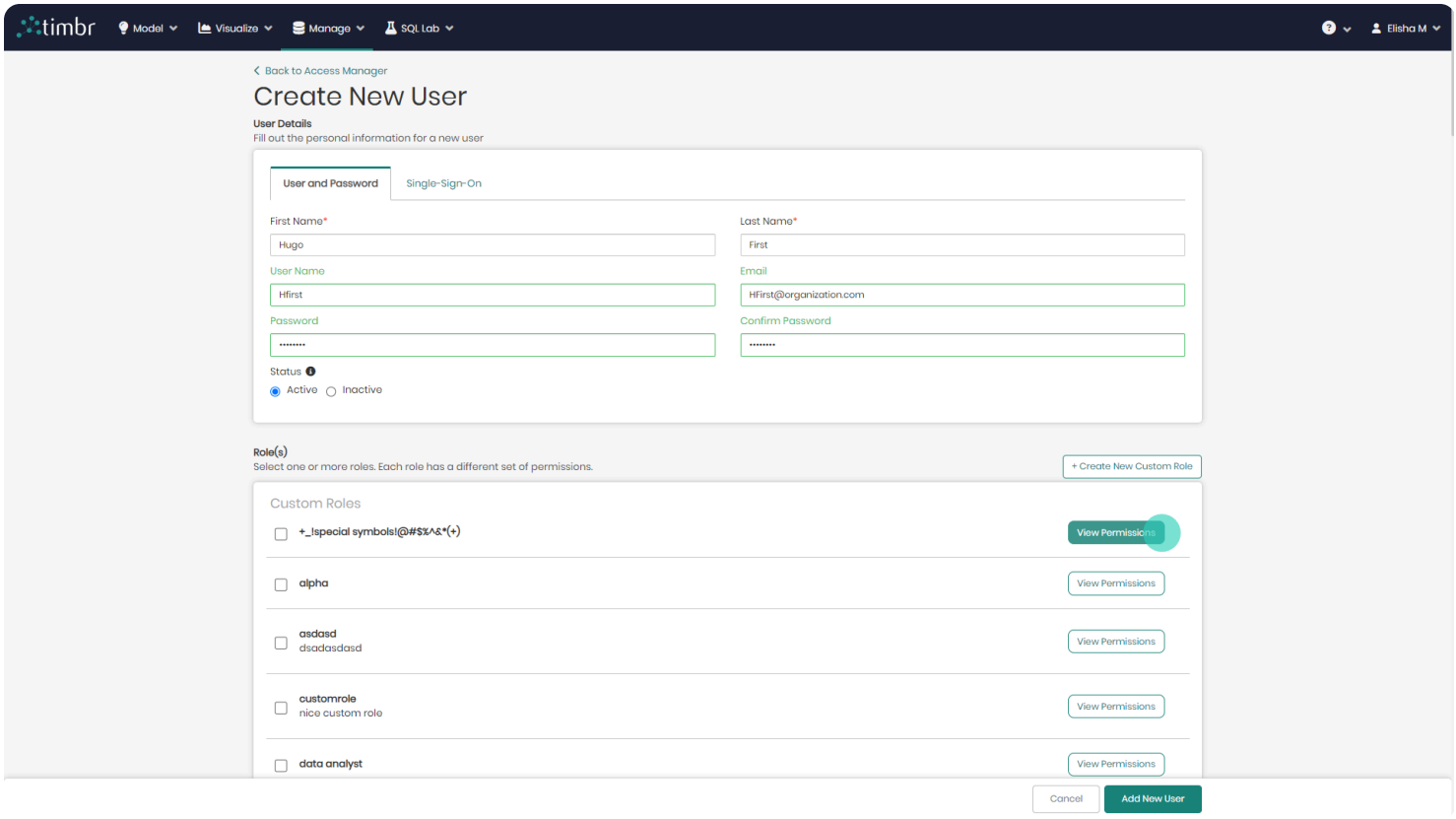
- **Single-Sign-On** - To create a user using this option, the only thing required is the user's *Email* used and authorized by the user's organization and supported by either Google or Microsoft Azure.



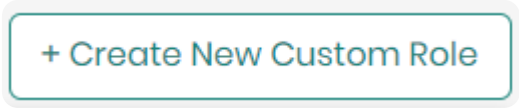
Once the new user's information is inserted, the user can now be added by clicking on **Add New User** on the bottom right of the screen.

If however while creating a user you would like to add various roles to the user which contain unique permissions, this can be accomplished in the two following ways:

- **Choosing existing roles** - Beneath the user information inserted to create the new user, is the list of roles (*Custom Roles*, *Default Roles*, *Default Datasource Roles*, and *Default Knowledge Graph Roles*) that can be assigned to the different users. Each role's permissions can be revealed by clicking on *View Permissions* to the right of the role.



- **Creating new custom roles** - The second option to add roles to a user is to create the new roles from scratch by clicking on + *Create New Custom Role* located on the top right of the list of roles.



The full process and options for creating user roles can be found in depth in the next section.

Creating User Roles

1. In the main *Access Manager* page switch from the default *Users* tab to the *Roles tab*.

You will now see all the different types of *Roles* that can be assigned to users divided by category which include:

Custom Roles - This category contains all the roles that were custom-made by the user.

Default Roles - This category contains Timbr's default roles which are included by default in the Timbr platform. The default roles are:

- **Admin** - Users given the *Admin* role will have full access and editing capabilities in the platform.
- **Analyst** - Users given the *Analyst* role will be able to query the specific datasources or knowledge graphs assigned by the admin.
- **Editor** - Users given the *Editor* role will be able to edit the specific datasources or knowledge graphs assigned by the admin.
- **Public** - Users given the *Public* role will have public access to the specific datasources or knowledge graphs assigned by the admin.
- **Viewer** - Users given the *Viewer* role will strictly have the ability to view the specific datasources or knowledge graphs assigned by the admin, without the ability to edit or query them.

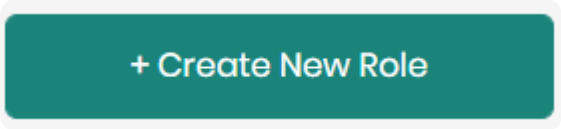
Default Datasource Roles - When a user creates a new datasource in Timbr, the following three default roles are automatically created:

- **Analyst** - Users given the *Analyst* role will be able to query the specific datasources assigned by the admin.
- **Editor** - Users given the *Editor* role will be able to edit the specific datasources assigned by the admin.
- **Viewer** - Users given the *Viewer* role will strictly have the ability to view the specific datasources assigned by the admin, without the ability to edit or query them.

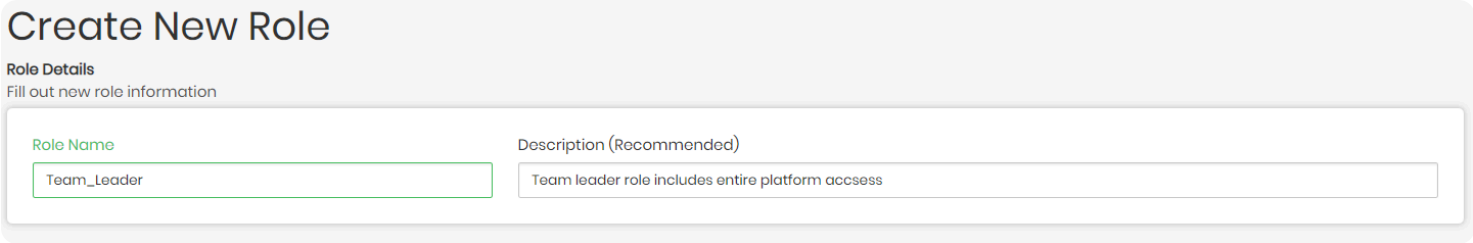
Default Knowledge Graph Roles - When a user creates a new knowledge graph in Timbr, the following three default roles are automatically created:

- **Analyst** - Users given the *Analyst* role will be able to query the specific knowledge graphs assigned by the admin.
- **Editor** - Users given the *Editor* role will be able to edit the specific knowledge graphs assigned by the admin.
- **Viewer** - Users given the *Viewer* role will strictly have the ability to view the specific knowledge graphs assigned by the admin, without the ability to edit or query them.

2. To create a new *custom role* click on **+ Create New Role** located on the top right above all the category lists.

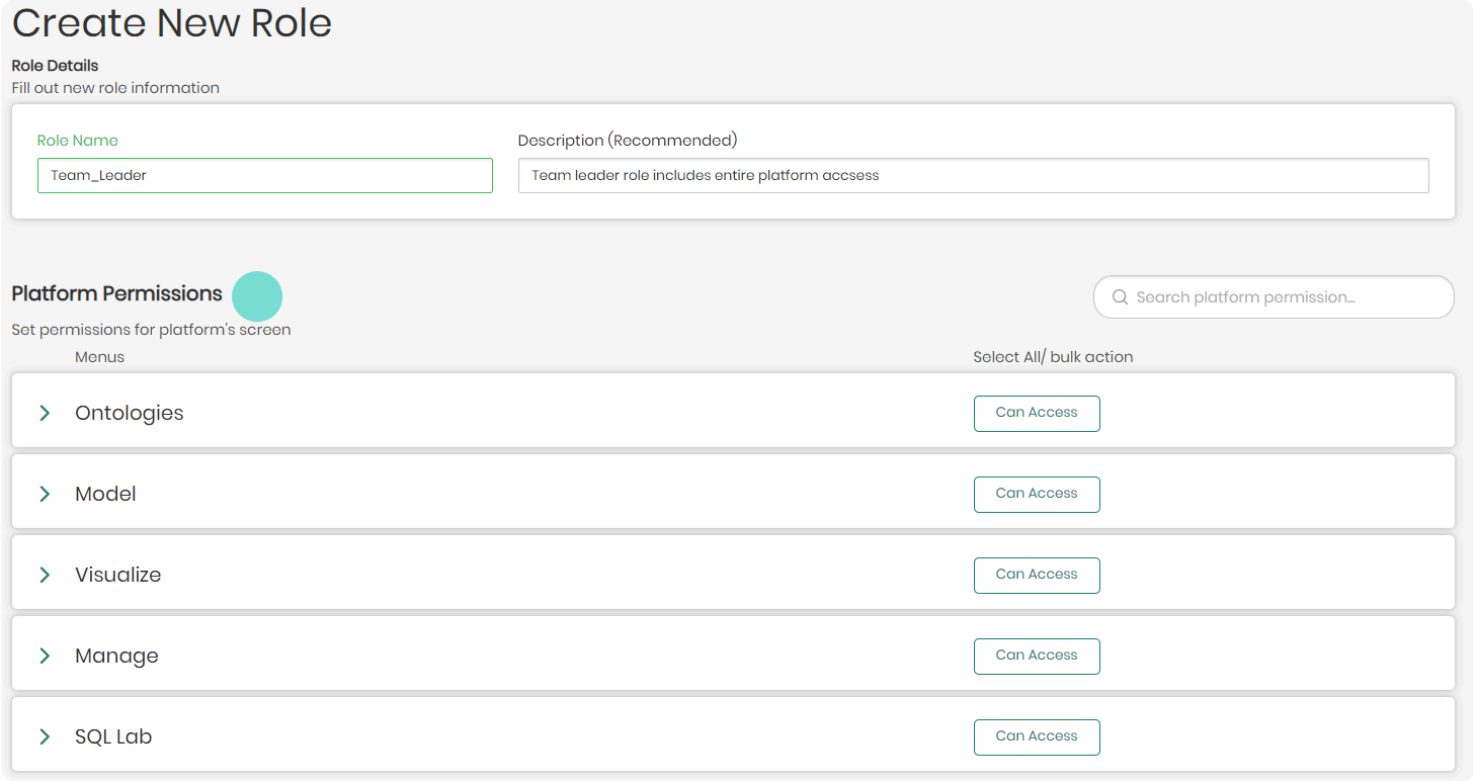


3. Once clicked, you will be asked to enter a *Name* for the role as well as an optional yet recommended description, describing the purpose of the created role.

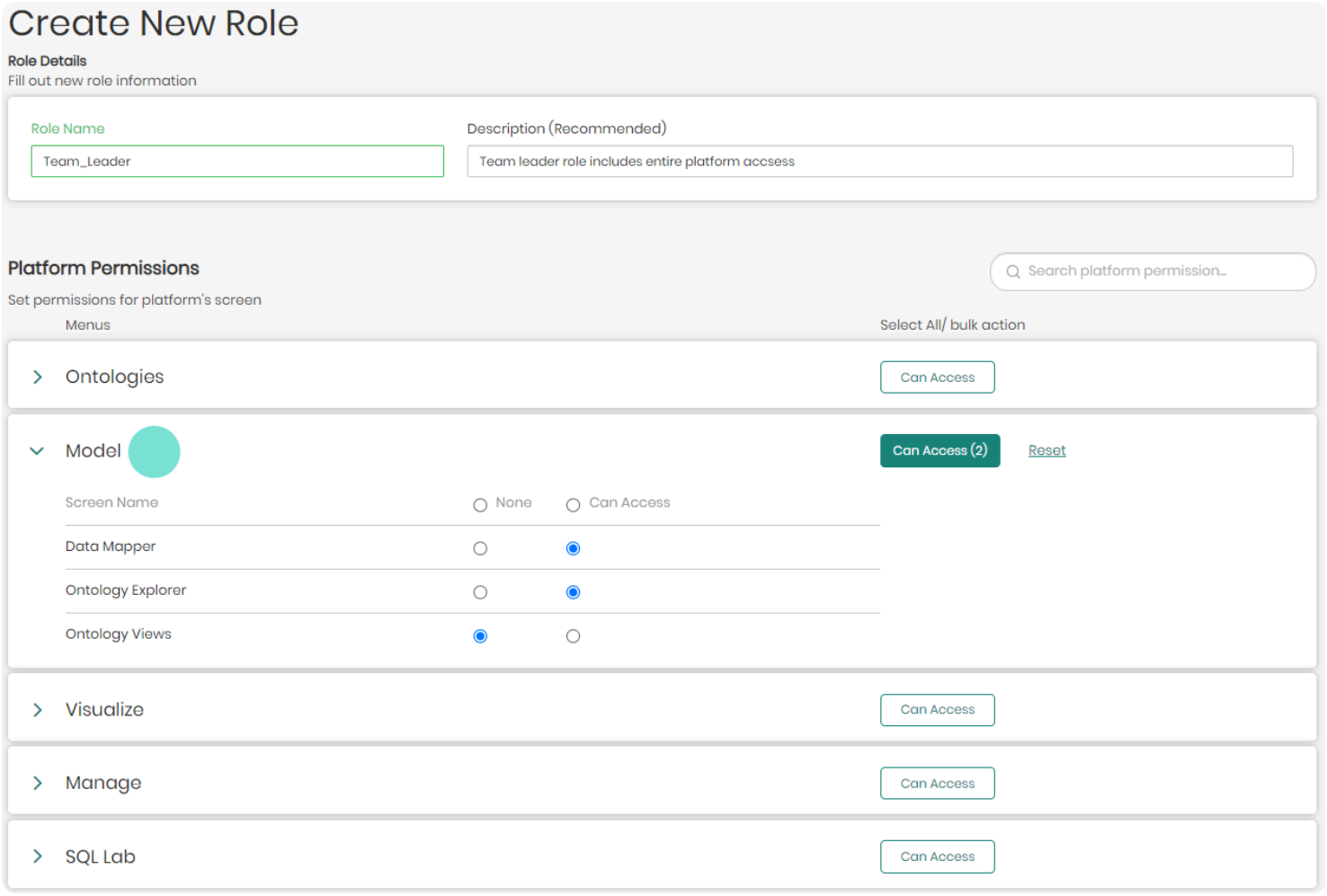


Beneath the name and description is the platform permissions which decide *What a user can do* in the platform when he is assigned this role. Here there is the option to pinpoint the exact possibilities the users will have when maneuvering through the platform with the given role.

Every possibility in the platform is located in the different platform screen menus seen below, which include: **Ontologies**, **Model**, **Visualize**, **Manage**, and **SQL Lab**.



Each platform menu can be clicked on and opened in order to select which aspects of the chosen platform menu the user will have access to when assigned this role.



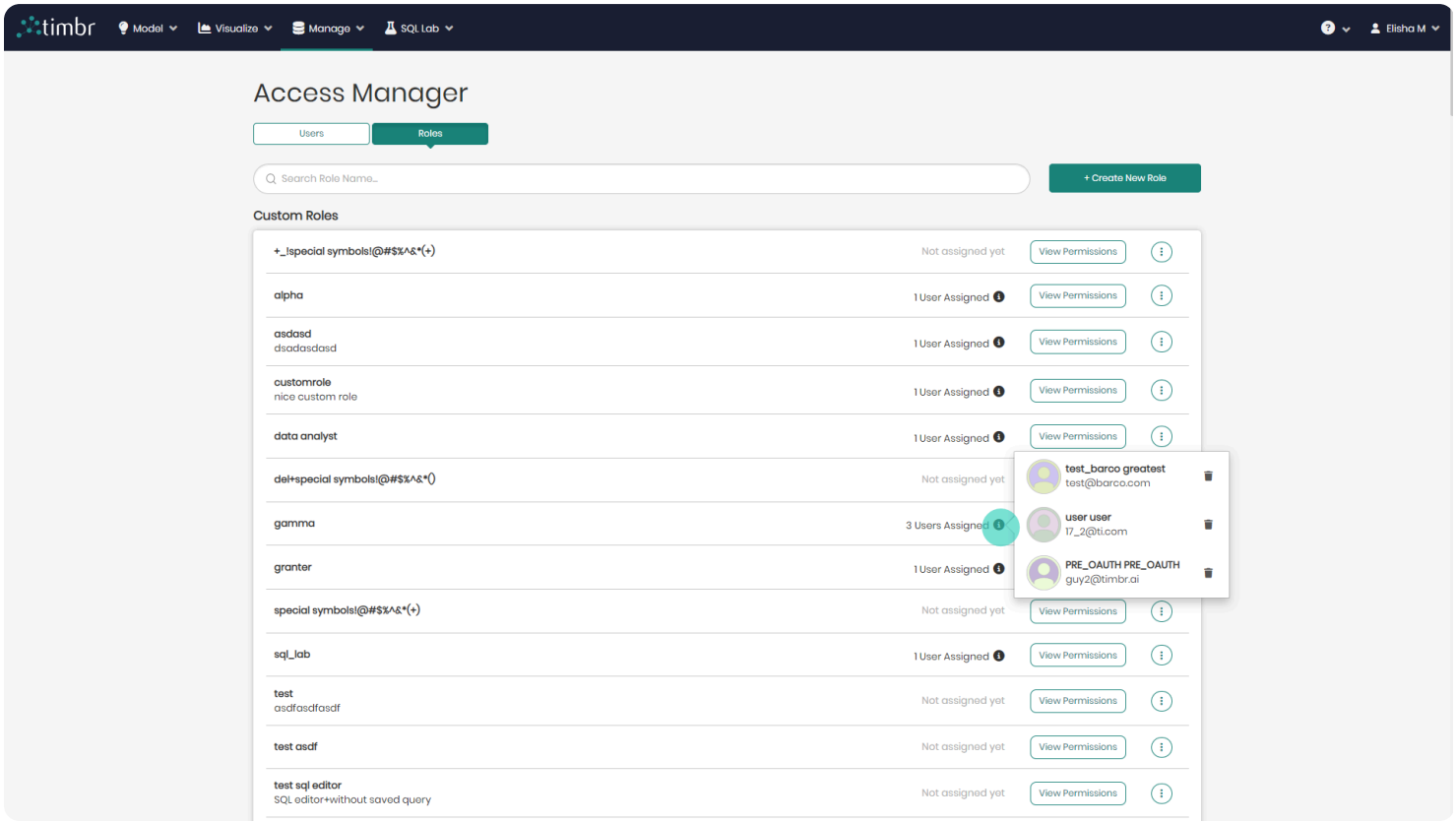
In cases where there are many options and you'd like to give access to all, you can use the *bulk options* on the right of each platform menu to select and give access to all by clicking on *Can Access*, or on the other hand, clicking on *Reset* to remove access to all.

4. Once the new roles information and permissions are in place, the role can now be added by clicking on **Add New Role** on the bottom right of the screen.

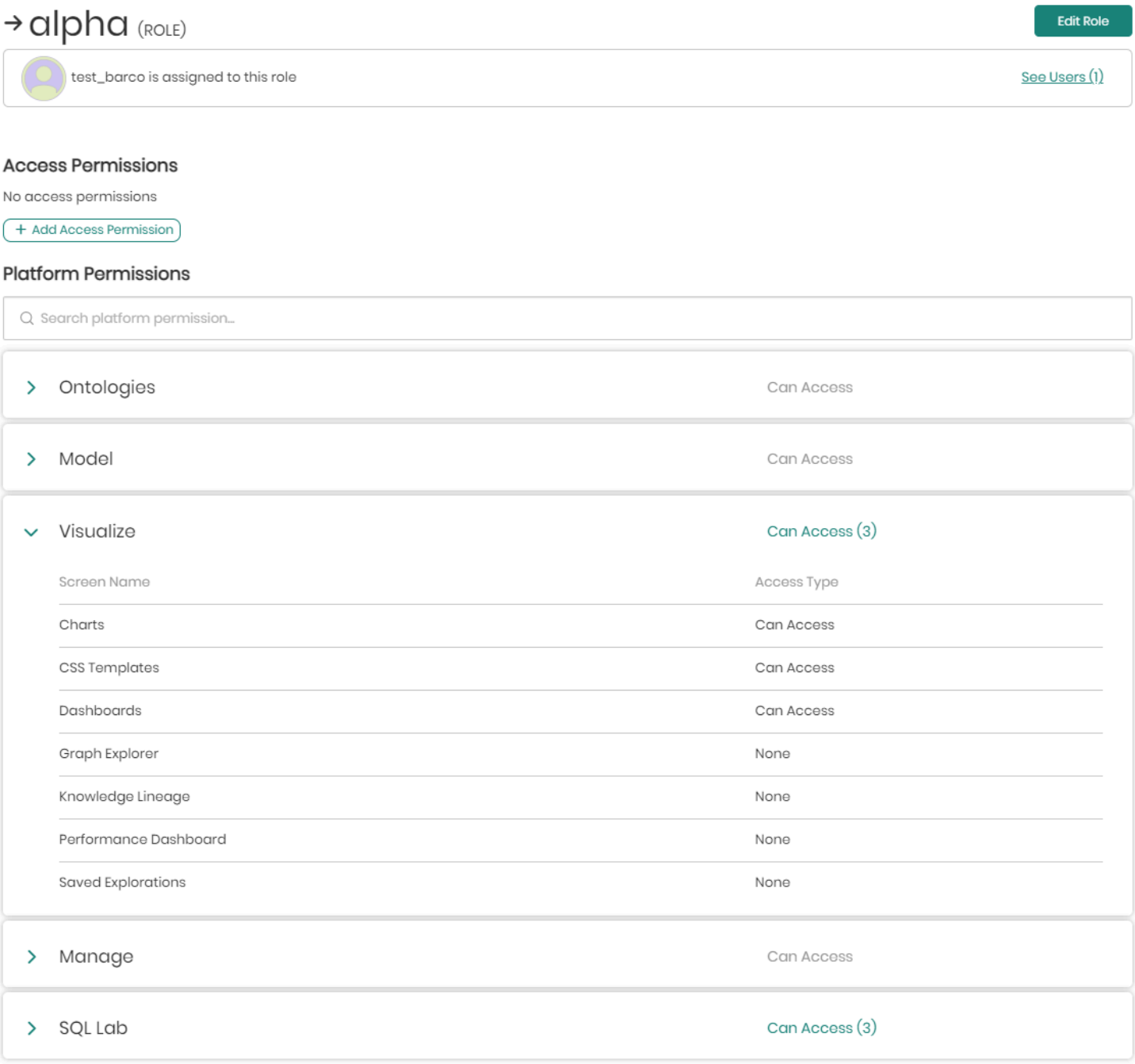
When returning to the main *Access Manager* page in the roles tab, to the right of each role on the list there are 3 options.

These 3 options are:

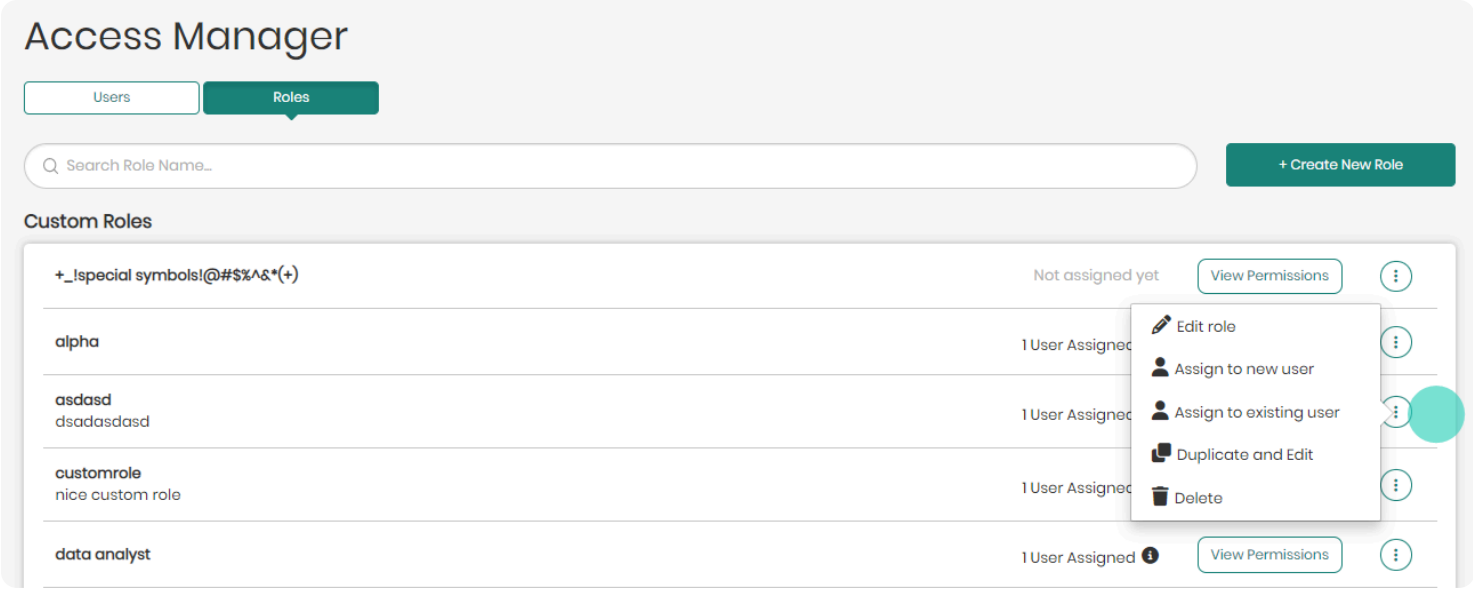
Assigned Vs. Not Assigned - This shows which roles are assigned to which users, as well as which roles have no users assigned to them.



View Permissions - This button will open a window with a detailed view of what platform permissions are attached to the selected role, as well as which users are assigned this role. Here you will be able to add permissions and edit the role.

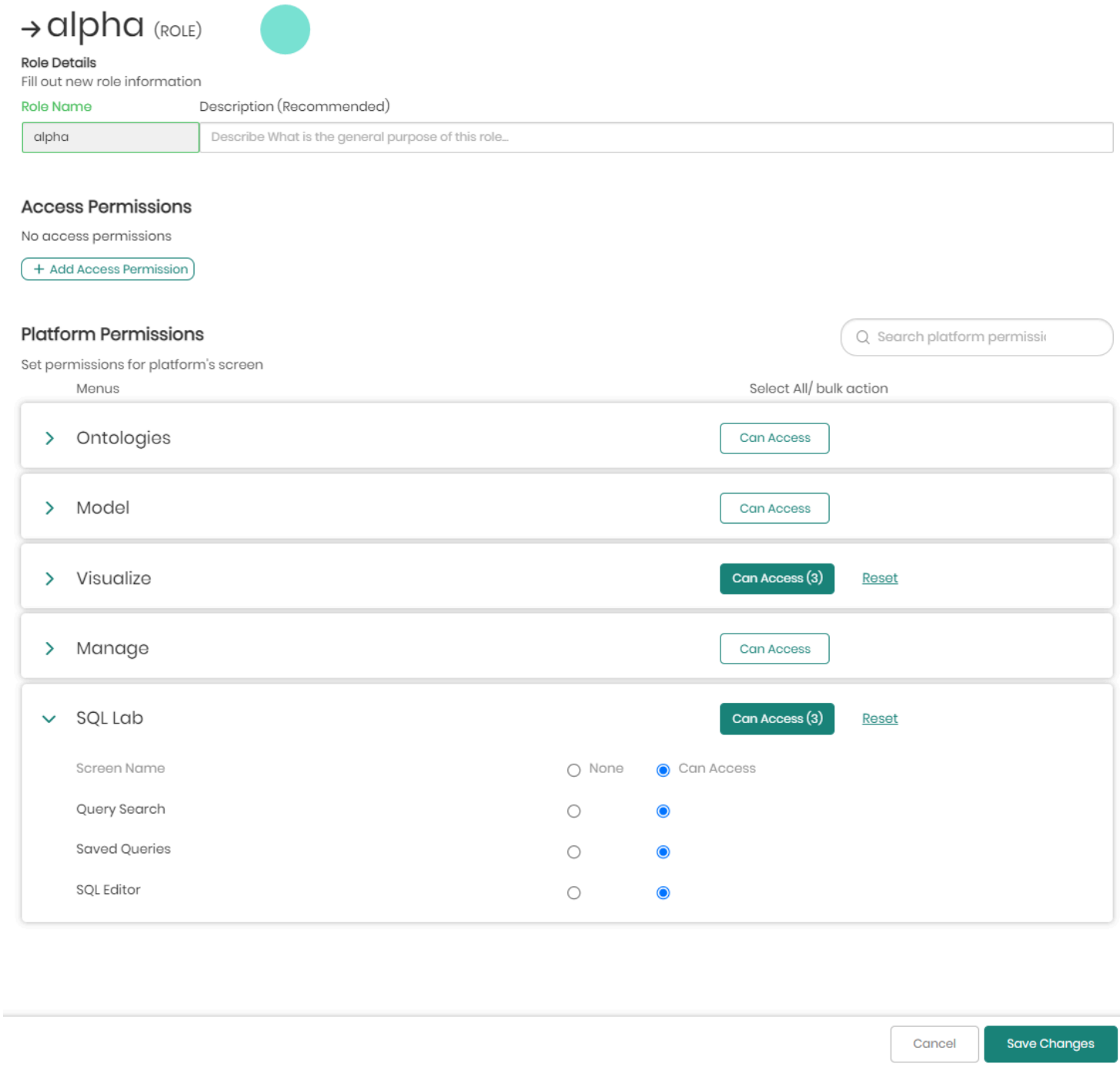


Additional Options - This button will open a small pop-up with additional options that can be performed on each role.

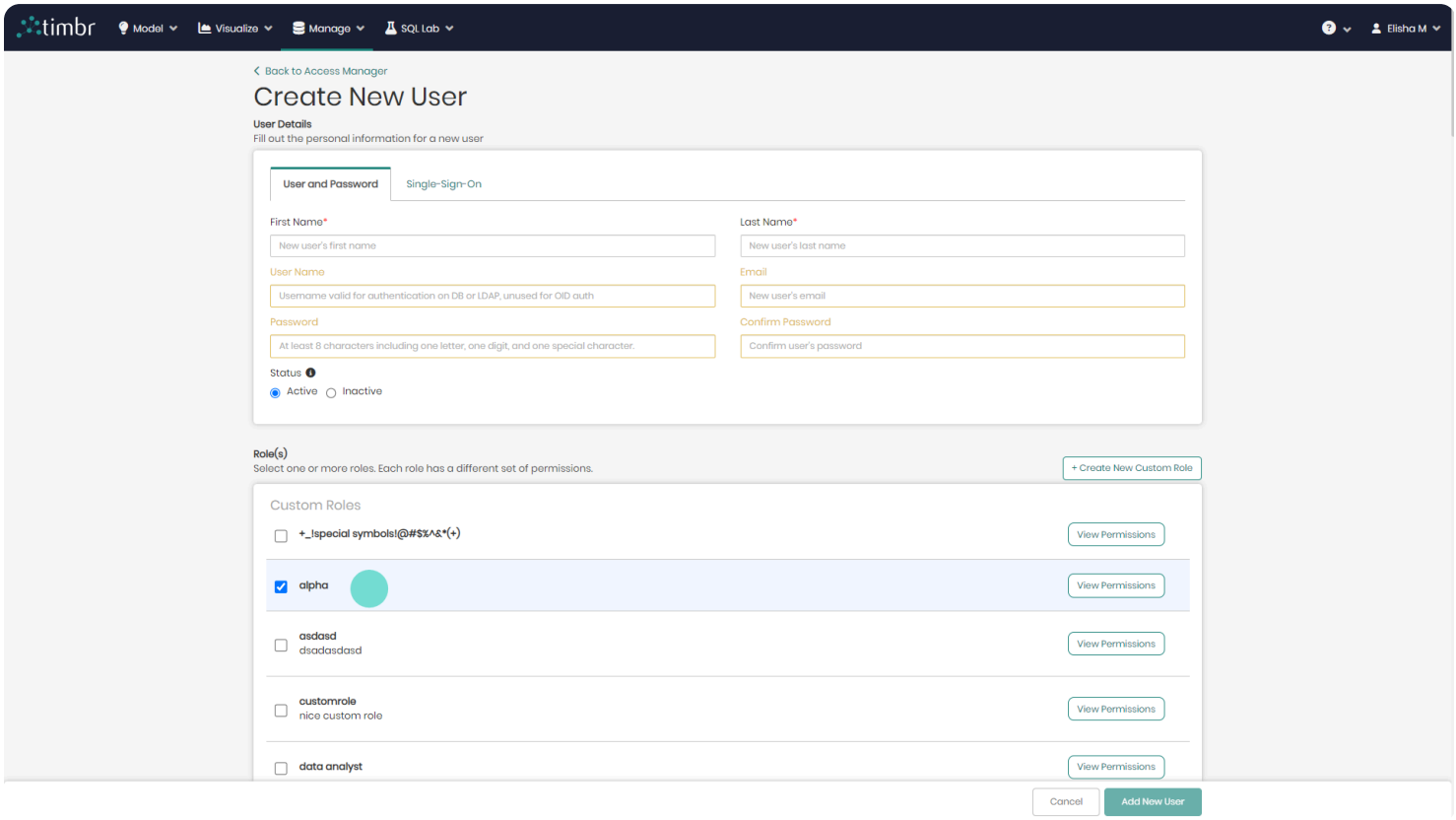


The following options that appear include:

Edit role - When clicked on, a window will appear allowing users to edit the selected role by adding or editing existing permissions to the role.

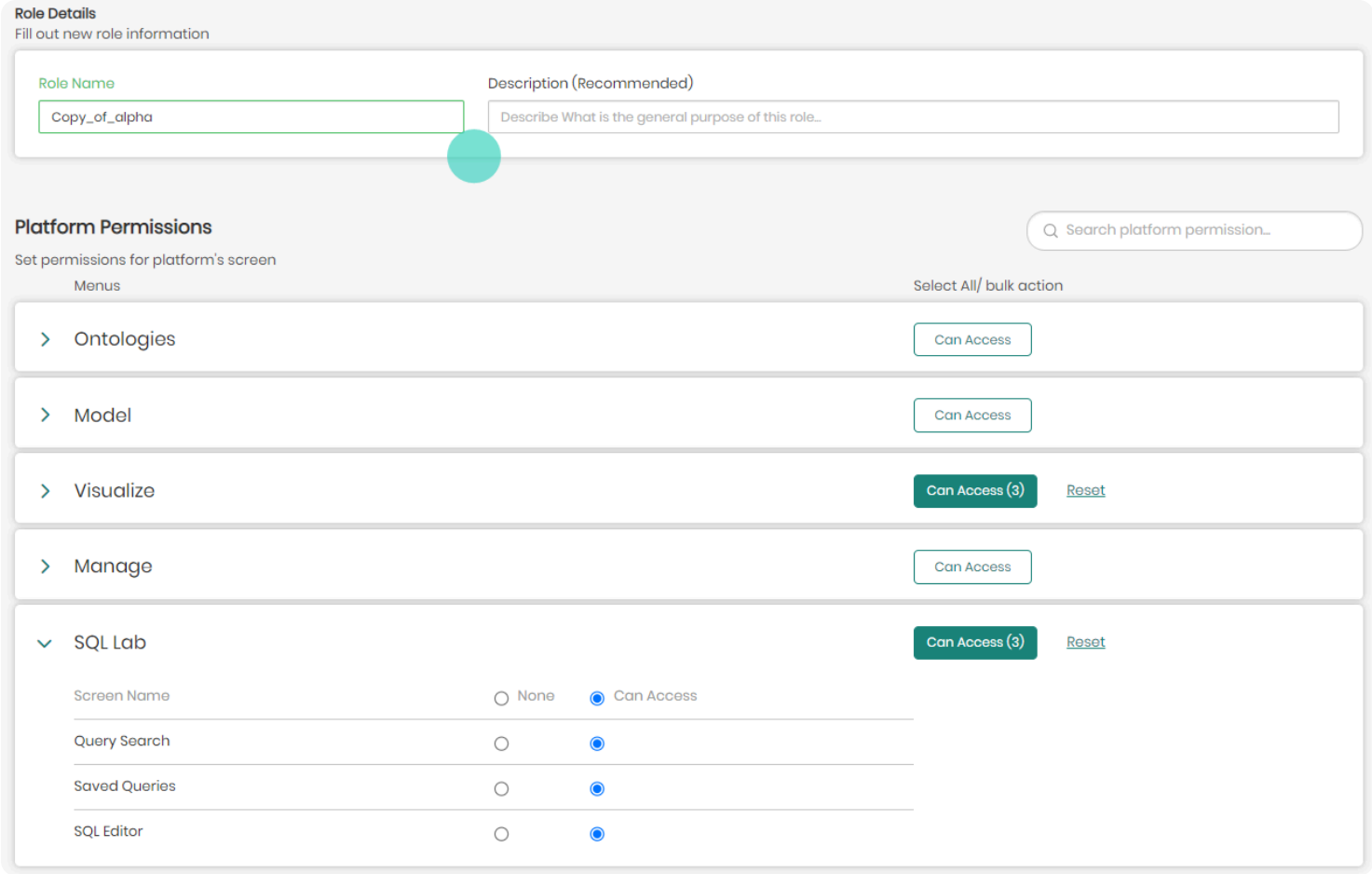


Assign to new user - When clicked on a *Create New User* window will appear with the selected role highlighted beneath. All that would be needed is to fill in the new user's information and click on *Add New User* on the bottom right.



Assign to existing user - When clicked on, an *Assign user* window will appear with the option to assign the selected role to any of the users on the list.

Duplicate and Edit - Duplicates the selected role enabling users to edit a copy of the role as well as edit its permissions.

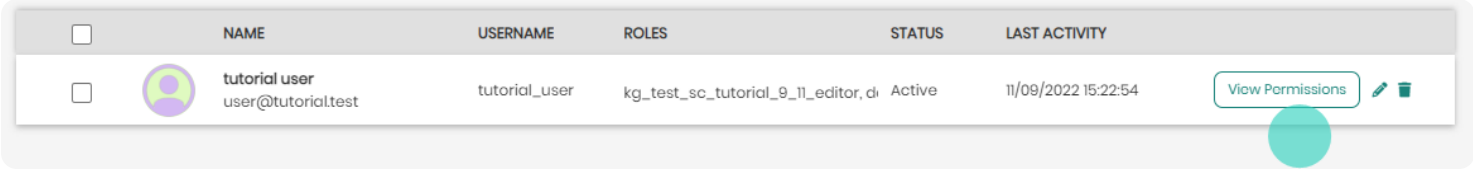


Delete - Deletes the selected role from the list of roles in the *Access Manager*.

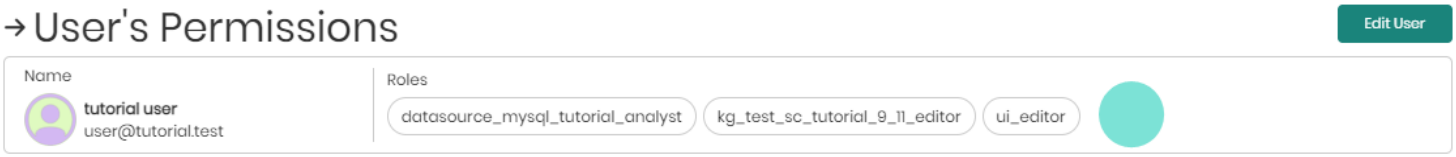
The features and options above can be performed on all categories as well, which include *Custom Roles*, *Default Roles*, *Default Datasource Roles* and *Default Knowledge Graph Roles*.

Adding and Editing User Access Permissions

As mentioned above, a user's access permissions can be accessed and viewed in the *Users* tab to the right of each user by clicking on **View Permissions**.

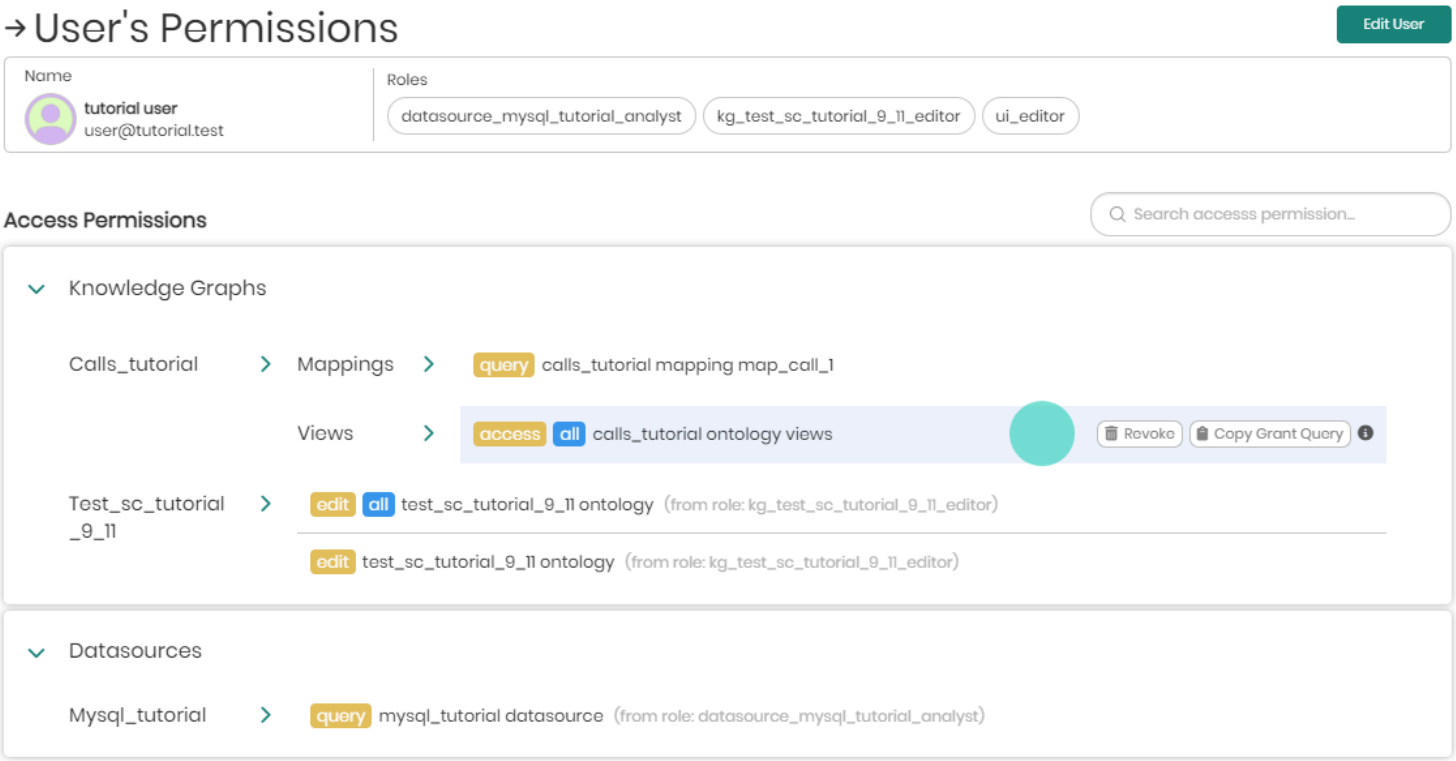


In the window that appears, on top will be the user's name and roles. Each user role can be accessed and edited by clicking on the role name.



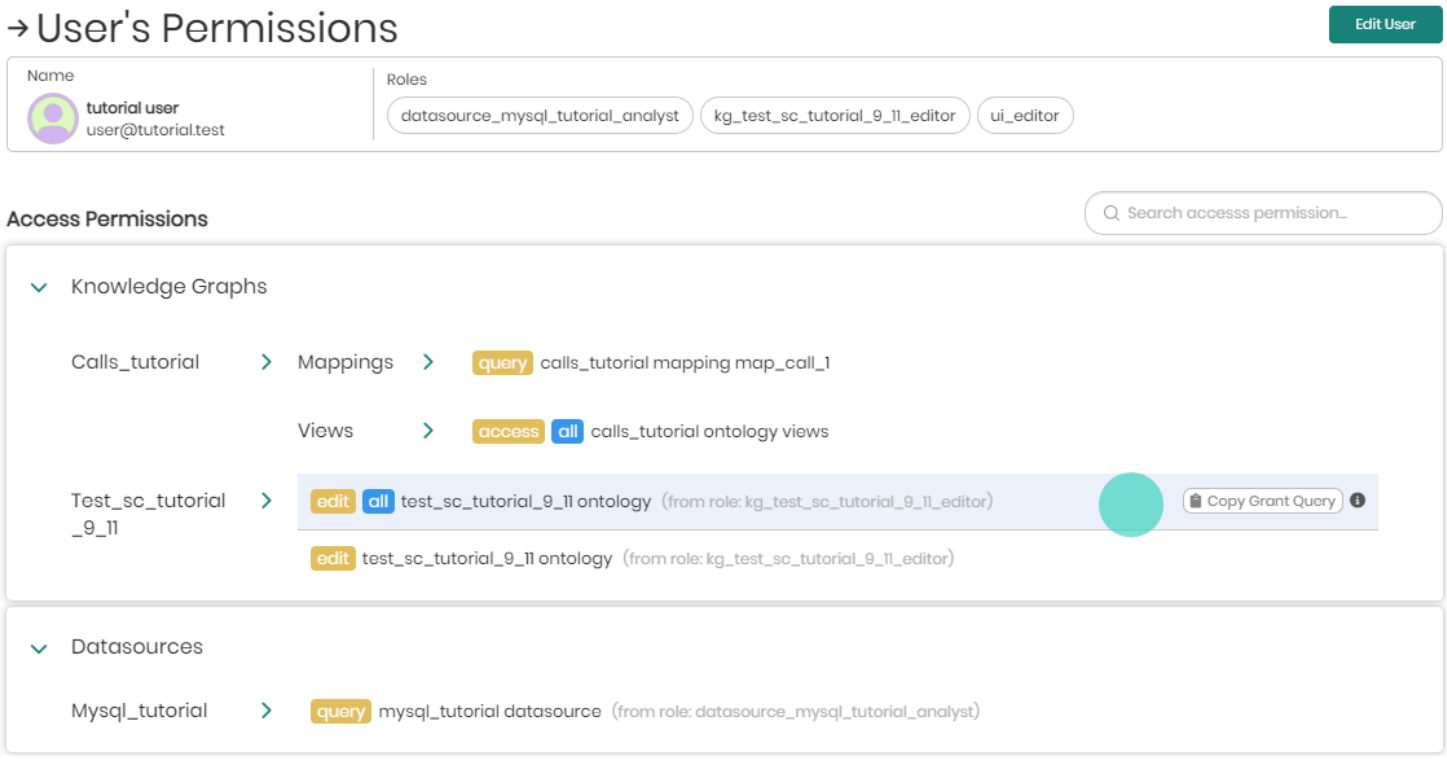
Please notice that when editing a role, it won't only affect the specific user you are trying to edit the role for, but will affect all users who were assigned that role.

Under the selected user's name and roles, will be the user's access permissions which define what a user can see in the platform. Existing permissions given to a specific user will have 3 options when hovering over the permission, which includes: **Revoke**, **Copy Grant Query**, and **Information**.



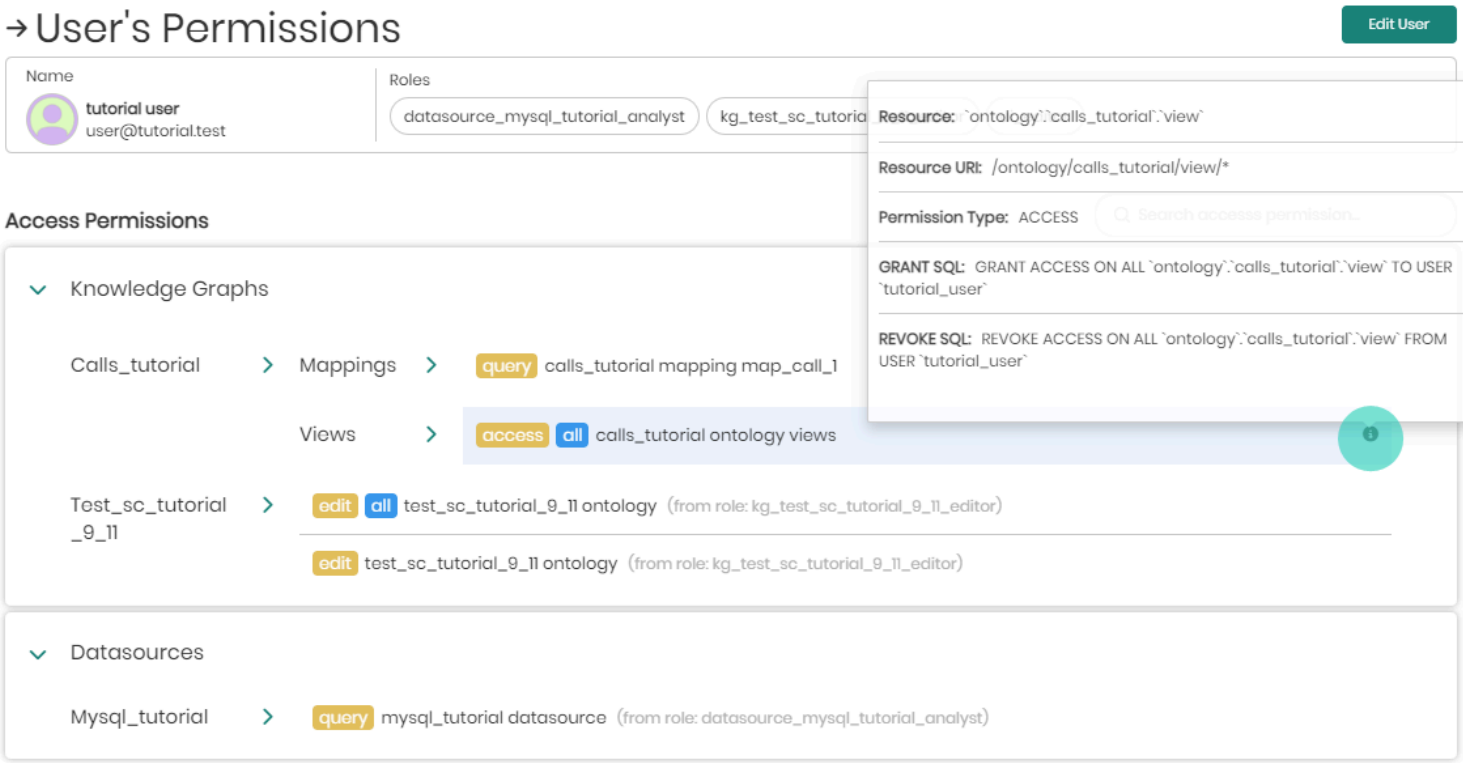
Revoke - Revokes the specific access given to the user.

Please notice that only permissions that were assigned to the specific user can be revoked from here. When a user is added to a role, revoking the permission is done by removing the user from the role he was assigned to through the roles tab.

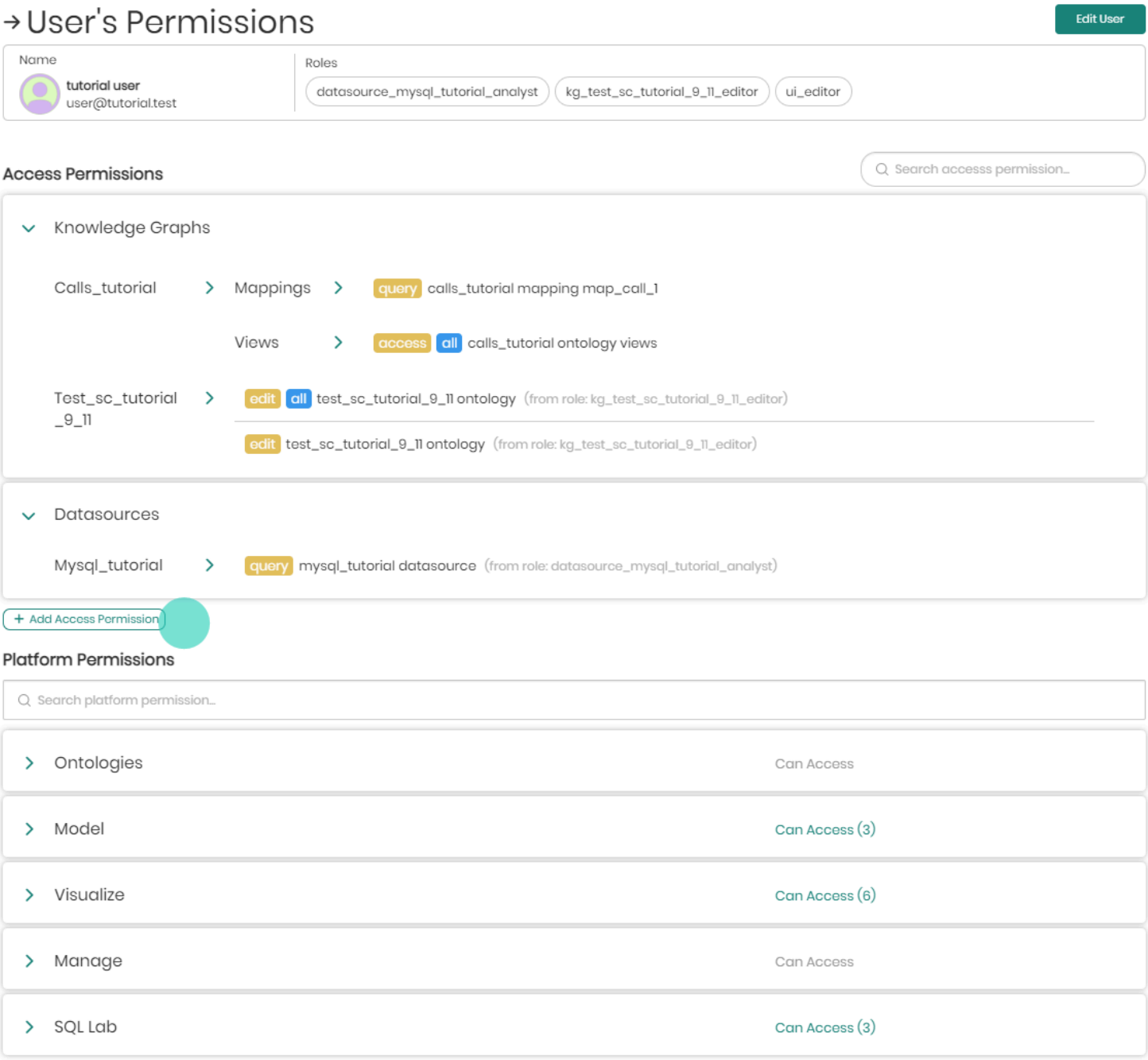


Copy Grant Query - Copies the permissions SQL Grant query to the clipboard.

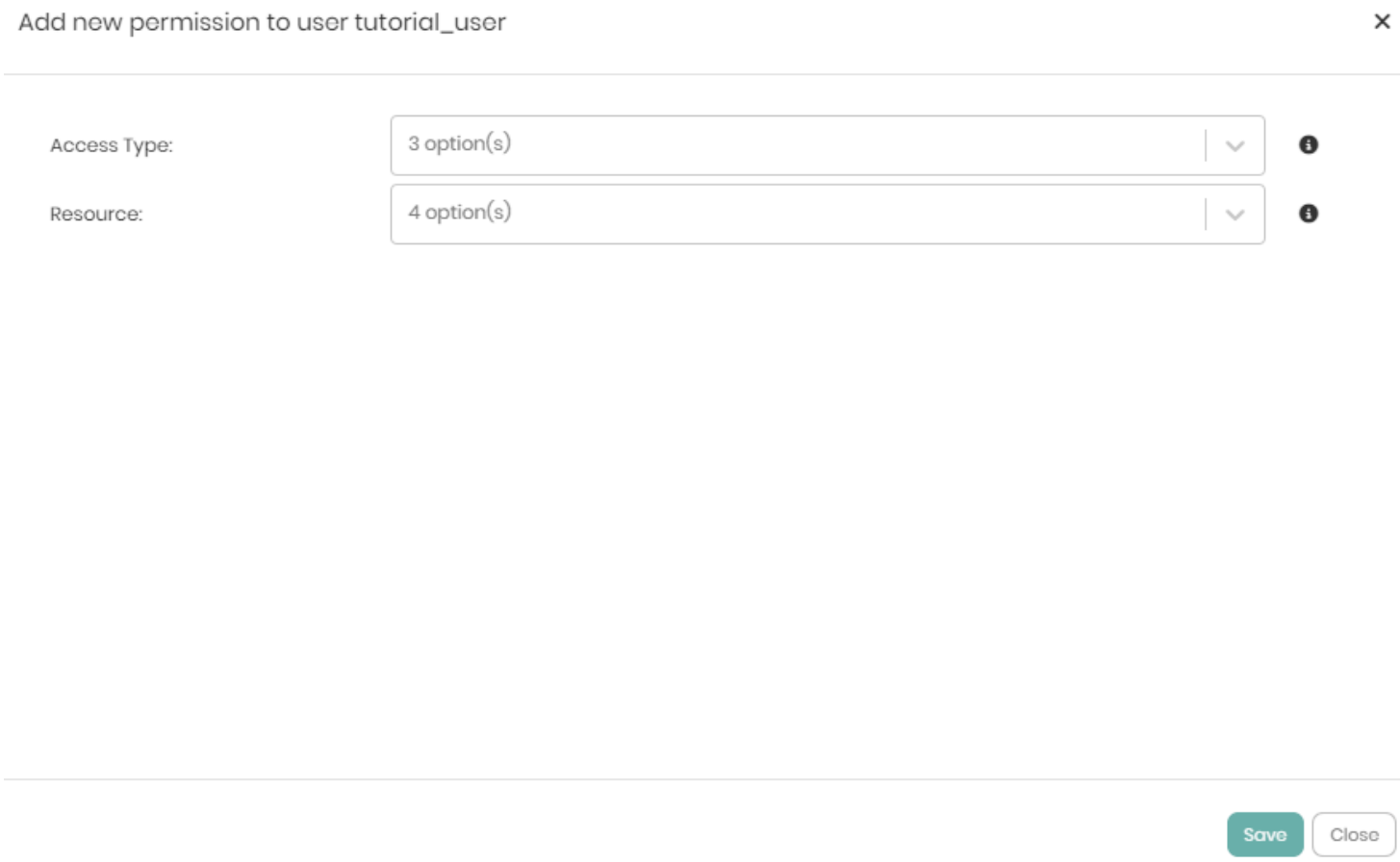
Information - When *Information* is hovered over, it will show all the information of the specific permission.



To add new access permissions to the user click on **+ Add Access Permissions** below the list of existing permissions.

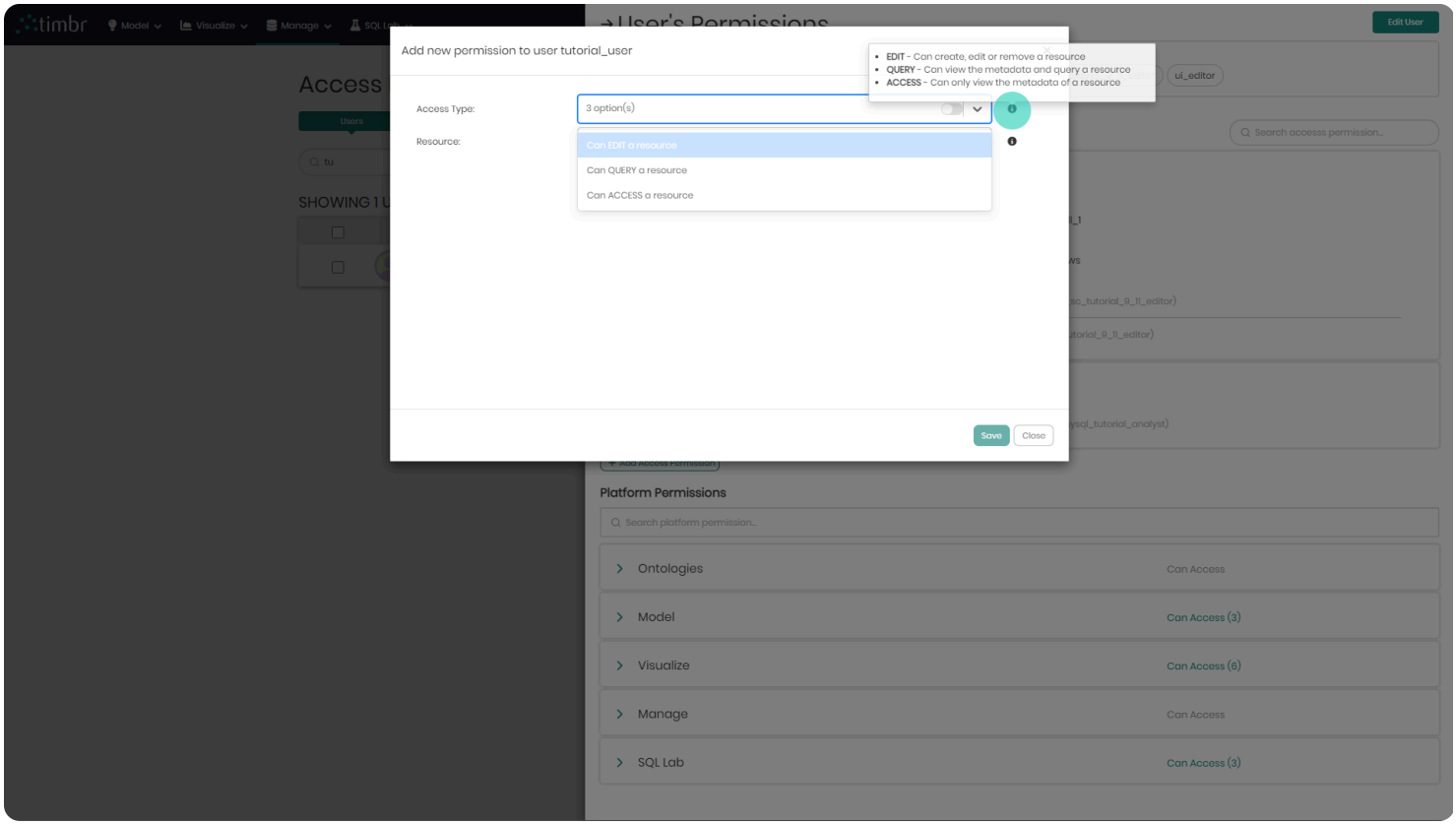


In the new window that appears, there are the initial 2 dropdowns that include **Access Type** and **Resource**.



The **Access Type** dropdown contains 3 options which are:

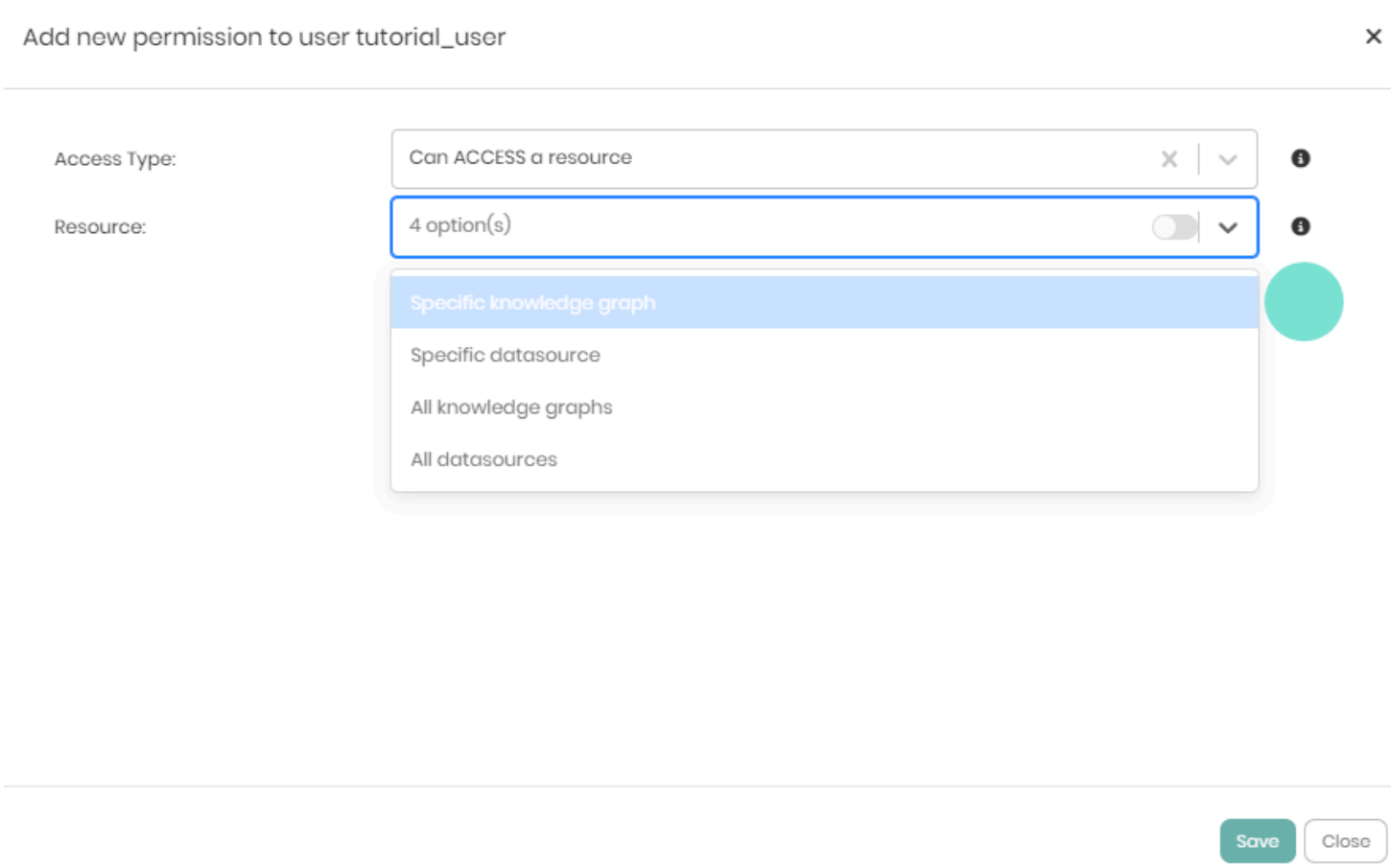
- **Access** - Users given the *Access* permission can only view the metadata of a chosen resource.
- **Query** - Users given the *Query* permission can view the metadata and query the chosen resource.
- **Edit** - Users given the *Edit* permission can create, edit or remove the chosen resource.



The **Resource** dropdown initially contains 4 options but the options change based on the first selection in the following way:

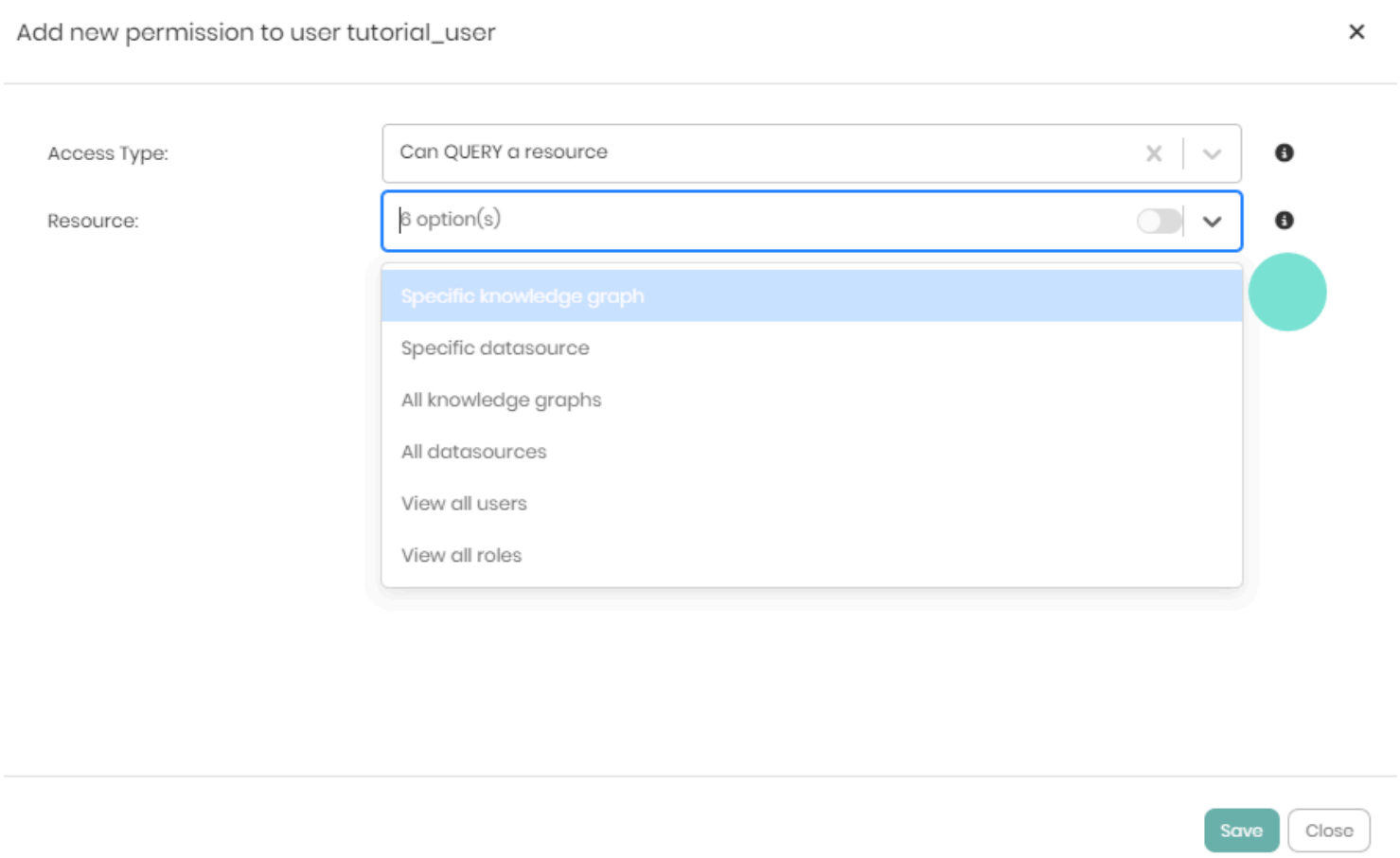
If the **Access** permission is given to the user which is the most basic permission then the *Resource* dropdown will show the following 4 options:

- **Specific knowledge graph** - Giving the user permission to view a specific knowledge graph.
- **Specific datasource** - Giving the user permission to view a specific datasource.
- **All knowledge graphs** - Giving the user permission to view all the knowledge graphs in the platform.
- **All datasources** - Giving the user permission to view all the datasources connected to the platform.



If the **Query** permission is given to the user then the *Resource* dropdown will show the following 6 options:

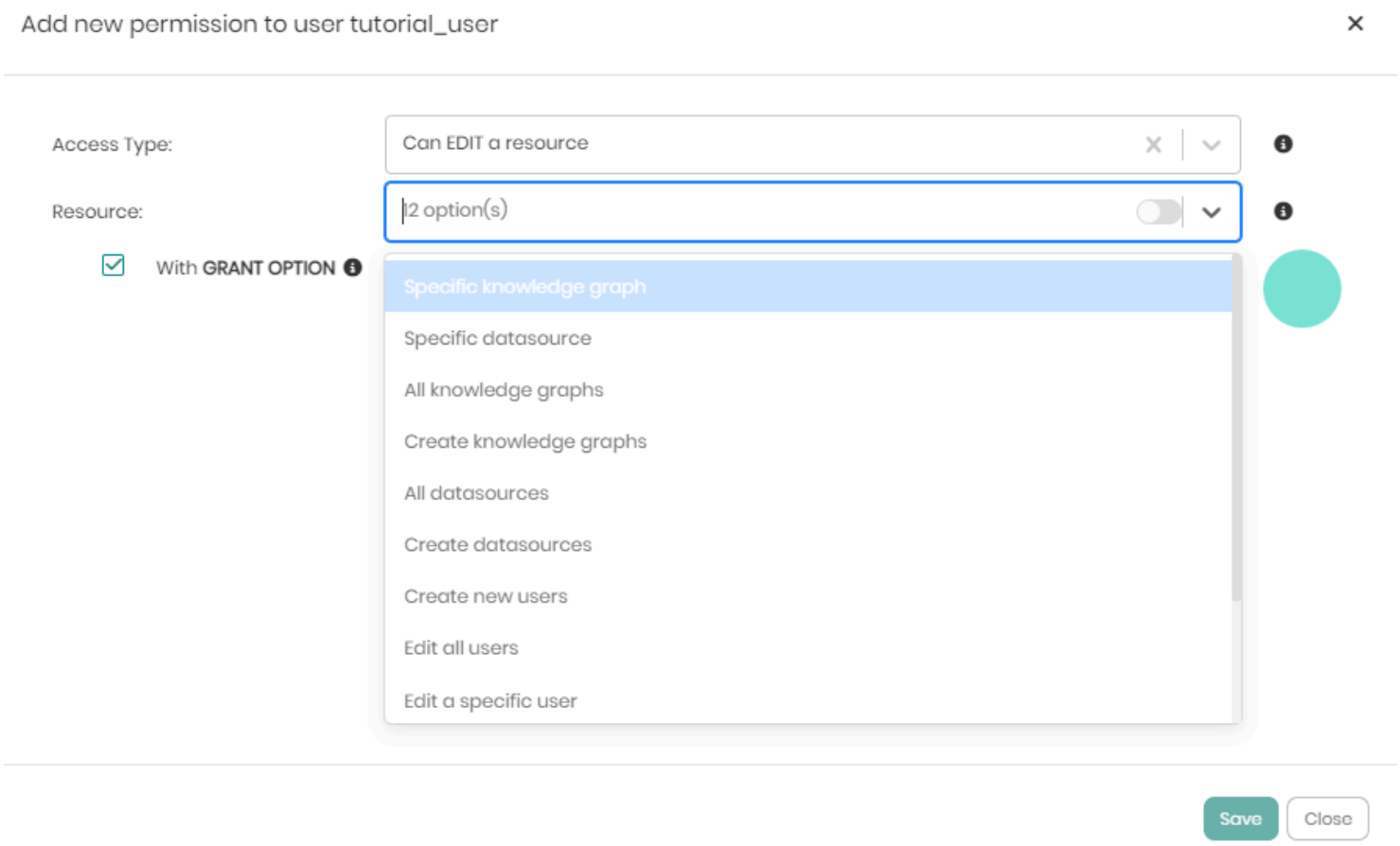
- **Specific knowledge graph** - Giving the user permission to view and query a specific knowledge graph.
- **Specific datasource** - Giving the user permission to view and query a specific datasource.
- **All knowledge graphs** - Giving the user permission to view and query all the knowledge graphs in the platform.
- **All datasources** - Giving the user permission to view and query all the datasources connected to the platform.
- **View all users** - Giving the user permission to view all users in the platform.
- **View all roles** - Giving the user permission to view all roles in the platform.



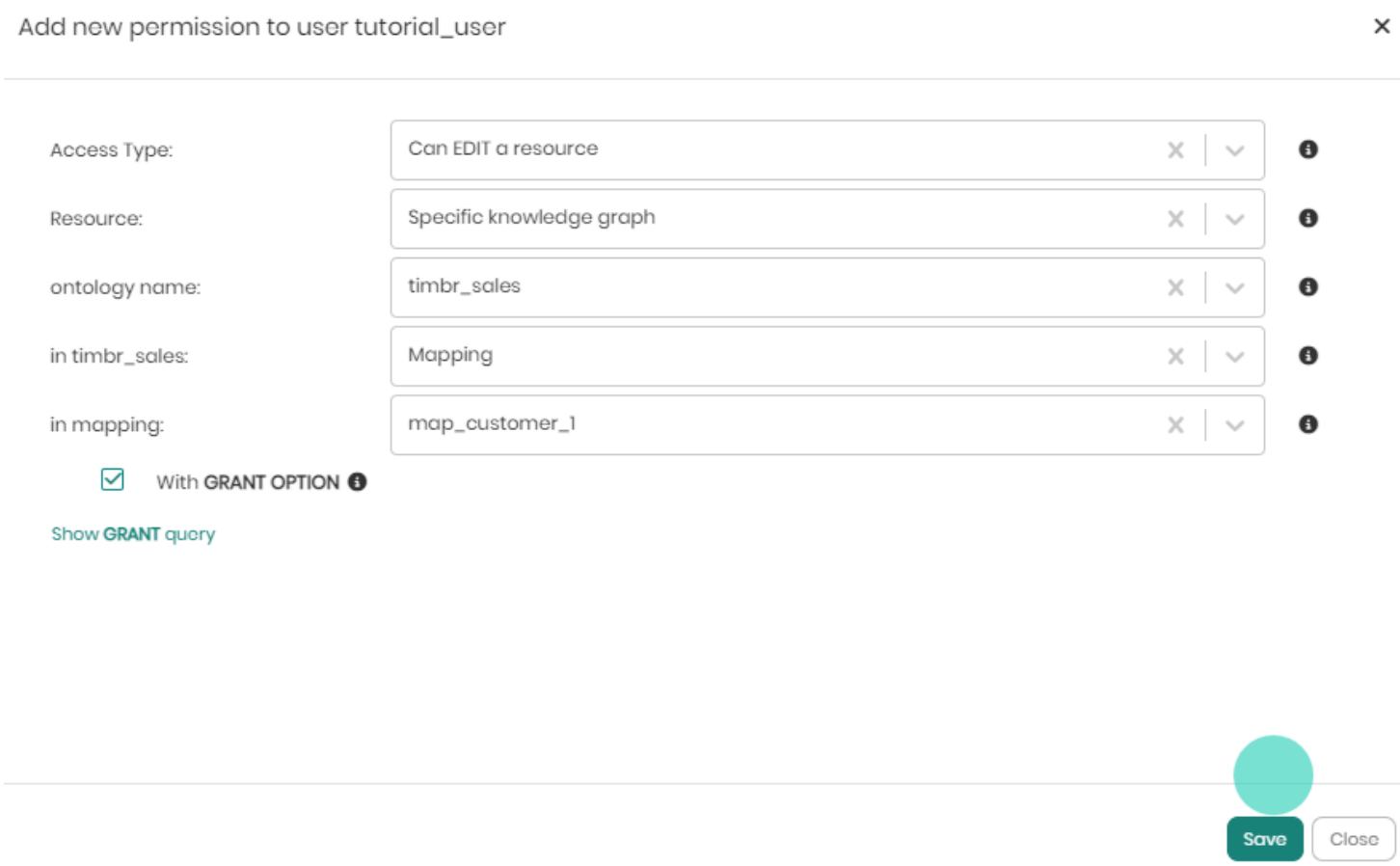
If the **Edit** permission is given to the user which includes the most advanced permissions then the *Resource* dropdown will show the following 12 options:

- **Specific knowledge graph** - Giving the user permission to view, edit or remove a specific knowledge graph.
- **Specific datasource** - Giving the user permission to view, edit or remove a specific datasource.
- **All knowledge graphs** - Giving the user permission to view, edit or remove all the knowledge graphs in the platform.
- **Create knowledge graphs** - Giving the user permission to create new knowledge graphs in the platform.
- **All datasources** - Giving the user permission to view, edit or remove all the datasources connected to the platform.
- **Create datasources** - Giving the user permission to create and add new datasources to the platform.
- **Create new users** - Giving the user permission to create new users in the platform.
- **Edit all users** - Giving the user permission to edit the details of all the users in the platform.
- **Edit a specific user** - Giving the user permission to edit a specific user in the platform.
- **Create new roles** - Giving the user permission to create new roles in the platform.
- **Edit all roles** - Giving the user permission to edit all roles in the platform.
- **Edit a specific role** - Giving the user permission to edit a specific role in the platform.

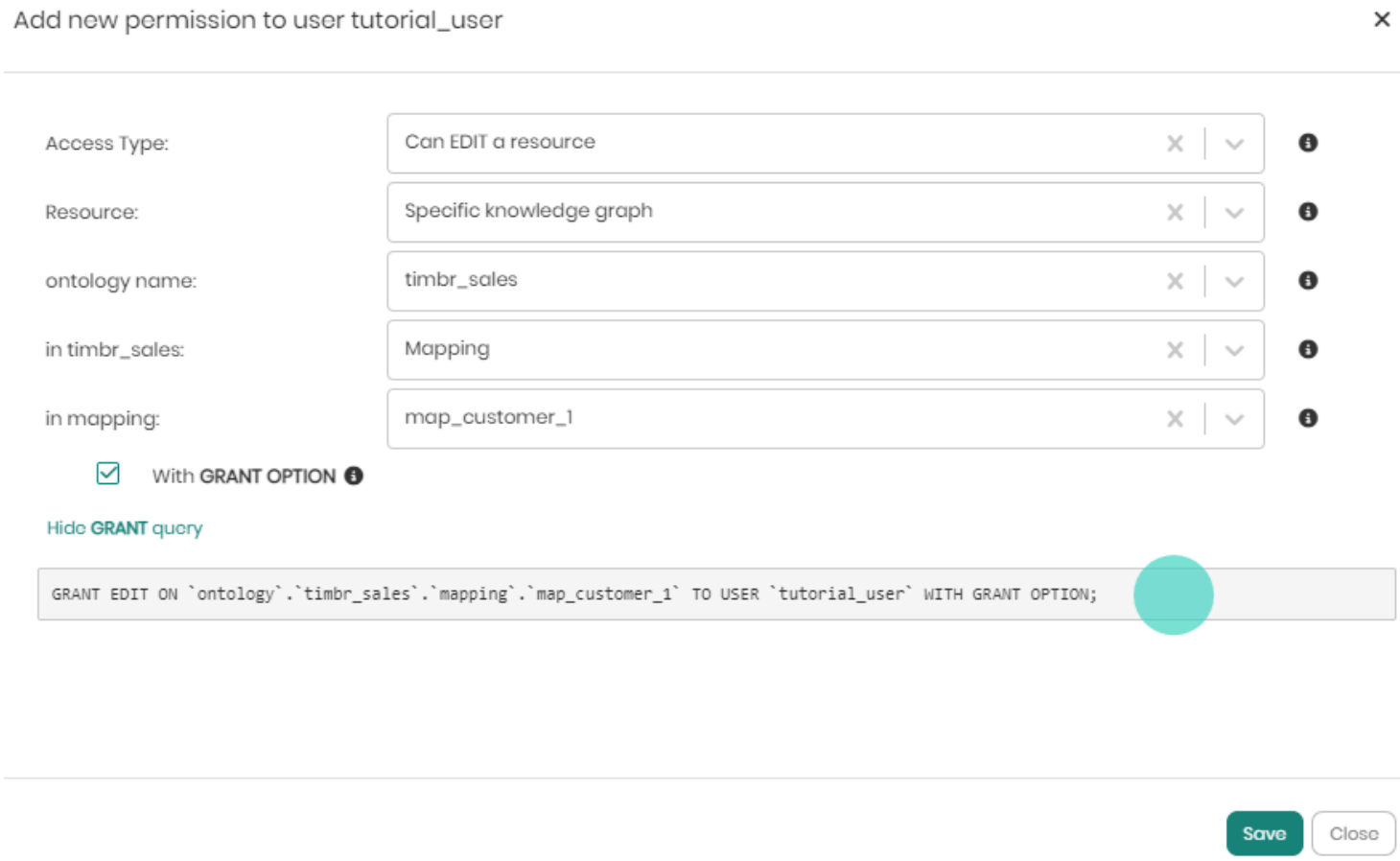
Notice that for the **Edit** permission only, there is a checkbox called with **Grant Option** below *Resource* that hands the specific user the ability to grant and revoke permissions from other users in the scope of the permission.



When the **Access Type** and **Resource** dropdowns are chosen, depending on the choices different dropdowns will appear to drill down to the intended selection. Once the dropdowns are chosen all that's left is to click on **Save** at the bottom right of the window. If before saving you would like to see the SQL syntax behind the grant query defining the permission, you can click on **Show GRANT query** beneath the *with GRANT Option* checkbox.

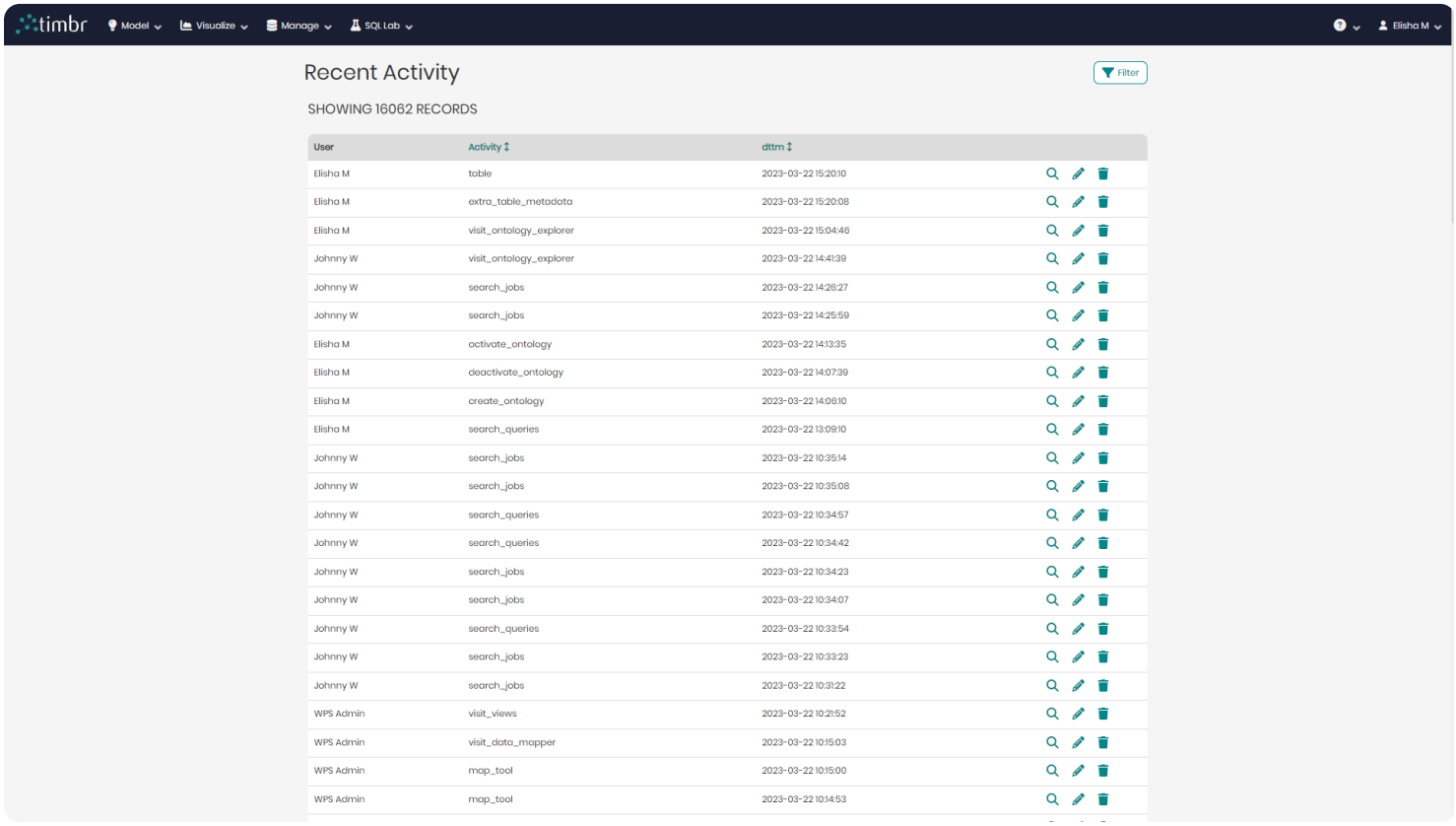


All the access permissions for both users and roles can also be granted or revoked in Timbrs SQL.


Add new permission to user tutorial_user
Access Type:
Can EDIT a resource
Resource:
Specific knowledge graph
ontology name:
timbr_sales
in timbr_sales:
Mapping
in mapping:
map_customer_1
With GRANT OPTION
Hide GRANT query
GRANT EDIT ON `ontology`.`timbr_sales`.`mapping`.`map_customer_1` TO USER `tutorial_user` WITH GRANT OPTION;
Save
Close

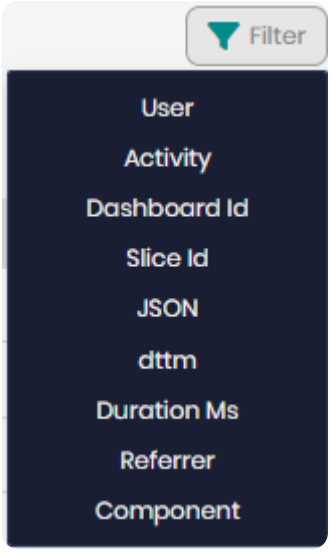
Recent Activity

The Recent Activity page is where users can review all the activity in the Timbr platform. Users can also use the system tables of Recent Activity and create a dashboard for monitoring specific activities across the platform.



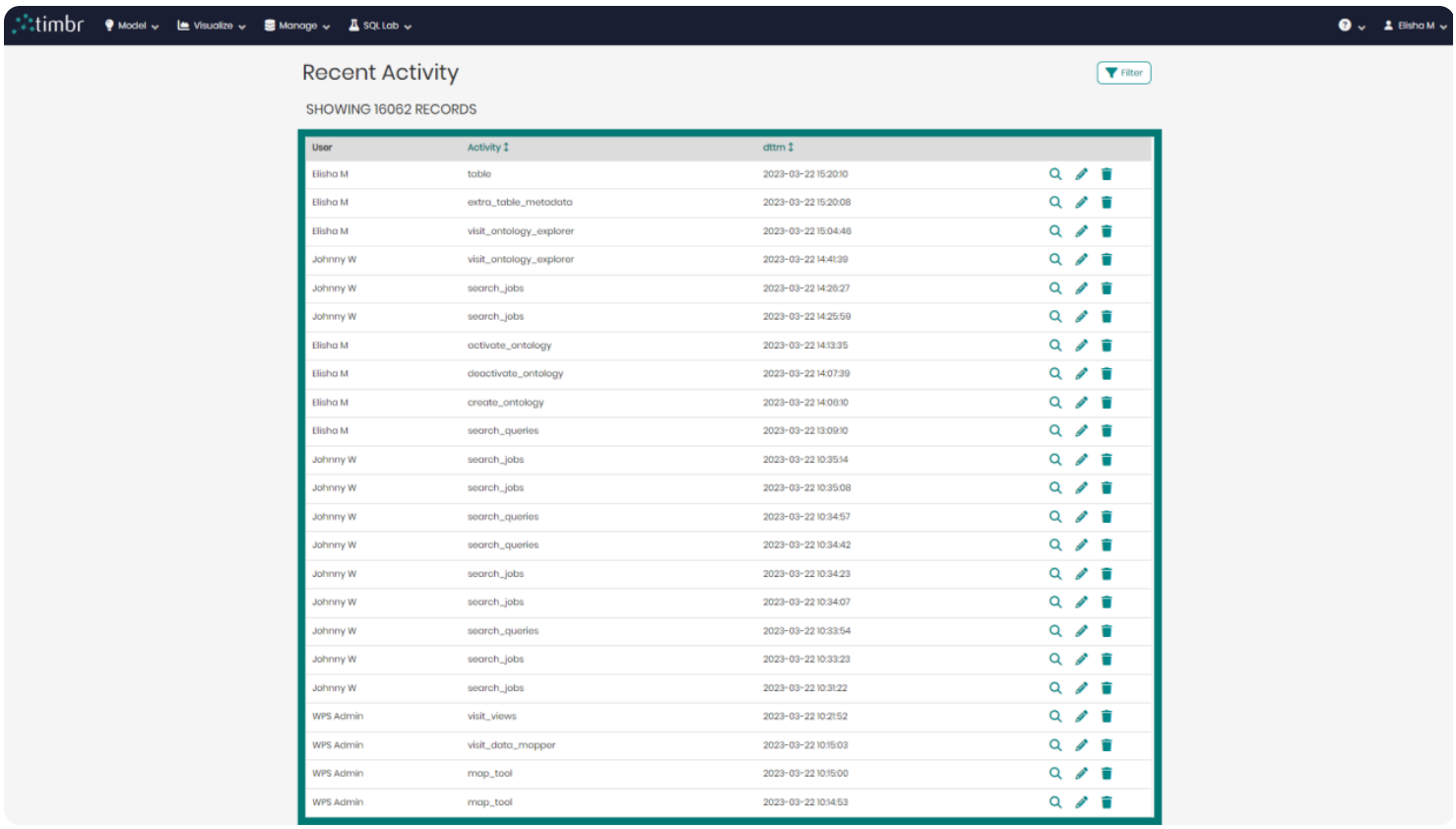
The **Recent Activity** can be accessed through the **Manage** tab and contains the following:

Recent Activity Filter



Filter - On the top right is filter which opens a pop-up with options to filter the recent activities by *User*, *Activity*, *Dashboard Id*, *Slice Id*, *JSON*, *dttm*, *Duration Ms*, *Referrer*, and *Component*.

Recent Activity list



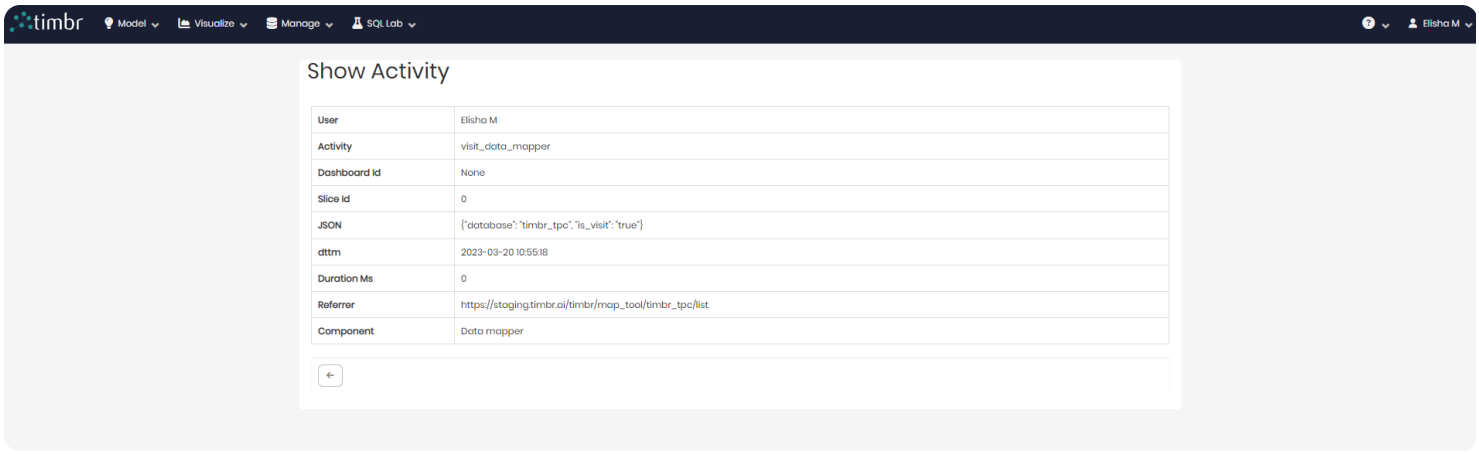
The Recent Activity list in the center of the screen contains the following columns:

User - Presents the user behind each Recent Activity.

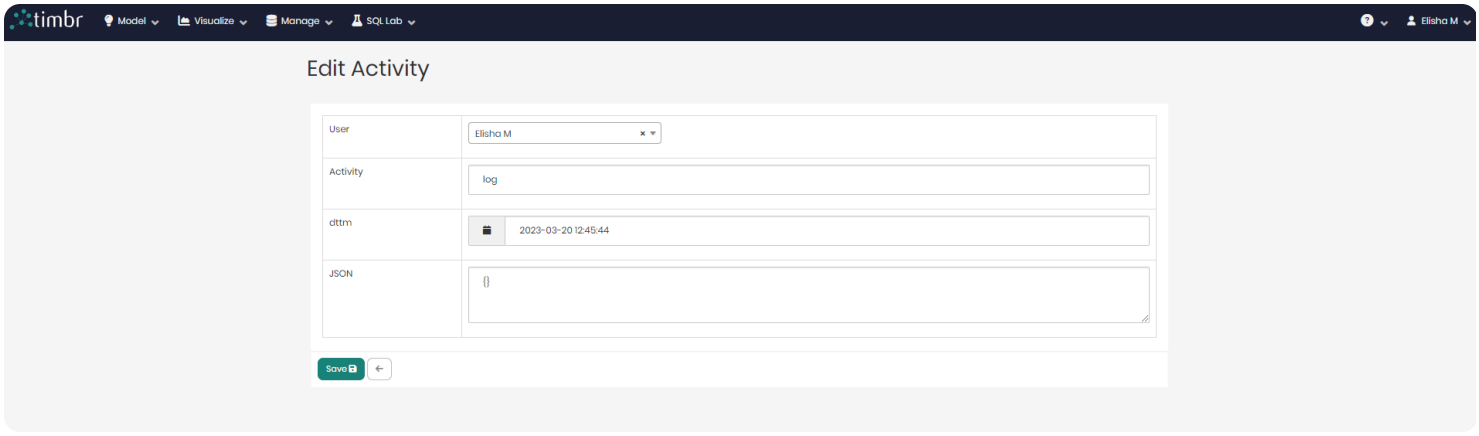
Activity - Presents the type of Recent Activity.

dttm - Presents the date and time at which the Recent Activity took place.

Show Recent Activity - Show Recent Activity is presented by a *Magnifying glass* icon that when clicked on opens a window with additional information about the Recent Activity.



Edit Recent Activity - Edit Recent Activity is presented by a *Pencil* icon that when clicked on opens a window that enables editing details regarding the Recent Activity.

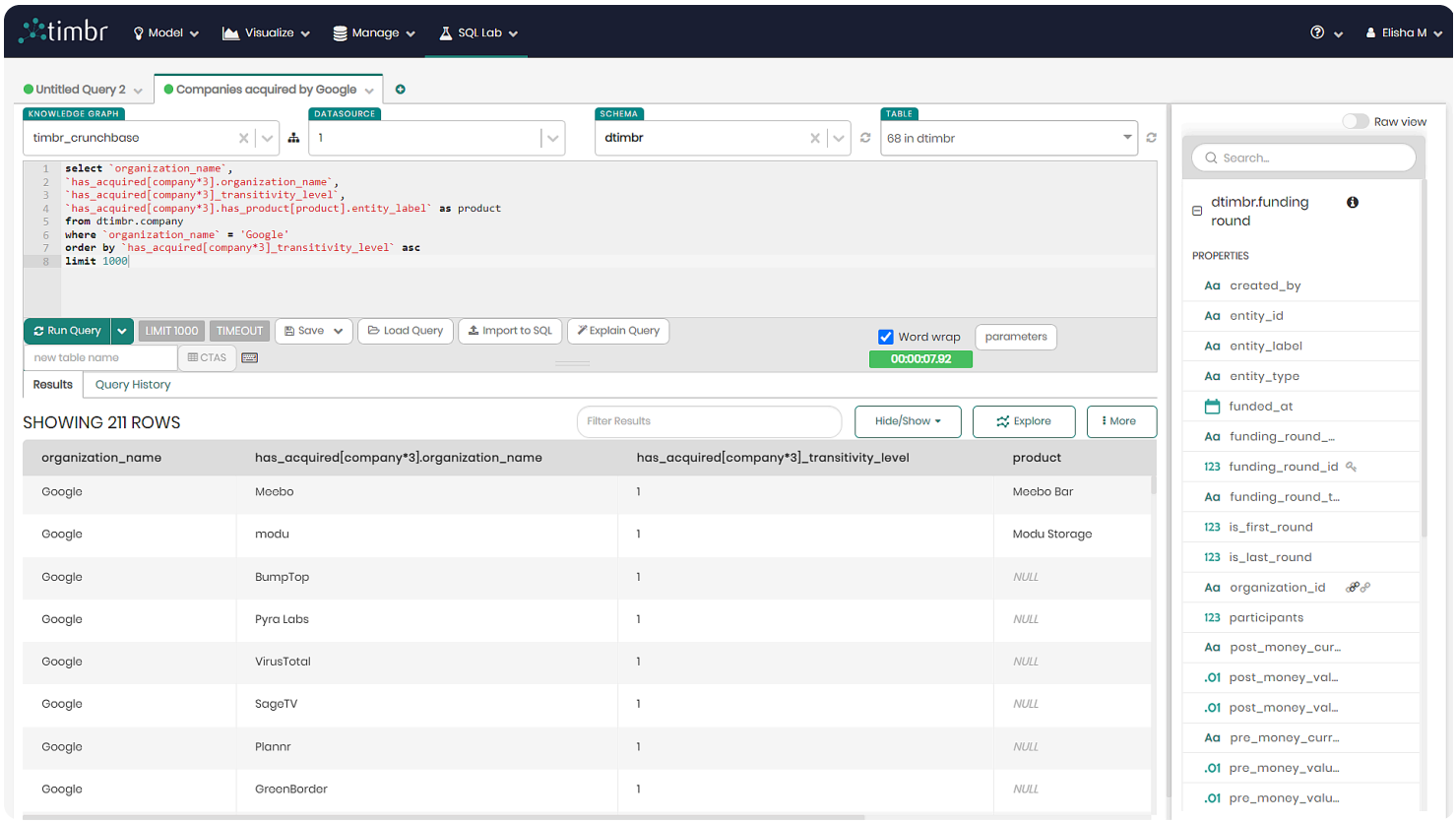


Delete Recent Activity - Delete Recent Activity is presented by a *Trash can* icon that when clicked on deletes the selected Recent Activity.

SQL Editor

The SQL Editor page contains a modern, feature-rich SQL editor that can query the data sources connected to the platform and the virtual semantic models (based on permissions). Users can filter or search the results set, rearrange hide or show columns, explore concepts metadata, and expand relationships (under the dtimbr schema) to easily copy & paste relationship paths that include multiple concepts to avoid writing complex queries.

Every operation that can be done in the platform can also be done directly in the SQL Editor page using SQL statements - connecting data sources, creating or dropping ontologies, users, managing permissions, concepts, mappings, views, cache jobs, etc.



OTHER METHODS TO QUERY THE KNOWLEDGE GRAPH

Users can also query the graph by using the Timbr REST API or using **Apache Spark**, **R**, **Python**, **Scala**, and **Java**.

Feature Overview

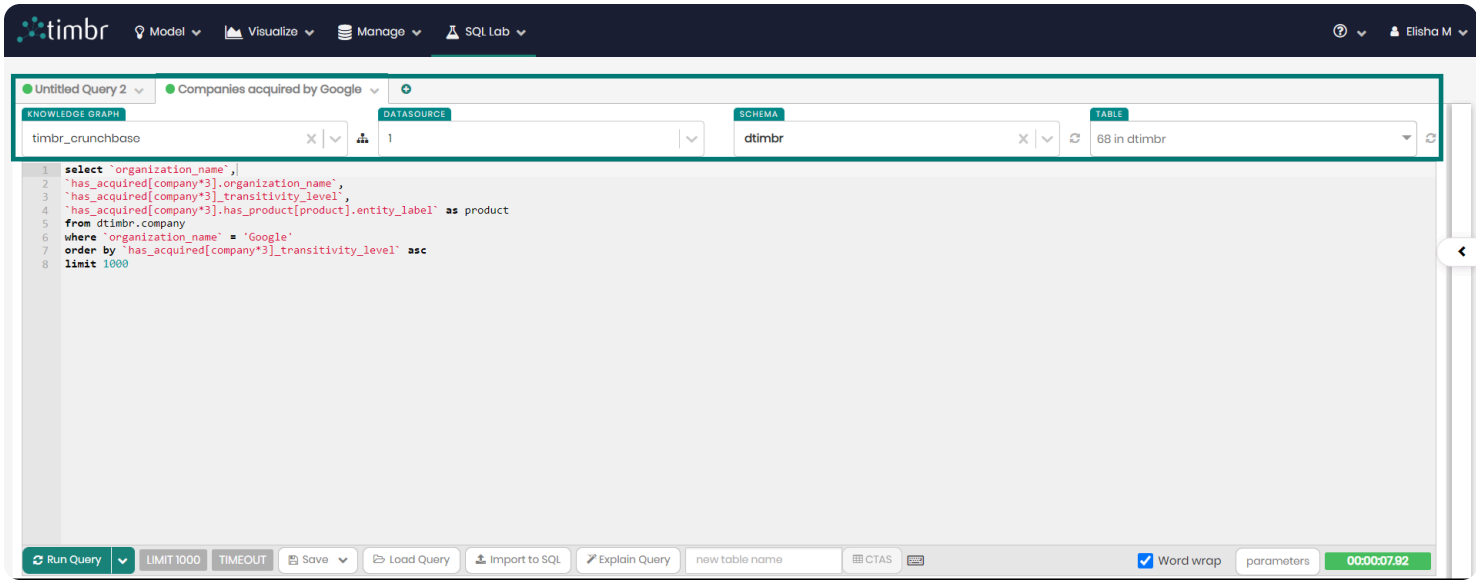
- Connects to just about any database backend
- A multi-tab environment to work on multiple queries at a time
- A smooth flow to visualize your query results using Timbr's rich visualization capabilities
- Browse database metadata: tables, columns, indexes, partitions
- Support for long-running queries
- A search engine to find queries executed in the past
- Supports templating using the Jinja templating language which allows for using macros in your SQL code

---

SQL Editor Page

The **SQL editor** page is used to run SQL queries and it can be found within the **SQL Lab** menu item and contains the following components:

Upper Query Pane



The upper query pane above the main query box is the starting point in the SQL editor. The upper pane contains two elements:

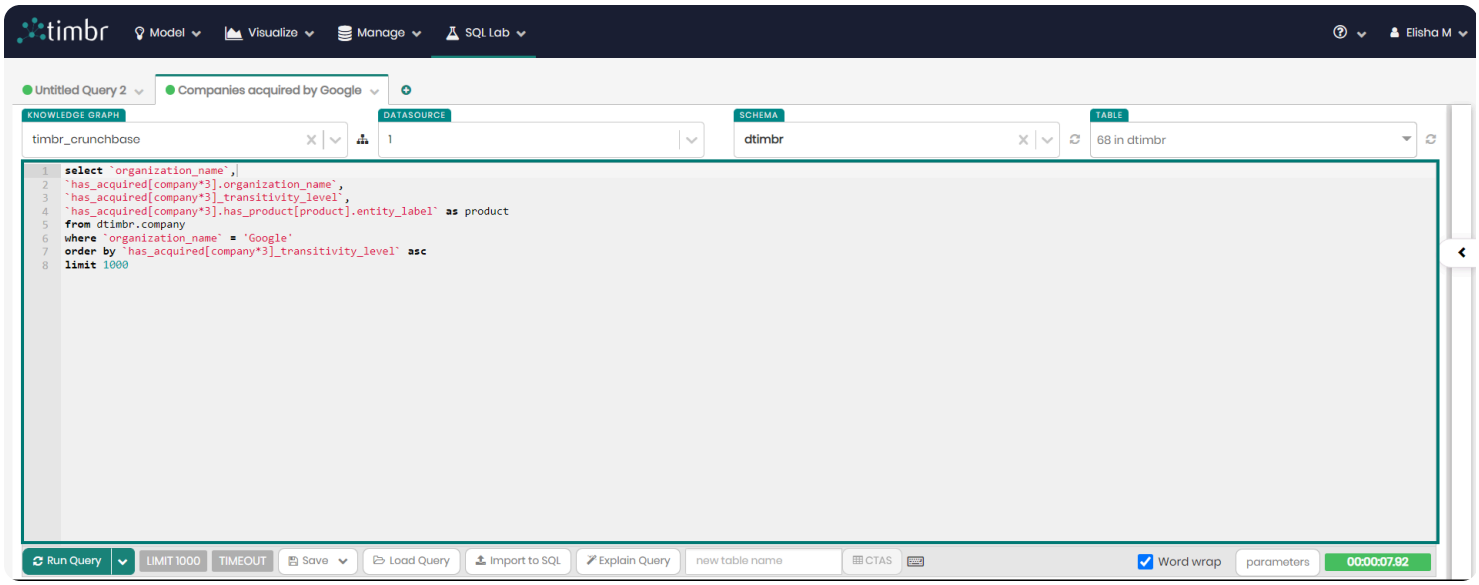
1. **Query tabs** - Choosing either to create a query in the default tab that opens when entering the SQL editor or instead opening a new tab which can be done by clicking on the green plus symbol to the right of the tabs.

2. **Data source selection** - Selecting the knowledge graph and datasource we'd like to query which is mandatory. The schema and table are optional and can be chosen to reveal the table columns and concept properties that will be revealed in the schema panel on the right side.

Additional features that can be found in the upper query pane are:

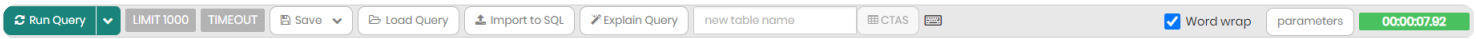
> - **Show selected ontology** - To the right of the chosen knowledge graph, there is a symbol of a hierarchy which when clicked on opens a small pop-up presenting the entire selected ontology model.
> - **Force refresh** - To the right of the chosen schema and table are the force refresh buttons, which can be used to refresh the schemas and tables if perhaps any changes have been made to them.

Main Query Box



The main query box is the space where users create, edit, and run their semantically enabled SQL queries.

Query Toolbar



The query toolbar can be found directly under the main query box. The toolbar includes the following features:

Run Query - Runs the query in the main query box. The run query button contains a downwards pointing arrow which enables users to run the query and download the query results as a CSV file.

Limit - Choosing the row limit to be queried. The default is 1000 rows.

Timeout - Determining the query timeout in seconds. When left empty default timeout will be in effect.

Save - When clicked on, a dropdown opens with the 3 following options:

> - **Save query** - Saves the current query which can later be found in the SQL lab tab under Saved Queries.
> - **Map to concept** - Enables to map the query results to a chosen concept in the knowledge graph.
> - **Create view** - This enables creating a view above the knowledge graph using the query results.

Load Query - This opens a window with all the saved queries, where any query or even multiple queries can then be loaded into the main query box in their own new tab.

Import to SQL - This enables uploading a CSV file and having Timbr convert the CSV file into an SQL query.

Explain Query - This opens an explain query window which presents the query that was pushed down behind the scenes of our current query to the datasource. This enables users to see what their query would look like without Timbr's relationships and business logic, where users would need to write up to 90% more code.

New table name - This enables typing a new table name and saving the query results to the new table being created.

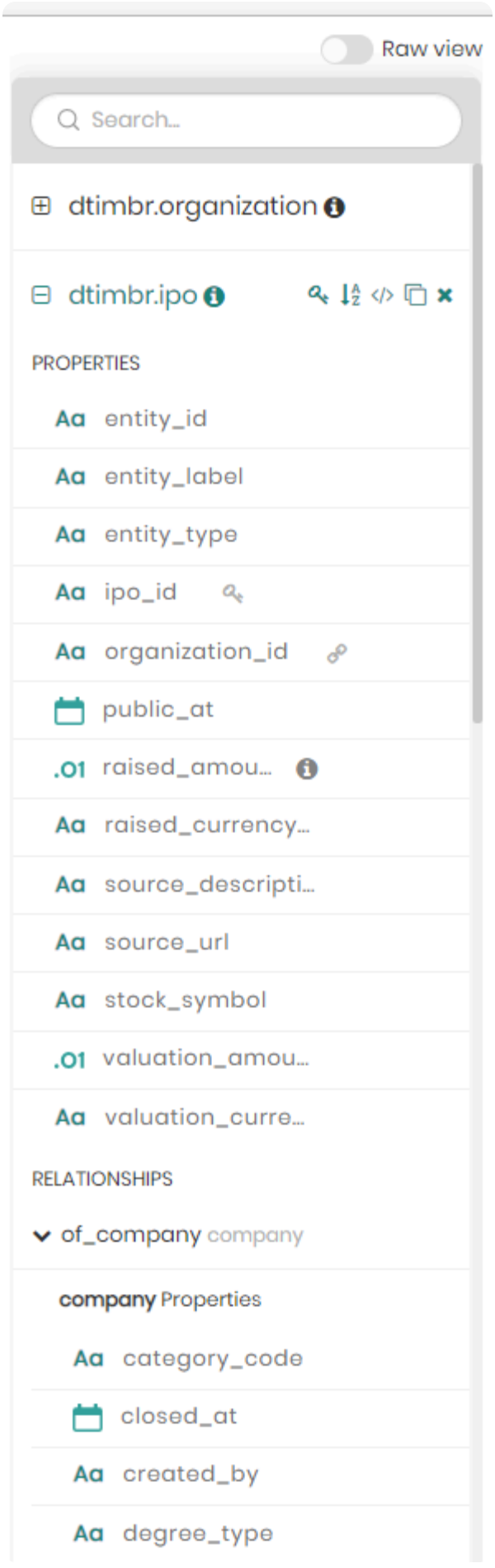
Keyboard shortcuts and snippets - Presents various shortcuts and snippets in the SQL editor helping users write and run queries faster.

Word wrap - When the word wrap checkbox is marked, word wrap will be activated and any text that exceeds the length of any specific row will be pushed down to the next row. When the word wrap checkbox is unmarked text on any row can continue until specified otherwise.

Parameters - This enables creating and editing templates using the Jinja templating language which allows for using macros in the SQL code.

Query timer - Shows the query run time once the run query button is clicked.

Schema Metadata Panel


Raw view
Search...
dtimbr.organization
dtimbr.ipo
PROPERTIES
Aa entity_id
Aa entity_label
Aa entity_type
Aa ipo_id
Aa organization_id
public_at
.01 raised_amou...
Aa raised_currency...
Aa source_descripti...
Aa source_url
Aa stock_symbol
.01 valuation_amou...
Aa valuation_curre...
RELATIONSHIPS
of_company company
company Properties
Aa category_code
closed_at
Aa created_by
Aa degree_type

The schema metadata panel can be found on the right side of the main query box once a schema and table are selected. The panel presents the selected tables' metadata containing their properties and relationships, enabling users to copy and paste the properties and relationships directly into the query.

When hovering over a table in the metadata, to the right of the schema and table name the following symbols will appear:

Keys and indexes - This enables users to view the different keys and indexes that exist in the current table.

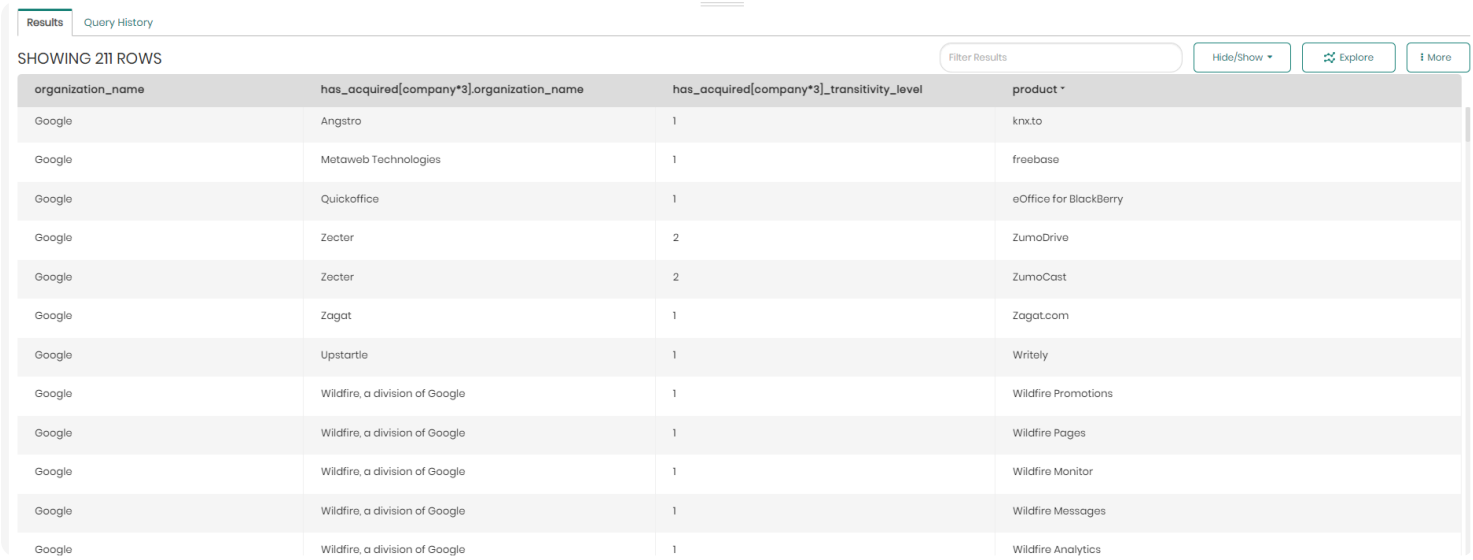
Sort columns alphabetically - Sorts the columns of the table in alphabetical order.

Copy full table name - Enables users to copy the full schema and table name that can then be pasted into the main query box.

Copy statement - Copies the entire metadata containing all the properties and relationships to the clipboard as a SELECT statement, that can then be pasted into the main query box.

Remove table preview - When the X is clicked the current table selected will be removed from the metadata panel.

Query Results



organization_name	has_acquired[company*3].organization_name	has_acquired[company*3]_transitivity_level	product
Google	Angstro	1	knx.to
Google	Metaweb Technologies	1	freebase
Google	Quickoffice	1	eOffice for BlackBerry
Google	Zecter	2	ZumoDrive
Google	Zecter	2	ZumoCast
Google	Zagat	1	Zagat.com
Google	Upstartle	1	Writely
Google	Wildfire, a division of Google	1	Wildfire Promotions
Google	Wildfire, a division of Google	1	Wildfire Pages
Google	Wildfire, a division of Google	1	Wildfire Monitor
Google	Wildfire, a division of Google	1	Wildfire Messages
Google	Wildfire, a division of Google	1	Wildfire Analytics

The query results can be seen on the bottom portion of the screen after a query has finished running. The query results section contains the following two main tabs:

Results

The results section presents the results of the query and offers additional features which can be found right above the results box on the upper right side which includes:

Filter Results - A search bar to type and filter for specific results.

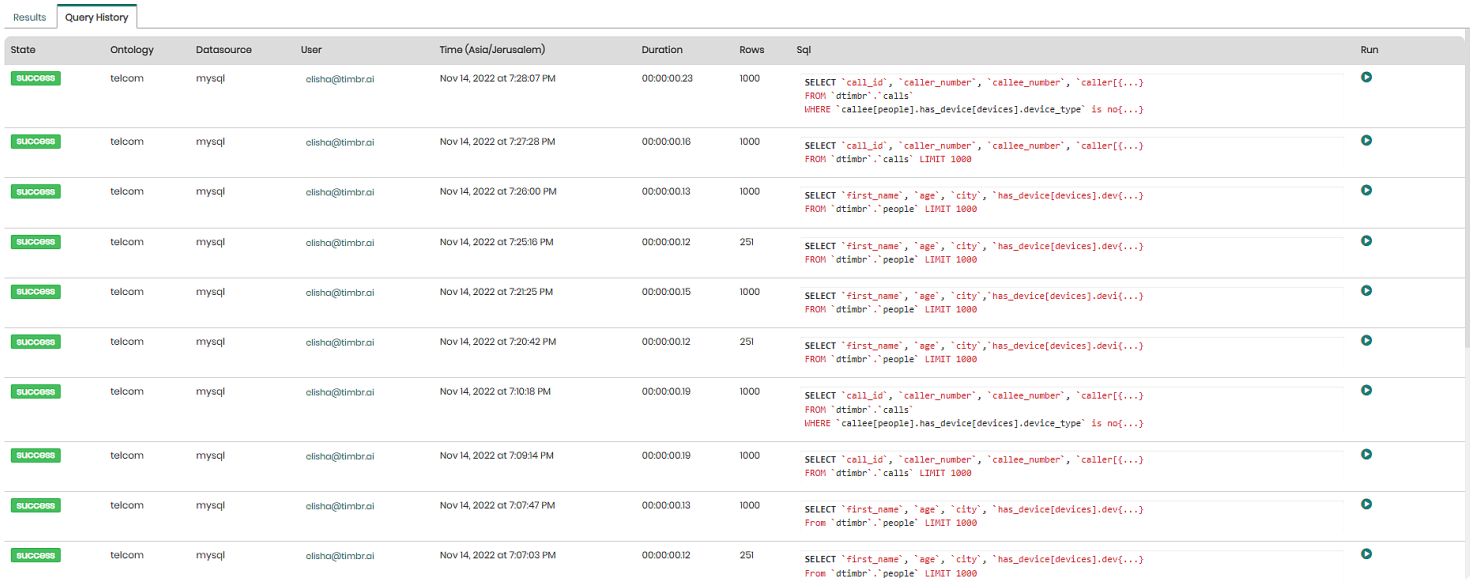
Hide/Show - Choosing which result columns to hide and which to show.

Explore - Enables to transfer the query results to Timbr's built-in BI module to explore and present the results using any of the various charts offered by Timbr.

More - The more button gives users additional options which include:

- **Remove null columns** - Removes all the null columns in the query results.
- **Export to CSV** - Exports the query results as a CSV file.
- **Results Stats** - Presents different stats of the query such as total rows, total columns, total cells, existing values, nulls, query limit, query runtime, query start, and end date.
- **Clipboard** - Copies the entire results to the clipboard to be used anywhere.
- **Show column stats** - Adds stats above the different columns such as unique values, null values, Min, Max, Avg, and more.

Query History



The query history tab presents the history of all queries that have been run in the past including additional information such as:

- **State** - Presents the state of each query, whether the query ran successfully or failed.
- **Ontology** - Presents the name of the Ontology that was used in each query.
- **Datasource** - Presents the Datasource that was used in each query.
- **User** - Presents the user behind each query.
- **Time** - States the exact time and date for each query at run time.
- **Duration** - States the amount of time it took to run each query.
- **Rows** - Shows the number of result rows that have been returned for each query.
- **SQL** - Shows a preview of each query and its SQL syntax. When clicked on, a popup window will appear presenting the entire selected query in cases where the query was too long to show in the preview.

- **Run** - When clicked on the selected query will start to run in the main query box.

Extra features

- Hit `alt + enter` as a keyboard shortcut to run your query

Templating with Jinja

```
SELECT *
FROM some_table
WHERE username = '{{ current_username() }}'
```

Templating unleashes the power and capabilities of a programming language within your SQL code.

Templates can also be used to write generic queries that are parameterized so they can be re-used easily.

Available macros

We expose certain modules from Python's standard library in Timbr's Jinja context:

- `time`: `time`
- `datetime`: `datetime.datetime`
- `uuid`: `uuid`
- `random`: `random`
- `relativedelta`: `dateutil.relativedelta.relativedelta`

Jinja's built-in filters can also be applied where needed.

superset.jinja_context.current_user_id

superset.jinja_context.current_username

superset.jinja_context.url_param

superset.jinja_context.filter_values

superset.jinja_context.CacheKeyWrapper.cache_key_wrapper

superset.jinja_context.PrestoTemplateProcessor

superset.jinja_context.HiveTemplateProcessor

Extending macros

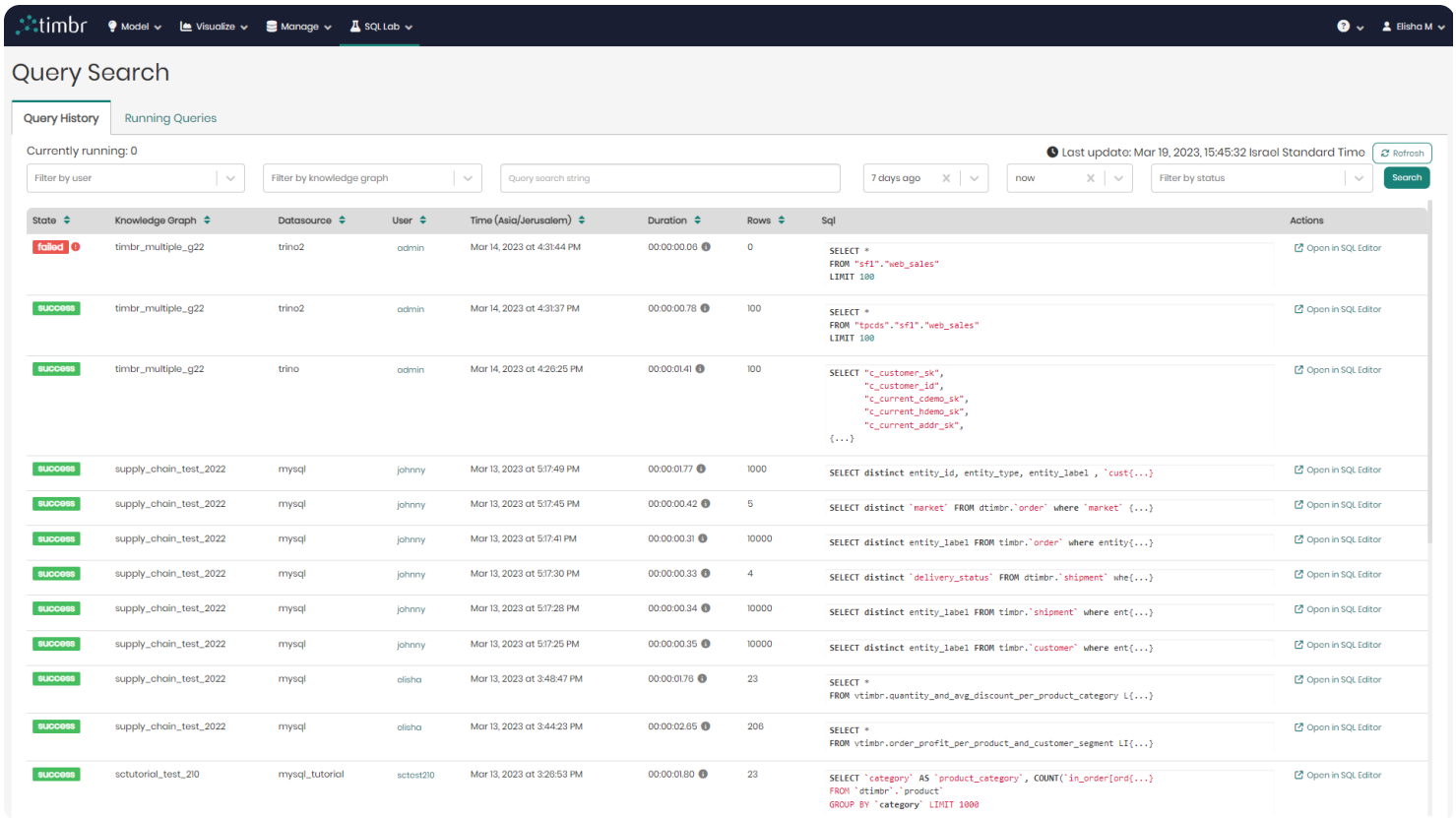
It's possible for administrators to expose more macros in their environment using the configuration variable `JINJA_CONTEXT_ADDONS`. All objects referenced in this dictionary will become available for users to integrate into their queries in the **SQL Editor**.

Query Search

In the Query Search page users can search historical queries based on a knowledge graph, database, query string, query status, timeframe, or users. The Query Search is also where users can stop running queries, monitor their status, or check for failed queries and the reason for why they failed.

The **Query Search** can be accessed through the **SQL Lab** tab and contains the following components:

Query History



When entering the Query Search page there are two tabs. The default tab that appears is the Query History tab, which presents all the information about previously asked queries.

beneath the tabs is the **upper query pane** which contains the following options:

Currently running - Presents the number of queries that are currently running.

Filter by user - Enables to search through and filter the queries asked by specific users.

Filter by knowledge graph - Enables to search through and filter the queries asked on specific knowledge graphs.

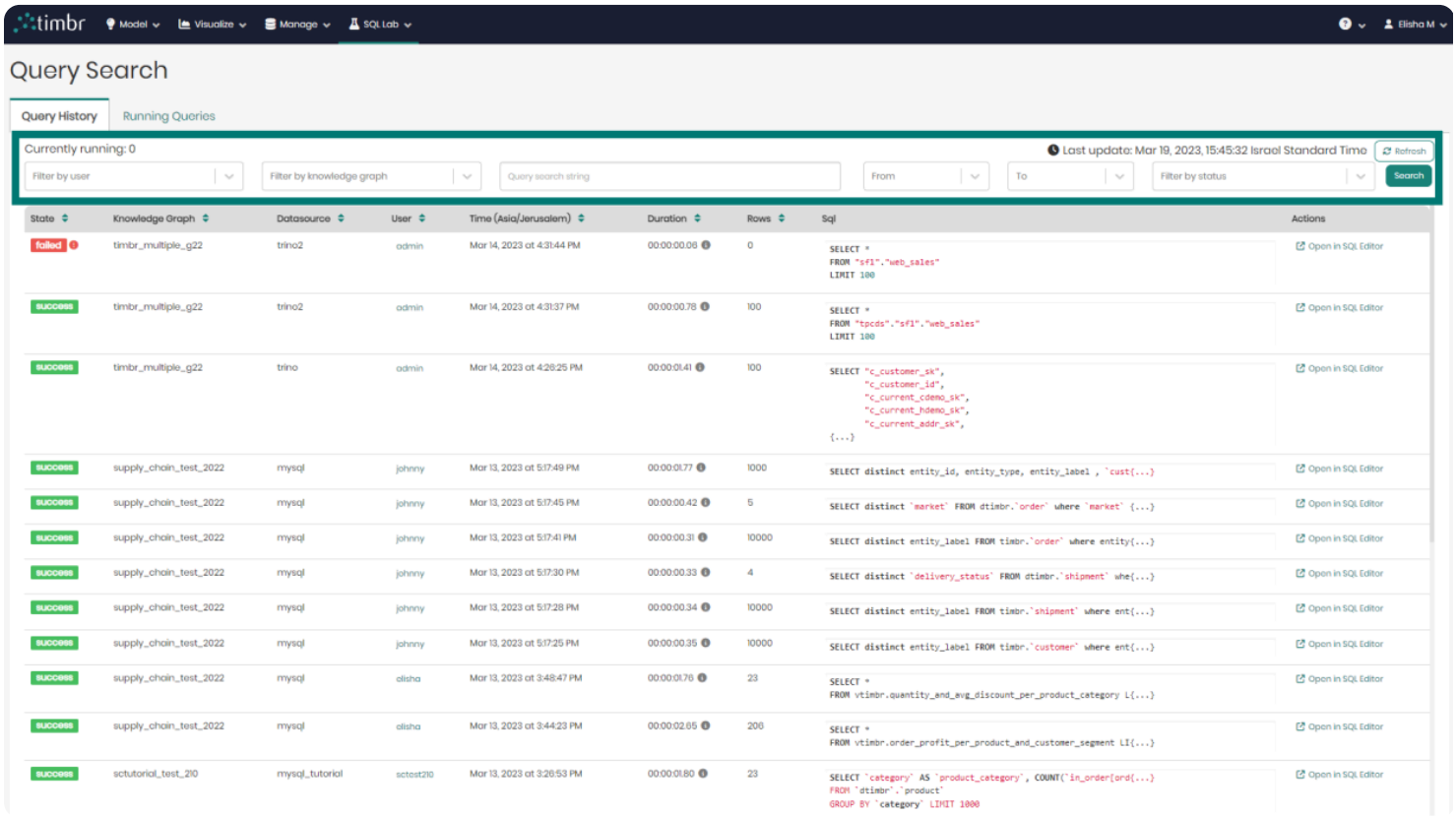
Query search string - Enables to search through and filter the queries based on specifically chosen text.

From & To - Enables to search through and filter the queries based on a specific time frame.

Filter by status - Enables to search through and filter the queries based on a specific query status.

Refresh - Performs a refresh on the list of queries.

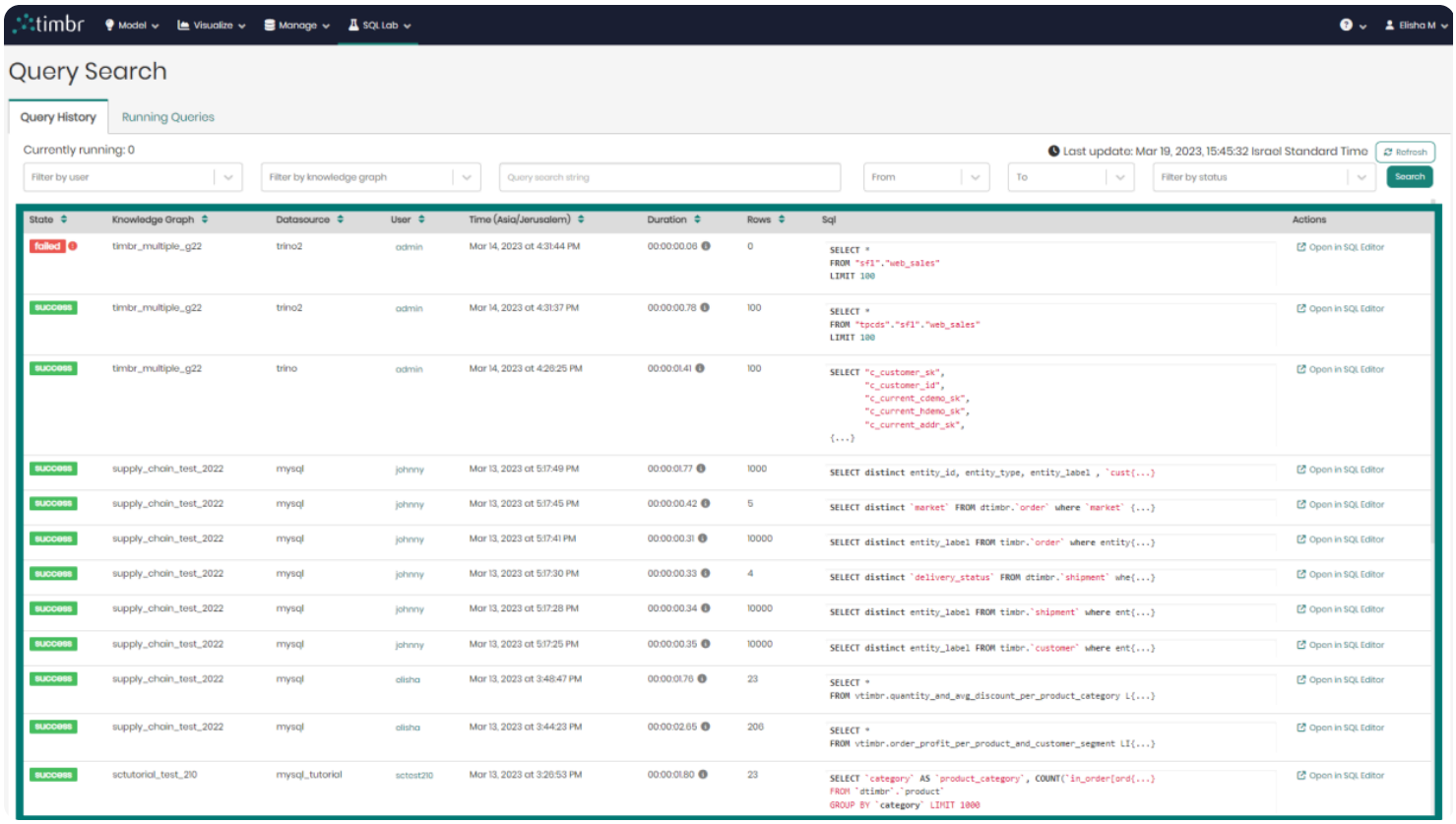
Search - Performs a search on the queries based on the parameters chosen in the various filters.



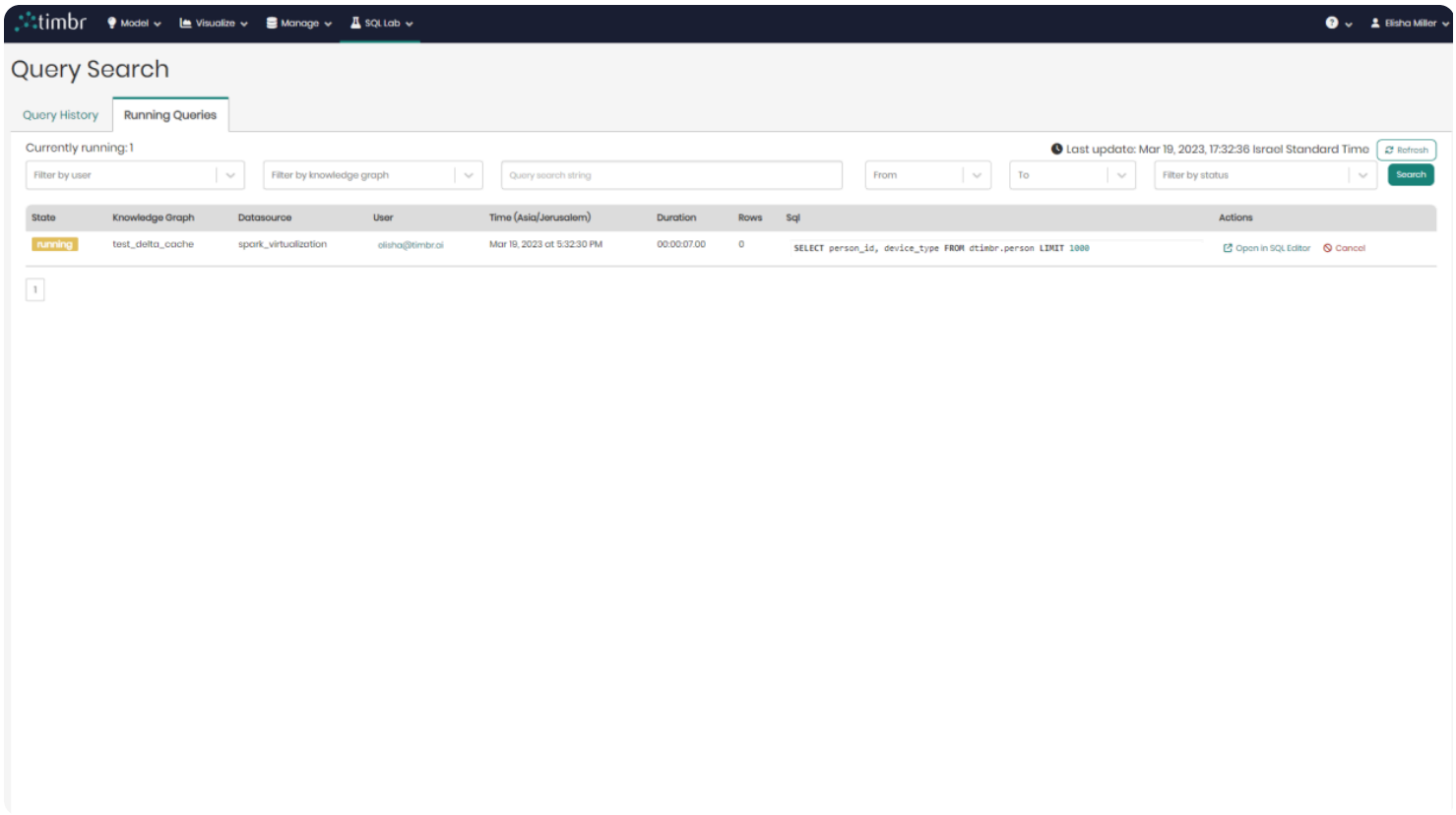
Beneath the upper query pane is the entire list of queries that were previously run. The list contains the following columns:

- **State** - Presents the state of each query, whether the query ran successfully or failed.
- **Knowledge Graph** - Presents the name of the Knowledge Graph that was used in each query.
- **Datasource** - Presents the Datasource that was used in each query.
- **User** - Presents the user behind each query.
- **Time** - States the exact time and date for each query at run time.
- **Duration** - States the amount of time it took to run each query.
- **Rows** - Shows the number of result rows that have been returned for each query.

- **SQL** - Shows a preview of each query and its SQL syntax. When clicked on, a popup window will appear presenting the entire selected query in cases where the query was too long to show in the preview.
- **Actions** - Contains a button called **Open in SQL Editor** that When clicked on will open the selected query in Timbr's SQL Editor.



Running Queries



The second tab that appears in the Query Search page is the Running Queries tab, which presents all the information about currently running queries.

Similarly to the Query History tab, here too there's an **upper query pane** that contains the following options:

Currently running - Presents the number of queries that are currently running.

Filter by user - Enables searching through the currently running queries and filtering the queries asked by specific users.

Filter by knowledge graph - Enables to search through the currently running queries and filter the queries asked on specific knowledge graphs.

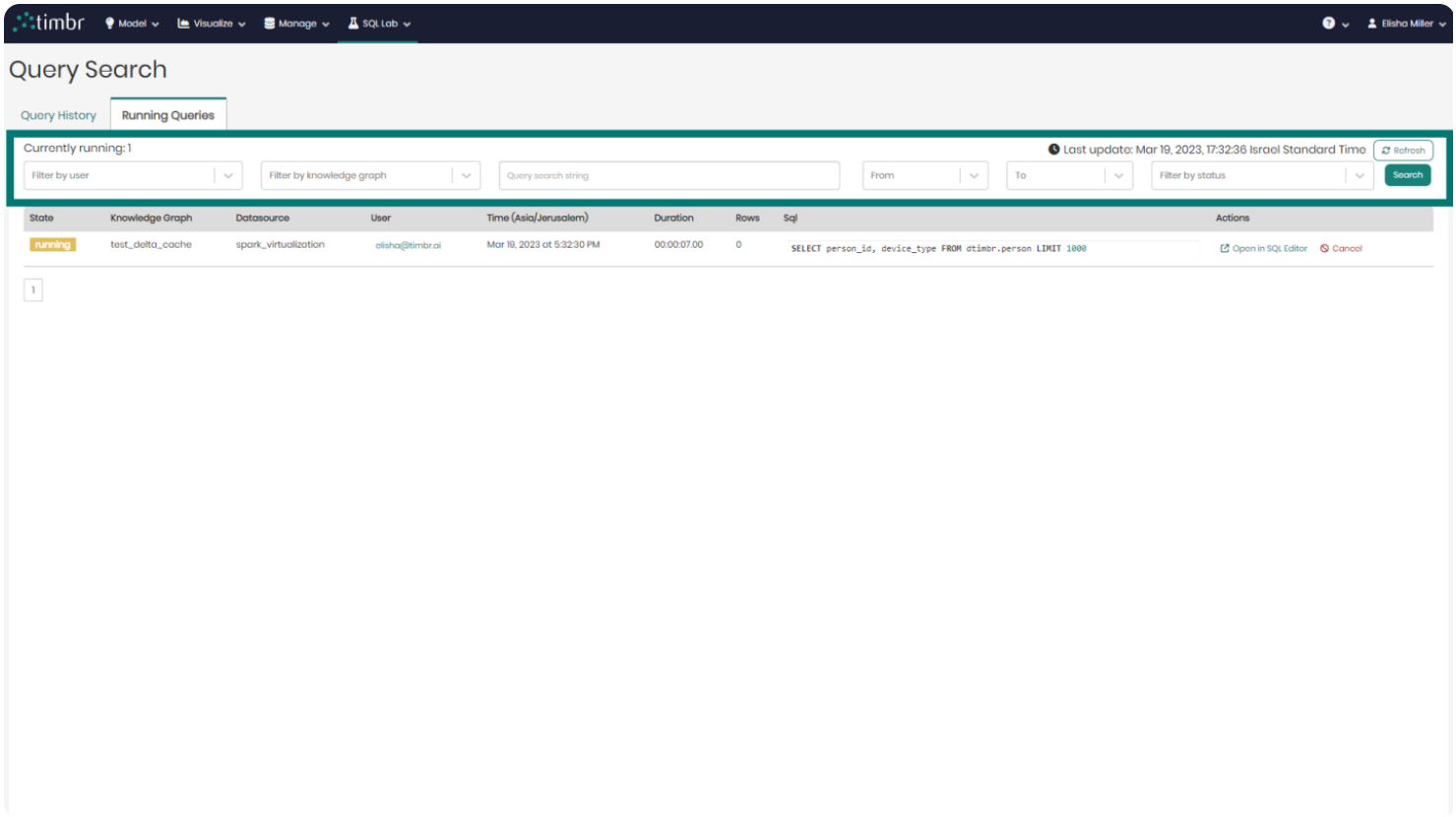
Query search string - Enables to search through the currently running queries and filter the queries based on specific chosen text.

From & To - Enables to search through the currently running queries and filter the queries based on a specific time frame.

Filter by status - Enables searching through the currently running queries and filtering the queries based on a specific query status.

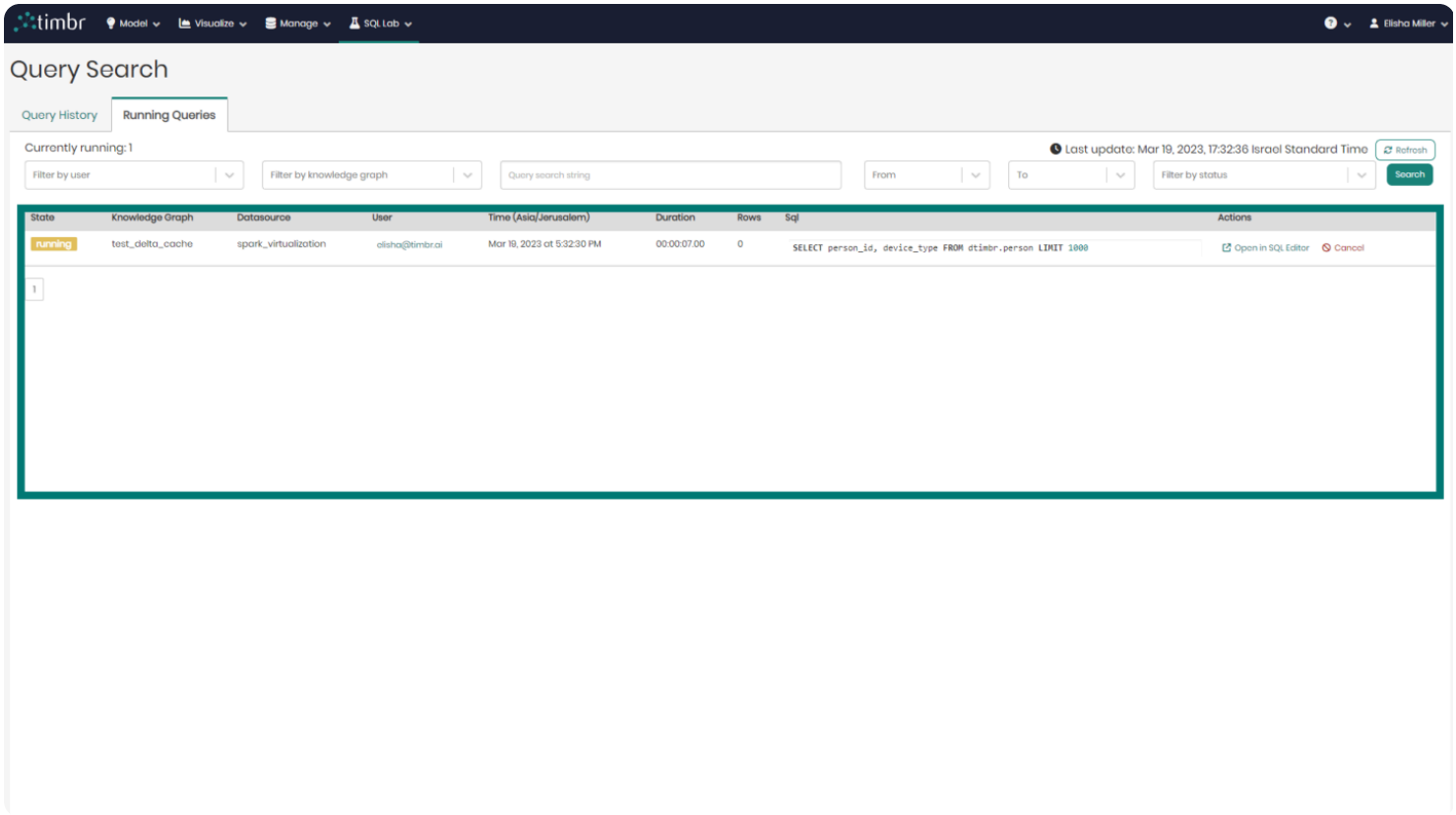
Refresh - Performs a refresh on the list of currently running queries.

Search - Performs a search on the currently running queries based on the parameters chosen in the various filters.



Beneath the upper query pane is the entire list of currently running queries. The list contains the following columns:

- **State** - Presents the state of each currently running query.
- **Knowledge Graph** - Presents the name of the Knowledge Graph that is being used in each query.
- **Datasource** - Presents the Datasource that is being used in each query.
- **User** - Presents the user behind each currently running query.
- **Time** - States the exact time and date for each query at run time.
- **Duration** - States the amount of time each query has been running for.
- **Rows** - Shows the number of result rows that have been returned for each query.
- **SQL** - Shows a preview of each query and its SQL syntax. When clicked on, a popup window will appear presenting the entire selected query in cases where the query was too long to show in the preview.
- **Actions** - Contains a button called **Open in SQL Editor** that When clicked on will open the selected query in Timbr's SQL Editor, as well as a **Cancel** button that when clicked on will cancel the currently running query selected.

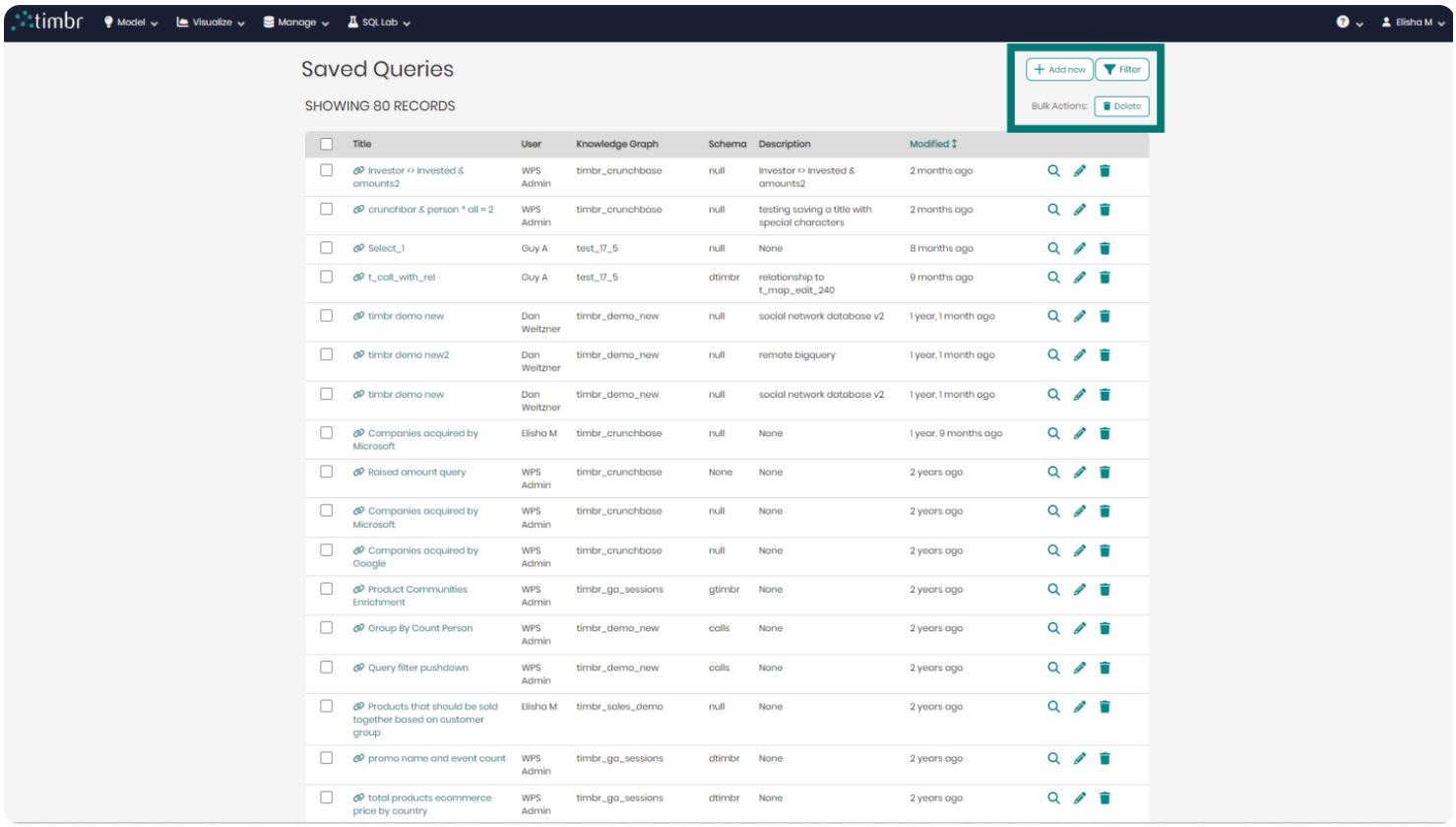

timbr
Model
Visualize
Manage
SQL Lab
Elisha Miller
Query Search
Query History
Running Queries
Currently running: 1
Last update: Mar 19, 2023, 17:32:36 Israel Standard Time
Refresh
Filter by user
Filter by knowledge graph
Query search string
From
To
Filter by status
Search
State
Knowledge Graph
Datasource
User
Time (Asia/Jerusalem)
Duration
Rows
Sql
Actions
running
test_delta_cache
spark_virtualization
elisha@timbr.ai
Mar 19, 2023 at 5:32:30 PM
00:00:07.00
0
SELECT person_id, device_type FROM dtimbr.person LIMIT 1000
Open in SQL Editor
Cancel
1

Saved Queries

In the Saved Queries page users can find a list of saved queries by platform users. Saved Queries can be searched and filtered according to labels, knowledge graphs, users, timeframes, or the last time a saved query was modified.

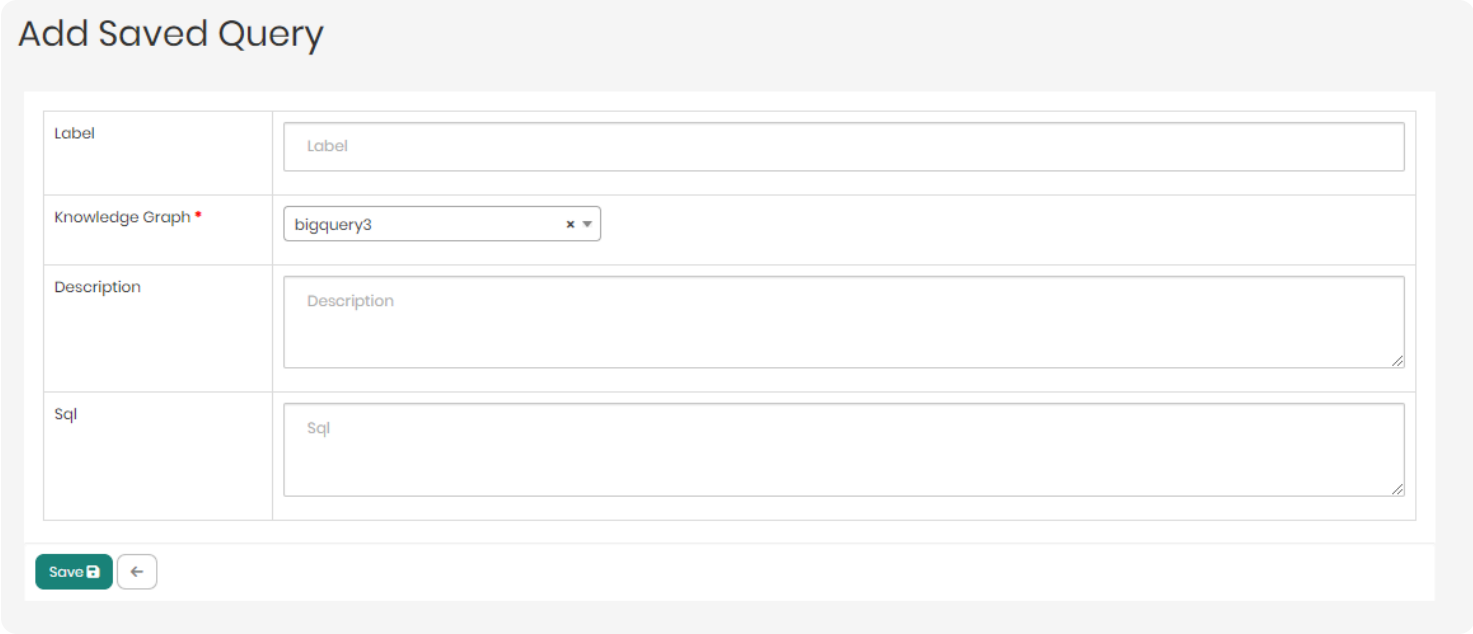
The **Saved Queries** can be accessed through the **SQL Lab** tab and it contains the following components:

Top right actions



The actions that can be performed on the top right of the screen include:

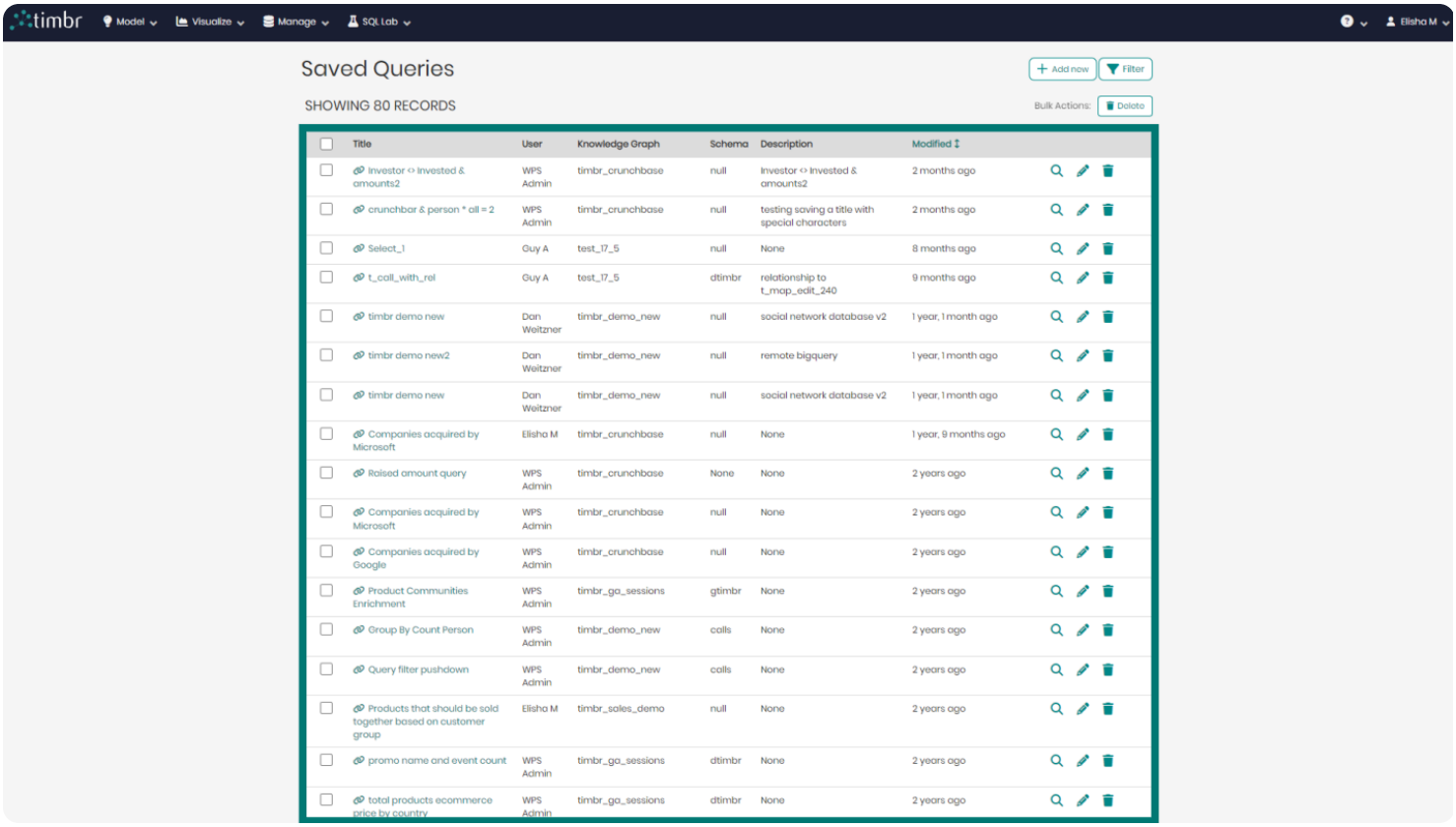
Add New - Opens a new window to create and save a new SQL query.



Filter - Opens a pop-up with options to filter the saved queries by *Label*, *User*, *Knowledge Graph*, *Schema*, and *Changed on*, which is the last modification date of the query.

Bulk Actions - Bulk actions contain the delete button which enables users to choose multiple queries to delete at once.

Saved Query list



The saved query list in the center of the screen contains the following columns:

Checkbox - Allows to click and choose specific queries to perform bulk actions on.

Title - Presents the title that was given to each saved query. By clicking on a title, the query with the selected title will open in the SQL Editor where the query can be reused or edited.

User - Presents the user behind each saved query.

Knowledge Graph - Presents the name of the knowledge graph behind the saved query.

Schema - Presents the name of the schema behind the saved query.

Description - Presents the description behind the saved query if any description was given.

Modified - Presents the last time the saved query was modified.

Show Saved Query - Show saved query is presented by a *Magnifying glass* icon that when clicked on opens a window with additional information about the saved query.

Edit Saved Query - Edit saved query is presented by a *Pencil* icon that when clicked on opens a window that enables editing the saved query.

Delete Saved Query - Delete saved query is presented by a *Trash can* icon that when clicked on deletes the selected saved query.

Introduction

This section provides a quick reference to timbr commands and queries. All the queries can be performed against any timbr-server from version `2.9.0` and onwards.

Table of contents

User Actions, Roles, and Permissions

This section is about granting and revoking permissions for users and roles, and other user / role actions in Timbr.

Granting and revoking user or role permissions

The pattern for **GRANTING PERMISSIONS** is as follows:

```
GRANT [EDIT|QUERY|ACCESS] ON <ALL> {resources} [TO USER|TO ROLE] {username_or_role_name} <WITH GRANT OPTION>
```

The pattern for **REVOKING PERMISSIONS** is as follows:

```
REVOKE [EDIT|QUERY|ACCESS] ON <ALL> {resources} [FROM USER|FROM ROLE] {username_or_role_name} <WITH GRANT OPTION>
```

Pattern explanation:

- **[EDIT|QUERY|ACCESS]** - First level permission. Must choose one of `EDIT`, `QUERY`, or `ACCESS`
  - `EDIT` - grants access to create, modify or remove a resource
  - `QUERY` - grants access to view the metadata and query a resource
  - `ACCESS` - grants access to view the metadata of a resource
- `ALL` - Optional flag that indicates the entire scope of a permission
  - In the context of `ontology`, means **[EDIT|QUERY|ACCESS]** to all resources for all ontologies, a specific ontologies, or all resources of a specific ontology.
  - In the context of `EDIT ON ALL datasources` means udpate or remove all datasources
  - In the context of `EDIT ON ALL users` can update or remove all users
  - In the context of `QUERY ON ALL users` to view all users
  - In the context of `EDIT ON ALL roles` can update or remove all roles
  - In the context of `QUERY ON ALL roles` to view all roles
- **{resources}** - A hierarchy of different resources, each with a scope drilldown denoted by a dot `.`. The list of first level resources are:
  - `ontology` - Second level permission. If only the first and second levels are specified, and it has the `ALL` flag set -Then it represents **[EDIT|QUERY|ACCESS]** permissions for all ontologies. To grant permission to create ontologies, dont specify the `ALL` flag, and the `EDIT` permission must be set - Then it means permissions to **Create ontologies**

        - **{ontology name}** - Third level permission. If only the first, second, and third levels are specified, and it has the `ALL` flag then it represents permissions for all resources within a specific ontology. If the next level resource (fourth) is specified then it must be one of the following options:
            - `mapping` - Fourth level permission. If only the first, second, third, and fourth levels are specified, and it has the `ALL` flag then it represents permissions for all mappings within a specific ontology. If the next level resource (fifth) is also specified then it **Can't contain the ALL flag**, and it represents a permission to a specific mapping in the ontology.
            - `view` - Fourth level permission. If only the first, second, third, and fourth levels are specified, and it has the `ALL` flag then it represents permissions for all views within a specific ontology. If the next level resource (fifth) is also specified then it **Can't contain the ALL flag**, and it represents a permission to a specific view in the ontology.
            - `schema` - Fourth level permission. If only the first, second, third, and fourth levels are specified, and it has the `ALL` flag then it represents permissions for all schemas within a specific ontology. If the next level resource (fifth) is also specified then it **Can't contain the ALL flag**, and it represents a permission to a specific schema in the ontology which must be one of `timbr`, `dtimbr`, `etimbr`, `vtimbr`, or `gtimbr`
            - `query` - Fourth level permission. If only the first, second, third, and fourth levels are specified, and it has the `ALL` flag then it represents permissions to view the **query history** and the **running queries** of a specific ontology. The next level resource (fifth) is also specified then it **Can't contain the ALL flag** and it must be one of `query_history` or `running_queries`
    - `datasource` - Second level permission. If only the first and second levels are specified, and it has the `ALL` flag set -Then it represents **[EDIT|QUERY|ACCESS]** permissions for all datasources. To grant permission to create datasource, don't specify the `ALL` flag, and the `EDIT` permission must be set
        - Then it means permissions to **Create datasources**
        - **{datasource name}** - Third level permission. If only the first, second, and third levels are specified it represents permissions for for a specific datasource
    - `user` - Second level permission. If only the first and second levels are specified, and it has the `All` flag set - Then it must have either the `EDIT` (first level) option set which represents creating / updating / removing all users. Or the `VIEW` (first level) option set which represents viewing all the users.
        - **{user name}** - Third level permission. Can be created only when the first level `EDIT` option is set, and it represents updating / removing a specific user
    - `role` - Second level permission. If only the first and second levels are specified, and it has the `All` flag set - Then it must have either the `EDIT` (first level) option set which represents creating / updating / removing all roles. Or the `VIEW` (first level) option set which represents viewing all the roles.
        - **{role name}** - Third level permission. Can be created only when the first level `EDIT` option is set, and it represents updating / removing a specific role
- `WITH GRANT OPTION` - Optional flag which represents whether the permission given can be also granted to other users or roles.

Example #1: Grant the user **robert** permission to create ontologies:

```
GRANT EDIT ON `ontology` TO USER `robert`;
```

Revoke the user **robert** permission to create ontologies:

```
REVOKE EDIT ON `ontology` FROM USER `robert`;
```

Example #2: Grant the user **robert** permission to modify and query all ontologies, and to grant the permission to other users:

```
GRANT EDIT ON ALL `ontology` TO USER `robert` WITH GRANT OPTION;
```

Revoke the user **robert** permission to modify and query all ontologies, and to grant the permission to other users:

```
REVOKE EDIT ON ALL `ontology` FROM USER `robert` WITH GRANT OPTION;
```

Example #3: Grant the user **robert** permission to query only a specific mapping `map_concept1` in the `test_ontology` ontology:

```
GRANT QUERY ON `ontology`.`test_ontology`.`mapping`.`map_concept1` TO USER `robert`;
```

Revoke the user **robert** permission to query only a specific mapping `map_concept1` in the `test_ontology` ontology:

```
REVOKE QUERY ON `ontology`.`test_ontology`.`mapping`.`map_concept1` FROM USER `robert`;
```

Example #4: Grant the role **manager** to edit all users and roles:

```
GRANT EDIT ON ALL `role` TO ROLE `manager`;
GRANT EDIT ON ALL `user` TO ROLE `manager`;
```

Revoke the role **manager** from editing all users and roles:

```
REVOKE EDIT ON ALL `role` FROM ROLE `manager`;
REVOKE EDIT ON ALL `user` FROM ROLE `manager`;
```

Example #5: Grant the role **ontology_manager** to modify the `test_ontology` ontology and view all of the ontology resources:

```
GRANT EDIT ON ALL `ontology`.`test_ontology` TO ROLE `ontology_manager`;
GRANT EDIT ON `ontology`.`test_ontology` TO ROLE `ontology_manager`;
```

Revoke the role **ontology_manager** from modifying the `test_ontology` ontology and view all of the ontology resources:

```
REVOKE EDIT ON ALL `ontology`.`test_ontology` FROM ROLE `ontology_manager`;
REVOKE EDIT ON `ontology`.`test_ontology` FROM ROLE `ontology_manager`;
```

Example #6: Grant the role **external_viewer** to see only the metadata of the `test_ontology` ontology:

```
GRANT ACCESS ON ALL `ontology`.`test_ontology` TO ROLE `external_viewer`
```

Revoke the role **external_viewer** to see only the metadata of the `test_ontology` ontology:

```
REVOKE ACCESS ON ALL `ontology`.`test_ontology` FROM ROLE `external_viewer`
```

Show the permissions of the current user

Returns the list of permissions for the current user

```
SHOW PERMISSIONS
```

Show the permission history log

Returns a list of permissions changes in an ontology according to the scope of permissions the user has

```
SHOW PERMISSION HISTORY
```

Show the users the current user has access to update

Shows a list of available users which the user has access to udpate

```
SHOW USERS
```

Show the roles the current user has access to update

Shows a list of available roles which the user has access to udpate

```
SHOW ROLES
```

Show which users are assigned which roles

Returns a list of users and associated roles.

```
SHOW USERS ROLES
```

Create a security policy over a property

Required information when creating a new security policy over a property (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The intended name for the new policy
- **{property_name}** - The property that's being assigned a new policy
- **{filter clause}** - The filtered parameter which defines the new policy

```
CREATE POLICY `{policy_name}` ON PROPERTY `{property_name}` WHERE {filter clause}
```

Create a security policy over a concept

Required information when creating a new security policy over a concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The intended name for the new policy
- **{concept_name}** - The concept that's being assigned a new policy
- **{filter clause}** - The filtered parameter which defines the new policy

```
CREATE POLICY `{policy_name}` ON CONCEPT `{concept_name}` WHERE {filter clause}
```

Create a security policy over a mapping

Required information when creating a new security policy over a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The intended name for the new policy
- **{mapping_name}** - The mapping that's being assigned a new policy
- **{filter clause}** - The filtered parameter which defines the new policy

```
CREATE POLICY `{policy_name}` ON MAPPING `{mapping_name}` WHERE {filter clause}
```

Create a security policy over a view

Required information when creating a new security policy over a view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The intended name for the new policy
- **{view_name}** - The view that's being assigned a new policy
- **{filter clause}** - The filtered parameter which defines the new policy

```
CREATE POLICY `{policy_name}` ON VIEW `{view_name}` WHERE {filter clause}
```

Removing a security policy

Removes the security policy from the knowledge graph.

Required information for removing a security policy (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The name of the policy being removed

```
DROP POLICY `{policy_name}`
```

Granting a security policy to a user

Required information for granting a security policy to a user (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The name of the policy being granted
- **{user_name}** - The user name being granted the policy

```
GRANT POLICY `{policy_name}` TO USER `{user_name}`
```

Revoking a security policy from a user

Required information for revoking a security policy from a user (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The name of the policy being revoked
- **{user_name}** - The user name being revoked from the policy

```
REVOKE POLICY `{policy_name}` FROM USER `{user_name}`
```

Granting a security policy to a role

Required information for granting a security policy to a role (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The name of the policy being granted
- **{role_name}** - The role name being granted the policy

```
GRANT POLICY `{policy_name}` TO ROLE `{role_name}`
```

Revoking a security policy from a role

Required information for revoking a security policy from a role (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{policy_name}** - The name of the policy being revoked
- **{role_name}** - The role name being revoked from the policy

```
REVOKE POLICY `{policy_name}` FROM ROLE `{role_name}`
```

Show the policies in the Knowledge Graph

Returns a list of policies that exist in the knowledge graph.

```
SHOW POLICIES
```

Show the policies assigned to users and roles

Returns a list showing which users and roles have which policies assigned to them.

```
SHOW ASSIGNED POLICIES
```

Query the system table of assigned policies

Returns a list from the system table showing which users and roles have which policies assigned to them.

```
SELECT * FROM timbr.SYS_ASSIGNED_POLICIES
```

Query the system table of existing policies

Returns a list from the system table showing the existing policies.

```
SELECT * FROM timbr.SYS_POLICIES
```

Get Timbr access token for current user

Returns the Timbr access token value for the current user.

```
SHOW TOKEN
```

Get Timbr access token for another user by username

Returns the Timbr access token value for a specific user.

Required information for retrieving the Timbr access token value for a user (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{user_name}** - The user name associated with the token value

```
SHOW TOKEN FOR {user_name}
```

Ontologies and Timbr Server information

This section is about getting metrics from ontologies and the Timbr server.

Show connections to Timbr

Shows a log of connections to Timbr server instances

```
SHOW CONNECTIONS
```

Show instance usage of Timbr server

Shows information regarding instances of the Timbr server

```
SHOW INSTANCES
```

Show the current license information

Returns information regarding the current Timbr license

```
SHOW LICENSE
```

Show the current running queries

Returns a list of the current running queries

```
SHOW RUNNING QUERIES
```

Show a list of available ontologies

Returns a list of the available ontologies for the current user.

```
SHOW ONTOLOGIES
```

Show the history of changes to ontologies

Returns a list of all the changes performed on all ontologies the user has access to.

```
SHOW ONTOLOGY HISTORY
```

Show the metadata operations of ontologies

Returns a list of all the metadata operations performed on all ontologies the user has access to.

```
SHOW OPERATIONS
```

Show the query history of ontologies

Returns a list of all the queries performed on all ontologies.

```
SHOW QUERY HISTORY
```

Ontology

This section is about running operations on ontologies (creating, modifying, duplicating, removing, backup and restore, etc)

Create a new ontology with a description

Creates a new ontology with a description associated to it.

Required information for creating an ontology with description (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the new ontology
- **{description}** - The new description associated with the ontology. This key can be ommitted to create an ontology without a description

```
CREATE ONTOLOGY `{ontology}` DESCRIPTION = '{description}'
```

Change the description of an ontology

Changes the description value of an ontology

Required information for changing the description of an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the ontology of which to change the description
- **{description}** - The new description associated with the ontology

```
"ALTER ONTOLOGY `{ontology}` SET DESCRIPTION = '{description}'
```

Show ontology configurations

Shows a list of all configurations for a specific ontology

```
SHOW CONFIG
```

Show ontology tags

Shows a list of all the tags associated with the resources (concepts, properties, ontology views) of a specific ontology

```
SHOW TAGS
```

Change the configuraiton paramenters for an ontology

Change the configuration value for a specific configuration in an ontology. **The list of available configuration keys are**

- `always_quote_identifier` - default value - `true` - Will always show the quote identifiers in SQL statements. Must be `true` or `false`
- `default_transitivity` - default value - `3` - The default relationship transitive value (unless explicitly specified)
- `entity_label_cast_type` - default value - `VARCHAR` - The CAST operation for entity label. Must be `VARCHAR` or `NVARCHAR`
- `graph_algorithm_output_schema` - default value - `graph_algorithms_output` - The default schema in which the output of graph algorithms presists
- `graph_depth` - default value - `1` - The number of hops in relationships to be returned from the dtimbr schema when querying for the metadata of a concept
- `graph_traversal_columns` - default value - `true` - Display the dtimbr schema relationships columns in metadata requests of a concept. Must be `true` or `false`
- `target_schema_cache` - default value - `48` - The target database cache refresh interval of a schema

Required information for changing the configuration of an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{config_value}** - The value of the config to be set.
- **{config_key_name}** - The config key name (must be one from the list above above)

```
UPDATE CONFIG SET value='{config_value}' WHERE name='{config_key_name}'
```

Backup a specific ontology

Backup the current state of an ontology. After creating the backup, a **version id** will be automatically created for the ontology

Required information for creating a backup of an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the ontology you want to backup
- **{comment}** - A description associated to the created backup, this field can be left empty.

```
BACKUP ONTOLOGY `{ontology}` '{comment}'
```

Shows a list of backups for ontologies

Shows a list of created backups of ontologies

```
SHOW ONTOLOGY BACKUP
```

Restore a specific ontology by version id

Restore a backup of an ontology that was created in Timbr.

Required information for restoring a backup (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the ontology you want to restore
- **{version}** - The version id of the backup you want to restore

```
RESTORE ONTOLOGY `{ontology}` '{version}'
```

Show the export of an ontology in SQL

Returns a dataset of the Timbr SQL representation of an ontology including the mappings and views,

```
EXPORT ONTOLOGY
```

Show the export of an ontology in OWL

Returns a dataset of the OWL representation of an ontology

```
EXPORT ONTOLOGY OWL
```

Refreshing the cache of an ontology

Refresh the cache of an ontology with respect to metrics regarding concepts, properties, relationships, queries, users, and roles.

```
REFRESH ONTOLOGY
```

Validating the integrity of an ontology

Validates the integrity of an ontology (concepts, relationships, properties, mappings, views). Will return **No response** if the validation is successful and no errors are found. If an error is found, results will return with the validation error message.

```
VALIDATE ONTOLOGY
```

Duplicate an ontology in the same environment

Duplicating an ontology in the same environment is like *exporting to* or *importing from* an existing ontology. In other words, both the **Export** and **Import** commands are identical in function and they only depend on the direction of the export/import

Duplicating an ontology using the Export

Required information for duplicating an ontology using the export command (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{source_ontology}** - The name of the ontology you want to duplicate from
- **{overwrite}** - Whether or not should overwrite an existing ontology if one with the same name already exists. In order to overwrite you should write `OVERWRITE`, otherwise you can leave it empty
- **{new_ontology}** - The name of the ontology to be created or overwritten. If the ontology doesn't exist a new ontology will be created. If it already exists and the `{overwrite}` flag is not set, the duplicate will fail unless the `{overwrite}` flag is set.

```
EXPORT ONTOLOGY `{source_ontology}` {overwrite} TO `{new_ontology}`
```

Duplicating an ontology using the Import

Required information for duplicating an ontology using the import command(the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{new_ontology}** - The name of the ontology to be created or overwritten. If the ontology doesn't exist a new ontology will be created. If it already exists and the `{overwrite}` flag is not set, the duplicate will fail unless the `{overwrite}` flag is set.
- **{overwrite}** - Whether or not should overwrite an existing ontology if one with the same name already exists. In order to overwrite you should write `OVERWRITE`, otherwise you can leave it empty
- **{source_ontology}** - The name of the ontology you want to duplicate from

```
IMPORT ONTOLOGY `{new_ontology}` {overwrite} FROM `{source_ontology}`
```

Export / Import ontologies between different environments

The export or import between different environment depends on which instance of Timbr the command is executed in. In other words, both the **Export** and **Import** commands are identical in function and they only depend on the direction of the export/import

Exporting an ontology to a remote / external environment

Required information for exporting an ontology to an external instance (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{source_ontology_name}** - The name of the ontology you want to export as its presented in the current Timbr instance
- **{overwrite}** - Whether or not should overwrite an existing ontology if one with the same name already exists. In order to overwrite you should write `OVERWRITE`, otherwise you can leave it empty
- **{hostname}** - The host name of the remote Timbr environment to which you want to export to. This is the host name of the `timbr-server`
- **{port}** - The port of the remote Timbr environment to which you want to export to. This is the port of the `timbr-server`
- **{new_ontology_name}** - The name of the ontology to be exported to. If it doesn't exist a new ontology will be created. If it already exists and the `{overwrite}` flag is not set, the export will fail unless the `{overwrite}` flag is set.
- **{username}** - The user name in the remote environment, you can use the user `token` if you want to connect using a timbr access token.

- **{password}** - The password to use with the user name in the remote environment. If you use a `token`, use the **token value** as the password.

```
EXPORT ONTOLOGY {source_ontology_name} {overwrite} TO EXTERNAL URL 'jdbc:hive2://{hostname}:
{port}/{new_ontology_name};transportMode=http' USER '{username}' PASSWORD '{password}'
```

Importing an ontology from a remote / external environment

Required information for importing an ontology from an external instance (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{new_ontology}** - The name of the ontology you want to import into. If it does not exist a new ontology will be created. If it already exists and the `{overwrite}` flag is not set, the impot will fail unless the `{overwrite}` flag is set.
- **{overwrite}** - Whether or not should overwrite an existing ontology if one with the same name already exists. In order to overwrite you should write `OVERWRITE`, otherwise you can leave it empty
- **{hostname}** - The host name of the remote Timbr environment from which you want to import from. This is the host name of the `timbr-server`
- **{port}** - The port of the remote Timbr environment to which you want to export to. This is the port of the `timbr-server`
- **{source_ontology_name}** - The name of the ontology from which to import.
- **{username}** - The user name in the remote environment, you can use the user `token` if you want to connect using a timbr access token.
- **{password}** - The password to use with the user name in the remote environment. If you use a `token`, use the **token value** as the password.

```
IMPORT ONTOLOGY {new_ontology} {overwrite} FROM EXTERNAL URL 'jdbc:hive2://{hostname}:
{port}/{source_ontology_name};transportMode=http' USER '{username}' PASSWORD '{password}'
```

Adding an existing datasource to an ontology

Adds an existing datasource to a specific ontology.

Required information for adding a datasource to an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the ontology as its presented in Timbr
- **{datasource}** - The name of the datasource as its presented in Timbr

```
ALTER ONTOLOGY {ontology} ADD DATASOURCE `{datasource}`
```

Removing an existing datasource from an ontology

Removes an existing datasource from an ontology. This will **not** remove the datasource from Timbr.

Required information for removing a datasource from an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology}** - The name of the ontology as its presented in Timbr
- **{datasource}** - The name of the datasource as its presented in Timbr

```
ALTER ONTOLOGY `{ontology}` DROP DATASOURCE `{datasource}`
```

Set active datasource for an ontology

If your ontology has more than one datasources associated to it, and you want to query either

1. Concept consisting of a mapping using that datasource
2. Table in the datasource

Required information for setting an active datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{datasource}** - The name of the datasource as its presented in Timbr

You should set the active datasource for that ontology by running the command

```
SET active_datasource = '{datasource}'
```

Removing an ontology

Removes an ontology from timbr

Required information for removing an ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology_name}** - The name of the ontology as its presented in Timbr

```
DROP ONTOLOGY `{ontology_name}`
```

Datasource

This section is about operating datasources in Timbr. Datasources are used as the execution engine for all the queries run on an ontology. Timbr connects to a datasource using a JDBC driver, and can theoretically connect to *any* datasource that has a JDBC driver. The types of datasources that are currently supported out-of-the-box are:

Supported Datasources

- `Amazon Athena` with driver: `com.simba.athena.jdbc.Driver`
- `Amazon Redshift` with driver: `com.amazon.redshift.jdbc.Driver`
- `Apache Drill` with driver: `org.apache.drill.jdbc.Driver`
- `Apache Hive` with driver: `org.apache.hive.jdbc.HiveDriver`
- `Apache Spark` with driver: `org.apache.hive.jdbc.HiveDriver`
- `Azure Blob Storage` with JSON connection template
- `Azure Datalake Storage` with driver: `org.apache.drill.jdbc.Driver`
- `Clickhouse` with driver: `com.clickhouse.jdbc.ClickHouseDriver`
- `Databricks` with driver: `com.simba.spark.jdbc.Driver`
- `Google BigQuery` with driver: `com.simba.googlebigquery.jdbc42.Driver`
- `Google Cloud Storage` with JSON connection template
- `Microsoft SQL Server` with driver: `com.microsoft.sqlserver.jdbc.SQLServerDriver`
- `MySQL` with driver: `com.mysql.jdbc.Driver`
- `Oracle` with driver: `oracle.jdbc.driver.OracleDriver`
- `Oracle 11` with driver: `oracle.jdbc.driver.OracleDriver`
- `PostgreSQL` with driver: `org.postgresql.Driver`
- `Presto` with driver: `com.facebook.presto.jdbc.PrestoDriver`
- `S3` with driver: `org.apache.drill.jdbc.Driver`
- `SAP Hana` with driver: `com.sap.db.jdbc.Driver`
- `Snowflake` with driver: `net.snowflake.client.jdbc.SnowflakeDriver`
- `Trino` with driver: `io.trino.jdbc.TrinoDriver`
- `Vertica Analytics Platform` with driver: `com.vertica.jdbc.Driver`

Create a datasource

Required information when querying to create a new datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{db_name}** - The name of the datasource as it will be presented in Timbr
- **{target_url}** - The JDBC URL of the datasource, usually including a protocol, hostname, port and sometimes database/catalog
- **{datasource_type}** - The type of the datasource, as its presented in the list of supported datasources above
- **{user}** - The username credentials for the datasource
- **{password}** - The password credentials for the datasource

To create a datasource you can run the command:

```
CREATE DATASOURCE `{db_name}` DESCRIPTION '{description}' OPTIONS(URL '{target_url}', DRIVER '{datasource_driver}',
TYPE '{datasource_type}', USER '{user}', PASSWORD '{password}') WITH VIRTUALIZATION
```

Optional keys are:

- **DESCRIPTION '{description}'** - Provide a string with description for the datasource
- **WITH VIRTUALIZATION** - Works only in the cases where you have the **Timbr Virtualization Service** or if the datasource type is **Apache Spark** or **Databricks**. This setting indicates whether the datasource serves as a virtualization engine. If you are using the **Timbr Virtualization Service** then your datasource type is **Apache Spark**

Create a datasource based on a json string

Some datasources like **S3**, **Azure Datalake Storage**, **Google Cloud Storage**, **Azure Blob Storage** require a JSON connection configuration template.

Required information when querying to create a new datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{db_name}** - The name of the datasource as it will be presented in Timbr
- **{json}** - A stringify version of the template JSON used for the connection
- **{datasource_type}** - The type of the datasource, as its presented in the list of supported datasources above
- **{json_secret}** - The password credentials for the datasource

If you want to create such datasource requiring a JSON string, you can do it programmatically by running

```
CREATE DATASOURCE `{db_name}` DESCRIPTION '{description}' OPTIONS(JSON '{json}', TYPE '{datasource_type}', PASSWORD
'{json_secret}') WITH VIRTUALIZATION
```

Optional keys are:

- **DESCRIPTION '{description}'** - Provide a string with description for the datasource
- **WITH VIRTUALIZATION** - Works only in the cases where you have the **Timbr Virtualization Service** or if the datasource type is **Apache Spark** or **Databricks**. This setting indicates whether the datasource serves as a virtualization engine. If you are using the **Timbr Virtualization Service** then your datasource type is **Apache Spark**

Show all datasources

Shows all the datasources associated with an ontology

```
SHOW DATASOURCES
```

Change the description of a datasource

Required information when changing the description of a datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{datasource}** - The name of the datasource as it will be presented in Timbr
- **{description}** - The description given to the datasource

When connected to a current ontology that has the datasource set as active, you can run the command

```
ALTER DATASOURCE `{datasource}` SET DESCRIPTION = '{description}'
```

Setting the virtualization of a datasource on / off

If a datasource was created with a **WITH VIRTUALIZATION** option, you can set it `ON` or `OFF`

Required information when setting the virutalization of a datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{datasource}** - The name of the datasource as its presented in Timbr
- **{value}** - `1` will set it **ON** and `0` will set it **OFF**

```
ALTER DATASOURCE `{datasource}` SET VIRTUALIZATION = {value}
```

Removing a datasource

Removes a datasource from Timbr.

> **IMPORTANT**
>
> In order to remove a datasource from timbr you need to make sure it is not associated/connected to any ontology. If the datasource is associated/connected to an ontology, you should first remove it from that ontology and then remove it from Timbr.

Required information for removing a datasource (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{datasource}** - The name of the datasource as its presented in Timbr

```
DROP DATASOURCE `{datasource}`
```

Concepts, Mappings, and Views

This section is about Timbr SQL Data Definition Language (DDL) queries to create concepts, mappings, and views in an ontology

---

Create or replace concept

Create or replace a concept in an ontology.

Required information for create or replace a concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to create
- **{property_name1}** - an example of a property name to be associated with the concept
- **{property_type1}** - The data type associated with the property
- **{property_name2}** - an example of a property name to be associated with the concept
- **{property_type2}** - The data type associated with the property
- **{inherits_from_concept}** - The parent concept of the concept you want to create
- **{concept_description}** - Optional. A description of the concept

```
CREATE OR REPLACE CONCEPT `{concept_name}` (`{property_name1}` {property_type1}, `{property_name2}` {property_type2},
    PRIMARY KEY(`{property_name1}`),
    LABEL(`{property_name2}`),
) INHERITS (`{inherits_from_concept}`)
DESCRIPTION '{concept_description}';
```

Create / Update a tag of a concept

Create or update a tag of a concept

Required information for create or update a tag of a concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to add the tag to
- **{tag_name}** - The tag key (name) to be added or updated to the concept
- **{tag_value}** - The tag value to be associated with the tag key of the concept

```
ALTER CONCEPT `{concept_name}` UPDATE TAG `{tag_name}`='{tag_value}';
```

Remove a tag from a concept

Remove a tag of a concept

Required information for removing a tag of a concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to remove the tag from
- **{tag_name}** - The tag key (name) to be removed from the concept

```
ALTER CONCEPT `{concept_name}` DROP TAG `{tag_name}`;
```

Create / Update a tag of a property

Create or update a tag of a property

Required information for create or update a tag of a property (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{property_name}** - The name of the property you want to add the tag to
- **{tag_name}** - The tag key (name) to be added or updated to the property
- **{tag_value}** - The tag value to be associated with the tag key of the property

```
ALTER PROPERTY `{property_name}` UPDATE TAG `{tag_name}`='{tag_value}';
```

Remove a tag from a property

Remove a tag of a property

Required information for removing a tag of a property (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{property_name}** - The name of the property you want to remove the tag from
- **{tag_name}** - The tag key (name) to be removed from the property

```
ALTER PROPERTY `{property_name}` DROP TAG `{tag_name}`;
```

Create or replace a calculated property

Create and edit calculated properties for a concept

Required information for creating or replacing a calculated property (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{property_name}** - The name of the property you want to create or replace
- **{property_type}** - The type of the property you want to create or replace
- **{description_text}** - Optional. The description of the property
- **{calculation_or_transformation}** - The calculation or transformation you want to apply on the property

Create a calculated property

```
CREATE PROPERTY `{property_name}` {property_type} DESCRIPTION '{description_text}' AS SELECT
{calculation_or_transformation};
```

Example:

```
CREATE PROPERTY `usd_revenue` decimal DESCRIPTION 'Euro Revenue calculated in USD currency' AS SELECT eu_revenue *
1.07;
```

Replace or edit a calculated property

```
CREATE OR REPLACE PROPERTY `{property_name}` {property_type} DESCRIPTION '{description_text}' AS SELECT
{calculation_or_transformation};
```

Example:

```
CREATE OR REPLACE PROPERTY `usd_revenue` decimal DESCRIPTION 'Euro Revenue calculated in USD currency as of Jan23' AS
SELECT eu_revenue * 1.07;
```

> **CREATING A CALCULATED PROPERTY OVER AN EXISTING PROPERTY**
>
> Users can create **calculated properties** over **existing properties** by simply following the SQL template above of - `CREATE OR REPLACE PROPERTY...`

Create or replace concept relationships

Create or replace a concept with relationship

Required information for create or replace a concept with a relationship (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to create
- **{property_name1}** - an example of a property name to be associated with the concept
- **{property_type1}** - The data type associated with the property
- **{property_name2}** - an example of a property name to be associated with the concept and used in the relationship
- **{property_type2}** - The data type associated with the property
- **{relationship_name}** - The relationship name you want to create as it will appear in this concept queries and metadata
- **{target_concept}** - The target concept to which the relationships point to
- **{target_concept_property}** - The property in the target concept which is related to the property in the source concept
- **{inverse_relationship_name}** - The relationship name as it will appear from the target concept context.
- **{inherits_from_concept}** - The parent concept of the concept you want to create
- **{concept_description}** - Optional. A description of the concept

```
CREATE OR REPLACE CONCEPT `{concept_name}` (`{property_name}` {property_type}, `{property_name2}` {property_type2},
    PRIMARY KEY(`{property_name}`),
    LABEL(`{property_name2}`),
    CONSTRAINT `{relationship_name}`
        FOREIGN KEY (`{property_name2}`)
        REFERENCES `{target_concept}` (`{target_concept_property}`)
        INVERSEOF `{inverse_relationship_name}`
) INHERITS (`{inherits_from_concept}`)
DESCRIPTION '{concept_description}';
```

Create or replace concept transitive relationships

Create or replace a concept with a transitive relationship.

Required information for create or replace a concept with a transitive relationship (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to create
- **{property_name1}** - an example of a property name to be associated with the concept
- **{property_type1}** - The data type associated with the property
- **{property_name2}** - an example of a property name to be associated with the concept and used in the relationship
- **{property_type2}** - The data type associated with the property

- **{relationship_name}** - The relationship name you want to create as it will appear in this concept queries and metadata
- **{target_concept}** - The target concept to which the relationships point to
- **{target_concept_property}** - The property in the target concept which is related to the property in the source concept
- **{inverse_relationship_name}** - The relationship name as it will appear from the target concept context.
- **{transitive_depth}** - The maximum number of transitive hops for the relationship
- **{inherits_from_concept}** - The parent concept of the concept you want to create
- **{concept_description}** - Optional. A description of the concept

```
CREATE OR REPLACE CONCEPT `{concept_name}` (`{property_name}` {property_type}, `{property_name2}` {property_type2},
    PRIMARY KEY(`{property_name}`),
    LABEL(`{property_name2}`),
    CONSTRAINT `{relationship_name}`
        FOREIGN KEY (`{property_name2}`)
        REFERENCES `{target_concept}` (`{target_concept_property}`)
        INVERSEOF `{inverse_relationship_name}`
        TRANSITIVE ({transitive_depth})
) INHERITS (`{inherits_from_concept}`)
DESCRIPTION '{concept_description}';
```

Create or replace logic concept

Create or replace a logic concept (based on a filter)

Required information for create or replace a logic concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to create
- **{inherits_from_concept}** - The parent concept of the concept you want to create
- **{concept_to_query}** - The parent concept to query for the filter
- **{sql_query_filter}** - The query filter to be performed on which the logic concept is based on

```
CREATE OR REPLACE CONCEPT `{concept_name}`
    INHERITS (`{inherits_from_concept}`)
    FROM `timbr`.`{concept_to_query}`
    WHERE {sql_query_filter};
```

Create or replace mapping

Create or replace a concept mapping

Required information for create or replace a concept mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to create
- **{target_concept}** - The name of the concept associated with the mapping
- **{table_column}** - The table column to be mapped (can also be a calculated column)
- **{property_of_concept}** - The property of the column associated with the table column (or calculated column)
- **{source_database}** - The source database from which the mapping will occur
- **{source_table}** - The source table from which the mapping will occur

```
CREATE OR REPLACE MAPPING `{mapping_name}` INTO (`{target_concept}`)
    AS
        SELECT `{table_column}` AS `{property_of_concept}`
        FROM {source_database}.{source_table};
```

Create or replace many-to-many mapping

Create or replace a many-to-many mapping to a concept. Many to many mappings are usually created between two concepts, or a concept and itself, where the mapping goes through an EAV table (Entity-Attribute-Value table) which stores the information about the relationship between two tables.

Required information for create or replace a many-to-many mapping to a concept (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to create
- **{relationship_name}** - The many-to-many relationship name you want to create as it will appear in this concept queries and metadata
- **{eav_column1}** - The table column from the EAV table to be mapped to the source concept
- **{source_concept}** - The name of the source concept associated with the many-to-many relationship
- **{source_concept_property}** - The property from the source concept to be related to the property of the target concept
- **{inverse_relationship_name}** - The many-to-many relationship name as it will appear from the target concept context
- **{eav_column2}** - The table column from the EAV table to be mapped to the target concept
- **{target_concept}** - The target concept to which the many-to-many relationship points to
- **{target_concept_property}** - The property in the target concept which is related to the property in the source concept
- **{eav_database}** - The source database from which the mapping will occur
- **{eav_table}** - The source table from which the mapping will occur

```
CREATE OR REPLACE MAPPING `{mapping_name}` (
    CONSTRAINT `{relationship_name}`
        FOREIGN KEY (`{eav_column1}`)
        REFERENCES `{source_concept}` (`{source_concept_property}`),
    CONSTRAINT `{inverse_relationship_name}`
        FOREIGN KEY (`{eav_column2}`)
        REFERENCES `{target_concept}` (`{target_concept_property}`))
    AS
        SELECT `{eav_column1}` AS `{source_concept_property}`,
            `{eav_column2}` AS `{target_concept_property}`
        FROM `{eav_database}`.`{eav_table}`;
```

Create or replace mv mapping

Create or replace a multi-value mapping

Required information for create or replace a multi-value mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to create
- **{added_property}** - The property to be added from the multi-value table
- **{added_property_type}** - The data type of the property to be added from the multi-value table
- **{mv_table_column}** - The related column of the table to be mapped from the mediary property of the concept
- **{target_concept}** - The name of the concept associated with the mapping
- **{mediary_property}** - The related property of the concept to be mapped to the mediary table column
- **{added_column}** - The table column (can also be a calculated column) to be mapped to the added property
- **{mv_database}** - The source database from which the mapping will occur
- **{mv_table}** - The source table from which the mapping will occur

```
CREATE OR REPLACE MAPPING `{mapping_name}` (`{added_property}` `{added_property_type}`,
    FOREIGN KEY (`{mv_table_column}`)
    REFERENCES `{target_concept}`(`{mediary_property}`))
    AS
        SELECT `{mv_table_column}` AS `{mediary_property}`,
            `{added_column}` AS `{added_property}`
        FROM `{mv_database}`.`{mv_table}`;
```

Create an ontology view

Create an ontology view which can be based on the ontology or the underlying datasources.

Required information for create an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the ontology view you want to create

- **{sql_for_view}** - The SQL statement for the view you want to create

```
CREATE ONTOLOGY VIEW `{view_name}` AS {sql_for_view};
```

> **CREATING A VIEW WITH TAGS**
>
> You can also specify creating a view with tags by default in the create statement.

- **{tag_name}** - The name of the tag to be associated with the view
- **{tag_value}** - The value for the tag associated with the view

```
CREATE ONTOLOGY VIEW `{view_name}` WITH TAGS (`{tag_name}` = `{tag_value}`) AS {sql_for_view};
```

> **CREATE OR REPLACE ONTOLOGY VIEW WITH OR WITHOUT PREVIOUS TAGS**
>
> Instead of running `CREATE ONTOLOGY VIEW...` command, some users prefer running `CREATE OR REPLACE ONTOLOGY VIEW` overwriting the previous ontology view. In those cases, a user may choose to **maintain** the previousely set tags of the ontology view or **remove** them
>
> - If you run a `CREATE OR REPLACE` command omitting the `WITH TAGS (...)` parts, then **the previously set tags will be maintained**
> - If you wish to remove the tags as well, then you can add to the `CREATE OR REPLACE STATEMENT` the part of `WITH TAGS()` and then the previousely set tags will also be removed.

Change ontology view description

Changes the description of an ontology view

Required information for changing the description of an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the ontology view you want to associate the description with
- **{view_description}** - A string representing the description associated with the ontology view

```
ALTER ONTOLOGY VIEW `{view_name}` SET DESCRIPTION='{view_description}'
```

Create or update a tag of an ontology view

Create or update a tag of an ontology view

Required information for create or update a tag of an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology_view_name}** - The name of the ontology view you want to add the tag to
- **{tag_name}** - The tag key (name) to be added or updated to the ontology view
- **{tag_value}** - The tag value to be associated with the tag key of the ontology view

```
ALTER ONTOLOGY VIEW `{ontology_view_name}` UPDATE TAG `{tag_name}`='{tag_value}';
```

Remove a tag from an ontology view

Remove a tag of an ontology view

Required information for removing a tag of an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{ontology_view_name}** - The name of the ontology view you want to remove the tag from
- **{tag_name}** - The tag key (name) to be removed from the ontology view

```
ALTER ONTOLOGY VIEW `{ontology_view_name}` DROP TAG `{tag_name}`;
```

Remove a concept

Removes a concept from an ontology.

Required information for removing a concept from the ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{concept_name}** - The name of the concept you want to remove

```
DROP CONCEPT `{concept_name}`;
```

Remove a mapping

Removes a mapping from an ontology.

Required information for removing a mapping from the ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to remove

```
DROP MAPPING `{mapping_name}`;
```

Remove an ontology view

Removes an ontology view from an ontology.

Required information for removing an ontology view from the ontology (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the mapping you want to remove

```
DROP ONTOLOGY VIEW  `{view_name}`;
```

Cache and Jobs

This section is with regards to caching resources which can be **mappings** or **ontology views**, and creating jobs to schedule the cachings.

Cache mappings

Create a cache for a mapping

Required information for creating a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to cache
- **{cache_schema}** - The schema name on which the cached mapping should be stored

```
CACHE MAPPING {mapping_name} OPTIONS (`schema`='{cache_schema}')
```

Cache a mapping with partition

Create a cache for a mapping with partition on a property

Required information for creating a cache for a mapping with a partition (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource which stores and runs the cached mapping
- **{cache_schema}** - The schema name on which the cached mapping should be stored
- **{partition_property}** - The name of the property by which to partition the cache

```
CACHE MAPPING {mapping_name} USING {virtualization_datasource}
    OPTIONS (`schema`='{cache_schema}',  partition='{partition_property}')
```

Cache a mapping with partition and split

Create a cache for a mapping with partition on a property and a split to split the partitioned data

Required information for creating a cache for a mapping with a partition and a split (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource which stores and runs the cached mapping
- **{cache_schema}** - The schema name on which the cached mapping should be stored
- **{partition_property}** - The name of the property by which to partition the cache
- **{split_value}** - The split value depends on the property, if its a `timestamp`, `date` or `datetime` then the value can be `daily`, `monthly`, or `yearly`. The split value can also be `partition` to split by the **{partition_property}**.

```
CACHE MAPPING {mapping_name} USING {virtualization_datasource}
  OPTIONS (`schema`='{cache_schema}', partition='{partition_property}', split='{split_value}')
```

Cache a mapping to object storage

Create a cache for a mapping to object storage. Timbr currently supports

- S3
- Azure Datalake Storage
- Google Cloud Storage

Required information for creating a cache for a mapping to object storage (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource on which to run the cache mapping job
- **{target_datasource_id}** - The name of the object store in which to store the cached mapping
- **{cache_schema}** - The schema name on which the cached mapping should be stored
- **{partition_property}** - The name of the property by which to partition the cache

```
CACHE MAPPING {mapping_name} USING {virtualization_datasource}
  INTO {target_datasource_id}
  OPTIONS (`schema`='{cache_schema}')
```

Cache an ontology view

Create a cache for an ontology view

Required information for creating a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the view you want to materialize
- **{cache_schema}** - The schema name on which the materialized view should be stored

```
CACHE VIEW {view_name} OPTIONS (`schema`='{cache_schema}')
```

Cache an ontology view with partition

Create a cache for an ontology view with partition on a property

Required information for creating a cache for an ontology view with a partition (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the view you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource which stores and runs the cached view
- **{cache_schema}** - The schema name on which the cached view should be stored
- **{partition_property}** - The name of the property by which to partition the cache

```
CACHE VIEW {view_name} USING {virtualization_datasource}
    OPTIONS (`schema`='{cache_schema}', partition='{partition_property}')
```

Cache an ontology view with partition and split

Create a cache for an ontology view with partition on a property and a split to split the partitioned data

Required information for creating a cache for an ontology view with a partition and a split (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the view you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource which stores and runs the cached view
- **{cache_schema}** - The schema name on which the cached view should be stored
- **{partition_property}** - The name of the property by which to partition the cache
- **{split_value}** - The split value depends on the property, if its a `timestamp`, `date` or `datetime` then the value can be `daily`, `monthly`, or `yearly`. The split value can also be `partition` to split by the **{partition_property}**.

```
CACHE VIEW {view_name} USING {virtualization_datasource}
  OPTIONS (`schema`='{cache_schema}', partition='{partition_property}', split='{split_value}')
```

Cache an ontology view to object storage

Create a cache for a view to object storage.

> **SUPPORTED OBJECT STORAGE DATA SOURCES**
>
> Timbr currently supports:
>
> - S3
> - Azure Datalake Storage
> - Google Cloud Storage

Required information for creating a cache for a view to object storage (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the view you want to cache
- **{virtualization_datasource}** - The name of the virtualization datasource on which to run the cache view job
- **{target_datasource_id}** - The name of the object store in which to store the cached view
- **{cache_schema}** - The schema name on which the cached view should be stored
- **{partition_property}** - The name of the property by which to partition the cache

```
CACHE VIEW {view_name} USING {virtualization_datasource}
  INTO {target_datasource_id}
  OPTIONS (`schema`='{cache_schema}')
```

Remove a cached mapping (uncache)

Removes a cache for a mapping.

Required information to remove a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the mapping you want to remove the cache from

```
UNCACHE MAPPING {mapping_name}
```

Remove a cached ontology view (uncache)

Removes a cache for an ontology view.

Required information to remove a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the view you want to remove the cache from

```
UNCACHE VIEW {view_name}
```

Full refresh a mapping cache

Refresh entirely the cache of a mapping

Required information to fully refresh a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the cache mapping you want to fully refresh

```
REFRESH CACHE MAPPING {mapping_name}
```

Incrementally refresh a mapping cache

Refresh only a partition of the cache for a mapping based on the partition property

Required information to incrementally refresh a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the cache mapping you want to incrementally refresh
- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{days_value}** - An integer representing the amount of days to subtract from the current date in order to apply the cache.

```
REFRESH CACHE MAPPING {mapping_name}
    WHERE {partition_property} > CURRENT_DATE - {days_value}
```

Custom partition refresh for a mapping cache

Refresh a particular part of the cache based on the partition property

Required information for custom refresh of a cache of a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{mapping_name}** - The name of the cache mapping you want to customly refresh
- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{sql_operator}** - SQL operators used in the `WHERE` clause of an SQL statement.
- **{comparator_value}** - The value to compare against the partition property

> **SUPPORTED SQL OPERATORS**
>
> The supported SQL operator are
>
> 1. `=` - Equals
> 2. `!=` or `<>` - Not equals
> 3. `IN(..)` - Includes
> 4. `NOT IN(..)` - Not includes
> 5. `LIKE` (can be used with wildcards combinations like `%` and `[]`) - Contains
> 6. `NOT LIKE` (can be used with wildcards combinations like `%` and `[]`) - Not contains
> 7. `>` - Greater than
> 8. `>=` - Greater than or equal
> 9. `<` - Less than
> 10. `<=` - Less than or equal
> 11. `IS NULL` - Null value
> 12. `IS NOT NULL` - Is not null value

```
REFRESH CACHE MAPPING {mapping_name}
    WHERE {partition_property} {sql_operator} {comparator_value}
```

Full refresh an ontology view cache

Refresh entirely the cache of an ontology view

Required information to fully refresh a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the cache for the ontology view you want to fully refresh

```
REFRESH CACHE VIEW {view_name}
```

Incrementally refresh an ontology view cache

Refresh only a partition of the cache for an ontology view based on the partition property

Required information to incrementally refresh a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the cache for the ontology view you want to incrementally refresh
- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{days_value}** - An integer representing the amount of days to subtract from the current date in order to apply the cache.

```
REFRESH CACHE VIEW {view_name}
    WHERE {partition_property} > CURRENT_DATE - {days_value}
```

Custom partition refresh for an ontology view cache

Refresh a particular part of the cache based on the partition property

Required information for custom refresh of a cache of an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{view_name}** - The name of the cached ontology view you want to customly refresh
- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{sql_operator}** - SQL operators used in the `WHERE` clause of an SQL statement.
- **{comparator_value}** - The value to compare against the partition property

SUPPORTED SQL OPERATORS

The supported SQL operators are

1. `=` - Equals
2. `!=` or `<>` - Not equals
3. `IN(..)` - Includes
4. `NOT IN(..)` - Not includes
5. `LIKE` (can be used with wildcards combinations like `%` and `[]`) - Contains
6. `NOT LIKE` (can be used with wildcards combinations like `%` and `[]`) - Not contains
7. `>` - Greater than
8. `>=` - Greater than or equal
9. `<` - Less than
10. `<=` - Less than or equal
11. `IS NULL` - Null value
12. `IS NOT NULL` - Is not null value

```
REFRESH CACHE VIEW {view_name}
    WHERE {partition_property} {sql_operator} {comparator_value}
```

Create job to schedule a full cache refresh on a mapping

Creates a recurring job to do a full cache refresh on a mapping

Required information to create a job that fully refresh a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to be created
- **{refresh_value}** - The interval for which the job should be run.
- **{mapping_name}** - The name of the cache for the mapping you want to fully refresh

INTERVALS FOR REFRESH_VALUE

Can be one of the following values:

1. `none` - Job runs only once
2. `daily` - Job will run once a day
3. `monthly` - Job will run once a month
4. `yearly` - Job will run once a year
5. `cron_expression` - A cron expression is a string comprising five or six fields separated by white space that represents a set of times, to schedule when the job should be executed. Visit here for more information

```
CREATE OR REPLACE JOB {job_name}
    REFRESH '{refresh_value}'
        AS REFRESH CACHE MAPPING {mapping_name}
```

Create job to schedule an incremental cache refresh on a mapping

Creates a recurring job to do a full cache refresh on a mapping

Required information to create a job that fully refresh a cache for a mapping (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to be created
- **{refresh_value}** - The interval for which the job should be run.
- **{mapping_name}** - The name of the cache for the mapping you want to incrementally refresh
- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{days_value}** - An integer representing the amount of days to subtract from the current date in order to apply the cache.

INTERVALS FOR REFRESH_VALUE

Can be one of the following values:

1. `none` - Job runs only once
2. `daily` - Job will run once a day
3. `monthly` - Job will run once a month
4. `yearly` - Job will run once a year
5. `cron_expression` - A cron expression is a string comprising five or six fields separated by white space that represents a set of times, to schedule when the job should be executed. Visit here for more information

```
CREATE OR REPLACE JOB {job_name}
    REFRESH {refresh_value}
    AS REFRESH CACHE MAPPING {mapping_name}
        WHERE {partition_property} > CURRENT_DATE - {days_value}
```

Create job to schedule a full cache refresh on an ontology view

Creates a recurring job to do a full cache refresh on an ontology view

Required information to create a job that fully refresh a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to be created
- **{refresh_value}** - The interval for which the job should be run.
- **{view_name}** - The name of the cache for the ontology view you want to fully refresh

> **INTERVALS FOR REFRESH_VALUE**
>
> Can be one of the following values:
>
> 1. `none` - Job runs only once
> 2. `daily` - Job will run once a day
> 3. `monthly` - Job will run once a month
> 4. `yearly` - Job will run once a year
> 5. `cron_expression` - A cron expression is a string comprising five or six fields separated by white space that represents a set of times, to schedule when the job should be executed. Visit here for more information

```
CREATE OR REPLACE JOB {job_name}
    REFRESH '{refresh_value}'
        AS REFRESH CACHE VIEW {view_name}
```

Create job to schedule an incremental cache refresh on an ontology view

Creates a recurring job to do a full cache refresh on an ontology view

Required information to create a job that fully refresh a cache for an ontology view (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to be created
- **{refresh_value}** - The interval for which the job should be run.
- **{view_name}** - The name of the cache for the ontology view you want to incrementally refresh

- **{partition_property}** - The name of the property by which the cache is partitioned by
- **{days_value}** - An integer representing the amount of days to subtract from the current date in order to apply the cache.

> **INTERVALS FOR REFRESH_VALUE**
>
> Can be one of the following values:
>
> 1. `none` - Job runs only once
> 2. `daily` - Job will run once a day
> 3. `monthly` - Job will run once a month
> 4. `yearly` - Job will run once a year
> 5. `cron_expression` - A cron expression is a string comprising five or six fields separated by white space that represents a set of times, to schedule when the job should be executed. Visit here for more information

```
CREATE OR REPLACE JOB {job_name}
    REFRESH {refresh_value}
    AS REFRESH CACHE VIEW {view_name}
        WHERE {partition_property} > CURRENT_DATE - {days_value}
```

Run a job

Executes an existing job manually

Required information to run a job (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to run

```
RUN JOB {job_name}
```

Remove a job

Deletes an existing job

Required information to remove a job (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_name}** - The name of the job to remove

```
DROP JOB {job_name}
```

Stop a running job

Stops the execution of an existing running job

Required information to stop a running job (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{job_id}** - The id of the job to stop

```
KILL JOB {job_id}
```

Graph algorithms

The currently supported graph algorithms can be found in the `gtimbr` schema. They can be run on **CPU** or **GPU** (if supported by the infrastructure)

The currently supported graph algorithms are:

Centrality

- **betweenness centrality** - Compute the shortest-path betweenness centrality for nodes. More information
- **katz** - Compute the Katz centrality for the nodes of the graph G. More information

Classification

- **node classification** - Node classification by Harmonic function. More information
- **pagerank** - Link classification. Returns the PageRank of the nodes in the graph. More information

Community Detection

- **louvain** - Find the best partition of a graph using the Louvain Community Detection Algorithm. Louvain Community Detection Algorithm is a simple method to extract the community structure of a network. This is a heuristic method based on modularity optimization.More information

Components

- **strongly cc** - Generate nodes in strongly connected components of graph. More information
- **weakly cc** - Generate weakly connected components of graph. More information

Cores

- **core number** - Returns the core number for each vertex. More information

Link Prediction

- **common neighbor centrality** - Compute the Common Neighbor and Centrality based Parameterized Algorithm(CCPA) score of all node pairs in ebunch. More information
- **jaccard** - Compute the Jaccard coefficient of all node pairs in ebunch. More information
- **jaccard fuzzy** - - Compute the Jaccard coefficient of all node pairs matched by fuzzy matching in ebunch. More information

Timbr can support any graph algorithm that is currently supported by:

1. CPU based NetworkX or cuGraph

2. Graph based cuGraph

Query a graph algorithm

To query a graph algorithm, all that is needed is to query the `gtimbr` schema for a supported graph algorithm specified above.

- Each graph algorithm may require different amount of inputs in order to run correctly.
- You can see the number of input variables required by querying the metadata of the graph algorithm selected in the `gtimbr` schema.
- The input variable of the graph on which to perform the graph algorithm is a *Timbr SQL query* that is passed as a parameter to the graph algorithm you want to run.
- The output of running a graph algorithm may have different number of columns depending on the graph algorithm that was executed.

Required information for querying a graph algorithm (the curly brackets `{}` **should not** be an input, they are used only as a variable substitution):

- **{graph_algorithm}** - The name of the graph algorithm you want to run
- **{timbr_sql_query}** - The SQL query which makes up the graph on which the graph algorithm is executed.

```
SELECT * from gtimbr.{graph_algorithm}(
{timbr_sql_query}
)
```

Example using the Louvain graph algorithm

```
SELECT * from gtimbr.louvain(
  select `organization_id`, `made_investment[funding_round].organization_id`
  from dtimbr.financial_organization
  limit 1000
)
```

Timbr REST API

Users can query the **Timbr REST API** endpoints in order to query their knowledge graph through HTTP/S. The Timbr REST API is exposed either directly on the **Timbr Platform** or it may also be run separately on a **Timbr REST API Service** Kubernetes container.

- The endpoint to query the API through the **Timbr Platform** is exposed through the path `/timbr/api/`
- The endpoint to query the API through the **Timbr REST API Service** instance are exposed through the paths `/timbr/api/` and `/api/`

Table of contents

Timbr API Service

The Timbr API Service is run in a Kubernetes container and can have different configuration flags to be passed down as environment variables.

Evnironment Variables

The following environment variables are **required** to run the Timbr API Service

Key	Default value	Options	Description
FLASK_APP	`./timbr_api_service/__init__.py`	relative or absoulte path inside the container	Location of the timbr api service init script
FLASK_ENV	`development`	`development` or `production`	The environment for which the service is running
THRIFT_HOST	`localhost`	Local or Remote DNS or IP	The **Timbr Server** host name or ip
THRIFT_PORT	`11000`	Port number	The **Timbr server** port

These environment variables are **optional**:

Key	Default value	Options	Description
ALLOWED_CONFIG_ENVS_IN_API	`TIMBR_DB_DOMAIN`	List of strings separated by comma	A list of environment variables that can be retrieved by querying the `get_config` endpoint. For example: `TIMBR_DB_DOMAIN,REDIS_HOST`
TIMBR_DB_DOMAIN	Timbr Server JDBC URI		A string representing the **JDBC URI** of the **Timbr Server**
TIMBR_DB_USER	token	Token or username	The username or token for authenticating with the **Timbr Server**. Needs to be `token` for token based authentication. **If another username is provided, the Timbr API Service becomes public**

Key	Default value	Options	Description
TIMBR_DB_PASSWORD	`**********`	Astrix or user password	**Must be 12 astrix** to be used with `token` based authentication.
API_AUTH_SALT	`xxxxxxxxxxxxx`	Any string up to 128 characters	Key used to decrypt a base64 username/password combination for the `get_token` request to retrieve a user's token
REDIS_HOST		Local or Remote DNS or IP	The host name of the **Redis** instance that stores the cache
REDIS_PORT		Port number	The port of the **Redis** instance that stores the cache
REDIS_DB		Valid string for a **Redis** DB name	The DB name of the **Redis** instance that stores the cache
REDIS_AUTH		Valid string for a **Redis** Password	The password for the **Redis** instance that stores the cache

THE API_AUTH_SALT FLAG

The use of the `API_AUTH_SALT` flag is used only for authenticating to get an API token. The process is further explained in Timbr API Requirements

API Requirements

Any call for the API **requires** the following headers in the `REQUEST` header **except for the** `get_token` **API call**

- `Content-Type` - The content type of the request must be `application/json`
- `x-api-key` - The token value for a user received from the Timbr-platform

For the `get_token` endpoint, the following headers are required instead of the `x-api-key` header

- `Authorization` - The authorization bearer of a base64 username/password combination encrypted with the salt as defined in the `API_AUTH_SALT` environment variable of the timbr-api service

---

How to get a token for a user from the Timbr platform

For the following example, we use the following assumption: The endpoint https://timbr-sample.company.com as if the **Timbr-Platform** is running and configured

1. Login to the timbr-platform endpoint (i.e. - https://timbr-sample.company.com)
2. Access the endpoint - https://timbr-sample.company.com/timbr/get_timbr_user_token/
3. Save the value of the `token` to be used as the `x-api-key` of the requests.

How to get a token from the Timbr API Service

To get the API token using the Timbr API Service you will need to generate an **Authorization Bearer token**. Timbr API Service uses **Fernet** symmetric encryption as defined in Python `cryptography` library to generate the **Authorization Bearer Token** The Bearer Token is used in the request for `get_token` in order to get a Timbr-api token.

Steps to get a Timbr-api token using an Authorization Bearer token:

1. Create an authorization bearer token by following and running the python script provided with the installation of the **Timbr API Service**
2. Post a request using the authorization bearer token with the following headers to `/timbr/api/get_token`

Header Key	Header Value	Sample	Description
`Content-Type`	`application/json`	`Content-Type: application/json`	Posts a query and returns results in JSON form
`Authorization`	`Bearer <token>`	`Authorization: Bearer gAAAAA....`	Used for authentication and retrieval of a timbr token to be used in following requests

Request:

```
curl -X 'POST' \
  'https://timbr-sample.company.com/timbr/api/get_token' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer gAAA...Gy8='
```

Response:

- Response code: 200
- Response body:

```
{
  "data": [
    "tk_ab...6af3"
  ],
  "status": "success"
}
```

Required Headers in Requests

The following headers are required in any API call request (except for the `get_token` request)

Header Key	Header Value	Sample	Description
`Content-Type`	`application/json`	`Content-Type: application/json`	Posts a query and returns results in JSON form
`x-api-key`	`**********`	`x-api-key: **********`	The header value is the token value for a user from the Timbr-Platform

Requests and Responses

Below is the usage of the current API implementation:

Endpoint	Method	Body	Description
`/timbr/api/get_ontologies/`	`GET`		Posts a query and returns results in JSON form
`/timbr/api/query/`	`POST`	`{ ontology_name: <ontology_name>, query: <SQL query to run> }`	Posts a query and returns results in JSON form
`/timbr/api/get_ontology/<ontology_name>`	`POST`		Gets the ontology details for a certain ontology
`/timbr/api/create_ontology/`	`POST`	`{ ontology_name: <new_ontology_name> }`	Creates a new ontology in the timbr
`/timbr/api/q/<ontology_name>/concept/<concept_name>?<filters>`	`GET`		Query a concept in a knowledge graph. A user can **optionally** specify `<filters>` for the properties of the concept
`/timbr/api/q/<ontology_name>/view/<view_name>?<filters>`	`GET`		Query a view in a knowledge graph. A user can **optionally** specify `<filters>` for the properties of the view
`/timbr/api/get_config/<config_name>`	`POST`		Gets the value of a whitelisted environment variable as specified in the `ALLOWED_CONFIG_ENVS_IN_API` environment variable
`/timbr/api/clear_cache/<ontology_name>`	`POST`		Clears the **Redis** cache for an ontology

Examples for calling the REST API endpoints

The following was tested using curl with MINGW64 on Windows 11

Get ontologies endpoint

Request:

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/get_ontologies/' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": ["timbr_supply_chain", "timbr_user_segmentation", ...],
  "status": "success"
}
```

Query endpoint

Request:

```
curl -X 'POST' \
  'https://timbr-sample.company.com/timbr/api/query/' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
  -H 'Content-Type: application/json' \
  -d '{
  "query": "select * from timbr.person limit 5",
  "ontology_name": "timbr_supply_chain"
}'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": [{
    "entity_id": "p:2",
    "entity_type": "person",
    ...
  },
  ...],
  "status": "success"
}
```

Get ontology endpoint

Assuming the `<ontology_name>` is `timbr_supply_chain`

Request:

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/get_ontology/timbr_supply_chain' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": {
    "datasources": [{ ... }, ...],
    "ontology": [ { ... }, ...],
    "properties": { ... },
    "quote": "`",
    "version": "52b2f455-bd0f-f34a-a9cc-9c799b8b9e6d",
    "views": { ... },
  },
  "status": "success"
}
```

Create ontology endpoint

Assuming the new ontology name we want to create is `timbr_finance`

Request:

```
curl -X 'POST' \
  'https://timbr-sample.company.com/timbr/api/create_ontology' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' \
  -d '{
  "ontology_name": "timbr_finance"
}'
```

Response:

- Response code: 200
- Response body:

```
{
  "status": "success"
}
```

Query concept endpoint

Assumptions

- `ontology_name` is `**timbr_supply_chain**
- `concept_name` is `**customer**
- `filters` Optional. Filters are a list of parameters for the properties or a relationship of a concept that translate into a `WHERE` statement in SQL. Omitting the filters will perform a `SELECT *` on the `timbr` schema of a concept.
- `nested` Optional. `false` by default. If `true` if the data in the results should be formatted in a nested value for relationships

> **NESTED IN HEADER**
>
> If the nested key is `true` then the results will be formatted only if **there is a relationship or a multi-value in the filters of the GET URI endpoint**

The filter paramteres can be 1..n of the following combinations:

1. `_subj_` - The subject of the filter – A property in the knowledge graph
2. `_op_` - The operator – The operator performed in the filter, must be one of
    i. `eq` - Equals, in SQL: `=`
    ii. `not_eq` - - Not equals, in SQL `!=` or `<>`
    iii. `in` - Includes, in SQL: `IN(..)`
    iv. `not_in` - Not includes, in SQL: `NOT IN(..)`
    v. `like` - Contains, in SQL: `LIKE` (can be used with wildcards combinations like `%` and `[]`)
    vi. `like` - Not contains, in SQL: `NOT LIKE` (can be used with wildcards combinations like `%` and `[]`)
    vii. `gt` - Greater than, in SQL `>`
    viii. `gte`- Greater than or equal, in SQL `>=`
    ix. `lt` - Less than, in SQL `<`
    x. `lte` - Less than or equal, in SQL `<=`
    xi. `is_null` - Null value, in SQL `IS NULL`
    xii. `not_null` - Is not null value, in SQL `IS NOT NULL`

3. `_comp_` - The comparator – The comparison value(s)
4. `_conj_` - The conjunction – For compound filters, must be either `AND` or `OR`

The order of the parameters also matter, but they can be sorted by the numbering (`_subj_`, `_subj1_`, `_subj2_` , ...)

For example: `/timbr/api/q/timbr_supply_chain/concept/person?_subj_=property1&_op_=eq&_comp_=val1&_conj_=AND&_subj1_=property2&_op1_=like&_comp1_=val2&_conj1_=OR_subj2_=property3&_op2_=IN&_comp2_=val2,val3,val4&_subj3_=....`

Request:

Querying the `customer` concept in the `timbr_supply_chain` knowledge graph. Applying the following filters:

- `customer_name` property should start with the string `Elizabeth P` *OR*
- `customer_id` equals to `5503`

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/q/timbr_supply_chain/concept/customer?
_subj_=customer_name&_op_=like&_comp_=Elizabeth%20P%&_conj_=OR&_subj1_=customer_id&_op1_=eq&_comp1_=5503' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": [{
    "customer_email": "XXXXXXXXX",
    "customer_id": "5503",
    "customer_name": "Tiffany Smith",
    "customer_password": "XXXXXXXXX",
    "customer_segment": "Consumer",
    "entity_id": "5503",
    "entity_label": "Tiffany Smith",
    "entity_type": "customer"
  }, {
    "customer_email": "XXXXXXXXX",
    "customer_id": "639",
    "customer_name": "Elizabeth Pittman",
    "customer_password": "XXXXXXXXX",
    "customer_segment": "Home Office",
    "entity_id": "639",
    "entity_label": "Elizabeth Pittman",
    "entity_type": "customer"
  }, {
    "customer_email": "XXXXXXXXX",
    "customer_id": "801",
    "customer_name": "Elizabeth Parker",
    "customer_password": "XXXXXXXXX",
    "customer_segment": "Consumer",
    "entity_id": "801",
    "entity_label": "Elizabeth Parker",
    "entity_type": "customer"
  }, {
    "customer_email": "XXXXXXXXX",
    "customer_id": "10058",
    "customer_name": "Elizabeth Proctor",
    "customer_password": "XXXXXXXXX",
    "customer_segment": "Corporate",
    "entity_id": "10058",
    "entity_label": "Elizabeth Proctor",
    "entity_type": "customer"
  }],
  "executed_query": "SELECT * FROM `dtimbr`.`customer` WHERE `customer_name` LIKE 'Elizabeth P%' OR `customer_id` =
'5503'",
  "status": "success"
}
```

Query ontology view endpoint

Assumptions

- `ontology_name` is **`timbr_supply_chain**
- `view_name` is **`customer_view**
- `filters` Optional. Filters are a list of parameters for the properties or a relationship of a view that translate into a `WHERE` statement in SQL. Omitting the filters will perform a `SELECT *` on the `vtimbr` schema.
- `nested` Optional. `false` by default. If `true` if the data in the results should be formatted in a nested value for relationships

NESTED IN HEADER

If the nested key is `true` then the results will be formatted only if **there is an `entity_id` column in the ontology view SQL query**

The filter paramteres can be 1..n of the following combinations:

1. `_subj_` - The subject of the filter – A property in the knowledge graph
2. `_op_` - The operator – The operator performed in the filter, must be one of
    i. `eq` - Equals, in SQL: `=`
    ii. `not_eq` - - Not equals, in SQL `!=` or `<>`
    iii. `in` - Includes, in SQL: `IN(..)`
    iv. `not_in` - Not includes, in SQL: `NOT IN(..)`
    v. `like` - Contains, in SQL: `LIKE` (can be used with wildcards combinations like `%` and `[]`)
    vi. `like` - Not contains, in SQL: `NOT LIKE` (can be used with wildcards combinations like `%` and `[]`)
    vii. `gt` - Greater than, in SQL `>`
    viii. `gte`- Greater than or equal, in SQL `>=`
    ix. `lt` - Less than, in SQL `<`
    x. `lte` - Less than or equal, in SQL `<=`
    xi. `is_null` - Null value, in SQL `IS NULL`
    xii. `not_null` - Is not null value, in SQL `IS NOT NULL`
3. `_comp_` - The comparator – The comparison value(s)
4. `_conj_` - The conjunction – For compound filters, must be either `AND` or `OR`

The order of the parameters also matter, but they can be sorted by the numbering (`_subj_`, `_subj1_`, `_subj2_` , ...)

For example: `/timbr/api/q/timbr_supply_chain/view/person?_subj_=property1&_op_=eq&_comp_=val1&_conj_=AND&_subj1_=property2&_op1_=like&_comp1_=val2&_conj1_=OR_subj2_=property3&_op2_=IN&_comp2_=val2,val3,val4&_subj3_=....`

Request:

Querying the `customer_view` view in the `timbr_supply_chain` knowledge graph without any filters

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/q/timbr_supply_chain/view/customer_view' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": [{...}],
  "executed_query": "SELECT * FROM `vtimbr`.`customer_view`",
  "status": "success"
}
```

Get config endpoint

Request:

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/get_config/TIMBR_DB_DOMAIN' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "data": {
    "TIMBR_DB_DOMAIN": "jdbc:hive2://172.21.57.160:11000"
  },
  "status": "success"
}
```

Clear cache endpoint

> **IMPORTANT!**
>
> Must have a **Redis** backend set and configured for this endpoint to properly work

Request:

```
curl -X 'GET' \
  'https://timbr-sample.company.com/timbr/api/clear_cache/timbr_supply_chain' \
  -H 'Content-Type: application/json' \
  -H 'x-api-key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Response:

- Response code: 200
- Response body:

```
{
  "status": "success"
}
```