Web scraper

Code

```
-- SI web tool for web scrapping
-- Stored proc that scrape from a website using beautiful soup
  -- NETWORK rule is part of db schema
CREATE OR REPLACE NETWORK RULE Snowflake_intelligence_WebAccessRule
  MODE = EGRESS
  TYPE = HOST_PORT
  VALUE_LIST = ('0.0.0.0:80', '0.0.0.0:443');

CREATE OR REPLACE EXTERNAL ACCESS INTEGRATION
Snowflake_intelligence_ExternalAccess_Integration
  ALLOWED_NETWORK_RULES = (Snowflake_intelligence_WebAccessRule)
  ENABLED = true;

CREATE OR REPLACE FUNCTION Web_scrape(weburl STRING)
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = 3.10
HANDLER = 'get_page'
EXTERNAL_ACCESS_INTEGRATIONS =
(Snowflake_intelligence_ExternalAccess_Integration)
PACKAGES = ('requests', 'beautifulsoup4')
AS
$$
import _snowflake
import requests
from bs4 import BeautifulSoup

def get_page(weburl):
  url = f"{weburl}"
  response = requests.get(url)
  soup = BeautifulSoup(response.text)
  return soup.get_text()
$$;

--test
select Web_scrape('https://www.snowflake.com/en/blog/ISO-IEC-42001-AI-certification/');
```

Tool Set up

Name:

Web_scraper

Resource type:

function

Parameter Description:

URL that shall be scraped.

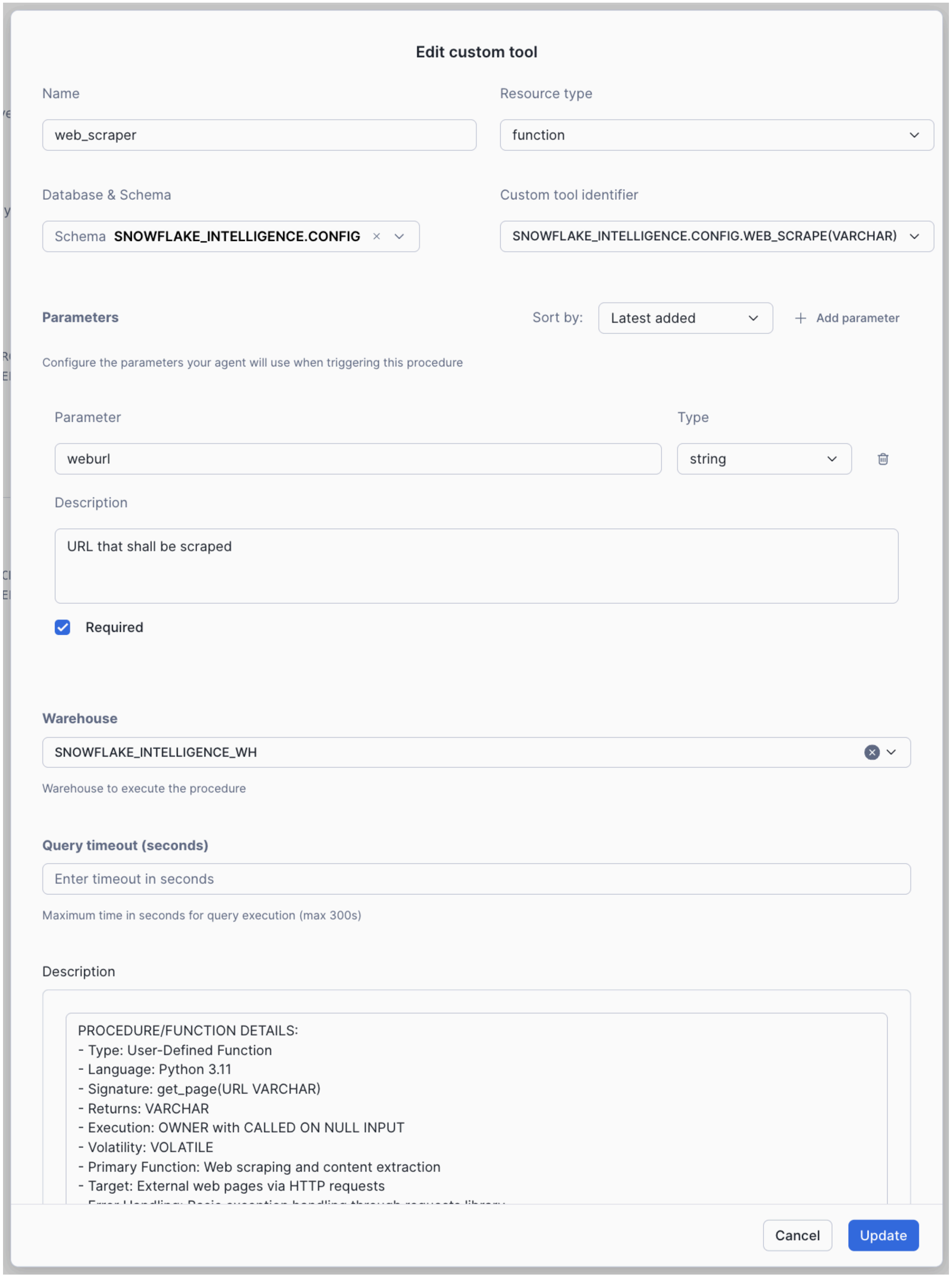
Tool Description:

PROCEDURE/FUNCTION DETAILS:
- Type: User-Defined Function
- Language: Python 3.11
- Signature: get_page(URL VARCHAR)
- Returns: VARCHAR
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: Web scraping and content extraction
- Target: External web pages via HTTP requests
- Error Handling: Basic exception handling through requests library

DESCRIPTION:
This Python-based function enables users to fetch and extract text content from web pages by providing a URL as input. The function performs HTTP requests to retrieve web page content and uses BeautifulSoup to parse HTML and extract clean text, making it valuable for data collection, content analysis, and web scraping workflows within the database environment. Since it executes with OWNER privileges and requires external network access through the AI_EXTERNAL_ACCESS_INTEGRATION, users should ensure they have appropriate permissions and comply with website terms of service and rate limiting policies. The function is marked as VOLATILE because it accesses external resources that can change between calls, and it will execute even when passed NULL input values. Organizations should implement proper governance around its usage to prevent abuse and ensure compliance with data privacy regulations when scraping external websites.

USAGE SCENARIOS:
- Content monitoring and analysis: Regularly extracting text from news websites, blogs, or competitor pages for market intelligence and trend analysis
- Data enrichment workflows: Supplementing existing datasets with publicly available information from corporate websites, product pages, or regulatory filings
- Development and testing: Creating sample datasets for testing applications by extracting content from various web sources during development cycles

Make sure you not only click SAVE but also REFRESH THE PAGE



Edit custom tool

Name: web_scraper

Resource type: function

Database & Schema: Schema **SNOWFLAKE_INTELLIGENCE.CONFIG**

Custom tool identifier: SNOWFLAKE_INTELLIGENCE.CONFIG.WEB_SCRAPE(VARCHAR)

Parameters

Sort by: Latest added

Add parameter

Configure the parameters your agent will use when triggering this procedure

Parameter: weburl

Type: string

Description: URL that shall be scraped

✓ Required

Warehouse

SNOWFLAKE_INTELLIGENCE_WH

Warehouse to execute the procedure

Query timeout (seconds)

Enter timeout in seconds

Maximum time in seconds for query execution (max 300s)

Description

```
PROCEDURE/FUNCTION DETAILS:
- Type: User-Defined Function
- Language: Python 3.11
- Signature: get_page(URL VARCHAR)
- Returns: VARCHAR
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: Web scraping and content extraction
- Target: External web pages via HTTP requests
```

Cancel

Update

Email sender

Code

```
-- SI web tool for email sending

-- notification integration required by the SP
use role accountadmin;
grant CREATE INTEGRATION on ACCOUNT to role snowflake_intelligence_admin_rl;
use role snowflake_intelligence_admin_rl;

CREATE OR REPLACE NOTIFICATION INTEGRATION ai_email_int
  TYPE=EMAIL
  ENABLED=TRUE;

-- Stored proc that sends email to a specified recipient
CREATE OR REPLACE PROCEDURE send_mail(recipient TEXT, subject TEXT, text TEXT)
RETURNS TEXT
LANGUAGE PYTHON
RUNTIME_VERSION = '3.11'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'send_mail'
AS
$$
def send_mail(session, recipient, subject, text):
    session.call(
        'SYSTEM$SEND_EMAIL',
        'ai_email_int',
        recipient,
        subject,
        text,
        'text/html'
    )
    return f'Email was sent to {recipient} with subject: "{subject}".'
$$;

--test
CALL send_mail('olivier.sinquin@snowflake.com', 'Test Email', 'This is a test email sent from
Snowflake.');
```

Tool Set up

Name:

email_sender

Resource type:

procedure

Parameter Description:

recipient
email address the email should be sent to

Subject
subject of the email

Content
content of the email

Tool Description:

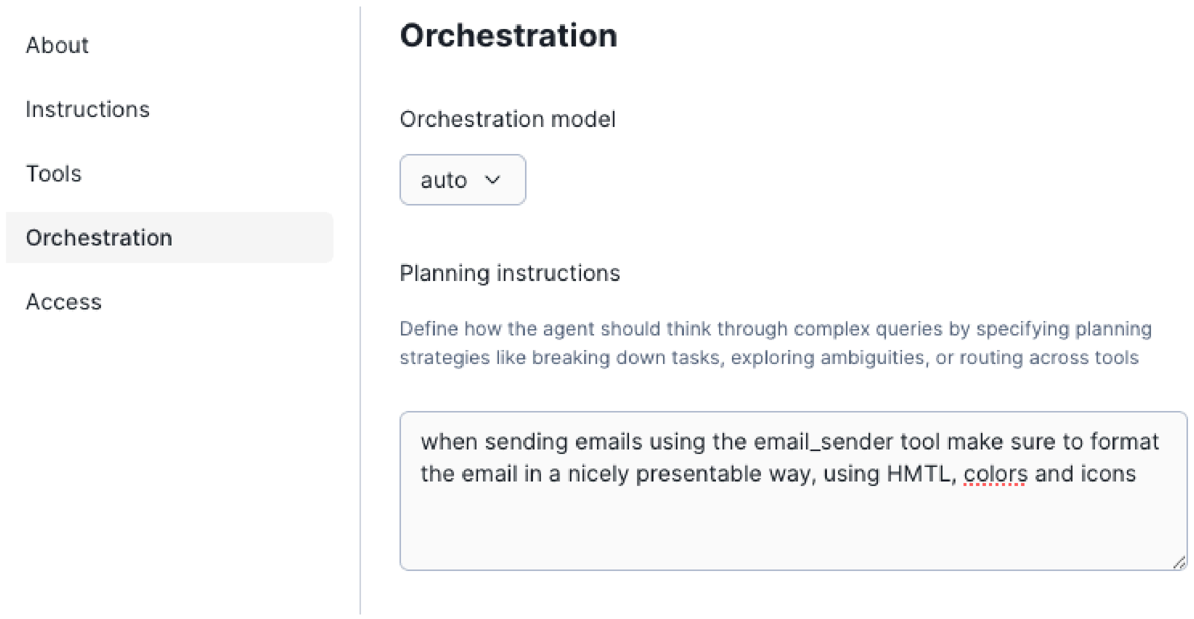
PROCEDURE/FUNCTION DETAILS:
- Type: Stored Procedure
- Language: Python 3.11
- Signature: send_mail(TEXT, TEXT, TEXT):
- Returns: TEXT
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: email sending

DESCRIPTION:
This Python-based stored procedure sends an email to the provided email address with the provided subject and content

INSTRUCTION:
When sending emails using the email_sender tool make sure to format the email in a nicely presentable way, using HMTL, colors and icons

Make sure you not only click SAVE but also REFRESH THE PAGE

If the email does not get as nicely formatted as you'd expect you can also use the agent orchestration tab to force it


About
Instructions
Tools
Orchestration
Access

Orchestration

Orchestration model

auto

Planning instructions

Define how the agent should think through complex queries by specifying planning strategies like breaking down tasks, exploring ambiguities, or routing across tools

when sending emails using the email_sender tool make sure to format the email in a nicely presentable way, using HMTL, colors and icons

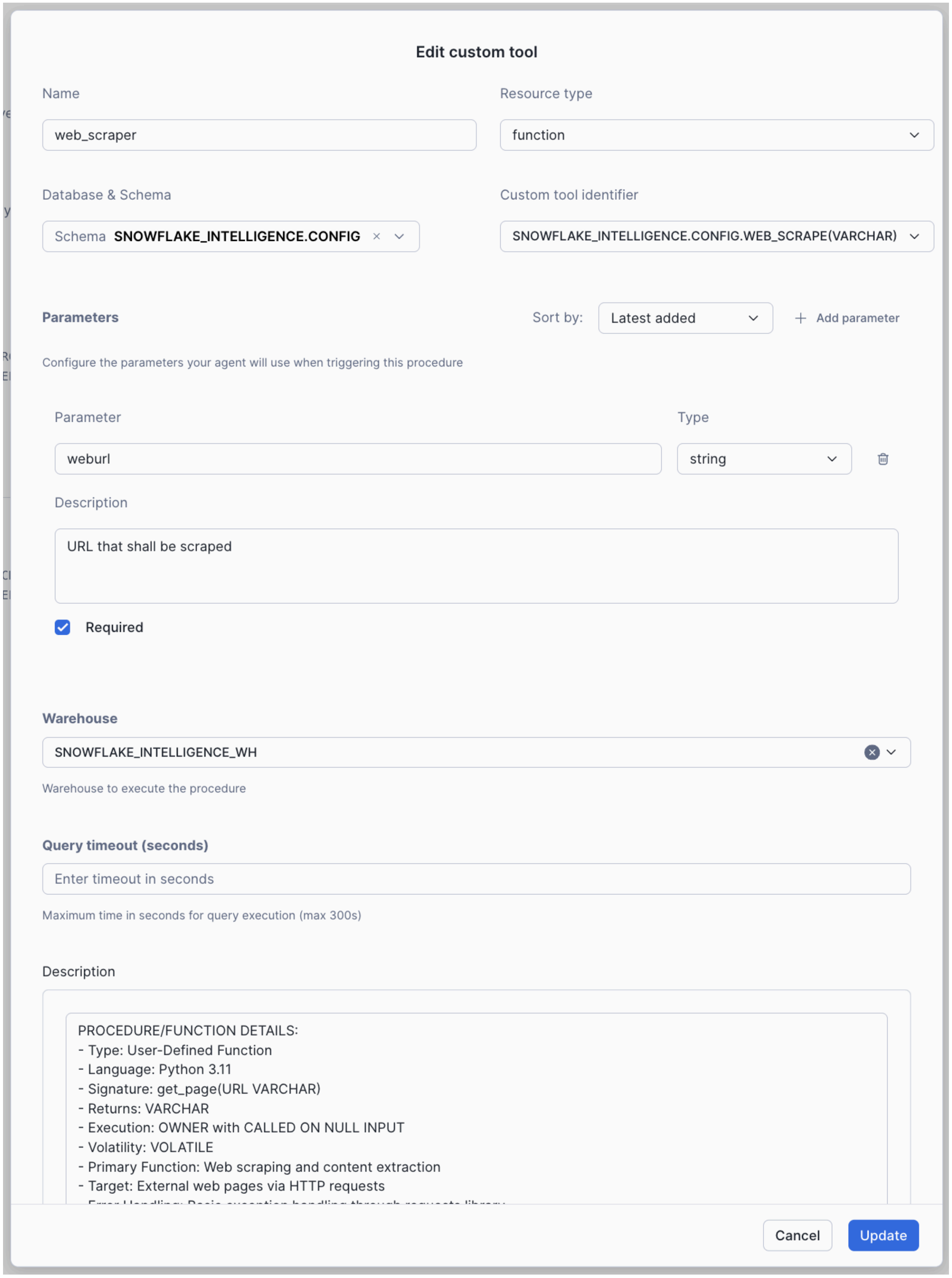
Test

send an email to olivier.sinquin@snowflake.com congratulating him for setting up with his first custom tools

Tool chaining (web scraping > email sending)

Test

Get the current number of customers and marketplace listings from Snowflake.com and email it to me


Edit custom tool

Name: web_scraper

Resource type: function

Database & Schema: Schema SNOWFLAKE_INTELLIGENCE.CONFIG

Custom tool identifier: SNOWFLAKE_INTELLIGENCE.CONFIG.WEB_SCRAPE(VARCHAR)

Parameters — Sort by: Latest added — Add parameter

Configure the parameters your agent will use when triggering this procedure

Parameter: weburl — Type: string

Description: URL that shall be scraped

☑ Required

Warehouse: SNOWFLAKE_INTELLIGENCE_WH

Warehouse to execute the procedure

Query timeout (seconds): Enter timeout in seconds

Maximum time in seconds for query execution (max 300s)

Description

```
PROCEDURE/FUNCTION DETAILS:
- Type: User-Defined Function
- Language: Python 3.11
- Signature: get_page(URL VARCHAR)
- Returns: VARCHAR
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: Web scraping and content extraction
- Target: External web pages via HTTP requests
```

Get Exchange Rate

Free signup https://www.exchangerate-api.com/

Code

```
CREATE OR REPLACE NETWORK RULE EXCHANGERATE_API
  MODE = EGRESS
  TYPE = HOST_PORT
  VALUE_LIST = ('v6.exchangerate-api.com');

CREATE OR REPLACE EXTERNAL ACCESS INTEGRATION
EXCHANGERATE_API_INTEGRATION
  ALLOWED_NETWORK_RULES = (EXCHANGERATE_API)
  ENABLED = true;

GRANT USAGE ON INTEGRATION EXCHANGERATE_API_INTEGRATION TO ROLE public;

CREATE OR REPLACE PROCEDURE get_exchange_rates(base_currency STRING)
RETURNS VARIANT
LANGUAGE PYTHON
RUNTIME_VERSION = '3.10'
HANDLER = 'get_rates'
PACKAGES = ('requests', 'snowflake-snowpark-python')
EXTERNAL_ACCESS_INTEGRATIONS = (EXCHANGERATE_API_INTEGRATION)
AS
$$
import requests
import json

def get_rates(base_currency):
    try:
        # Validate input - ensure it's 3 characters and uppercase
        if not base_currency or len(base_currency.strip()) != 3:
            return {"error": "Currency code must be exactly 3 characters (e.g., USD, EUR, GBP)"}

        base_currency = base_currency.strip().upper()

        # Call the exchange rate API with the provided base currency
        url = f"https://v6.exchangerate-api.com/v6/<APIKEY>/latest/{base_currency}"
        response = requests.get(url, timeout=10)

        if response.status_code == 200:
            return response.json()
        elif response.status_code == 404:
            return {"error": f"Currency code '{base_currency}' not supported by the API"}
        else:
            return {"error": f"API returned status code: {response.status_code}"}

    except Exception as e:
        return {"error": f"Failed to fetch exchange rates: {str(e)}"}
$$;


GRANT USAGE ON PROCEDURE
SNOWFLAKE_INTELLIGENCE.TOOLS.get_exchange_rates(string) TO ROLE public;
GRANT USAGE ON PROCEDURE
SNOWFLAKE_INTELLIGENCE.TOOLS.get_exchange_rates(string) TO ROLE accountadmin;
```

Tool Set up

Name:

get_exchange_rate

Resource type:

sproc

Parameter Description:

3 alphabet currency code to use as the base currency for exchange rate.

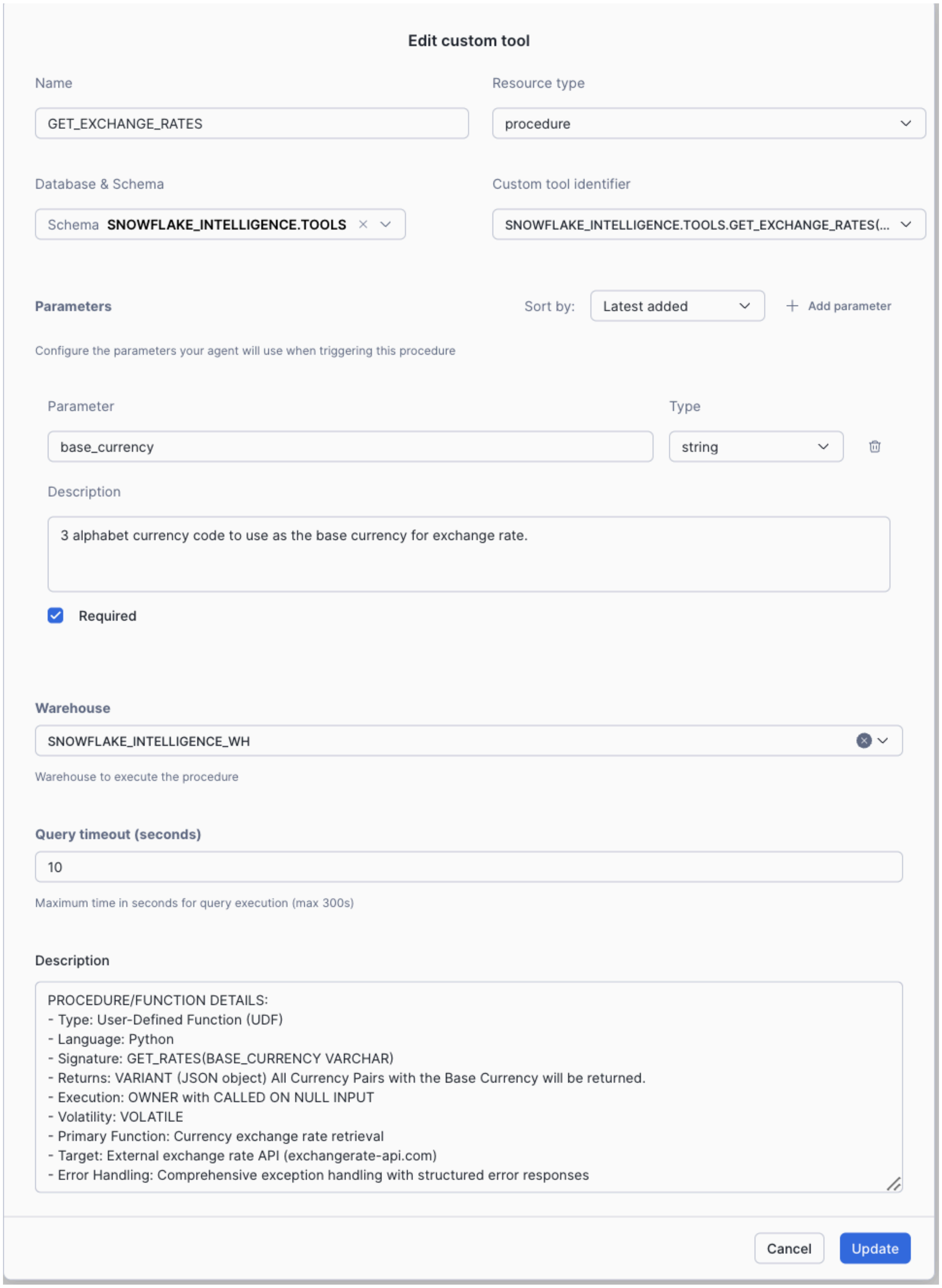
Tool Description:

PROCEDURE/FUNCTION DETAILS:
- Type: User-Defined Function (UDF)
- Language: Python
- Signature: GET_RATES(BASE_CURRENCY VARCHAR)
- Returns: VARIANT (JSON object) All Currency Pairs with the Base Currency will be returned.
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: Currency exchange rate retrieval
- Target: External exchange rate API (exchangerate-api.com)
- Error Handling: Comprehensive exception handling with structured error responses

DESCRIPTION:
This Python-based user-defined function retrieves real-time currency exchange rates from an external API service, allowing users to obtain current conversion rates for any supported base currency against all other available currencies. The function accepts a 3-character currency code (such as USD, EUR, or GBP) as input and returns a JSON object containing comprehensive exchange rate data, including conversion rates to multiple target currencies, timestamp information, and API response metadata. Built with robust error handling, the function validates input parameters, manages API timeouts, and provides clear error messages for unsupported currencies or connection issues. This function is particularly valuable for

financial applications, international business operations, and any scenario requiring up-to-date currency conversion data directly within the database environment. Since it executes with OWNER privileges and makes external API calls, users should be aware that it requires appropriate network access permissions and may be subject to API rate limits or availability constraints.

USAGE SCENARIOS:
- Financial reporting and analytics: Automatically convert multi-currency transactions to a standard base currency for consolidated financial statements and cross-border business analysis
- E-commerce and pricing systems: Dynamically update product prices and transaction amounts in real-time based on current exchange rates for international customers
- Data pipeline integration: Incorporate live currency conversion capabilities into ETL processes, data warehousing operations, and automated reporting workflows that handle international financial data



Edit custom tool

Name: GET_EXCHANGE_RATES

Resource type: procedure

Database & Schema: Schema **SNOWFLAKE_INTELLIGENCE.TOOLS**

Custom tool identifier: SNOWFLAKE_INTELLIGENCE.TOOLS.GET_EXCHANGE_RATES(...

Parameters

Sort by: Latest added

+ Add parameter

Configure the parameters your agent will use when triggering this procedure

Parameter	Type
base_currency	string

Description: 3 alphabet currency code to use as the base currency for exchange rate.

☑ Required

Warehouse

SNOWFLAKE_INTELLIGENCE_WH

Warehouse to execute the procedure

Query timeout (seconds)

10

Maximum time in seconds for query execution (max 300s)

Description

```
PROCEDURE/FUNCTION DETAILS:
- Type: User-Defined Function (UDF)
- Language: Python
- Signature: GET_RATES(BASE_CURRENCY VARCHAR)
- Returns: VARIANT (JSON object) All Currency Pairs with the Base Currency will be returned.
- Execution: OWNER with CALLED ON NULL INPUT
- Volatility: VOLATILE
- Primary Function: Currency exchange rate retrieval
- Target: External exchange rate API (exchangerate-api.com)
- Error Handling: Comprehensive exception handling with structured error responses
```

Web search

```
CREATE OR REPLACE NETWORK RULE Snowflake_intelligence_WebAccessRule
  MODE = EGRESS
  TYPE = HOST_PORT
  VALUE_LIST = ('0.0.0.0:80', '0.0.0.0:443');

CREATE OR REPLACE EXTERNAL ACCESS INTEGRATION
Snowflake_intelligence_ExternalAccess_Integration
  ALLOWED_NETWORK_RULES = (Snowflake_intelligence_WebAccessRule)
  ENABLED = true;


CREATE OR REPLACE FUNCTION Web_search(query STRING)
RETURNS STRING
LANGUAGE PYTHON
RUNTIME_VERSION = 3.10
HANDLER = 'search_web'
EXTERNAL_ACCESS_INTEGRATIONS =
(Snowflake_intelligence_ExternalAccess_Integration)
PACKAGES = ('requests', 'beautifulsoup4')
AS
$$
import _snowflake
import requests
from bs4 import BeautifulSoup
import urllib.parse
import json

def search_web(query):
    encoded_query = urllib.parse.quote_plus(query)
    search_url = f"https://html.duckduckgo.com/html/?q={encoded_query}"

    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.124 Safari/537.36'
    }

    try:
        response = requests.get(search_url, headers=headers, timeout=10)
        response.raise_for_status()

        soup = BeautifulSoup(response.text, 'html.parser')

        search_results_list = []

        results_container = soup.find(id='links')

        if results_container:
```

```
        for result in results_container.find_all('div', class_='result'):
            # Check if the result is an ad and skip it.
            if 'result--ad' in result.get('class', []):
                continue

            # Find title, link, and snippet.
            title_tag = result.find('a', class_='result__a')
            link_tag = result.find('a', class_='result__url')
            snippet_tag = result.find('a', class_='result__snippet')

            if title_tag and link_tag and snippet_tag:
                title = title_tag.get_text(strip=True)
                link = link_tag['href']
                snippet = snippet_tag.get_text(strip=True)

                # Append the result as a dictionary to our list.
                search_results_list.append({
                    "title": title,
                    "link": link,
                    "snippet": snippet
                })

            # Break the loop once we have the top 3 results.
            if len(search_results_list) >= 3:
                break

        if search_results_list:
            # Return the list of dictionaries as a JSON string.
            return json.dumps(search_results_list, indent=2)
        else:
            # Return a JSON string indicating no results found.
            return json.dumps({"status": "No search results found."})

    except requests.exceptions.RequestException as e:
        return json.dumps({"error": f"An error occurred while making the request: {e}"})
    except Exception as e:
        return json.dumps({"error": f"An unexpected error occurred during parsing: {e}"})
$$;
```

DESCRIPTION for the tool in Agents

PROCEDURE/FUNCTION DETAILS:
Type: User-Defined Function
Language: Python 3.10
Signature: Web_search(query STRING)
Returns: STRING (specifically, a JSON-formatted string)
Execution: OWNER with CALLED ON NULL INPUT
Volatility: VOLATILE
Primary Function: Web search, result extraction, and structured output generation
Target: External search engine (DuckDuckGo HTML endpoint) via HTTP requests

Error Handling: Returns a JSON object with an "error" key upon request or parsing failure. Returns a JSON object with a "status" key when no results are found.

DESCRIPTION:
This Python-based function acts as a web search tool, designed to find and return a structured list of search results for a given query. It performs an HTTP request to a specialized HTML endpoint of the DuckDuckGo search engine. The function automatically filters out sponsored advertisements and extracts the title, URL, and content snippet from the top three organic search results. The output is a machine-readable JSON string, making it an ideal first-step tool for an AI agent or any automated workflow.

The function is marked as VOLATILE because its results depend on external, unpredictable data. It requires external network access through the Snowflake_intelligence_ExternalAccess_Integration, and users should be mindful of permissions and adherence to search engine policies.

USAGE SCENARIOS:
AI Agent Orchestration: Serves as the initial tool for an AI agent to find relevant URLs for a query. The agent can then parse the JSON output, extract the links, and use a subsequent tool like Web_scrape to retrieve the full content of those pages.
Automated Research: Programmatically identifying and collecting information on specific topics or keywords, providing a list of top-ranked sources without manual Browse.
Content Discovery: Finding new articles, blogs, or websites related to a topic for content curation or monitoring.
Link Analysis: Gathering a set of external links to analyze for authority, relevance, or other metrics.

Make sure you not only click SAVE but also REFRESH THE PAGE