This document describes some capabilities in private preview and thus subject to the confidentiality of the preview agreement. If you have any questions, please contact your Databricks account team.

Databricks - Genie Conversation APIs

Private Preview

Last updated: Jan 23, 2025

Preview terms

- The product is in private preview, is not intended for use in production, and is provided AS-IS consistent with your agreement with Databricks.
- Although the preview is not intended for use in production, you may still incur charges for platform usage DBUs.
- Non-public information about the preview (including the fact that there is a preview for the feature/product itself) is confidential.
- We may change or discontinue the preview at any time without notice. These include breaking changes. We may also choose not to make the preview generally commercially available.
- We may charge for this preview offering in the future.
- Previews are not included in the SLA and do not have formal support. **If you have questions or feedback, please reach out to your Databricks contact.**

Feature Summary

This feature allows you to ask questions to an existing Genie space and retrieve query results via API. Customers can leverage their own interfaces as a front-end user-facing experience with the Genie engine.

The following endpoints are available via API:

	Description	Type	Endpoint	API Responses
Start conversation	Starts a new chat thread with an initial question	POST	`/api/2.0/genie/spaces/{space_id}/start-conversation`	Details here
Create new conversation message	Ask a follow up question to an existing chat thread	POST	`/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages`	Details here
Get conversation	Track the status of your question +	GET	`/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}`	Details here

message	retrieve the generated SQL & description			
Get SQL query results	Retrieve the query results from the executed SQL	GET	`/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}/query-result`	Details here
Re-execute SQL query	Re-execute SQL that may have expired query results	POST	`/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}/execute-query`	Details here

Important: The API does not yet support creating, modifying, or deleting Genie spaces via API.

Prerequisites

- Must be an AWS or (non GFM) Azure account
- Must have a preconfigured Genie space that is ready to receive questions via API
    - In this context, a preconfigured Genie space should include instructions, attached tables, and has been tested already.
- Authentication to submit REST API requests (details here).
    - Users associated with your authentication tokens must have Can Run or above permissions on the Genie space.
- Accept the private preview terms so that Databricks can enable this feature in the requested workspace(s).
    - Enablement is done at the workspace level.

How do permissions work with the Genie API?

By default, your Genie space will be using Run as viewer permissions as documented below.

Run as viewer: If you want different users of your Genie application to have different access to data, make sure your Genie space has the "Run as viewer" settings option selected. Each different user must have Databricks workspace access and authentication tokens. Your application must manage these different tokens when their associated users call on the API. Finally, these different users must also have the permissions on the following:

- Genie space: "Can Run" or above permissions
- Attached SQL warehouse: "Can Use" or above permissions on the warehouse
- Included tables: "Select" or above permissions on all of the tables added to the Genie space. This is where you can apply RLS permissions for different users so that they see different data.

If your Genie space is enabled to "Run as viewer," but only one user's authentication token is used by your application, everyone will use that one user's credentials on the data (the same effect as embedding that user's credentials).

NOTE: If you have also enabled the Genie Embedded Credentials private preview, the Genie API can leverage Embedded Credentials. Reach out to your account team if you want this feature and haven't been enabled yet (described here).

Embedded credentials: If you want all of your users to have access to the same data, you can configure your Genie space + API to do so (assuming you are in the Embedded Credentials private preview). You can achieve this by selecting the "Embedded Credentials" option on the Genie space. Additionally, you must ensure that the authentication tokens used by your application are associated with users that have Can Run or above permissions on the Genie space.

- Details: Embedded credentials will create one service principal that mirrors the last editor's permission attached tables only. The last editor is the person who most recently edited the data room settings such as changing warehouse, list of attached tables, descriptions, etc. Modifications to Instructions do not make a user the "last editor."

How to use the Conversation APIs

Setup

- Locate the ID for your Genie space. It can be found in the URL
  - https://{PLACEHOLDER}.databricks.com/genie/rooms/01ef274d35a310b5bffd01dadcbaf577
- Ensure your application is configured to use Databricks REST APIs.
  - Databricks authentication information, such as a Databricks personal access token
  - The workspace instance name of your Databricks deployment

```
HOST= "<WORKSPACE_INSTANCE_NAME>"
TOKEN="<your_authentication_token>"
HEADERS = {'Authorization': 'Bearer {}'.format(TOKEN)}
```

Ask an initial question + retrieve SQL results

Create a new chat thread with an initial question

A `POST` request to create a new chat thread with an initial question will look as follows:

```
POST /api/2.0/genie/spaces/{space_id}/start-conversation
Authorization: <your_authentication_token>
{
    "content": "What are the biggest open sales opportunities this year?",
}
```

An example API response and detailed descriptions are available here.

The important fields for next steps are:

- `conversation_id:` unique id for the created conversation thread. This ID is a necessary path parameter for all of the other Genie conversation functions
- `message_id:` unique id of the submitted question. Use this ID to track its execution status and get SQL query results in next steps.

Get status of your submitted question + retrieve generated SQL

Now that you have submitted your question, you can poll the "Get conversation message" endpoint to fetch its execution status.

A `GET` request to fetch status and SQL statement with description (if available) will look as follows:

```
GET /api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}
Authorization: Bearer <your_authentication_token>
```

An example API response and the detailed descriptions are available here**.**

The important field for next steps is the `status` field, which reports the status of the execution.

The options below delineate the different statuses and what they mean:

- Generation still in progress
    - `SUBMITTED`: The initial question has been submitted to the LLM.
    - `FILTERING_CONTEXT`: Genie is determining relevant context to be included for the LLM.
    - `FETCHING_METADATA`: The LLM is fetching metadata from the data sources.
    - `ASKING_AI`: Waiting for the LLM to come up with a response based on metadata + question.
- Executing SQL query
    - `EXECUTING_QUERY`: Genie will respond with a generated SQL statement, but the query is still executing. The SQL statement and its description will be available in the `attachments` field.

- Conclusive status
  - `COMPLETED`: The entire messaging process has completed:
    - Genie can respond with a follow-up question rather than an answer. In these cases, the `attachments` array field will contain a `text` object with the follow-up question in the `content` field.
    - Genie can respond with a generated SQL statement. In these cases, the `attachments` array field will have a `query` object that contains...
      - `title` string: Name of the query
      - `query` string: AI generated SQL query
      - `description` string: Description of the response
      - `last_updated_timestamp` int64: Time when the user updated the query last
    - NOTE: Genie Conversation API currently doesn't support returning SQL Function responses. This guide will be updated once fixed.
  - `FAILED`: Generating a response or executing the query failed. Additional details will be available in the `error` field.
  - `CANCELLED`: Message has been terminated.
    - This condition can be produced when SQL warehouse is stopped, or other external factors.
    - Currently no way to cancel a submitted question via API.
  - `QUERY_RESULT_EXPIRED`: The query results for the message have expired (after 7 days) and the user needs to execute the query again.

Fetch query results

Once the "Get conversation message" endpoint shows a `status` field that is "COMPLETED", you can retrieve the actual query result from the "Get conversation message SQL query result" endpoint.

A `GET` request to fetch SQL results (if available) will look as follows:

```
GET
/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}/query-result
Authorization: Bearer <your_authentication_token>
```

An example API response and the detailed descriptions are available here.

The important fields to note are:

- `status:` object describing various aspects of SQL statement execution
  - `state:` the statement execution state:
    - `PENDING`: waiting for warehouse
    - `RUNNING`: running the query
    - `SUCCEEDED`: execution was successful, result data available for fetch
    - `FAILED`: execution failed; reason for failure described in accompanying error message

        - CANCELED: user canceled; can come from explicit cancel call, or timeout with on_wait_timeout=CANCEL
        - CLOSED: execution successful, and statement closed; result no longer available for fetch
    - `error:` the different error codes that can be generated from the SQL execution.
        - `Error_code: UNKNOWN | INTERNAL_ERROR | TEMPORARILY_UNAVAILABLE | IO_ERROR | BAD_REQUEST | SERVICE_UNDER_MAINTENANCE | WORKSPACE_TEMPORARILY_UNAVAILABLE | DEADLINE_EXCEEDED | CANCELLED | RESOURCE_EXHAUSTED | ABORTED | NOT_FOUND | ALREADY_EXISTS | UNAUTHENTICATED`
- `manifest:` object providing the schema and metadata for the result set
- `result:` object containing the result data

Once the `state` field is "SUCCEEDED", you can retrieve the query result. The code snippet below is an example of how to retrieve the result set in Python. There is a placeholder get() function that manages the GET request:

```
while True:
     resp =
    get(f"/api/2.0/genie/spaces/{SPACE_ID}/conversations/{conversation_id}/message
    s/{message_id}/query-result")['statement_response']
     state = resp['status']['state']
     if state == 'SUCCEEDED':
       data = resp['result']
       meta = resp['manifest']
       rows = [[c['str'] for c in r['values']] for r in data['data_typed_array']]
       columns = [c['name'] for c in meta['schema']['columns']]
       df = spark.createDataFrame(rows, schema=columns)
       display(df)
       return df
     elif state == 'RUNNING' or state == 'PENDING':
       print(f"Waiting for query result...")
       time.sleep(5)
     else:
       print(f"No query result: {resp['state']}")
       return None
```

Ask a follow-up question + retrieve SQL results

You can also ask follow up questions to an existing conversation thread for deeper data insights (just like in the UI) by leveraging the "Create conversation message" function.

Use an existing chat thread to ask a follow-up question

To ask a follow-up question, you need an existing conversation thread.

This `POST` request to create a new chat thread with an initial question will look as follows:

```
POST /api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages
Authorization: <your_authentication_token>
{
    "content": "What about the year before?",
}
```

An example API response and the detailed descriptions are available here.

The important field to note:

- `id:` unique message id of the submitted question. Use this ID to track its status and get SQL query results in future steps.

Get status of your submitted question + retrieve generated SQL

The steps are the same as outlined in the steps above: linked here. Be sure to use the new message id generated from this follow up question.

Fetch query results

The steps are the same as outlined in the steps above: linked here

Re-execute SQL from previous message

At times, you may want to re-execute a SQL query that has expired query results (previously executed SQL becomes expired after 7 days). You can do this with the "Execute SQL query in a conversation message" endpoint.

It's important to understand that this re-executes the previously generated SQL for the message. It doesn't regenerate a new SQL response.

Execute SQL query in a conversation message

This `POST` request will re-execute a message's SQL will look as follows::

```
POST
/api/2.0/genie/spaces/{space_id}/conversations/{conversation_id}/messages/{message_id}/exec
ute-query
```

An example API response and the detailed descriptions are available here.

The important field to note:

- `statement_id:` this statement ID field is returned upon successfully submitting a SQL statement

Fetch query results
The steps here are the same as outlined in the steps above: linked here

Rate Limits

The Genie Conversation APIs have the following rate limits:

- 20 questions per minute per workspace across all Genie spaces (start conversation + create conversation message POST requests)
- The other endpoints (Get conversation message, Execute SQL query in a conversation message, and Get conversation message SQL query result) have a combined limit of 100 requests per second per workspace across all Genie spaces
- Query results from Genie responses are limited to 5000 rows in both UI and API

FAQs

- **Can you cancel a message that you sent?**
  - No, you can't currently cancel a message that you've already sent via API.
- **How many follow-up questions can I ask in a conversation thread via API?**
  - Just like in the UI, you can ask an unlimited number of questions in a chat thread.
  - The same constraints that apply to questions generally in the UI (# of tables and columns attached to the spaces, # of instructions, etc.) apply here as well.
- **Can I increase my Genie API rate limit above 20 question requests per minute?**
  - There is not a way to raise this. If you are frequently pressing against this limit, please reach out to your Solutions Architect and the Genie product team with more details on your use case.
- **What if the user with the authentication token leveraged in requests loses access to the Genie space?**
  - The Genie Conversation APIs will return a 403 error code.
- **How do Conversation API requests appear on the Genie monitoring page?**
  - The Monitoring page will contain the messages sent to the Genie space.
  - The User column will show the name of the user with the authentication token leveraged in requests.
- **How do Conversation API requests appear in the query history UI and query history system table?**
  - The Query History UI today will show the queries generated by API.

    - The User column will show
        - The service principal name if embedded credentials are used (no matter whose auth token is used)
        - If "Run as viewer" credentials are used, the User column will display the username of the user whose auth token is used by the API call.
- **Are Conversation API requests emitted in the audit log?**
    - Audit logs are not currently supported for Genie but are planned in the future.
- **What if I get a 501 error code that says "This API is not yet supported."**
    - If you receive this message, this means that your workspace has not been enabled with this private preview.
    - Please consult with your Solutions Architect to secure enablement.
- **What error messages can I receive when using the Conversation APIs?**
    - 400 Bad Request
    - 403 Permission Denied
    - 404 Not found, table does not exist
    - 409 Resource conflict
    - 429 Resource Exhausted
        - This is the error code when exceeding 20 QPM
    - 500 Internal Error or Data Loss
    - 501 Not Implemented
    - 503 Unavailable or Temporarily Unavailable
    - 504 Deadline exceeded
- **Who do I contact if I run into issues or have feedback?**
    - If you have feedback or encounter an issue not addressed in this user guide, please reach out to your Solutions Architect first and ask them to discuss with the product team supporting the preview.
    - Alternatively, as part of participating in this preview, you will have a session with the product team supporting this preview and can also raise there.

API Response Field Descriptions

Start Conversation

Example API Response	Description
{ "conversation_id": "6a64adad2e664ee58de08488f986af3e", "conversation": { "created_timestamp": 1719769718, "id": "6a64adad2e664ee58de08488f986af3e",	conversation_id uuid Conversation ID
	conversation object - created_timestamp int64 Timestamp when the message was created - id uuid Conversation ID

<table>
<tr>
<td>

```
    "last_updated_timestamp": 1719769718,
    "space_id":
"3c409c00b54a44c79f79da06b82460e2",
    "title": "Give me top sales for last
month",
    "user_id": 12345
  },
   “message_id":
"e1ef34712a29169db030324fd0e1df5f",
  "message": {
    "attachments": null,
    "content": "Give me top sales for last
month",
    "conversation_id":
"6a64adad2e664ee58de08488f986af3e",
    "created_timestamp": 1719769718,
    "error": null,
    "id":
"e1ef34712a29169db030324fd0e1df5f",
    "last_updated_timestamp": 1719769718,
    "query_result": null,
    "space_id":
"3c409c00b54a44c79f79da06b82460e2",
    "status": "IN_PROGRESS",
    "user_id": 12345
  }
}
```

</td>
<td>

- last_updated_timestamp int64 Timestamp when the message was last updated
- space_id uuid Genie space ID
- title string Conversation title
- user_id uuid ID of the user who created the conversation

message_id uuid Message ID

message object
- id uuid Message ID
- space_id uuid Genie space ID
- conversation_id uuid Conversation ID
- user_id int64 ID of the user who created the message
- created_timestamp int64 Timestamp when the message was created
- last_updated_timestamp int64 Timestamp when the message was last updated
- status string Enum: FETCHING_METADATA | ASKING_AI | EXECUTING_QUERY | FAILED | COMPLETED | SUBMITTED | QUERY_RESULT_EXPIRED | CANCELLED
- content string User message content
- attachments Array of object AI produced response to the message
- query_result object The result of SQL query if the message has a query attachment
- error object Error message if AI failed to respond to the message

</td>
</tr>
</table>

Get conversation message

<table>
<tr>
<th>Example API Response</th>
<th>Description</th>
</tr>
<tr>
<td>

```
{
  "attachments": {
    "query": {
      "description": "Query description in
a human readable format",
      "last_updated_timestamp": 1719769718,
      "query": "SELECT * FROM
top_performer",
      "title": "Query title"
    }
  },
  "content": "Give me top sales for last
month",
  "conversation_id":
"6a64adad2e664ee58de08488f986af3e",
  "created_timestamp": 1719769718,
  "error": null,
  "id": "e1ef34712a29169db030324fd0e1df5f",
  "last_updated_timestamp": 1719769718,
  "query_result": {
    "row_count": 10,
```

</td>
<td>

attachments Array of object AI produced response to the message
- query object
  - title string Name of the query
  - query string AI generated SQL query
  - description string Description of the query
  - last_updated_timestamp int64 Time when the user updated the query last
- text object name of the query
  - content string AI generated message

content string User message content

conversation_id uuid Conversation ID

created_timestamp int64 Timestamp when the message was created

error object Error message if AI failed to respond to the message
- error string

</td>
</tr>
</table>

"statement_id": "9d8836fc1bdb4729a27fcc07614b52c4" }, "space_id": "3c409c00b54a44c79f79da06b82460e2", "status": "EXECUTING_QUERY", "user_id": 12345 }	- type string id uuid Message ID last_updated_timestamp int64 Timestamp when the message was last updated query_result object The result of SQL query if the message has a query attachment - statement_id string Statement Execution API statement id. Use Get status, manifest, and result first chunk to get the full result data. - row_count int64 Row count of the result space_id uuid Genie space ID status string: FETCHING_METADATA | ASKING_AI | FILTERING_CONTEXT | EXECUTING_QUERY | FAILED | COMPLETED | SUBMITTED | QUERY_RESULT_EXPIRED | CANCELLED user_id int64 ID of the user who created the message

Create conversation message

Example API Response	**Description**
{ "attachments": null, "content": "Give me top sales for last month", "conversation_id": "6a64adad2e664ee58de08488f986af3e", "created_timestamp": 1719769718, "error": null, "id": "e1ef34712a29169db030324fd0e1df5f", "last_updated_timestamp": 1719769718, "query_result": null, "space_id": "3c409c00b54a44c79f79da06b82460e2", "status": "IN_PROGRESS", "user_id": 12345 }	attachments Array of object AI produced response to the message content string User message content conversation_id uuid Conversation ID created_timestamp int64 Timestamp when the message was created error object Error message if AI failed to respond to the message id uuid Message ID last_updated_timestamp int64 Timestamp when the message was last updated query_result object The result of SQL query if the message has a query attachment space_id uuid Genie space ID status string Enum: FETCHING_METADATA | ASKING_AI | EXECUTING_QUERY | FAILED | COMPLETED | SUBMITTED | QUERY_RESULT_EXPIRED | CANCELLED user_id int64 ID of the user who created the message

Get conversation message SQL query result & Execute SQL query in a conversation message

** Both endpoints respond with the same format*

Example API Response	**Description**

<pre>{
  "statement_response": {
  "statement_id": "string",
  "status": {
  "state": "PENDING",
  "error": {
  "error_code": "UNKNOWN",
  "message": "string"
  }
  },
 	
  "manifest": {
  "format": "JSON_ARRAY",
  "schema": {
  "column_count": 0,
  "columns": [
  {
  "name": "string",
  "type_text": "string",
  "type_name": "BOOLEAN",
  "type_precision": 0,
  "type_scale": 0,
  "type_interval_type":
"string",
  "position": 0
  }
  ]
  },
 	
  "total_chunk_count": 0,
  "chunks": [
  {
  "chunk_index": 0,
  "row_offset": 0,
  "row_count": 0,
  "byte_count": 0
  }
  ],
  "total_row_count": 0,
  "total_byte_count": 0,
  "truncated": true
  },
  "result": {
  "chunk_index": 0,</pre>
	statement_response (object): SQL Statement Execution response
	- statement_id (string): The statement ID is returned upon successfully submitting a SQL statement, and is a required reference for all subsequent calls.
	- status: The status response includes execution state and if relevant, error information.
	- state: see above
	- error: see above
	- error_code:see above
	- message: A brief summary of the error condition.
	
	manifest object: The result manifest provides schema and metadata for the result set.
	- format string: JSON_ARRAY | ARROW_STREAM | CSV
	- schema object: The schema is an ordered list of column descriptions.
	  - column_count integer: # of columns
	  - columns Array of object
	  - name string: The name of the column.
	  - type_text string: The full SQL type specification.
	  - type_name string the name of the base data type. This doesn't include details for complex types such as STRUCT, MAP or ARRAY.
	  - type_precision int32: Specifies the number of digits in a number. This applies to the DECIMAL type.
	  - type_scale int32: Specifies the number of digits to the right of the decimal point in a number. This applies to the DECIMAL type.
	  - type_interval_type string: The format of the interval type.
	  - position int32: The ordinal position of the column (starting at position 0).
	- total_chunk_count integer: The total # of chunks the result set has been divided into.
	- chunks Array of object: Array of result set chunk metadata.
	  - chunk_index integer: The position within the sequence of result set chunks.
	  - row_offset int64: The starting row offset within the result set.
	  - row_count int64: The number of rows within the result chunk.
	  - byte_count int64: The number of bytes in the result chunk. This field is not available when using INLINE disposition.
	- total_row_count int64: The total number of rows in the result set.

```
      "row_offset": 0,
      "row_count": 0,
      "byte_count": 0,
      "next_chunk_index": 0,
      "next_chunk_internal_link":
"string",
      "data_array": [
        [
          "string"
        ]
      ],
      "external_links": [
        {
          "chunk_index": 0,
          "row_offset": 0,
          "row_count": 0,
          "byte_count": 0,
          "next_chunk_index": 0,
          "next_chunk_internal_link":
"string",
          "external_link": "string",
          "expiration":
"2019-08-24T14:15:22Z",
          "http_headers": {
            "property1": "string",
            "property2": "string"
          }
        }
      ]
    }
  }
}
```


- total_byte_count int64: The total number of bytes in the result set. This field is not available when using INLINE disposition.
- truncated boolean: Indicates whether the result is truncated due to row_limit or byte_limit. Row limit is 5,000. Byte limit is 35MB.

result object: Contains the result data of a single chunk when using INLINE disposition. When using EXTERNAL_LINKS disposition, the array external_links is used instead to provide presigned URLs to the result data in cloud storage. Exactly one of these alternatives is used. (While the external_links array prepares the API to return multiple links in a single response. Currently only a single link is returned.)

- chunk_index integer: The position within the sequence of result set chunks.
- row_offset int64: The starting row offset within the result set.
- row_count int64: The number of rows within the result chunk.
- byte_count int64: The number of bytes in the result chunk. This field is not available when using INLINE disposition.
- next_chunk_index integer: When fetching, provides the chunk_index for the *next* chunk. If absent, indicates there are no more chunks. The next chunk can be fetched with a statementexecution/getstatementresultchunkn request.
- next_chunk_internal_link string When fetching, provides a link to fetch the *next* chunk. If absent, indicates there are no more chunks. This link is an absolute path to be joined with your $DATABRICKS_HOST, and should be treated as an opaque link. This is an alternative to using next_chunk_index.
- data_array Array of Array of string: The JSON_ARRAY format is an array of arrays of values, where each non-null value is formatted as a string. Null values are encoded as JSON null.
- external_links Array of object
- chunk_index integer The position within the sequence of result set chunks.
- row_offset int64: The starting row offset within the result set.
- row_count int64: The number of rows within the result chunk.
- byte_count int64: The number of bytes in the result chunk. This field is not available when using INLINE disposition.
- next_chunk_index integer: When fetching, provides the chunk_index for the *next* chunk. If absent, indicates there are no more chunks. The next chunk can be fetched with a statementexecution/getstatementresultchunkn request.
- next_chunk_internal_link string: When fetching, provides a link to fetch the *next* chunk. If absent,

	indicates there are no more chunks. This link is an absolute path to be joined with your $DATABRICKS_HOST, and should be treated as an opaque link. This is an alternative to using next_chunk_index. - external_link string: A presigned URL pointing to a chunk of result data, hosted by an external service, with a short expiration time (<= 15 minutes). As this URL contains a temporary credential, it should be considered sensitive and the client should not expose this URL in a log. - expiration date-time Indicates the date-time that the given external link will expire and becomes invalid, after which point a new external_link must be requested. - http_headers object HTTP headers that must be included with a GET request to the external_link. Each header is provided as a key-value pair. Headers are typically used to pass a decryption key to the external service. The values of these headers should be considered sensitive and the client should not expose these values in a log.