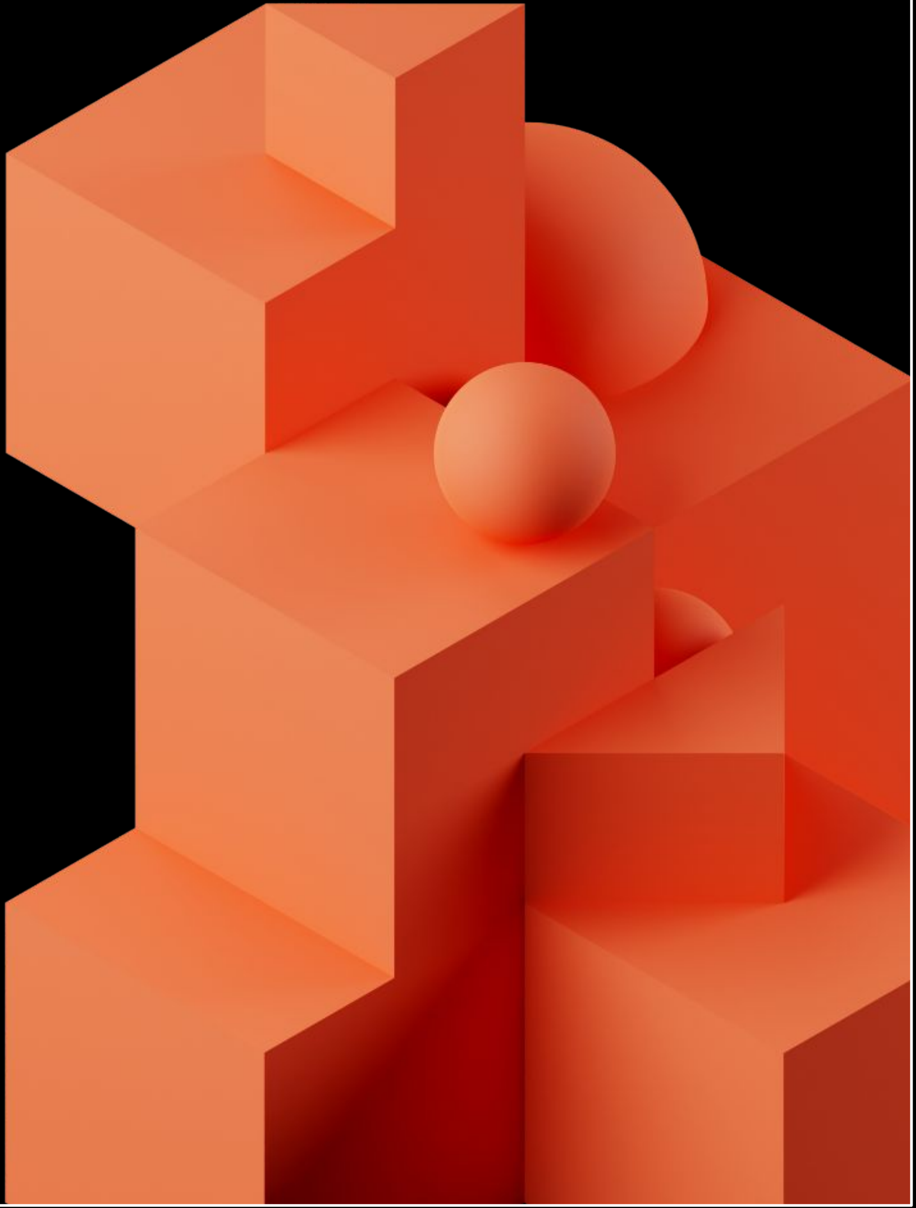
DATA+AI SUMMIT
BY databricks

Building apps for the Lakehouse with Databricks SQL

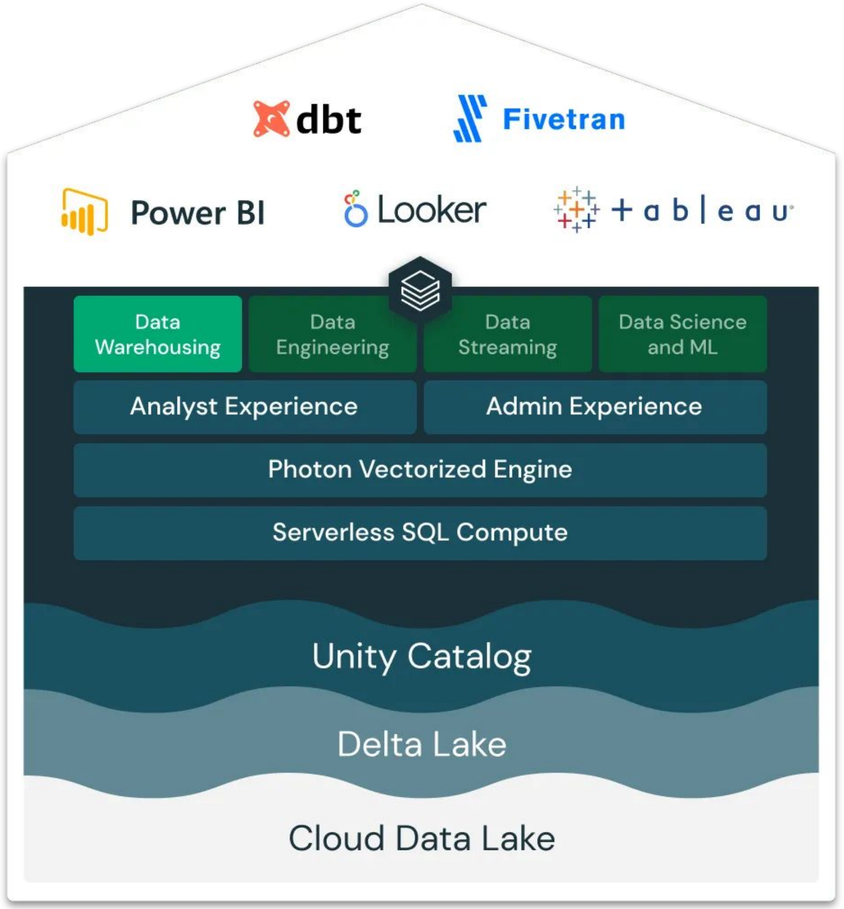
Data + AI Summit 2023



Adriana Ispas, Chris Stevens
Databricks R&D

Databricks SQL

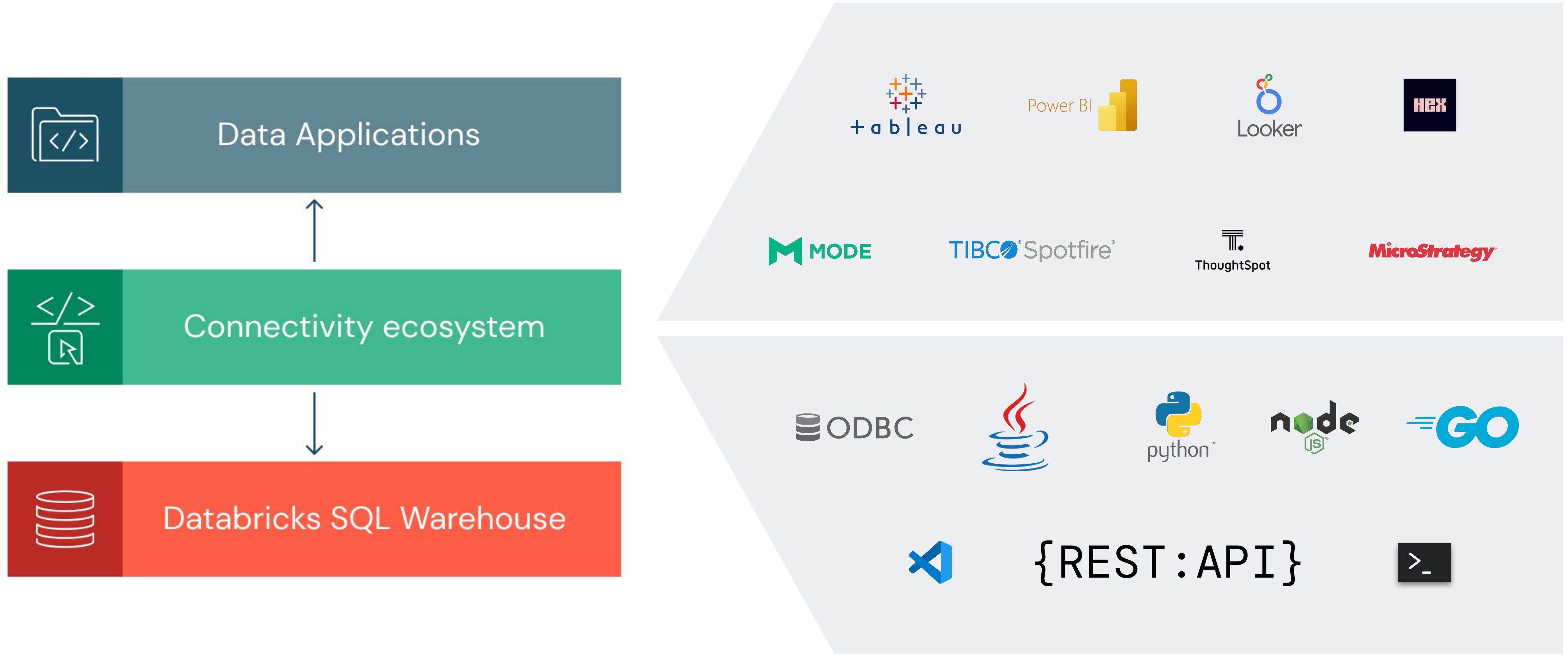
Run SQL on the Lakehouse with your tools of choice



- Connect to the lakehouse using well established BI tools, e.g., Power BI, Tableau, or Looker.
- Easily ingest and transform data in-place using your favorite tools like Fivetran or dbt.
- **Leverage existing applications to find insights or build data apps with tools and languages you already know.**

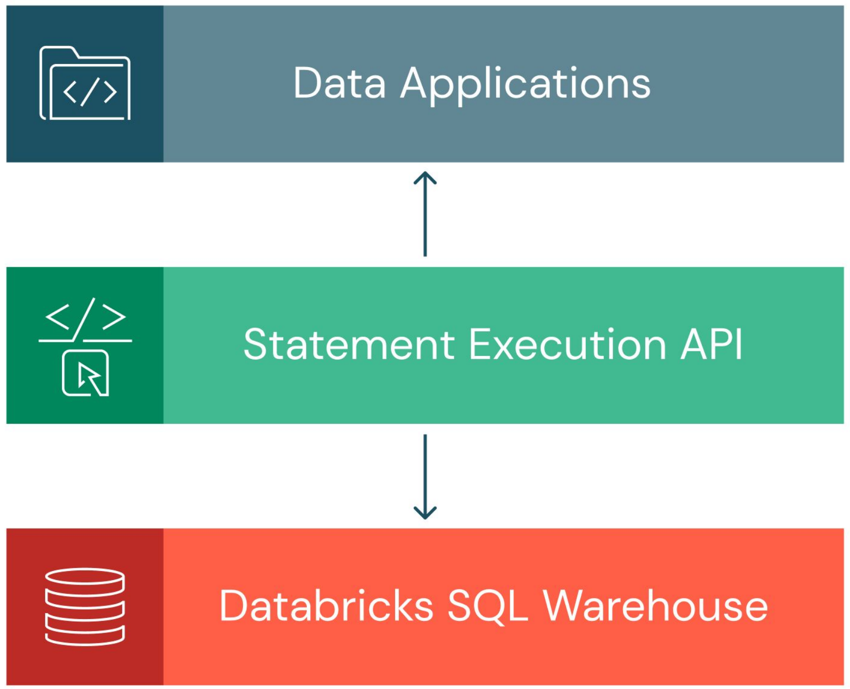
Build apps powered by the Lakehouse

Benefit from a rich ecosystem



SQL Statement Execution API

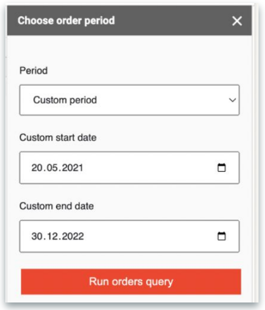
Access and manage data by executing SQL statements over HTTP



- Build custom data applications
- Integrate with a wide range of applications and computing devices
- Create a generic integration layer for enterprise services
- Create client libraries for your programming language of choice

SQL Statement Execution API

Access and manage data by executing SQL statements over REST

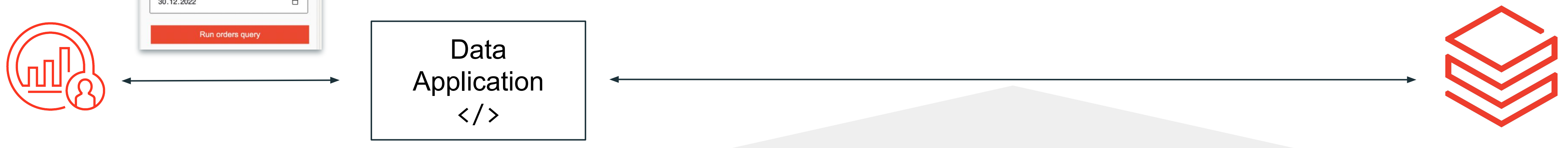


Choose order period

Period: Custom period

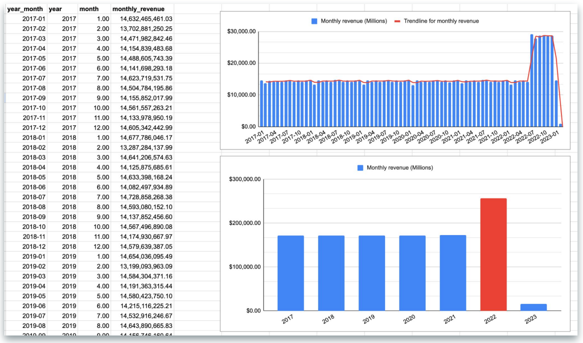
Custom start date: 20.05.2021

Custom end date: 30.12.2022



Business user

SQL Warehouse



year_month	year	month	monthly_revenue
2017-01	2017	1.00	14,632,465,461.03
2017-02	2017	2.00	13,702,881,250.25
2017-03	2017	3.00	14,471,982,842.46
2017-04	2017	4.00	14,154,839,483.68
2017-05	2017	5.00	14,488,605,743.39
2017-06	2017	6.00	14,141,698,293.18
2017-07	2017	7.00	14,623,719,531.75
2017-08	2017	8.00	14,504,784,195.86
2017-09	2017	9.00	14,155,852,017.99
2017-10	2017	10.00	14,561,557,263.21
2017-11	2017	11.00	14,133,978,950.19
2017-12	2017	12.00	14,605,342,442.99
2018-01	2018	1.00	14,677,786,046.17
2018-02	2018	2.00	13,287,284,137.99
2018-03	2018	3.00	14,641,206,574.63
2018-04	2018	4.00	14,125,875,685.61
2018-05	2018	5.00	14,633,398,168.24
2018-06	2018	6.00	14,082,497,934.89
2018-07	2018	7.00	14,728,858,268.38
2018-08	2018	8.00	14,593,080,152.10
2018-09	2018	9.00	14,137,852,456.60
2018-10	2018	10.00	14,567,496,890.08
2018-11	2018	11.00	14,174,930,667.97
2018-12	2018	12.00	14,579,639,387.05
2019-01	2019	1.00	14,654,036,095.49
2019-02	2019	2.00	13,199,093,963.09
2019-03	2019	3.00	14,584,304,371.16
2019-04	2019	4.00	14,191,363,315.44
2019-05	2019	5.00	14,580,423,750.10
2019-06	2019	6.00	14,215,116,225.21
2019-07	2019	7.00	14,532,916,246.67
2019-08	2019	8.00	14,643,890,665.83

Submit SQL statement for execution

`POST /sql/statements`

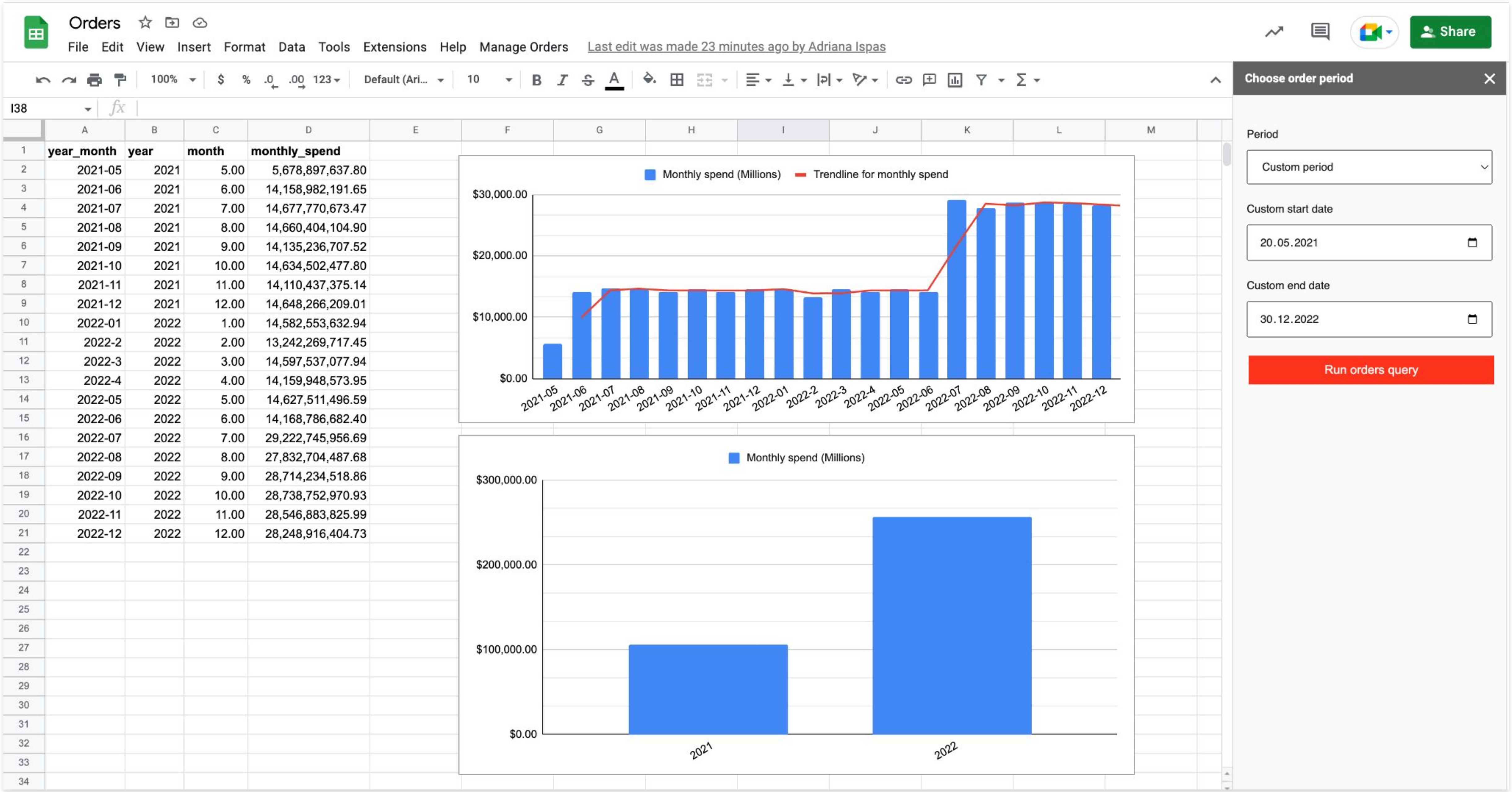
Check the execution status and retrieve results

`GET  /sql/statements/{statement_id}`

Cancel the execution of a SQL statement

`POST /sql/statements/{statement_id}/cancel`

Integrate with Google Sheets



year_month	year	month	monthly_spend
2021-05	2021	5.00	5,678,897,637.80
2021-06	2021	6.00	14,158,982,191.65
2021-07	2021	7.00	14,677,770,673.47
2021-08	2021	8.00	14,660,404,104.90
2021-09	2021	9.00	14,135,236,707.52
2021-10	2021	10.00	14,634,502,477.80
2021-11	2021	11.00	14,110,437,375.14
2021-12	2021	12.00	14,648,266,209.01
2022-01	2022	1.00	14,582,553,632.94
2022-2	2022	2.00	13,242,269,717.45
2022-3	2022	3.00	14,597,537,077.94
2022-4	2022	4.00	14,159,948,573.95
2022-05	2022	5.00	14,627,511,496.59
2022-06	2022	6.00	14,168,786,682.40
2022-07	2022	7.00	29,222,745,956.69
2022-08	2022	8.00	27,832,704,487.68
2022-09	2022	9.00	28,714,234,518.86
2022-10	2022	10.00	28,738,752,970.93
2022-11	2022	11.00	28,546,883,825.99
2022-12	2022	12.00	28,248,916,404.73

Choose order period

Period: Custom period

Custom start date: 20.05.2021

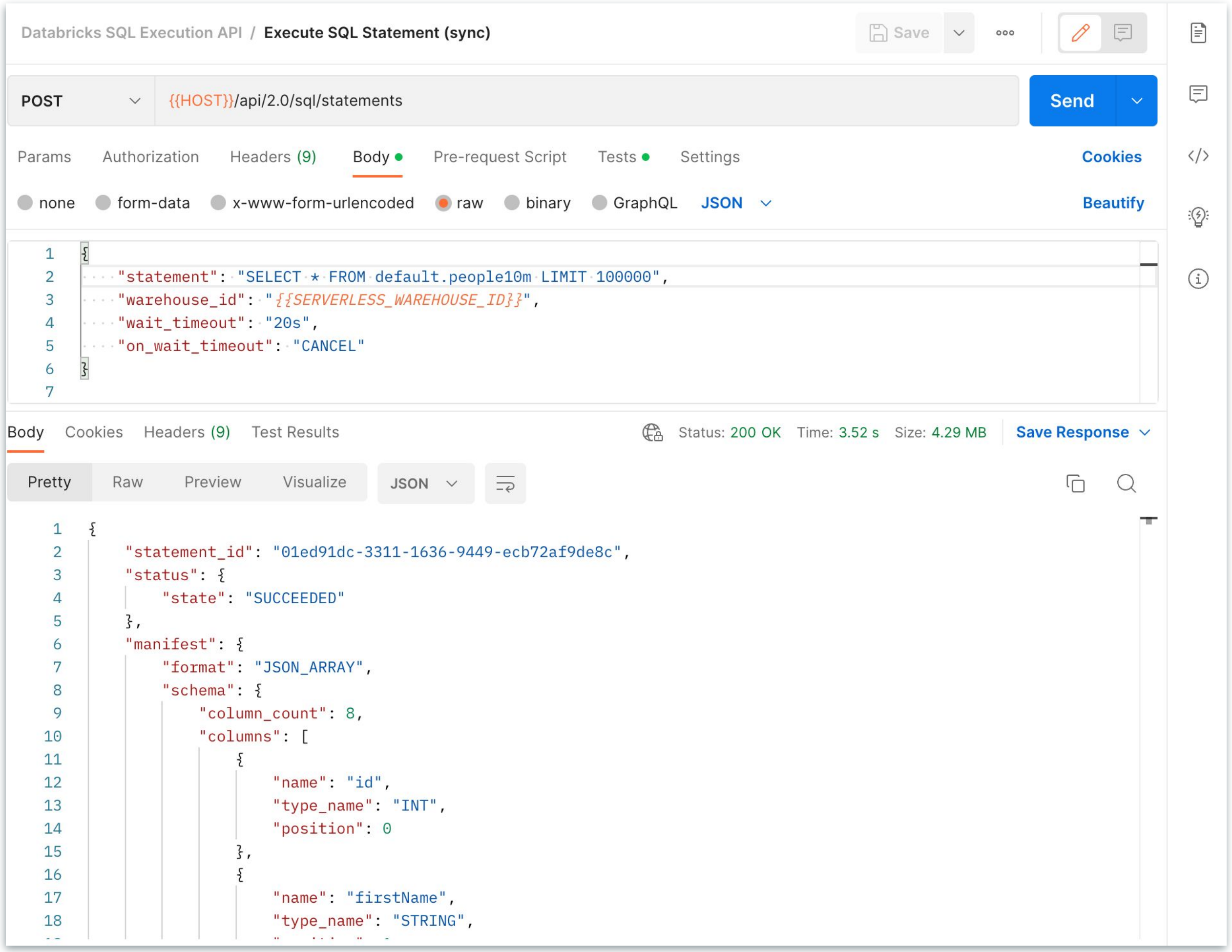
Custom end date: 30.12.2022

Run orders query

Blog post: https://www.databricks.com/blog/2023/03/07/databricks-sql-statement-execution-api-announcing-public-preview.html

Demo

Basics of the API using Postman



POST {{HOST}}/api/2.0/sql/statements

```
{
    "statement": "SELECT * FROM default.people10m LIMIT 100000",
    "warehouse_id": "{{SERVERLESS_WAREHOUSE_ID}}",
    "wait_timeout": "20s",
    "on_wait_timeout": "CANCEL"
}
```

Status: 200 OK Time: 3.52 s Size: 4.29 MB

```
{
    "statement_id": "01ed91dc-3311-1636-9449-ecb72af9de8c",
    "status": {
        "state": "SUCCEEDED"
    },
    "manifest": {
        "format": "JSON_ARRAY",
        "schema": {
            "column_count": 8,
            "columns": [
                {
                    "name": "id",
                    "type_name": "INT",
                    "position": 0
                },
                {
                    "name": "firstName",
                    "type_name": "STRING",
```

Submit SQL statement for execution

Three modes: synchronous, asynchronous and hybrid

Default: `wait_timeout ∈ [5, 50]s`. Then, continue async & fetch results in subsequent calls via ID

```
POST /sql/statements
      statement : "SELECT * FROM my_table"
      wait_timeout : "15s"
      on_wait_timeout: "CONTINUE"
```

→

```
# wait up to 15s ...
statement_id: "ID123"
status: { state: "RUNNING" }
```

Asynchronous: `wait_timeout = 0` → Execute async & fetch results in subsequent calls via ID

```
POST /sql/statements
      statement : "SELECT * FROM my_table"
      wait_timeout : "0s"
```

→

```
# no wait
statement_id: "ID123"
status: { state: "PENDING" }
```

Synchronous: `wait_timeout ∈ [5, 50]s` and return results in the same call. Otherwise, cancel.

```
POST /sql/statements
      statement : "SELECT * FROM my_table"
      wait_timeout : "15s"
      on_wait_timeout: "CANCEL"
```

→

```
# wait up to 15s, then cancel
statement_id: "ID123"
status: { state: "SUCCEEDED" }
manifest: { ... }
result: { ... }
```

Fetching results

Two modes: INLINE or EXTERNAL_LINKS

Inline: Results returned as payload, limited to 16 MiB, usually chunked, formats: JSON/CSV

```
POST /sql/statements
     …
     disposition: “INLINE”
```

→

```
…
result:
  chunk_index: 0,
  row_offset: 0,
  row_count: 1000,
  data_array: [["1234","3.14159"],...]
```

External links: Results returned via resolved pre-signed URLs; 100 GB, formats: JSON/CSV/Arrow

```
POST /sql/statements
     …
     disposition :
     “EXTERNAL_LINKS”
```

→

```
…
result:
  external_links:
  - chunk_index: 0,
    row_offset: 0,
    row_count: 257500,
    next_chunk_index": 1,
    next_chunk_internal_link: "/api/2.0/sql/statements/…/result/chunks/1?row_offset=..."
    external_link: "https://cloud.store/path/chunk00_abc?token=YYZ"
    expiration: "2022-09-22T19:21:03Z"
```

Retrieve results w/ EXTERNAL_LINKS (1)

Check execution status using the handle & retrieve the 1st result chunk if ready

```
GET /sql/statements/ID123
```

→

```
total_chunk_count: 3
chunks:
  -   chunk_index: 0
      row_offset: 0
      row_count: 257500
      ...
result:
  external_links:
  - chunk_index: 0,
    row_offset: 0,
    row_count: 257500,
    next_chunk_index": 1,
    next_chunk_internal_link:
"/api/2.0/sql/statements/.../result/chunks/1?row_offset=..."
    external_link: "https://cloud.store/path/chunk00_abc?token=YYZ"
    expiration: "2022-09-22T19:21:03Z"
```

```
GET https://cloud.store/path/chunk00_abc?token=YYZ
```

→

```
[["4444","2.0"],...]
```

Note: the disposition = EXTERNAL_LINKS is specified when submitting the execution request

Retrieve results w/ EXTERNAL_LINKS (2)

Retrieve a specific chunk in the result set

```
GET /sql/statements/ID123/result/chunks/1/
?row_offset=...
```

→

```
external_links:
  - chunk_index: 1,
    row_offset: 257500,
    row_count: 257500,
    next_chunk_index": 2,
    next_chunk_internal_link:
"/api/2.0/sql/statements/…/result/chunks/2?row_offset=..."
    external_link: "https://cloud.store/path/chunk00_abc?token=YYZ"
    expiration: "2022-09-22T19:21:03Z"
```

```
GET https://cloud.store/path/chunk01_abc?token=YYZ
```

→

```
[["4444","2.0"],...]
```

Note: the disposition = EXTERNAL_LINKS is specified when submitting the execution request

Parameterized SQL statements

Improved security and reusability

```
POST /api/2.0/sql/statements HTTP/1.1
Host: <base_HOST>
Authorization: Bearer <personal_access_token>
Content-Type: application/json
{
    "statement": "SELECT * FROM stores WHERE store_id = :store_id",
    "warehouse_id": "<warehouse_ID>",
    "parameters": [
       {
        "name":"store_id",
        "type":"INT",
        "Value": 1234
       }
   ]
}
```

Build a data app

Acme, Inc’s Data API

Manage stores and their sales

Get all stores

```
GET /stores
```

→

```
state: “SUCCEEDED”
stores: [[“123”, “Acme, Inc”, …], [“456”, “Databricks”, …], …]
```

Get sales for a store

```
GET /stores/<store_id>/sales
      request_id: Optional[token]
      limit: Optional[int]
      format: “CSV”
```

→

```
request_id: “ID456”
state: “PENDING”
links: Optional[Array]
```

Create new sale

```
POST /stores/<store_id>/sales
      date: “2023-06-29”
      quantity: 10
      price: 2.50
      item_id: 1234
```

→

```
sale_id: “ID789”
```

List Stores

Synchronous mode, inline small data

Acme Inc's API Request

```
GET /stores
```

→

SQL Statement Execution API Request

```
POST /sql/statements
      statement: “SELECT * FROM stores”
      wait_timeout: “50s”
      on_wait_timeout: “CANCEL”
```

Acme Inc's API Response

```
state: “SUCCEEDED”
stores: [
  [“123”, “Acme, Inc”, …],
  [“456”, “Databricks”, …]
]
```

←

SQL Statement Execution API Response

```
statement_id: “ID123”
status: { state: “SUCCEEDED” }
manifest: { ... }
result: {
  data_array: [
    [“123”, “Acme, Inc”, …],
    [“456”, “Databricks”, …]
  ]
}
```

Download Sales for a Store

Asynchronous mode, large data with external links

Acme Inc's API Request

```
GET /stores/123/sales
      format: “CSV”
```

→

SQL Statement Execution API Request

```
POST /sql/statements
      statement: “SELECT * FROM stores where store_id = :store_id”
      parameters: [
        { name: “store_id”, value: “123”, type: “INT” }
      ]
      disposition: “EXTERNAL_LINKS”
      wait_timeout: “0s”
      on_wait_timeout: “CONTINUE”
```

Acme Inc's API Response

```
request_id: “ID123”
state: “RUNNING”
```

←

SQL Statement Execution API Response

```
statement_id: “ID123”
status: { state: “RUNNING” }
```

Download Sales for a Store

Asynchronous mode, large data with external links

Acme Inc’s API Request

```
GET /stores/123/sales
      request_id: “ID123”
```

→

SQL Statement Execution API Request

```
GET /sql/statements/ID123
```

Acme Inc’s API Response

```
request_id: “ID123”
state: “SUCCEEDED”
links: [
  “https://cloud.store/path/chunk00_abc?token=YYX”,
  “https://cloud.store/path/chunk01_def?token=YYW”,
  “https://cloud.store/path/chunk02_ghi?token=YYZ”
]
```

←

SQL Statement Execution API Response

```
statement_id: “ID123”
status: { state: “SUCCEEDED” }
manifest: { total_chunk_count: 3 }
result: {
  external_links: [
    {
      external_link:
       “https://cloud.store/path/chunk00_abc?token=YYZ”
    }
  ]
}

GET /sql/statements/ID123/result/chunks/1
GET /sql/statements/ID123/result/chunks/2
```

Create a new sale

Hybrid, DML

Acme Inc’s API Request

```
POST /stores/123/sales
      date: “2023-06-29”
      quantity: 10
      price: 2.50
      item_id: 1234
```

→

SQL Statement Execution API Request

```
POST /sql/statements
      statement: “INSERT INTO …”
      Parameters: [ … ]
      wait_timeout: “50s”
      on_wait_timeout: “CONTINUE”
```

Acme Inc’s API Response

```
sale_id: “ID456”
state: “SUCCEEDED”
```

←

SQL Statement Execution API Response

```
statement_id: “ID123”
status: { state: “SUCCEEDED” }
```

Create a new sales order

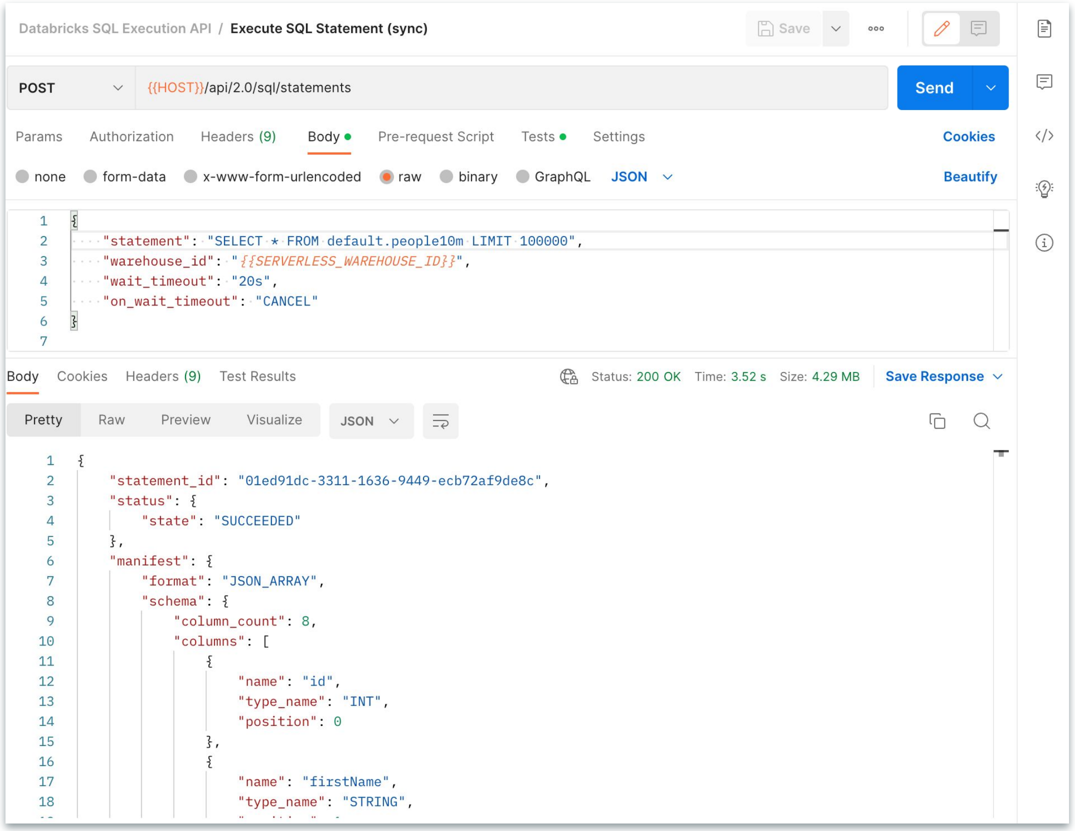
Use parameters for improved security

```
POST /api/2.0/sql/statements HTTP/1.1
Host: <base_HOST>
Authorization: Bearer <personal_access_token>
Content-Type: application/json
{
    "statement":
      "INSERT INTO store_sales (ss_sold_date_sk, ss_ticket_number, ss_store_sk, ss_item_sk, ss_quantity, ss_sales_price)
       VALUES (:sold_date, :sale_id, :store_id, :item_id, :quantity, :sales_price)",
    "parameters": [
       { "name": "sold_date", "type": "DATE", "value": "2023-06-29" },
       { "name": "sale_id", "type": "BIGINT", "value": "1234" },
       { "name": "store_id", "type": "INT", "value": "567" },
       { "name": "item_id", "type": "INT", "value": "890" },
       { "name": "quantity", "type": "INT", "value": "10" },
       { "name": "sales_price", "type": "DECIMAL(7,2)", "value": "2.50" }
   ]
}
```

Further code samples

Check out our Git repo: Postman, cURL, notebooks, etc.

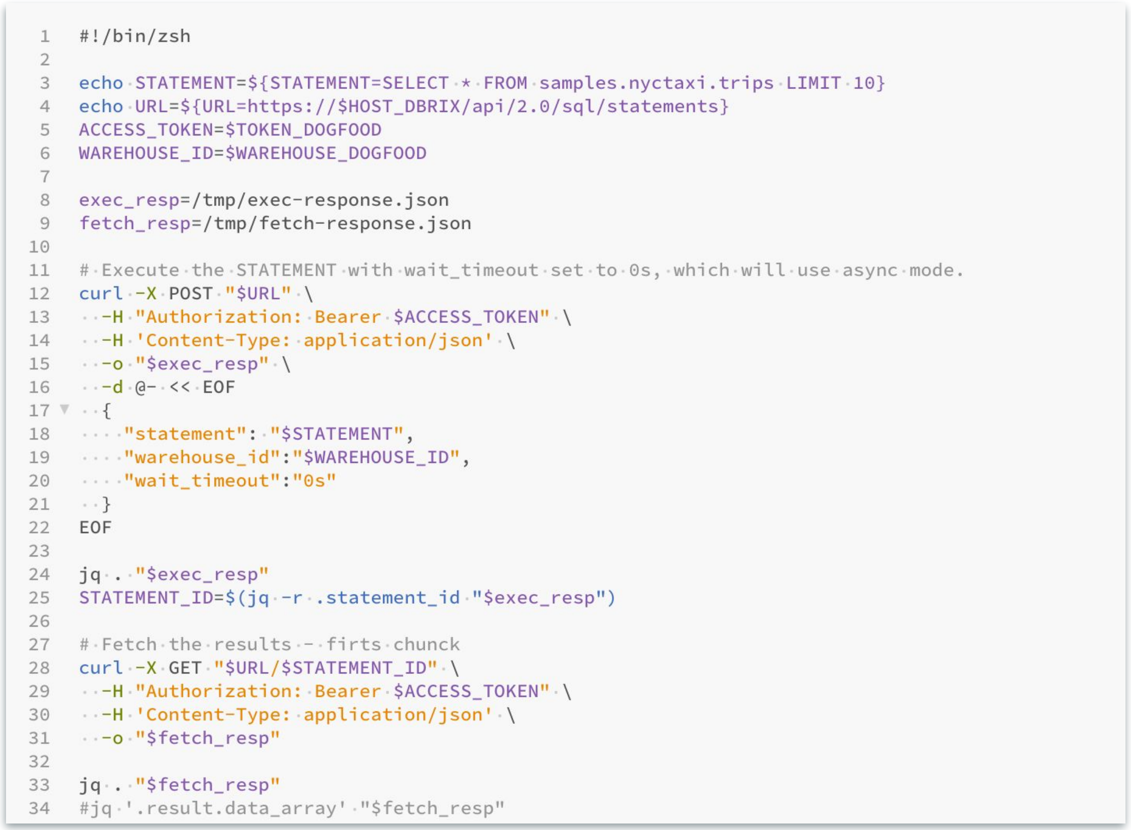
github.com/databricks-demos/dbsql-rest-api



```json
{
    "statement": "SELECT * FROM default.people10m LIMIT 100000",
    "warehouse_id": "{{SERVERLESS_WAREHOUSE_ID}}",
    "wait_timeout": "20s",
    "on_wait_timeout": "CANCEL"
}

```

```json
{
    "statement_id": "01ed91dc-3311-1636-9449-ecb72af9de8c",
    "status": {
        "state": "SUCCEEDED"
    },
    "manifest": {
        "format": "JSON_ARRAY",
        "schema": {
            "column_count": 8,
            "columns": [
                {
                    "name": "id",
                    "type_name": "INT",
                    "position": 0
                },
                {
                    "name": "firstName",
                    "type_name": "STRING",
```



```bash
#!/bin/zsh

echo STATEMENT=${STATEMENT=SELECT * FROM samples.nyctaxi.trips LIMIT 10}
echo URL=${URL=https://$HOST_DBRIX/api/2.0/sql/statements}
ACCESS_TOKEN=$TOKEN_DOGFOOD
WAREHOUSE_ID=$WAREHOUSE_DOGFOOD

exec_resp=/tmp/exec-response.json
fetch_resp=/tmp/fetch-response.json

# Execute the STATEMENT with wait_timeout set to 0s, which will use async mode.
curl -X POST "$URL" \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H 'Content-Type: application/json' \
  -o "$exec_resp" \
  -d @- << EOF
  {
    "statement": "$STATEMENT",
    "warehouse_id":"$WAREHOUSE_ID",
    "wait_timeout":"0s"
  }
EOF

jq . "$exec_resp"
STATEMENT_ID=$(jq -r .statement_id "$exec_resp")

# Fetch the results - firts chunck
curl -X GET "$URL/$STATEMENT_ID" \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H 'Content-Type: application/json' \
  -o "$fetch_resp"

jq . "$fetch_resp"
#jq '.result.data_array' "$fetch_resp"
```

SQL Statement Execution API

Simplified data access using a programming language of your choice

- Removes the need to install drivers and manage Cloud infrastructure, or manage connections
- Allows querying and manipulating data, or defining data objects (DDL, DML, DQL, DCL)
- Allows different execution modes: synchronous, asynchronous, or hybrid
- Allows efficient access to large data sets with EXTERNAL_LINKS
- Leverages authentication options supported by Databricks REST APIs.

Build apps powered by the Lakehouse

Learn more and get started

→ Learn more about our connectors and tools

https://docs.databricks.com/dev-tools/index-driver.html

→ Learn about apps in the marketplace

https://www.databricks.com/blog/introducing-lakehouse-apps

Related talks at Data+AI Summit

Customer Talks

- **Akamai |** Internet-Scale Analytics: Migrating a Mission Critical Product to the Cloud
- **AT&T |** Building and Managing Data Platform for 13+ PB Delta Lake and 1000s of Users: AT&T's Story
- **S&P GLOBAL |** Using Databricks to Power Insights and Visualizations on the S&P Global Marketplace
- **Land O'Lakes |** Self-Service Geospatial Analysis Leveraging Databricks, Apache Sedona, And R
- **American Airlines |** Making Travel More Accessible For Customers Bringing Mobility Devices
- **Collins Aerospace |** Jet Streaming Data and Predictive Analytics: How the Lakehouse and Apache Spark™ Enable Collins Aerospace to Keep Aircraft Flying
- **Banco Bradesco |** Data Democratization with Lakehouse: An Open Banking Application Case
- **Michelin |** Data Democratization at Michelin
- **Zurich Insurance |** dbt Labs | Modernizing the Data Stack: Lessons Learned From the Evolution at Zurich Insurance
- **Rec Room |** How Rec Room Processes Billions of Events Per Day with Databricks and RudderStack
- **RaceTrac Inc. |** Unlocking the Power of Real-Time Data to Maximize Data Insights
- **dbt Labs |** Modernizing the Data Stack: Lessons Learned From the Evolution at Zurich Insurance

Related talks at Data+AI Summit

Product Deep Dive

- **Databricks |** Databricks SQL: Why The Best Serverless Data Warehouse Is A Lakehouse
- **Databricks |** What's New In Databricks SQL -- With Live Demos
- **Databricks |** Databricks SQL Serverless Under the Hood: How We Use ML to Get the Best Price/Performance
- **Databricks |** Best Practices For Setting Up Databricks SQL At Enterprise Scale
- **Databricks |** Building Apps on the Lakehouse with Databricks SQL
- **Databricks |** Unlock The Next Evolution Of The Modern Data Stack With The Lakehouse Revolution -- With Live Demos
- **Databricks |** Unleashing Large Language Models with Databricks SQL's AI Functions
- **Databricks |** Under the Hood: Intelligent Workload Management
- **Databricks |** Going Beyond SQL: Python UDFs in Unity Catalog for all your Lakehouse

databricks