Axon DataStore v2 User Guide

Rev: 27 Jan 2026

Axon Enterprise, Inc.
17800 N 85th St
Scottsdale AZ 85255
USA

Contents

What's New?

This guide includes the following updates:

- Fine-grained access control using default and custom access profiles

Axon DataStore v2

The Axon DataStore offers a structured and performance-optimized platform for accessing Axon Records and Axon Standards data. This section covers key improvements over DataStore v1, including enhanced data modeling, automatic view updates, incremental data extraction, and support for Master Indices. It also outlines access control options, connectivity requirements, disaster recovery planning, integration guidance, and recommendations for reliable querying and dashboard development.

Introduction

The Axon DataStore is a read-only Azure SQL database designed to support reporting and analytics for Axon Records and Axon Standards. It provides structured access to Axon Records and Axon Standards data through SQL views, powers Axon's in-product Analytics dashboards, as well as allows seamless querying and reporting with third-party tools. When your organization's personnel use Axon Records and Axon Standards in the normal course of operations, data is synchronized to the DataStore where it is processed and presented in SQL views.

You can use the DataStore to support varying data needs, including:

- Building analytics reports with tools like Power BI, Crystal Reports, and Tableau
- Running ad hoc queries for deeper insights
- Transferring data into data warehouses
- Integrating with downstream systems

The original DataStore (DataStore v1) has received detailed feedback highlighting long-standing issues, including lack of clear entity relationships, delays in accessing JSON-extracted columns, slow query performance with JSON operators, and difficulty in identifying and loading incremental data changes. To address these issues, Axon is introducing a new DataStore version 2 that improves usability and understandability, performs faster, and allows for direct data access.

The following sections outline the improvements in DataStore v2.

1. Structured data access with clear entity relationships

In DataStore v1, the data is either obscured in complex JSON blobs or relationships between views are not easily understandable, making it difficult to query the needed data.

DataStore v2 structures all data fields in an explicitly declared tabular format with clear entity relationships that can be directly accessed via standard SQL queries. Because all data fields are pre-extracted into columns, the DataStore v2 does not provide raw JSON blobs. Queries do not require extracting or referencing values stored in JSON blobs, which:

- Makes specific data fields easier to find
- Eliminates the performance hit incurred by performing data extraction from inside of JSON blobs

2. Data Model automatically derived from application forms

DataStore v2 structures all data fields in an explicitly declared tabular format that follows the structure and naming convention of the forms that organization personnel interact with in Axon Records and Axon Standards. This helps IT personnel and data analysts better understand the source of data in any given table, reducing the risk of errors.

3. Automatic table updates when forms change

With DataStore v1, you had to either wait for Axon Support to create custom views (which could take several weeks), or resort to JSON extraction from raw data to query data related to any form changes. In DataStore v2, structural changes for forms will automatically reflect in the DataStore as follows:

- Columns are added for any new fields added to forms.
- Columns are retained for fields removed from forms if data exists in any row.
- Columns are removed for any completely empty columns that correspond to fields removed from forms.
- New views are added for net-new custom forms.

4. Reliable, incremental data extracts

The lack of incremental data-pulling support in DataStore v1 required you to track changes across multiple views or frequently replace large datasets, creating inefficiencies and unnecessary strain on both your systems and the DataStore.

To address this, the DataStore v2 supports industry-standard Change Data Capture (CDC) mechanisms based on incremental data-pulling, providing a transaction record that is faithful to all transactions performed in the database over a defined time period, including **`DELETE`** operations. This provides a well-defined interface where you can detect any changes in the DataStore and reliably build your data extraction pipelines.

5. Standardized incident reporting and Master Name Indices

With DataStore v1, assembling complete and accurate data often proved challenging. Retrieving the latest incident information required piecing together multiple rows, each representing a different version of the same report. Similarly, the absence of a Master Indices (MxI) forced you to manually extract and compile identities from related reports, leading to time-consuming, inconsistent, and inefficient processes.

DataStore v2 streamlines reporting by providing structured identity tracking and incident reporting. Master Indices (MxI) eliminate the need for manual identity extraction, offering a pre-built, standardized, and up to date list for persons, vehicles, organizations, and locations. The **`LatestIncidentReport`** view consolidates the most recent version of an incident into a single row, reducing complexity. These enhancements improve accuracy, minimize manual effort, and simplify data analysis.

6. Self-Service access control

The "all-or-none" access model in DataStore v1 was often frustrating, since users either had full access or none at all. Managing access also required relying entirely on Axon Support, leaving administrators unable to see who had active credentials or to grant or revoke access themselves. This lack of visibility and control made it harder to enforce security policies, onboard new users efficiently, and remove access when necessary.

DataStore v2 introduces a more secure and streamlined access control mechanism, enabling administrators to manage credentials independently without relying on Axon Support. Administrators can generate and revoke access in a self-service manner, eliminating the need for manual requests. Additionally, by the end of 2025, you will be able to implement fine-grained access controls, restricting user access to specific views to ensure data security and compliance.

Uptime and updates

Axon guarantees a minimum of 99.9% uptime for the Axon solutions 7 days per week on a 24-hour basis, apart from scheduled downtime, scheduled maintenance, and emergency maintenance.

Axon Records and Axon Standards are designed and operated as highly available cloud applications. Multiple redundant components are used throughout the system architecture to ensure high levels of reliability.

Data freshness

Data entered into Axon Records or Axon Standards reports does not appear in the DataStore in real-time. Instead, it appears in the DataStore after 15 to 30 minutes.

DataStore releases

DataStore code updates are performed on a 2-week cycle. Release notes for view changes are added to the Axon Records and Axon Standards release documentation and can be found on Axon Help.

Generally, changes will only add columns or make performance improvements and will not break an existing view or column. A breaking change is typically considered one where a column is removed or a column data type is changed.

Note

If a breaking change is introduced to the DataStore, that change will be first announced in release notes with a published date in the future for when the change will take place.

The release notes will also include the appropriate replacement for what should be used instead. The replacement will generally be available at the same time the breaking change is announced so you can start making changes right away.

Recommendations

In future releases of the Axon Records and Axon Standards DataStore, Axon may augment the definition of a view by adding columns to the end of the column list. We recommend that you do NOT use the syntax **`SELECT * FROM <view name>`** in production code. This syntax will pull more data than necessary and slow the performance of your query. Additionally, because the number of columns returned might change, this syntax could well break your application.

We also recommend that you do NOT use ordinal positioning in production code as there is no guarantee that column ordering will remain the same. To avoid any issues, use column names in your production code.

Disaster recovery

In the event of a major disaster that results in a full loss of a Microsoft Azure region, Axon has created the Axon Cloud Services Information System Contingency Plan (ISCP). The ISCP focuses on the recovery of Axon Records and Axon Standards to a secondary Microsoft Azure region.

Axon is confident that in the event of the complete destruction of a primary Microsoft Azure region, the Axon application services can be recovered and restored in the secondary Microsoft Azure region within, at most, a 24-hour window. However, Axon views the likelihood of such an occurrence as negligible, given the architecture of the underlying Microsoft Azure services.

Firewall requirements

Connectivity to the Axon DataStore requires that you adjust your organization's IP restrictions and network requirements.

IP restriction

All IP traffic to the DataStore is blocked by default. To access the DataStore, your organization's **public** IP address(es) must be added to the allow list. You can manage this allow list from the **Access Policy** tab in the DataStore Settings tool.

Note

The DataStore uses IPv4 (Internet Protocol Version 4) and does NOT currently support IPv6 (Internet Protocol Version 6).

Public vs private IP addresses

When submitting your request for Firewall Access, ensure the IP is a public IP and not a private IP. If your organization uses a private network, you can identify your public IP using any online service such as https://www.whatismyip.com/ or https://www.showmyip.com/.

What are private IP addresses?

Private IP addresses are a subset of IP addresses designated for use within private networks. These addresses are not routable on the public internet, meaning they cannot be used to communicate directly with devices outside the local network, such as the Axon DataStore.

How do I recognize a private IP address?

Private IP addresses are divided into three classes, as defined by the Internet Engineering Task Force (IETF) in RFC 1918:

Class	Range	Prefix	Usage
A	10.0.0.0 to 10.255.255.255	10	Large organizations and enterprises
B	172.16.0.0 to 172.31.255.255	172	Schools, universities, and businesses
C	192.168.0.0 to 192.168.255.255	192	Small office or home networks

Network requirements

Your organization's firewall must allow outbound traffic to the Azure Gov SQL IP Ranges on port 1433.

Refer to the following links for a weekly updated list of the region's SQL IP addresses that your organization should allow:

- Azure IP Ranges and Service Tags - US Government Cloud: US-based organizations
- Azure IP Ranges and Service Tags – Public Cloud: Non-US based organizations

Use the "Sql" Service Tag to make this easier to manage.

Tips

- You can detect updates from one publication to the next by noting increased changeNumber values in the JSON file. Each subsection (e.g., Storage.WestUS) has its own changeNumber that is incremented as changes occur. The top level of the file's changeNumber is incremented when any of the subsections is changed.
- For examples of how to parse the service tag information (e.g., get all address ranges for Storage in WestUS), see the Service Tag Discovery API PowerShell documentation.
- When new IP addresses are added to service tags, they will not be used in Azure for at least one week. This gives you time to update any systems that might need to track the IP addresses associated with service tags.
- You can also ensure connectivity by using Azure's Fully Qualified Domain Name (FQDN), using the server provided in the DataStore access control tool.
  - For information on how to set up access control, please submit a ticket in the Axon Support portal or reach out to softwaresupport@axon.com.

Connect to the DataStore

You can connect to the DataStore using Visual Studio Code or Microsoft SQL Server Management Studio.

Warning

Previously, you could connect to the DataStore using Azure Data Studio. However, this program is being deprecated in February 2026. To continue to access the DataStore, use either VS Code or Microsoft SQL Server Management Studio.

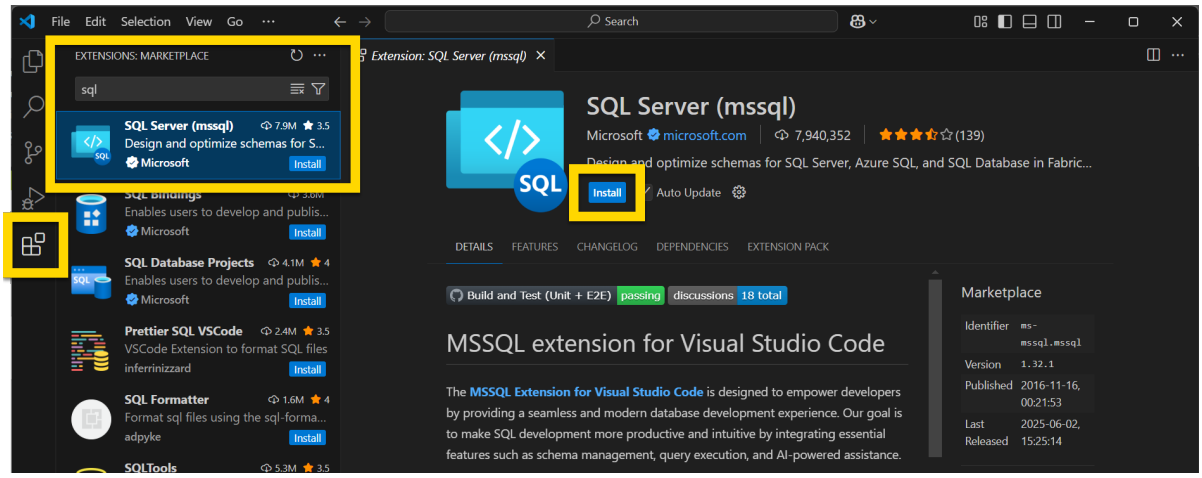
You can generate a DataStore account using the DataStore Secret Generation tool in the Administrator Console.

If you have questions about connecting to the DataStore, submit a ticket using the TSS portal or email softwaresupport@axon.com.

Connect with Visual Studio Code

To connect to the DataStore using VS Code, take these steps:

1. Open VS Code.
   - VS Code is a free software that can be downloaded here.
2. Select **Extensions** in the side menu.
3. Enter "sql" in the search box and select **SQL Server (mssql)**.
4. Select **Install**.

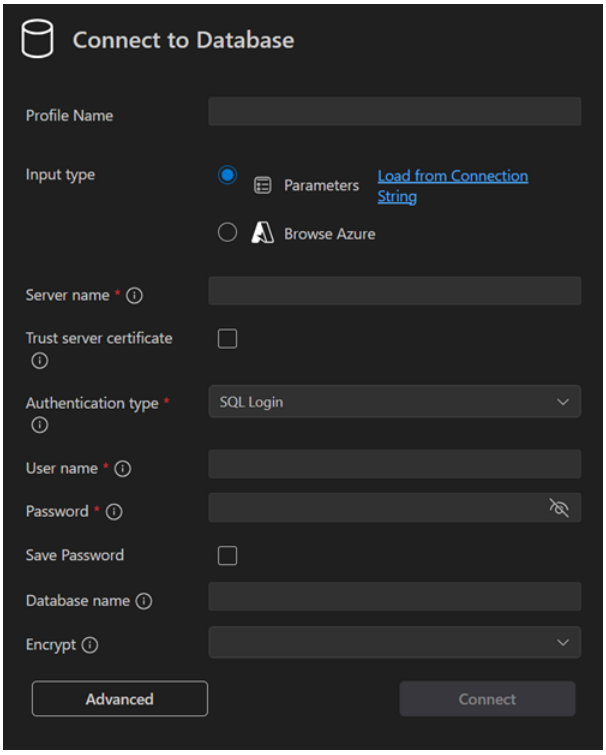


5. After a successful install, **SQL Server** will appear as an option in the side menu. Select this option.

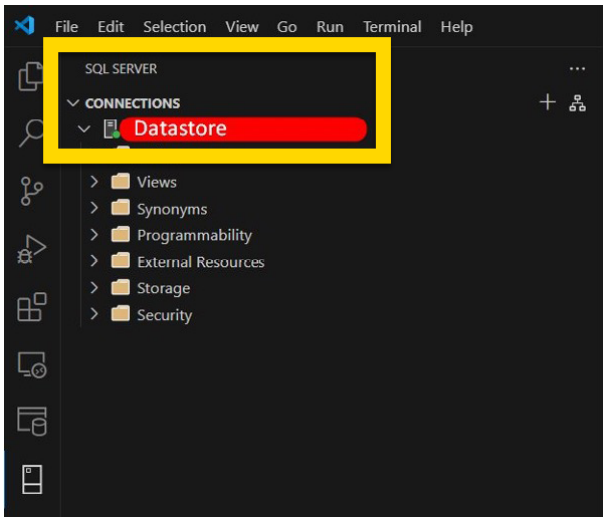
6. Select **Add Connection**.

7. Complete the following fields in the Connect to Database window:
   - **Profile name:** This name will appear in your list of available connections.
     - Recommended format: “Axon [Records or Standards] [PROD or Training] [PW or Query] Datastore”
       - Example: Axon Records PROD PW Datastore
   - **Sever name:** The name of the server you are connecting to
   - **Authentication type:** SQL Login
   - **User name**
   - **Password**
   - **Save Password:** Check this box to avoid entering your password each time you use VS Code.
   - **Database name:** The name of the database you are connecting to

8. Select **Connect**.



9. Following a successful connection, the Profile name will appear in your Connections list.

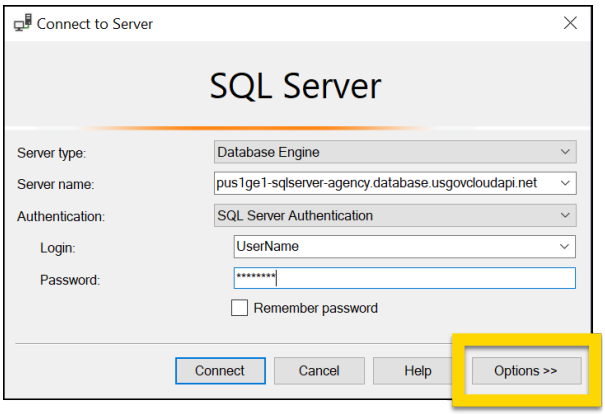


Connect with Microsoft SQL Server Management Studio

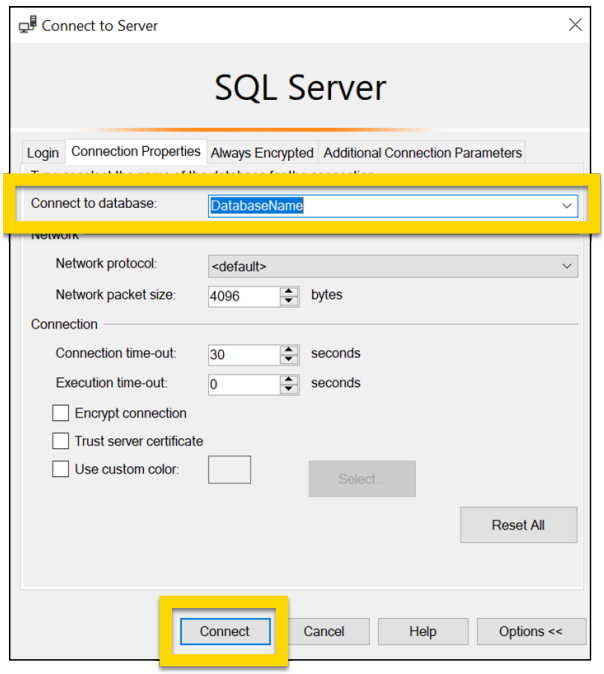
To perform queries using Microsoft SQL Server Management Studio, take these steps:

1. Open Microsoft SQL Server Management Studio.
2. A secondary Connect to Server will open.

3. Enter the Server Name, Username, and Password
4. Select **Options**.



5. Enter the Database name in the **Connect to database** field.
6. Select **Connect**.



DataStore integrations

Connecting the DataStore to an external product is possible with any database compatible software. The DataStore can be connected via user or service accounts, using a SQL Server or ODBC connection. Axon Support can provide basic instructions on how to integrate the DataStore with your existing reporting software.

Note

If you integrate the DataStore with external products, it is your organization's responsibility to build and maintain any external reports or analytics.

Axon Analytics and Power BI

The Analytics module in Axon Records and Axon Standards uses Power BI to visualize data. The video below provides a comprehensive overview of setting up a dashboard in Power BI, from data import to final customization and filtering.

This video covers the following topics:

- Importing data into Power BI
- Preparing and transforming data
- Building and customizing visualizations
- Customizing dashboard layouts
- Using slicers for data filtering

Once a Power BI dashboard has been created, it can be uploaded into the Analytics module in Axon Axon Records and Axon Standards. The following video uses the Use of Force dashboard in Axon Standards to explain how users can interact with the Power BI dashboards and visualizations.

This video covers the following topics:

- Using tab and visualization filters
- Exploring map visualizations
- Viewing tool tips
- Interacting with tables and hyperlinks
- Dashboard privileges
- Activity log tabs
- Editing and exporting dashboards

Concepts and features

This section introduces the core structure and concepts behind DataStore v2, which powers advanced reporting and analytics for Axon Records and Standards. It explains how tables are named and organized, the roles of entity and relation types, how to join across tables using primary and foreign keys, and how time zones and data types are handled. You'll also find guidance on versioning, schema evolution, command hierarchy modeling, and how to replicate or query your data using DataStore's flexible, read-only architecture.

Naming conventions

The Data Model V2 uses a consistent global naming convention for all tables to improve clarity and organization. Table names are composed of multiple parts written in PascalCase and joined by underscores (_). Each part may include one of the following prefixes to indicate the table type:

- **Ent**: Entity
- **Rel**: Relationship
- **Sub**: Sub-document

For example, in a table named **`IncidentReport_EntPerson`**,

- **`IncidentReport`**: Parent table
- **`EntPerson`**: Related child table

Additionally, prefixes appear at the start of view and column names to indicate the source of the data:

- **Axon**: Data from a generic (standard) form or field
- **Custom**: Data from a custom form or field

These naming rules ensure consistency across the DataStore and help you quickly understand the structure and source of each table and field.

Schemas

In DataStore V2, all tables are accessed through base schemas, with each product—such as Axon Records, Axon Standards, or Personnel—having its own dedicated schema. For example, all base tables for Axon Records are found in the **`records.*`** schema.

Base schemas are implemented as views that have a one-to-one mapping with their corresponding physical tables. For instance, **`records.*`** views map directly to tables in the **`dboRecords.*`** schema. For simplicity, this document uses the terms *base schema* and *physical table* interchangeably.

In addition to base schemas, the following schemas are available for managing reusable queries and supporting analytics:

- **`axonV2.*`**: Used to store frequently used queries.
  - To store a query in **`axonV2.*`**, contact Axon Support.
- **`analyticsV2.*`** Used to store queries that power datasets on analytics dashboards.

These schemas help organize data access and support efficient query reuse across various use cases.

Table types

The following table types appear in DataStore v2:

- Report (or Document) tables
- Embedded (or Child) tables
- Entity tables
- Relation tables
- SubDocument tables
- Array tables

Report (or Document) tables

In DataStore V2, each report type—whether out-of-the-box reports (e.g., Incident Report, Field Interview) or custom forms created in Form Builder—has its own dedicated table. The table name matches the form name defined in Form Builder and does not include any prefixes. Each report table name consists of a single part, such as:

- **`Records.IncidentReport`**
- **`Records.FieldInterview`**

This is a key difference from DataStore V1, where all report data was stored in a single view, **`axon.Reports`**, and individual report types were distinguished using the **`ReportTypeName`** column.

By separating report types into individual tables, DataStore V2 improves clarity, simplifies querying, and aligns more closely with how reports are defined and used.

Embedded (or Child) tables

An embedded table is a child table that stores data directly related to a specific parent table, preserving a one-to-many relationship between them. Embedded tables typically contain detailed or nested information that extend the parent table's data model.

Embedded table names use the parent table as a prefix to clearly indicate the relationship between the two tables.

For example, in a table named **`IncidentReport_EntPerson`**:

- **`IncidentReport`**: Parent table
- **`EntPerson`**: Embedded table

This relationship is maintained through a Primary Key (PK) in the parent table and a Foreign Key (FK) in the embedded table. See Identifiers for additional details on PK/FK mappings.

An embedded table can belong to one of the following types:

- Entity table
- Relation table
- SubDocument table
- Array table

Entity tables

An entity table represents a core entity such as an Incident, Case, Person, Vehicle, or Offense. These tables follow a naming convention with the **`Ent`** prefix, such as **`EntPerson`**.

There are two types of entity tables in DataStore V2:

1. **Embedded entity table:** Captures a snapshot of the entity as it appeared at the time the report was written
   - This is a child table nested under a report table (e.g., **`IncidentReport_EntPerson`**).
   - Useful for historical context and audit trails.
2. **Master entity table:** Contains the most recent version of the entity, updated based on the latest report in the system
   - This is a standalone table (e.g., **`EntPerson`**) that is not embedded within a report table.
   - These are also referred to as MxI (Master Indices) tables.

Example scenario (illustration only)

Consider a person involved in two different offenses documented three years apart. In the first report, the person is described as having brown hair, while in the second report, they are described as having black hair.

- The **`EntPerson`** table (master entity) reflects the latest information about this person—black hair.
- The **`IncidentReport_EntPerson`** table (embedded entity) from the first report retains the original information—brown hair.

This structure ensures both the most current and historical representations of an entity are preserved and accessible.

Relation tables

A relation table defines the relationship between two entities or between a report and an entity. These tables are essential for modeling many-to-many relationships. Relation tables follow a naming convention that uses the **`Rel`** prefix—for example, **`RelCaseIncident`**.

Each relation table includes two key columns:

- **`FromId`**: The ID of the first entity or report in the relationship.
- **`ToId`**: The ID of the second entity or report.

The order of these fields follows the sequence in the table name. For example, in **`RelCaseIncident`**:

- **`FromId`**: Case ID
- **`ToId`**: Incident ID

DataStore V2 supports two types of relation tables:

1. **Embedded relation table:** Represents relationships between two embedded entities within the same report
   - Example: **`IncidentReport_RelPersonToOffense`** shows how a person was involved in a specific offense documented in an incident report.
2. **Master relation table:** Represents relationships between two master entities (e.g., **`RelCaseIncident`**)
   - This can also represent relationships between a report and a master entity (e.g., **`RelIncidentDocument`**).

SubDocument tables

A SubDocument table stores supplementary information related to a primary report. These tables capture additional structured data that extend the main report's content and are especially useful for specialized forms or modules.

SubDocument tables follow a naming convention that includes the **Sub** prefix. For example, in **`IncidentReport_SubDomesticViolence`**:

- **`IncidentReport`**: The primary report
- **`SubDomesticViolence`**: The Domestic Violence sub-form within the primary report

Array tables

An array table is used to store fields that contain multiple values or lists of objects with their own attributes. These are typically generated from multi-select fields (e.g., checkboxes) or nested repeatable sections within a report or entity.

Each value or object in the list is stored as a separate row in a child table that maintains a one-to-many relationship with the parent table using a Primary Key (PK) and Foreign Key (FK).

Examples

- In the Offense section of an Incident Report, multiple selected values for Bias Motivation are stored in the child table **`IncidentReport_EntOffense_AxonBiasMotivationInvolved`**. Each selected value appears as a separate row, with the value stored in the **`Key`** column.
- A person's multiple Markings (e.g., tattoos, scars) are stored in the child table **`EntPerson_AxonMarkings`**, with one row per marking.

Array tables can also represent arrays of arrays, where a list of objects contains nested lists or additional multi-value fields. Each child table name extends its parent table name using underscores (_), reflecting the data hierarchy.

Identifiers

Understanding how identifiers work in DataStore V2 is key to navigating table relationships and querying data effectively. This section explains the use of the following identifiers across different table types:

- Primary keys
- Foreign keys
- Other ID columns

Primary keys

Primary keys are defined differently depending on the table type:

- For most tables, the primary key is the **`Id`** column.
- For embedded entity, subdocument, and embedded relation tables, the primary key is a composite of (**`Id`**, **`ReportId`**).
- For master relation tables, the primary key consists of (**`FromId`**, **`ToId`**), representing the source and target entities or reports in the relationship.

In DataStore v2, the **`Id`** field serves as a globally unique identifier—equivalent to the **`ExternalId`** in v1. It uniquely identifies reports, entities, and subdocuments across products like Axon Records and Axon Standards.

In the embedded entity table the **`Id`** field will always be the Entity Id. For example:

- **`IncidentReport_EntPerson.Id`**: Person Id
- **`IncidentReport_EntVehicle.Id`**: Vehicle Id

Note

In array tables, the **`Id`** is auto-generated by THE DataStore and is only meaningful within the context of DataStore itself—it is not globally unique or externally meaningful.

Foreign keys

DataStore V2 maintains Parent Key / Foreign Key (PK/FK) relationships between:

- Master tables (e.g., Reports, Master Entities, Master Relation Tables)
- their corresponding child and grandchild tables (e.g., embedded entities, array tables, subdocuments, array-of-array tables)

This structure ensures hierarchical integrity and enables recursive navigation through complex, nested datasets.

Other ID columns

Each entity and report table, along with all of their child (embedded or array) tables, includes a set of standard ID columns used to link related data across the schema:

- **`ReportId`**: A UUID that uniquely identifies a report (formerly **`ExternalId`** in V1).
  - This column is present in all child tables of a report and is used to join data across embedded entities, sub-documents, and array tables associated with the same report.
  - The **`Id`** in the root report table is equivalent to **`ReportId`**.

- **`<Entity>Id`** (e.g., **`ArrestId`**, **`IncidentId`**): A UUID that uniquely identifies an entity (also formerly **`ExternalId`** in V1).
  - This column is available in all child tables of the same entity and is used to group or join related records.
  - The **`Id`** in the root embedded entity table is equivalent to the corresponding <Entity>Id. For example, **`Id`** in **`IncidentReport_EntArrest`** is equivalent to **`ArrestId`**.
- **`<Entity>FriendlyId`**: A user-friendly, organization-wide unique identifier corresponding to **`<Entity>Id`** (e.g., **`ReportFriendlyId`**, **`IncidentFriendlyId`**).
  - This value appears in Axon Records and Axon Standards, including in URLs, links, and titles. It replaces fields such as **`IncidentNumber`** and **`ReportNumber`** from V1.
- **`ParentId`**: In array tables, this column serves as a foreign key that links each row to its parent entity or report.
  - It helps maintain the hierarchical relationship between parent and child records.

Summary of table types, and their primary and foreign keys

Table type hierarchy	Example	Primary Key	Foreign Key
Report (or Document) table	`IncidentReport`	`Id`	N/A
Array, array of array table of Report (or Document)	`IncidentReport_AxonParties`	`Id`	`ParentId`
Embedded Entity table	`IncidentReport_EntPerson`	`Id, ReportId`	`ReportId`
Array, array of array table of embedded Entity	`IncidentReport_EntPerson_AxonMarkings`	`Id`	`ParentId, ReportId`
Embedded SubDocument table	`IncidentReport_SubDomesticViolence`	`Id, ReportId`	`ReportId`
Array, array of array table of SubDocument	`IncidentReport_SubDomesticViolence_AxonRestrainingOrder`	`Id`	`ParentId, ReportId`
Embedded Relation table	`IncidentReport_RelPersonToOffense`	`Id, ReportId`	`ReportId, FromId, ToId`
Master Entity table	`EntPerson`	`Id`	N/A

Table type hierarchy	Example	Primary Key	Foreign Key
Array and array of array table of Master Entity	`EntPerson_AxonMarkings`	`Id`	`ParentId`
Master Relation table	`RelCaseIncident`	`FromId, ToId`	N/A

Column data types

In addition to identifier columns—such as **`Id`**, **`<Entity>Id`**, **`<Entity>FriendlyId`**, and **`ParentId`**—each base view contains columns that store single-value fields describing reports, entities, relationships, or metadata. Below is a summary of supported data types:

- **nvarchar(100)**
  - Used only for ID columns (**`Id`**, **`FromId`**, **`ToId`**, etc.).
  - Character limit is enforced to support indexing for query performance.
- **nvarchar(max)**
  - Used for all string-type columns (e.g., names, descriptions, addresses).
  - Supports flexible string lengths as defined in the UI.
- **int**
  - Used only for incremental ID columns, typically internal counters.
- **decimal**
  - Used for all numeric fields, including monetary values and measurements.
- **bit**
  - Used for boolean (True/False) fields (e.g., **`IsDeleted`**, **`IsPrimary`**).

Datetime fields and time zone handling

Timestamp data type

In DataStore V2, all timestamp fields use the **`datetime2`** data type, and all values are stored in UTC to ensure consistency across systems and time zones.

Common timestamp fields include:

- **CreatedAt**: When the report, entity, or relation was first created in Axon Records or Standards
- **UpdatedAt** : When the report, entity, or relation was last modified in Axon Records or Standards
- **LastUpdated**: When the data was last ingested or updated in the DataStore

Time zone conversions

For optimal performance when filtering by date, always compare against UTC-stored values. Use **`AT TIME ZONE`** to convert your local datetime to UTC before comparing.

Recommended query pattern

```
WHERE r.CreatedAt >= CONVERT(datetime2, '2024-01-01T00:00:00.000') AT TIME ZONE 'Central
Standard Time' AT TIME ZONE 'UTC'
```

How it works

1. The input value is first converted to Central Standard Time (CST): **`2024-01-01 00:00:00 -06:00`**.
2. That CST value is then converted to UTC: **`2024-01-01 06:00:00 +00:00`**.
3. The UTC result is compared against the **`CreatedAt`** column, which is indexed and stored in UTC—yielding the most efficient query performance.

Common time zones

The commonly used time zones include:

- Eastern Standard Time
- Central Standard Time
- Mountain Standard Time
- US Mountain Standard Time (recommended for Arizona)
- Pacific Standard Time

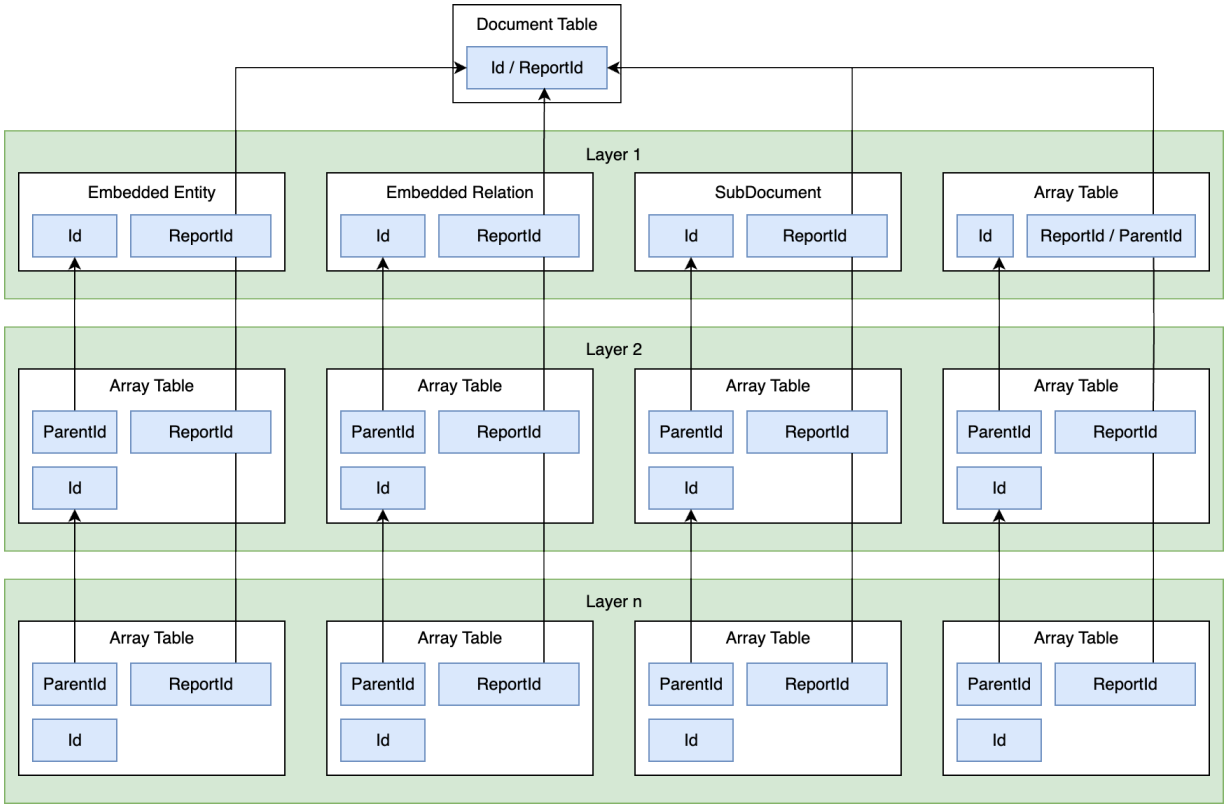
For a full list of valid time zone names, refer to Microsoft's documentation here.

Joining

The following sections explain how to join in DataStore v2 across the different table types:

- Joining across report and child tables
- Joining master entity tables with their array tables
- Joining using master relation tables

Joining across report and child tables



When working with report tables and their child structures in DataStore V2, it's important to follow consistent join patterns to ensure data integrity and accurate results.

- **Use `ReportId` to join across child tables of the same report.** The `ReportId` value is consistent across all embedded, sub-document, and array tables linked to a single report. Always use `ReportId` as the join key to connect related data.

Example

```
FROM IncidentReport r
LEFT JOIN IncidentReport_SubDomesticViolence dv
ON dv.ReportId = r.ReportId
```

- **For second-level (or deeper) child tables, use both `ParentId` and `ReportId`** to join the child to its immediate parent and ensure the connection stays within the correct report.

Example

```
FROM standards.UseOfForce R
INNER JOIN standards.UseOfForce_AxonApplicationOfForce A
ON R.ReportId = A.ReportId
INNER JOIN standards.UseOfForce_AxonApplicationOfForce_Force F
ON R.ReportId = F.ReportId AND A.Id = F.ParentId
```

- **When joining between an entity array and an embedded entity, use both the entity's `Id` and the shared `ReportId`** to ensure the entity is correctly scoped to the report.

Example

```
FROM IncidentReport_AxonParties AP
JOIN IncidentReport_EntPerson EP
ON EP.Id = AP.IndividualPerson AND EP.ReportId = AP.ReportId
```

- **For embedded relation views**, the first entity listed in the view name corresponds to **`FromId`**, and the second corresponds to **`ToId`**. These must be joined using both the entity **`Id`** and **`ReportId`** to ensure proper matching within the context of a report.

Example

```
IncidentReport_RelPersonToOffense.FromId = IncidentReport_EntPerson.Id
AND IncidentReport_RelPersonToOffense.ReportId = IncidentReport_EntPerson.ReportId
```

Joining master entity tables with their array tables

Similar to report tables, when joining master entity tables with their child tables, use the corresponding **`<Entity>Id`** (e.g., **`PersonId`**, **`IncidentId`**) consistently across all joins to ensure accuracy. For first-level child tables, joining on **`<Entity>Id`** is sufficient, as it is directly available in both the master entity and its immediate child.

However, when working with second-level child tables or deeper, you must also include the **`ParentId`** in the join condition. This ensures the nested data is correctly linked to its immediate parent within the entity hierarchy.

Example

```
FROM EntPerson p
JOIN EntPerson_AxonContactEmails e
ON e.PersonId = p.PersonId

FROM EntCallForService cfs
JOIN EntCallForService_AxonLocation l
ON l.CallForServiceId = cfs.CallForServiceId
JOIN EntCallForService_AxonLocation_Alias a
ON a.ParentId = l.Id AND a.CallForServiceId = l.CallForServiceId
```

Joining using master relation tables

Master relation tables in DataStore v2 define relationships between master entities and report tables. These tables are especially useful when linking reports to global entities that exist independently of a specific report context.

For example, to connect an Incident Report to its corresponding Incident entity (e.g., to retrieve the incident number, clearance details, etc.), you can use the master relation table **`RelIncidentDocument`**.

This table stores the association between the Incident entity and its related Incident Report(s):

- **FromId**: ID of the entity (e.g., **`EntIncident`**)
- **ToId**: ID of the report (e.g., **`IncidentReport`**)

Example: Join incident report with the incident entity

The following query retrieves key report-level and entity-level details using the **`RelIncidentDocument`** master relation table:

```
SELECT ir.ReportFriendlyId, ir.AxonIncidentFromDate, ir.AxonIncidentToDate,
inc.IncidentFriendlyId, inc.AxonClearedExceptionally, inc.AxonClearanceDate,
inc.AxonUcrDisposition

FROM Records.IncidentReport ir
JOIN Records.RelIncidentDocument er ON er.ToId = ir.Id
JOIN Records.EntIncident inc ON inc.Id = er.FromId
```

Data dictionary

Before writing a SQL query, it’s important to have a clear understanding of the data you need and where it is likely to be stored within the DataStore V2 model.

For example, if you're looking for a person's age, it may be found in the **EntPerson** table—either as part of an embedded entity within a report table (e.g., **IncidentReport_EntPerson**) or in the master entity (MNI) table (e.g., **EntPerson**), depending on whether you need the historical snapshot or the latest record. (See Entity tables for more information.)

Some values, especially multi-select fields or repeatable data, may not be stored directly in the entity or report table. Instead, they may be normalized into an array table (e.g., **EntPerson_AxonMarkings** or **IncidentReport_EntOffense_AxonBiasMotivationInvolved**). Be sure to check for this structure if the column is not present in the base table.

If you're unsure where a specific field is located, use the **DataDictionary** view, which provides a searchable reference for available tables and columns across DataStore V2. This is a valuable tool for identifying the correct data source. Below is the query to get data, and meaning of its returned columns.

```
SELECT *
FROM RECORDS.DataDictionary
```

Column name	Meaning
Product	Indicates which Axon product (module) the view belongs to—either Standards or Records. (Note: "Product" would be more accurately named "Module")
SchemaName	The name of the schema where the view is located (e.g., RECORDS, STANDARDS, or PERSONNELS).
ViewName	The name of the view.
ColumnName	The name of the column within the view.
DataType	The SQL data type of the column (e.g., nvarchar(max), decimal, datetime2).
IsPrimaryKey	True if the column is a *Primary Key*, otherwise False.
IsIndexed	True if the column is indexed, otherwise False.
IsForeignKey	True if the column is a *Foreign Key* (i.e., part of a child table referencing a parent), otherwise False.
ReferenceTable	If the column is a foreign key, this indicates the *parent table* it references.
ReferenceColumn	The *parent column* in the referenced table (usually a primary key) that this foreign key points to.
RootParentView	The name of the *ultimate parent table* (e.g., document, master entity, or master relation) that the current view is descended from.

Column name	Meaning
`RootParentViewType`	The type of the root parent table (e.g., Document, Entity, Relation).
`ViewType`	The type of the *current table* (e.g., SubDocument, Array, Entity, Relation, etc.).

Latest incident report

In Axon Records, once an incident report is finalized, it becomes read-only and cannot be edited. Any changes or corrections must be made by submitting a supplement report, which clones the existing report and creates a new version. Each new version receives a numeric suffix (e.g., -2, -3, etc.), forming a version history for the same incident.

In DataStore v2, the **`IncidentReport`** table—along with all of its child tables (e.g., **`IncidentReport_EntPerson`**, **`IncidentReport_EntOffense`**, etc.)—stores every version of an incident report. This includes all supplements, providing a complete version history.

In DataStore v1, incident reports were referred to as general offense reports.

For analytics and reporting purposes, users often only need the most recent version of a report. To support this, DataStore v2 has a dedicated set of views: **`LatestIncidentReport`** and its child views (e.g., **`LatestIncidentReport_EntPerson`**, etc.). These views provide a filtered 1:1 mapping with the **`IncidentReport`** tables, but only include the latest (most current) version of each incident report. This simplifies querying and ensures that analytics reflect the most up-to-date information.

Notes

Versioning support is exclusive to incident reports.

Other report types do not follow a versioning model and always appear as single-version entries in their respective tables.

Schema evolution

DataStore v2 supports automatic schema evolution to keep your data model in sync with form changes in Axon Records and/or Standards.

New report forms and new fields added to existing forms are automatically detected and updated in DataStore within 24 hours. This ensures that newly created forms and fields are queryable without requiring manual intervention.

However, new entities (i.e., new business objects not currently modeled in the schema) require engineering development and will be prioritized according to Axon's product roadmap.

Change log

All schema changes in DataStore v2 are tracked in the **`schema_change_log`** table. You can reference this table to audit when new tables or columns were added, providing transparency and traceability over time.

Schema changes are designed to be backward-compatible. Specifically:

- New tables and columns may be added as part of ongoing schema evolution.
- Existing tables or columns with data will not be modified or deleted, ensuring existing queries remain stable.

In the rare event of a breaking change, Axon will provide advance notice through release notes, with a minimum of 90 days before the change is applied.

Read-Only access and data replication

DataStore v2 provides read-only access to **`dbo`** tables and enables Change Data Capture (CDC) on all tables. This setup gives you the flexibility to replicate DataStore data into your own systems for downstream analytics, warehousing, or backup purposes.

You can choose from several data replication options depending on your infrastructure and technical preferences:

Option 1: Integration tools

Most off-the-shelf data integration tools support replication from SQL Server / Azure SQL using a combination of initial load + incremental CDC sync.

Replication specifications

The following are the key specifications for setting up data replication from DataStore v2 using integration tools:

- **Initial Load**: Replication can begin with a full snapshot of all `dbo` tables. This is performed using read-only access and provides a baseline for future incremental updates.
- **Incremental Sync**: Integration tools can access Azure Change Data Capture (CDC) enabled on all tables to capture incremental changes, including inserts, updates, and deletes.
- **Schema Evolution**: Organizations are responsible for handling schema changes. Depending on the capabilities of the selected tool, this may require manual re-syncing of affected tables when changes are detected.
- **Connectivity**: Tools connect directly to the database using TLS/SSL-secured connections.
- **Access**: Read-only access is granted to `dboRecords`, `dboStandards`, and other supported `dbo` tables required for replication.

Limitations

- Write access is not supported (read-only).
- Some tools require elevated privileges during setup (e.g., sysadmin for enabling CDC or Change Tracking).
- Initial full load may be time-consuming for very large datasets.

Vendor Examples (off-the-shelf tools)

Vendor	Incremental Sync	Connectivity	Documentation
Stitch	Key-based incremental, CDC (log-based)	Direct SQL, SSL/TLS, optional SSH tunnel	Stitch Azure SQL docs
Airbyte	CDC (SQL Server CDC / binlog), Incremental append	Direct SQL, SSL/TLS, SSH tunnel	Airbyte MSSQL docs
Fivetran	Change Tracking (CT), CDC, Binary Log Reader, Teleport Incremental	Direct SQL, SSL/TLS, enterprise features (AlwaysOn, TDE)	Fivetran SQL Server docs

This approach is suitable for most agencies because the setup is straightforward and maintenance is minimal. Once configured, the integration tool handles the ongoing replication reliably.

Option 2: Custom scripting

For advanced users or custom pipelines, you can implement your own script-based solution to handle CDC-based replication.

See Incremental pulling for more information.

Command Hierarchy

The Command Hierarchy in DataStore v2 models your organizational structure as a tree, where each node represents a team, group, or unit.

Each node in the tree has a unique **`HierarchyId`**, a path-like string that encodes its position in the structure. This format allows for intuitive querying of team relationships. For example, with the hierarchy structure below, the **`HierarchyId`** of Patrol Division is **`/1/1/`**.

```
/1/  Headquarters
├── /1/1/  Patrol Division
│   ├── /1/1/1/  North Precinct
│   │   ├── /1/1/1/1/  North Day Shift
│   │   └── /1/1/1/2/  North Night Shift
│   └── /1/1/2/  Bike Patrol Unit
├── /1/2/  Investigations Division
│   ├── /1/2/1/  Special Victims Unit
├── /1/3/  Special Operations
│   ├── /1/3/1/  SWAT
│   ├── /1/3/2/  K-9 Unit
```

To query the full command hierarchy, use the following query:

```
SELECT * FROM Records.CommandHierarchy
```

Example 1: Get all teams under Special Operations

This query returns all teams under the Special Operations unit, such as SWAT, K-9 Unit.

```
SELECT ch.Id, ch.HierarchyID, ch.Name
FROM Records.CommandHierarchy ch
CROSS APPLY
( SELECT HierarchyID FROM Records.CommandHierarchy WHERE Name = 'Special Operations') AS
h
WHERE ch.HierarchyID LIKE h.HierarchyID + '%'
```

Example 2: Get all officers under SWAT

This query retrieves all officers assigned to the SWAT team or any subteams beneath it.

```
SELECT ch.Name AS teamName, o.ExternalId AS OfficerId, o.BadgeNumber, o.FirstName,
o.LastName, o.Username, o.District, o.DistrictDescription
```

```
FROM Records.CommandHierarchy ch
JOIN Records.TeamOfficerRelation team ON ch.Id = team.TeamId
JOIN Records.Officer o ON o.ExternalId = team.OfficerId
CROSS APPLY
(SELECT HierarchyID FROM Records.CommandHierarchy WHERE Name = 'SWAT') AS h
WHERE ch.HierarchyID LIKE h.HierarchyID + '%'
```

Example 3: Get chain of command for Bike Patrol

This query shows all parent teams above the Bike Patrol unit, forming its chain of command.

```
SELECT ch.Id, ch.HierarchyID, ch.Name
FROM Records.CommandHierarchy ch
CROSS APPLY
(SELECT HierarchyID FROM Records.CommandHierarchy WHERE Name = 'Bike Patrol Unit'
) AS h
WHERE h.HierarchyID LIKE ch.HierarchyID + '%'
```

Incremental pulling

You can perform incremental pulling in the Axon DataStore to retrieve only newly added or updated data from DataStore instead of replacing all data. This data sync method improves efficiency and reduces system strain while enabling users to manage data expungement without full deletions.

Generally you will use this in two scenarios: incremental pull and cold start.

Glossary

When using this feature, it's helpful to understand the following terms:

- **Expungement data:** Data completely deleted from DataStore, considered redundant or obsolete
- **CDC:** Change Data Capture, tracks database record changes (`INSERT`, `UPDATE`, `DELETE`)
- **DML:** Data Manipulation Language (`SELECT`, `INSERT`, `UPDATE`, `DELETE`)
- **Sproc:** Stored procedure in SQL
- **UserTableName:** Name of the table in the your Azure SQL database
- **TableName:** Name of the table in the Axon DataStore
- **TablePrimaryKey:** Primary key of a table (e.g., `Id` in IncidentReport or EntIncident)

Recommendations

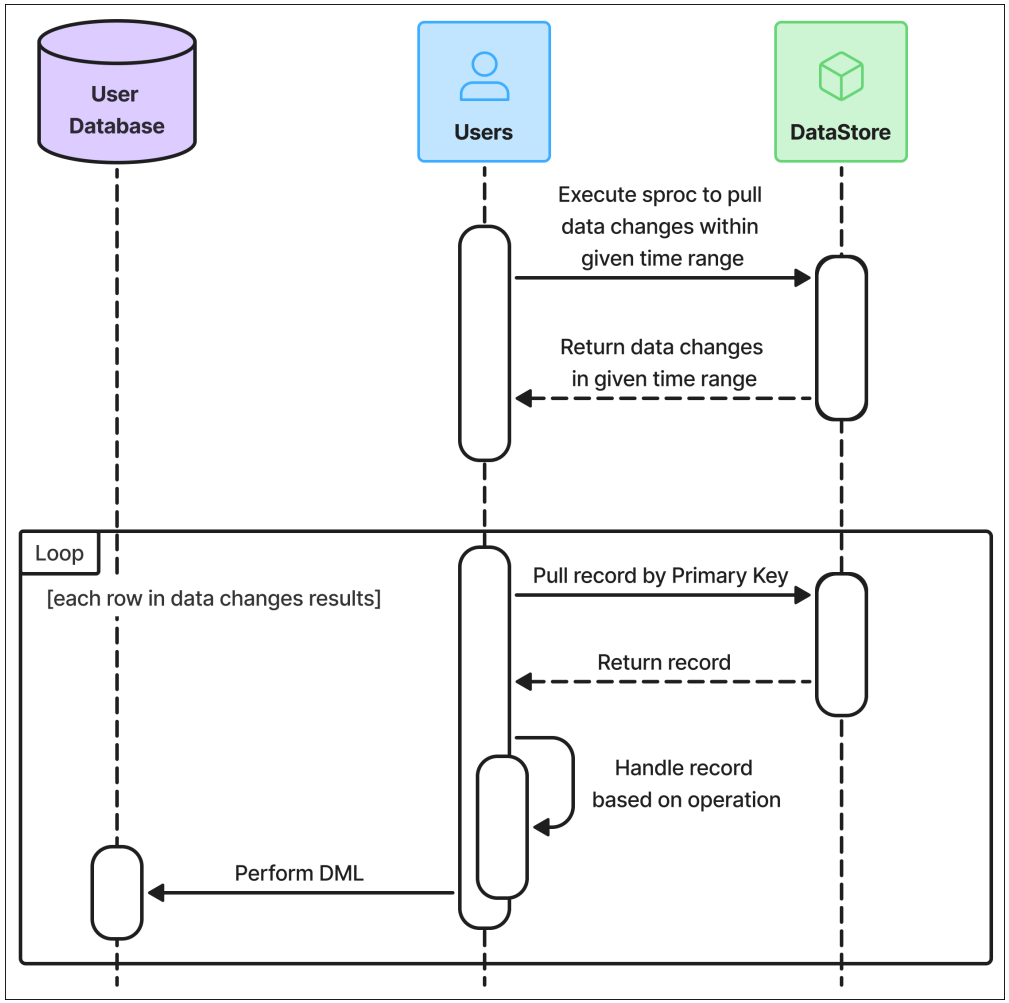
Following these guidelines ensures your organization's incremental pulling works as expected:

- Use the same schema, table name, and column name as DataStore in your Azure SQL database. If these items do not match, you must map the tables between your database and the DataStore.
  - You can find all supported tables and columns by retrieving the information schema.

- When new tables are added to the DataStore in their corresponding label group, CDC is automatically enabled. When made aware of these changes, you should wait to consume new tables until your schema is ready.
- If a record fails, you can pull it again using its primary key.
- Perform an incremental pull after a cold start to handle expungement data.
- Do not place foreign keys on tables. This complicates deletion, requiring child entities to be deleted before the parent is deleted.

Incremental pulling

The Axon DataStore team provides a stored procedure you can execute to retrieve data changes, including primary keys and associated operations. The diagram below illustrates this process:



Retrieve data changes

The first step in performing an incremental pull is to execute the **`axon.usp_PullDataByRange`** stored procedure. This procedure returns a list of data changes for a specific table within a given time range, based on the timestamp when each change was captured in the CDC tables.

```
EXEC axon.usp_PullDataByRange @BeginTime @EndTime @SchemaName @TableName
```

Use the following inputs for this query:

- **`BeginTime:`** The beginning time to start the pull; input in SQL datetime2 format and UTC timezone
- **`EndTime:`** The end time to finish the pull; input in SQL datetime2 format and UTC timezone
- **`SchemaName:`** **`Records`** or **`Standards`**
- **`TableName:`** Specific table name within the given time range

> **Note**
>
> After each pull, save the value used for **`EndTime`**. On the next pull, use that **`EndTime`** value as the new **`BeginTime`**. This ensures you only retrieve new changes with each pull.

For example:

```
-- Execute sproc to retrieve one specific table
EXEC axon.usp_PullDataByRange N'2024-01-01 00:00:00', N'2024-01-02 00:00:00', N'Records', N'IncidentReport';
```

The result of this query is a table that contains the following columns:

- **`SchemaName`**: **`Records`** or **`Standards`**
- **`TableName`**: The table name in DataStore (**`Reports`**, **`Cases`**, etc.)
- **`Operation`**: **`DELETE`** or **`UPSERT`**

- **`Primary Key`**: Contains one or more pairs of columns (composite keys):
  - **`PK_Column1Name`**: The name of the primary key column
  - **`PK_Column1Value`**: The value in the primary key column

See below for an example result you would see after querying **`IncidentReport`**:

SchemaName	TableName	PK_Column1Name	PK_Column1Value	Operation
RECORDS	IncidentReport	Id	63575741-cf6f-4a73-adaa-11451f4798a5	UPSERT
RECORDS	IncidentReport	Id	72b3563b-141b-4e7e-af99-1d053063eaba	DELETE
RECORDS	IncidentReport	Id	e4d7e926-26a9-4af2-8604-3447b66338f0	UPSERT
RECORDS	IncidentReport	Id	8805387e-5413-47f7-aaaa-7a056eb44aaa	DELETE

The example result below shows that **`IncidentReport_EntPerson`** has primary key columns of **`ID`** and **`ReportId`**, indicating that you must query both columns.

SchemaName	TableName	PK_ Column1Name	PK_ Column1Value	PK_ Column2Name	PK_ Column2Value	Operation
RECORDS	IncidentReport	Id	63575741-cf6f-4a73-adaa-11451f4798a5	ReportId	0ae2e41a-239d-45c6-b9e6-797ef909739f	UPSERT
RECORDS	IncidentReport	Id	72b3563b-141b-4e7e-af99-1d053063eaba	ReportId	ca6eca55-464b-4926-bc0d-3a913fe2174c	DELETE
RECORDS	IncidentReport	Id	e4d7e926-26a9-4af2-8604-3447b66338f0	ReportId	057160fe-3ec3-48f4-a2d1-9ccafca9ef4b	UPSERT
RECORDS	IncidentReport	Id	8805387e-5413-47f7-aaaa-7a056eb44aaa	ReportId	b0161993-0087-4347-9595-c68e3a6c33d5	DELETE

Perform an operation on each change

After running the procedure above, iterate through each row of the result table and perform either an **`UPSERT`** or **`DELETE`** operation:

- For **`DELETE`** operations, use the primary key column to locate and remove the corresponding record from your database.
- For **`UPSERT`** operations, retrieve the record from the DataStore using the primary key and then perform the **`UPSERT`** in your database.

Include all primary key columns in the condition to ensure you retrieve the correct record.

Example **`DELETE`** operation

```
-- Sample query in your Azure SQL database
DELETE FROM <UserTableName>
WHERE <PK_Column1Name>=<PK_Column1Val>
    AND <PK_Column2Name>=<PK_Column2Val> ...
```

In this example, primary key is a placeholder. Each table has a different name for its primary key. For example:

- **`DELETE FROM IncidentReport WHERE Id=XXXXYYYY`**
- **`DELETE FROM IncidentReport_EntPerson WHERE ID=XXX AND ReportId=YYY`**

Example **`UPSERT`** operation

```
>-- Sample query in your Azure SQL database
-- Retrieve record from the DataStore
SELECT * FROM &lt;SchemaName&gt;.&lt;TableName&gt;
WHERE &lt;PK_ColumnName&gt;=&lt;PK_Column1Value&gt;
    AND &lt;PK_Column2Name&gt;=&lt;PK_Column2Value&gt; ...

-- Perform Upsert
IF EXISTS(SELECT 1 FROM &lt;UserTableName&gt;
```

```
    WHERE &lt;PK_Column1Name&gt;=&lt;PK_Column1Value&gt;
        AND &lt;PK_Column2Name&gt;=&lt;PK_Column2Value&gt;))
BEGIN
    UPDATE &lt;UserTableName&gt;
    SET ColumnA='', ColumnB='',...
    WHERE &lt;PK_Column1Name&gt;=&lt;PK_Column1Value&gt;
        AND &lt;PK_Column2Name&gt;=&lt;PK_Column2Value&gt;;
END
ELSE
    INSERT INTO &lt;UserTableName&gt; (ColumnA, ColumnB, ...)
    VALUES (ValueColumnA, ValueColumnB, ...)
```

Use the following inputs for this query:

- **`SchemaName:`** **`Records`** or **`Standards`**
- **`TableName:`** The table name in DataStore (**`Reports`**, **`Cases`**, etc.)
- **`TablePrimaryKey:`** Primary key of the table (e.g., **`ReportId`** in Report, **`IncidentId`** in Incident)
- **`UserTableName:`** Name of the table in the your database
- **`PK_Column{i}Name:`** Name of the **`ith`** column in composite primary key
- **`PK_Column{i}Value:`** Calue of the **`ith`** column in composite primary key

You can find all supported tables and their primary keys by retrieving the information schema.

Cold start

Change data in each table is retained for up to three months. If you need older data, you must perform a cold start and pull data into empty tables.

Use Cases

You may need to perform a cold start when first syncing with the DataStore or if the data in your database has been out-of-sync with the DataStore for more than three months.It's recommended that you truncate the tables before initiating a cold start to avoid stale or duplicated data.

In this case, you would run the cold start to pull in data for a specific time period, such as **`2010-01-01 00:00:00`** to **`2024-01-01 00:00:00`**. After performing the cold start, you would begin an incremental pull starting at **`2024-01-01 00:00:00`**.

The timestamp in CDC tables will always be greater than or equal to the delta time timestamp in the tables (i.e., **`CreatedAt`** and **`UpdatedAt`**). As a result, duplicate imports may occur. However, this is acceptable because the **`UPSERT`** operation during incremental pulling ensures synchronization by updating new or modified records since the cold start point.

Expunged Data

When you perform a cold start, you must also perform an incremental pull to properly handle expunged data. If you do not, expunged data may remain in your tables. As a result, if you choose to perform a cold start, your database should be empty before pulling data. This ensures you retrieve only the most up-to-date records. If the table is not cleared before a cold start, there is a risk of retaining stale data or expunged records that should no longer exist.

Cold Start Query

Use the following query to perform a cold start. It is recommended that you pull data in one-month segments to avoid excessive CPU and RAM usage.

```
EXEC axon.usp_PullTableByRange @BeginTime @EndTime @SchemaName @TableName
```

Use the following inputs for this query:

- **`BeginTime:`** The beginning time to start the pull; input in SQL datetime2 format and UTC timezone
- **`EndTime:`** The end time to finish the pull; input in SQL datetime2 format and UTC timezone

- **`SchemaName: Records`** or **`Standards`**
- **`TableName:`** Specific table name within the given time range

The sproc returns data equal to and greater than **`BeginTime`** and smaller than **`EndTime`**. For example:

```
-- Sample query in your Azure SQL Database
-- Cold Start from 2020 to 2024
-- Batch in 1 month
EXEC axon.usp_PullTableByRange N'2020-01-01 00:00:00' N'2020-02-01 00:00:00' N'Records' N'IncidentReport'
```

This query returns data from the specified table (e.g., **`Reports`**), including all columns similar to a **`SELECT`** query.

- Data is retrieved within a specified time range (e.g., **`2020-01-01 00:00:00`** to **`2020-01-31 23:59:59`**).
- The ending boundary (23:59:59) is excluded, and the date continues to increment for subsequent batches.

The query internally pulls data using a delta time strategy:

1. **`UpdatedAt`** is used if available.
2. If **`UpdatedAt`** is not available, **`CreatedAt`** is used.
3. If neither is available, **`LastUpdated`** (the time the a row was last updated in the DataStore) is used, as it serves the same purpose.

See below for an example result you would see after running a cold start query:

Report Id	IncidentNumber	ReportNumber	ReportType	ReportCategory	UpdatedAt
3c868b65-7a61-4ee0-944e-1144d92236a7	56565656	56565656-80	GENERAL_OFFENSE	RecordsReport	2020-08-11 00:49:56.2800000
d35aef50-b795-4b31-8aca-3d47beb4b6d0	48645418	48645418-1	GENERAL_OFFENSE	RecordsReport	2022-03-17 22:48:43.6130000

Retrieve Information Schema

The DataStore has a view you can access to get all table names that support historical tables for incremental pulling. You can:

- Get tables with incremental pulling enabled
- Get column names of a table
- Get primary key of a table

Get tables with incremental pulling enabled

To get all tables with incremental pulling enabled, use the following procedure:

```
SELECT * FROM [axon].[HistoricalTables]
```

See below for an example result you would see after running this query:

SchemaName	TableName
RECORDS	InciddntReport
RECORDS	EntIncident

You can also query to get only tables that have changed within specified time range. Use the following procedure:

```
EXEC axon.usp_GetTablesHaveChanges @BeginTime, @EndTime, @SchemaName
```

Use the following inputs for this query:

- **`BeginTime:`** The beginning time to start the pull; input in SQL datetime2 format and UTC timezone
- **`EndTime:`** The end time to finish the pull; input in SQL datetime2 format and UTC timezone
- **`SchemaName: Records`** or **`Standards`**

Example query:

```
EXEC axon.usp_GetTablesHaveChanges N'2024-01-01 00:00:00', N'2024-01-02 00:00:00', N'RECORDS'
```

Get column names of a table

Use the query below to get the column names for a table:

```
SELECT TABLE_NAME,
    COLUMN_NAME,
    DATA_TYPE,
    CHARACTER_MAXIMUM_LENGTH,
    IS_NULLABLE,
    is_primary_key,
    is_fk
FROM axon.uf_GetTableColumns(@Schema, @TableName);
```

Example query:

```
select TABLE_NAME,
    COLUMN_NAME,
    DATA_TYPE,
    CHARACTER_MAXIMUM_LENGTH,
    IS_NULLABLE,
    is_primary_key,
    is_fk
from axon.uf_GetTableColumns('RECORDS', 'IncidentReport_EntPerson');
```

See below for an example result you would see after running this query:

TABLE_NAME	COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	IS_NULLABLE	is_primary_key	is_fk
IncidentReport_EntPerson	Id	nvarchar	100	NO	true	false
IncidentReport_EntPerson	ExternalId	nvarchar	100	YES	false	false
IncidentReport_EntPerson	ReportId	nvarchar	100	NO	true	true
IncidentReport_EntPerson	ReportFriendlyId	nvarchar	100	YES	false	false
IncidentReport_EntPerson	PersonFriendlyId	nvarchar	100	YES	false	false
IncidentReport_EntPerson	CreatedAt	datetime2	null	YES	false	false
IncidentReport_EntPerson	UpdatedAt	datetime2	null	YES	false	false
IncidentReport_EntPerson	AxonAge	decimal	null	YES	false	false
IncidentReport_EntPerson	AxonSex	nvarchar	-1	YES	false	false
IncidentReport_EntPerson	AxonRace	nvarchar	-1	YES	false	false

Get primary key of a table

Use the query below to get all columns in the composite primary key of a table:

```
SELECT * FROM axon.uf_GetPrimaryKeyColumns(@SchemaName, @TableName)
```

Use the following inputs for this query:

- **`TableName:`** The name of the table you want to retrieve
- **`SchemaName: Records`** or **`Standards`**

See below for an example result you would see after querying the **`IncidentReport_EntPerson`** table with the **`Records`** schema:

TableName	ColumnName
IncidentReport_EntPerson	Id
IncidentReport_EntPerson	ReportId

Demo python script

You can use the following Python script to pull data with full flow. Copy the code below or download this file, which contains the full code shown below.

1. Determine **`begin_time`** and **`end_time`**, assuming data pull every 24 hours:

```
begin_time_utc = datetime.now(timezone.utc) - timedelta(hours=24)
begin_time = begin_time_utc.strftime('%Y-%m-%d %H:%M:%S')

end_time_utc = datetime.now(timezone.utc)
end_time = end_time_utc.strftime('%Y-%m-%d %H:%M:%S')
```

2. Retrieve a list of table names that have changed within a specified time range for a given schema:

```
def get_changes_tables(SCHEMA: STR, begin_time: STR, end_time: STR,
        CURSOR: pyodbc.Cursor) -&gt; list[STR]: query = f"""
    EXEC axon.usp_GetTablesHaveChanges N'{begin_time}', N'{end_time}', N'{schema}'
    """ TABLES = [] try: cursor.execute(query) TABLES = [row.TableName for row in cursor.fetchall()]
EXCEPT
EXCEPTION AS e: print(f"Error fetching tables have changes: {e}") RETURN TABLES
```

3. Get all columns for each table:

```
def fetch_columns_in_a_table(SCHEMA: STR, TABLE: STR,
        CURSOR: pyodbc.Cursor): query = """
    SELECT COLUMN_NAME
    FROM axon.uf_GetTableColumns(?, ?)
    """ columns = [] try: cursor.execute(query, (SCHEMA, TABLE)) columns =
    [row.COLUMN_NAME for row in cursor.fetchall()]
EXCEPT
EXCEPTION AS e: print(f"Error fetching columns: {e}") RETURN columns
```

4. Get primary keys for each table:

```
def fetch_primary_keys(SCHEMA: STR, TABLE: STR,
            CURSOR: pyodbc.Cursor): query = """
        select PRIMARY_KEY_COLUMN as COLUMN_NAME from axon.uf_GetPrimaryKeyColumns
        (?, ?)
    """ primary_keys = [] try: cursor.execute(query, (SCHEMA, TABLE)) primary_keys = [row.COLUMN_NAME for row in cursor.fetchall
()]
EXCEPT
EXCEPTION AS e: print(f"Error fetching primary keys: {e}") RETURN primary_keys
```

5. Other useful utilities:

```
def concatenate_table_name_and_schema_name(SCHEMA: STR, TABLE: STR) -> STR:
    RETURN f"[{schema}].[{table}]"
def format_sql_value(row_data):
    value = row_data
    IF value IS NONE:
        RETURN 'null'
    RETURN f"'{value}'"
```

6. Get updated IDs in CDC:

```
statement = f"EXECUTE axon.usp_PullDataByRange N'{begin_time}', N'{end_time}', {schema_name}, {table_name}"
cursor.execute(statement)
data = cursor.fetchall()
```

7. Iterate every row and perform an **`UPSERT`** or **`DELETE`** procedure:

```
for row in data:
   try:
    key_values = {}
    for i in range(num_primary_keys):
        name_pattern = f"PK_Column{i + 1}Name"
        value_pattern = f"PK_Column{i + 1}Value"
        key_values[getattr(row, name_pattern, None)] = getattr(row, value_pattern,
        None)

    prepare_where_statement = " AND ".join([
        f"{name}='{key_values[name]}'" for name in key_values
```

```
])

prepare_statement = f"""
    SELECT {', '.join(columns)}
    FROM [{schema_name}].[{table_name}]
    WHERE ({prepare_where_statement})
"""

datastore_cursor.execute(prepare_statement)
row_data = datastore_cursor.fetchone()
operation = getattr(row, 'OperationType', None)

# Execute in User Database
if operation == 'UPSERT':
    column_names_without_primary_keys =
    datastore_cursor.description[len(primary_keys):]

    upsert_prepare_statement = f"""
        IF EXISTS (
            SELECT 1
            FROM {concatenate_table_name_and_schema_name(user_schema_name,
            user_table_name)}
            WHERE ({prepare_where_statement})
        )
        BEGIN
            UPDATE {concatenate_table_name_and_schema_name
            (user_schema_name, user_table_name)}
            SET {", ".join([
                f"{column_name[0]}={format_sql_value(getattr(row_data,
            column_name[0]))}"
                for column_name in column_names_without_primary_keys
            ])}
            WHERE ({prepare_where_statement})
        END
        ELSE
        BEGIN
            INSERT INTO {concatenate_table_name_and_schema_name
            (user_schema_name, user_table_name)}
            ({', '.join([column[0] for column in
            datastore_cursor.description])})
            VALUES ({", ".join([f"{format_sql_value(i)}" for i in
            row_data])})
```

```
            END
        """

        user_cursor.execute(upsert_prepare_statement)
        user_cursor.commit()

    elif operation == "DELETE":
        delete_prepare_statement = f"""
            DELETE FROM {concatenate_table_name_and_schema_name(user_schema_name,
            user_table_name)}
            WHERE ({prepare_where_statement})
        """
        user_cursor.execute(delete_prepare_statement)
        user_cursor.commit()

    except Exception as e:
        print(f"SKIPPING - Error while running row data with {prepare_where_statement}
        - "
            f"table {table_name} - primary keys value {key_values} - err:
            {str(e)}")
```

- Example **UPSERT** in a row:

```
IF EXISTS(SELECT 1 FROM [dboRECORDS].[EntPerson_AxonIdentityDocuments]
            WHERE  (Id='5090'))
    BEGIN
    UPDATE [dboRECORDS].[EntPerson_AxonIdentityDocuments]
        SET Id='5090', ParentId='6a19ae77-4467-48b6-ae00-8c4f4e94ffdb', PersonFriendlyId='PER2500001289',
PersonId='6a19ae77-4467-48b6-ae00-8c4f4e94ffdb', CreatedAt=null, UpdatedAt=null, Type='DRIVERS_LICENSE', MoreInfo=null,
SsnNumber=null, OtherNumber=null, PassportNumber=null, PassportCountry=null, OtherDescription=null, DriverLicenseClass=null,
DriverLicenseNumber=null, OtherExpirationDate=null, BirthCertificateNumber=null, PassportExpirationDate=null,
CommonAccessCardIssuer=null, CommonAccessCardNumber=null, DriverLicenseRestrictions=null, DriverLicenseIssuingState=null,
DriverLicenseExpirationDate=null, BirthCertificateIssuingState=null, CommonAccessCardExpirationDate=null,
FederalBureauOfInvestigationNumberNumber=null, OriginatingAgencyPoliceOrIdNumberNumber=null, LastUpdated='2025-02-20
13:26:07.058000'
        WHERE (Id='5090')
    END
ELSE
    INSERT INTO [dboRECORDS].[EntPerson_AxonIdentityDocuments] (Id, ParentId, PersonFriendlyId, PersonId, CreatedAt,
UpdatedAt, Type, MoreInfo, SsnNumber, OtherNumber, PassportNumber, PassportCountry, OtherDescription, DriverLicenseClass,
DriverLicenseNumber, OtherExpirationDate, BirthCertificateNumber, PassportExpirationDate, CommonAccessCardIssuer,
CommonAccessCardNumber, DriverLicenseRestrictions, DriverLicenseIssuingState, DriverLicenseExpirationDate,
BirthCertificateIssuingState, CommonAccessCardExpirationDate, FederalBureauOfInvestigationNumberNumber,
OriginatingAgencyPoliceOrIdNumberNumber, LastUpdated)
    VALUES ('5090', '6a19ae77-4467-48b6-ae00-8c4f4e94ffdb', 'PER2500001289',
'6a19ae77-4467-48b6-ae00-8c4f4e94ffdb', null, null, 'DRIVERS_LICENSE', null, null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, null, null, null, '2025-02-20 13:26:07.058000')
```

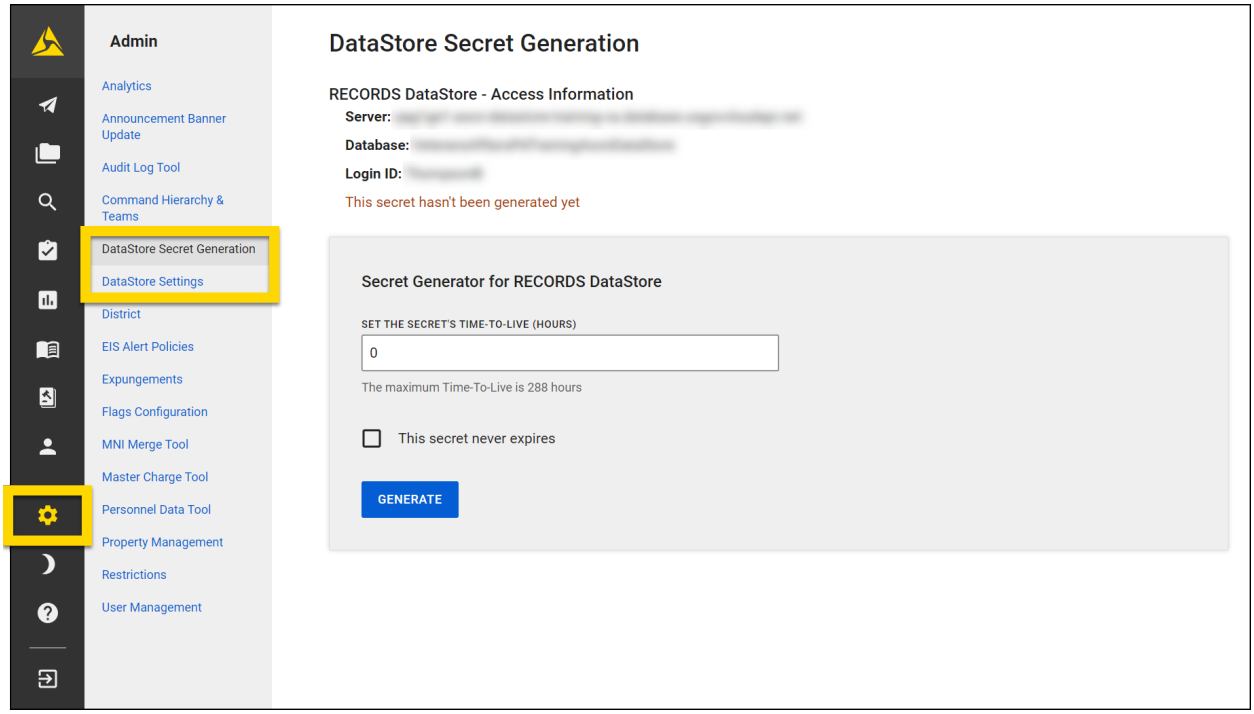
- Example **DELETE** in a row:

```
DELETE FROM [dboRECORDS].[EntPerson_AxonIdentityDocuments]
WHERE (Id='5090')
```

Access control

The DataStore tools in the Administrator Console let administrators with the appropriate privileges manage which users can access the DataStore. Using these tools, administrators can:

- DataStore Settings: Lets users manage DataStore configurations and user credentials
- DataStore Secret Generation: Lets users create secrets that allow direct access to the DataStore



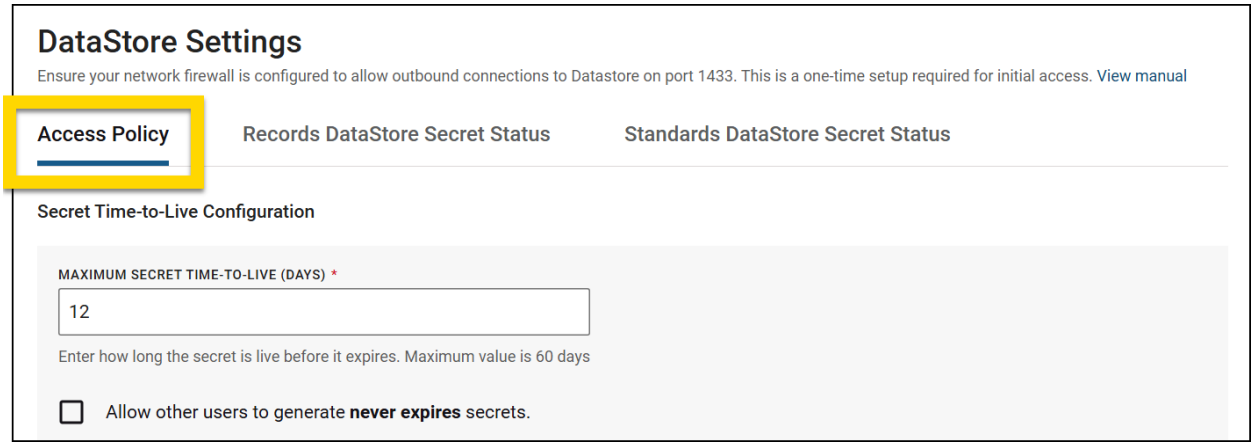
DataStore settings

Users who belong to Groups or teams with the DataStore Management privilege can view the DataStore Settings tool. Using this tool, users can manage the organization's access policy and secret statuses for Axon Records and Axon Standards users.

Access policy

From the **Access Policy** tab in the DataStore Settings tool, you can adjust the following settings:

- Secret time-to-live configuration
- Allowed IP addresses for DataStore access



Secret time-to-live configuration

Enter a number in the **Maximum Secret Time-To-Live (Days)** field to indicate the maximum number of days a secret can be available after it is generated.

Note that during secret generation, you can enter a TTL in hours, which allows for more granular control than the days setting in the DataStore Settings tool. However, the number of hours you enter during secret generation must be equal to or fewer than the day duration specified in this field in the DataStore Settings tool.

Select the **Allow other users to generate never expires secrets** checkbox to allow users to generate secrets that never expire.

After making changes, select **Save Settings**.

Allowed IP addresses for DataStore access

The IP addresses listed in this table are the only addresses from which the DataStore can be accessed.

To add a new IP address:

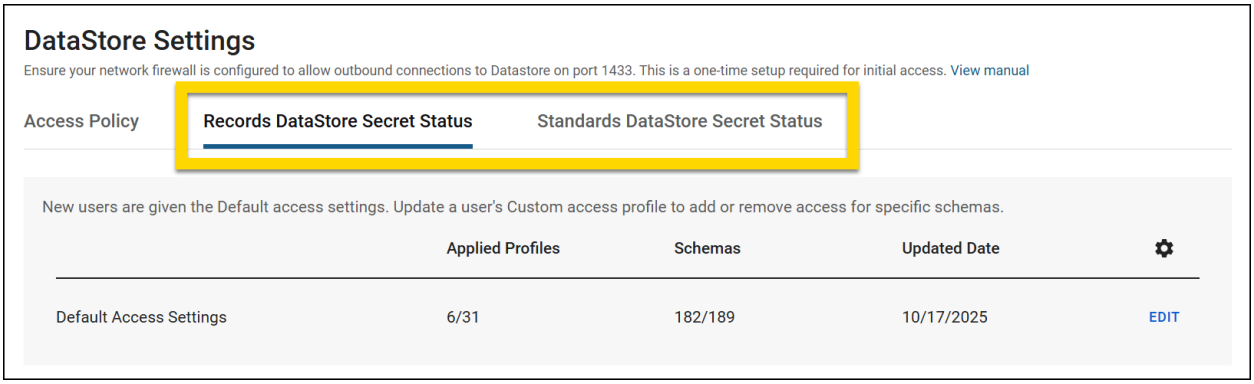
1. Select **Add IP**.
2. Select the type of address:
   - Single IP address: Lets you add one address
   - Range of IP address: Lets you add an IP address range. You must provide a starting IP address and an Ending IP address.
3. Enter either the single IP address or the range.
4. Select **Add**.
5. Select **Save Settings**.

To remove an IP address (either a single IP address or a range):

1. Select **Remove** in the row corresponding to the IP address you want to remove.
2. Select **Remove** again in the confirmation window that appears.
3. Select **Save Settings**.

Secret status

The **Secret Status** tabs in the DataStore Settings tool display a list of all users who have been given access to the DataStore. If your organization is configured for both Axon Records and Axon Standards, you will see two **Secret Status** tabs, one for Axon Records and one for Axon Standards.



These tabs are split into two sections:

- Default access settings
- Access profile list

Default access settings

The top section on the **Secret Status** tabs displays the settings for your organization's default access profile. The default access profile is applied whenever you give a new user access to the DataStore and gives users access to a predefined set of accessible views and schemas.

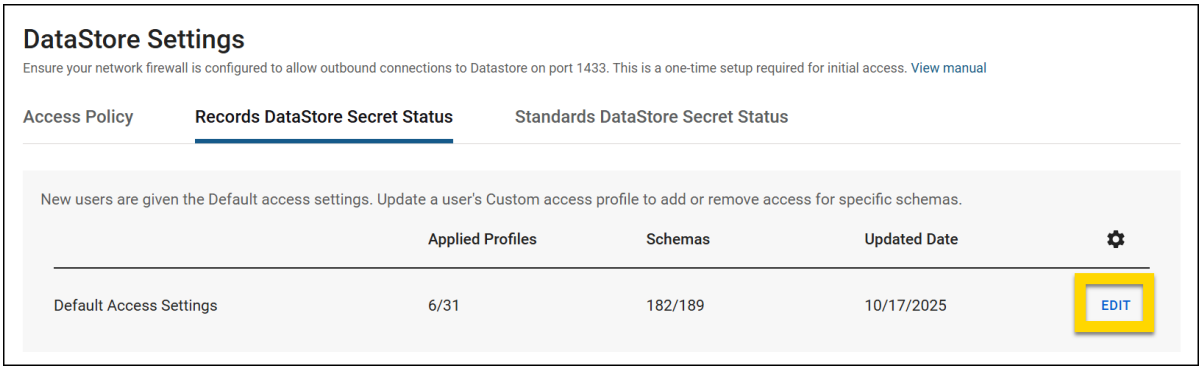
The Default Access Settings section displays the following information:

- **Applied Profiles:** The number of users who have been given default access
- **Schemas:** The number of tables/views out of the total number of tables/views in the entire DataStore that are included in the default access profile.
  - For example, if this section displays `182/189`, there are 189 total tables/views in the DataStore but the default access profile only includes 182.
- **Updated Date:** The last date when the default access settings were updated.

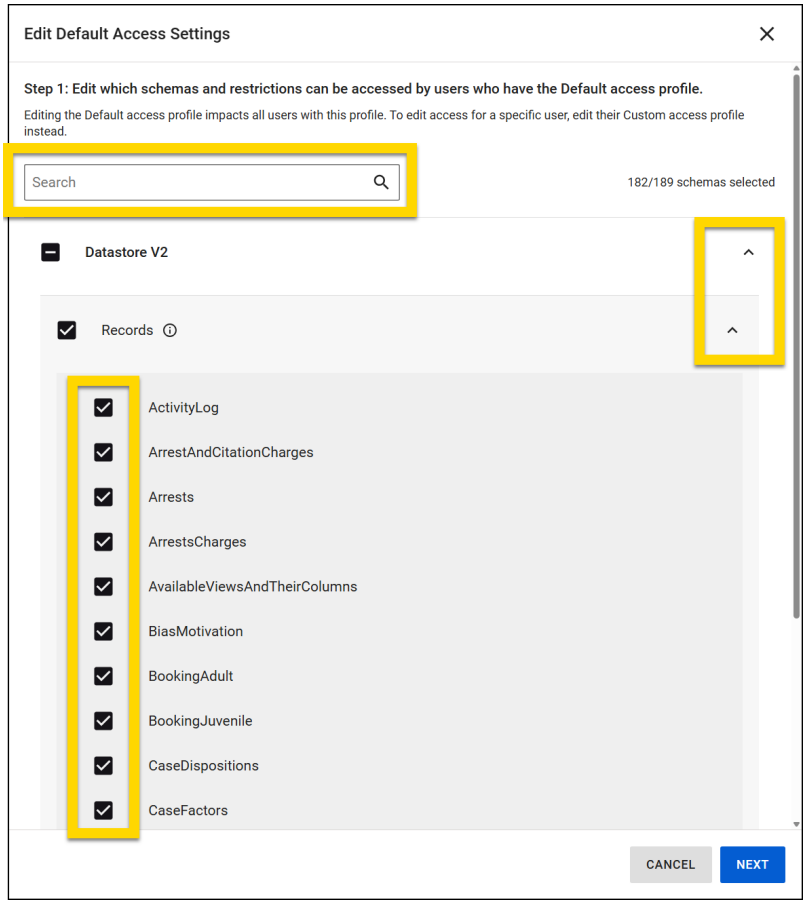
When you edit your DataStore's default access settings, all users who have been given default access will receive the access updates you make. If you only want to adjust access settings for a single user, edit their custom access profile instead.

To edit the default access settings:

1. Select **Edit** in the Default Access Settings section at the top of the **Secret Status** tab.



2. Use the checkboxes to set which schemas are included in the default access profile.
   - Use the search bar to find specific tables or views.
   - Hover over the information icon to view more details about that schema. See Concepts and features to learn more about schemas.
   - Select the down arrows to reveal the nested schema levels.



3. After adjusting the schemas, select **Next**.
4. Use the checkboxes to set which users have the default access profile.
   - All users who have previously been given the default access profile are pre-selected.
   - Use the search box to find specific users.
5. After adjusting the user list, select **Save**.
   - It may take several minutes for your updates to save. **Do not close the tab during this process.**

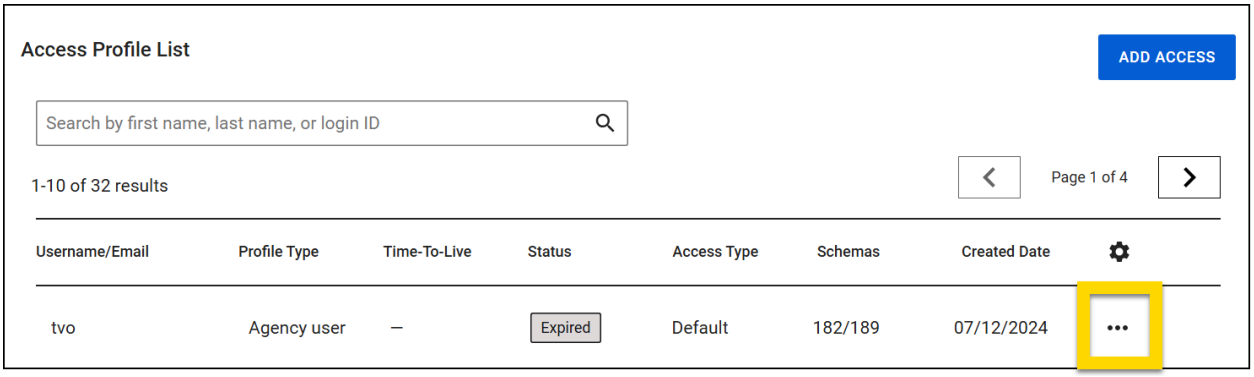
Access profile list

The table in the Access Profile List section provides detailed information about all user profiles that have access to DataStore, along with their current access status. The following information is included about each profile:

- **Username/Email:** The username or email associated with the profile
- **Profile Type:**
  - Agency user: Users who have an Axon Evidence account with the agency and for whom a DataStore secret has been generated.
  - Third party: Users who belong to third-party organizations but can still access the organization's DataStore.
  - Axon: Axon representatives who have been granted access to the organization's DataStore.
- **Time-To-Live:** Shows the TTL that was specified when the profile's secret was generated
- **Status:**
  - Setup Pending: A secret has NOT yet been generated.
  - Active: A secret has been generated and is currently active.
  - Expired: A secret was generated, but the TTL has passed and the secret has expired.
- **Access Type:** Whether the profile has default or custom access
- **Schemas:** The number of tables/views out of the total number of tables/views in the entire DataStore that are included in the default access profile.
  - For example, if this section displays `182/189`, there are 189 total tables/views in the DataStore but the default access profile only includes 182.
  - If a profile has default access, this schema count is the same as the count in the Default Access Settings section.
- **Created Date:** The date the access profile was created

Depending on the profile type and status, various options appear in the **More Actions [...]** menu:

- Edit access control: Appears for all profiles and statuses
- Regenerate secret: Appears for third-party profiles that are in Setup Pending or Active status
- Revoke secret: Appears for all profiles that are in Active status
- Remove access profile: Appears for all Axon profiles in any status

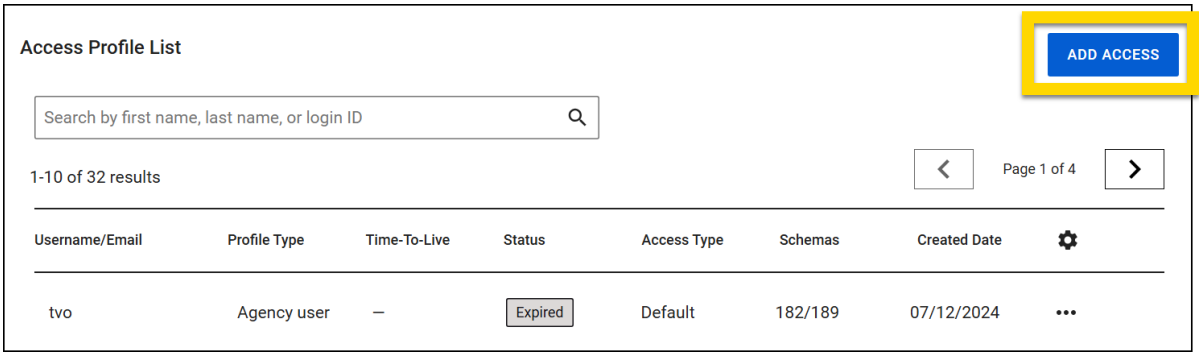


Create access profile

To grant DataStore access to an agency user, first generate a secret using the DataStore Secret Generation tool. Once the secret has generated, an access profile appears in the Access Profile List where you can edit the access control if you want the profile to have more or less access than is granted by the default access profile.

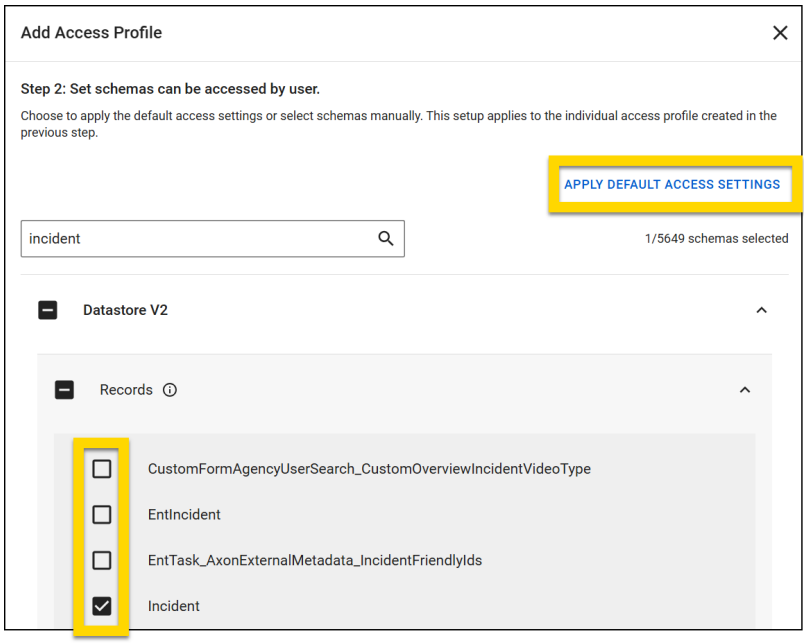
To create a new access profile for an Axon representative or a third-party user:

1. Select **Add Access**.



2. Specify if the user is an Axon representative or a third-party user.
   - If the user is an Axon representative, enter their email address.
   - If the user is a third party, enter a username.

3. Select the checkbox to acknowledge that you are a designated administrator responsible for granting DataStore access to users outside your organization.
4. Select **Next**.
5. Set which schemas the profile can access.
   - To grant default access, select **Apply Default Access Settings**.
   - To grant custom access, use the checkboxes to select specific schemas.
     - Tip: To use the default profile as a starting point, select **Apply Default Access Settings** then adjust the checkboxes to refine the schemas included in the profile's access.



6. Select **Add Access Profile** to create the profile.
7. When you create a profile for a third-party user, you can immediately generate a secret for the profile.
   a. Select a TTL option.
      - Options include days, hours, or never expires.
      - The maximum number is controlled by the maximum TTL setting on the **Access Policy** tab.
   b. Select **Generate**.
   c. The secret will generate and appear below the **Generate** button. **This secret must be copied immediately, as it will not be displayed again.**
   d. Select **Copy Secret and Close**.
   e. Follow your organization's security practices to safely share the secret with the user.

8. When you create a profile for an Axon representative, they receive an email directing them to log into the internal Axon secured administration portal to generate a secret.
    - Axon representatives can never generate “never expire” secrets. Any secrets they generate will adhere to the maximum TTL setting on the **Access Policy** tab.

Edit access control

You can edit a profile’s access settings at any time. Changing access does not affect the user’s existing secret; it only modifies their ability to SELECT tables or views within a schema. When new access is granted, the user will be able to see the additional tables or views shortly after the update. Conversely, if a user’s access to specific tables or views is revoked, their access to those resources will be removed immediately. All access changes—both granting and revoking—take effect within approximately 30 seconds.

To edit access:

1. Use the search bar to find a profile.
2. Select **More Actions [...] > Edit Access Control**.
3. Use the checkboxes to set which schemas are included in the profile.
    - If a profile previously had the default access profile and you add or remove access to any schemas, their access type changes to Custom.
    - To return a user to the default access profile, select **Apply Default Access Settings**.
4. After making all necessary adjustments, select **Save**.

Regenerate secret

Regenerating a secret lets you refresh secrets for third-party users outside your organization. Agency users and Axon representatives who have access to the DataStore can generate secrets for themselves as necessary.

To regenerate a secret for a third-party user:

1. Use the search bar to find a profile.
2. Select **More Actions [...] > Regenerate Secret**.
3. Select the check box to acknowledge that you are a designated administrator responsible for granting DataStore access to users outside your organization.
4. Select a TTL option.
    - Options include days, hours, or never expires.
    - The maximum number is controlled by the settings on the **Access Policy** tab.
5. Select **Generate**.

6. The secret will generate and appear below the **Generate** button. **This secret must be copied immediately, as it will not be displayed again.**
7. Select **Copy Secret and Close**.
8. Follow your organization's security practices to safely share the secret with the user.

Revoke secret

Once a secret is revoked, it immediately changes to an Expired status and can no longer be used to access the DataStore. If the user is currently accessing the DataStore when this occurs, they will no longer be able to perform any queries and will be logged out when their session times out.

Secrets for Axon representatives can't be revoked. Instead, the access profile must be removed.

To revoke a secret:

1. Use the search bar to find a profile.
2. Select **More Actions [...] > Revoke**.
3. Select **Revoke** in the confirmation window that appears.

Remove access profile

Access profiles for Axon representatives can be removed at any time. Once an access profile is removed, it cannot be restored.

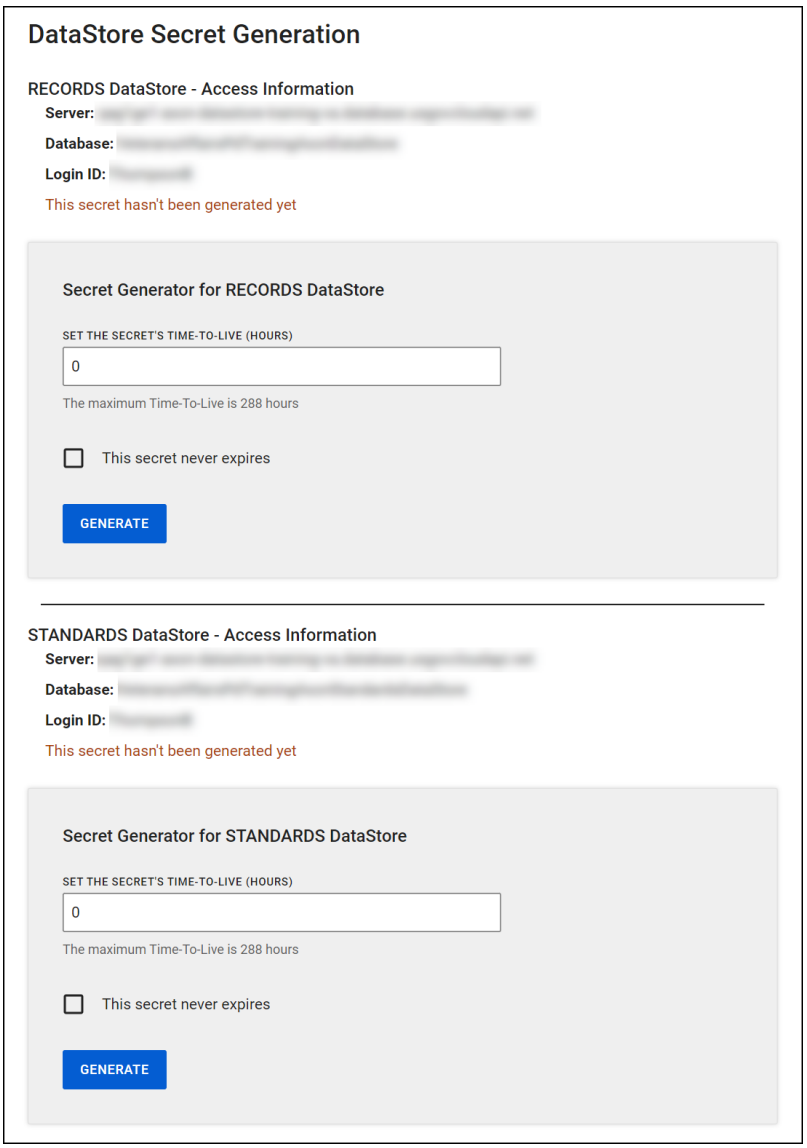
To remove an access profile:

1. Use the search bar to find a profile.
2. Select **More Actions [...] > Remove access profile**.
3. Select **Remove** in the confirmation window that appears.

Secret generation

Users who belong to Groups or teams with the DataStore Access privilege can view the DataStore Secret Generation tool. Using this tool, users can generate secrets that allow access to the DataStore. If your organization is configured for both Axon Records and

Axon Standards, you will see sections on this page: one for generating Axon Records secrets and one for Axon Standards.



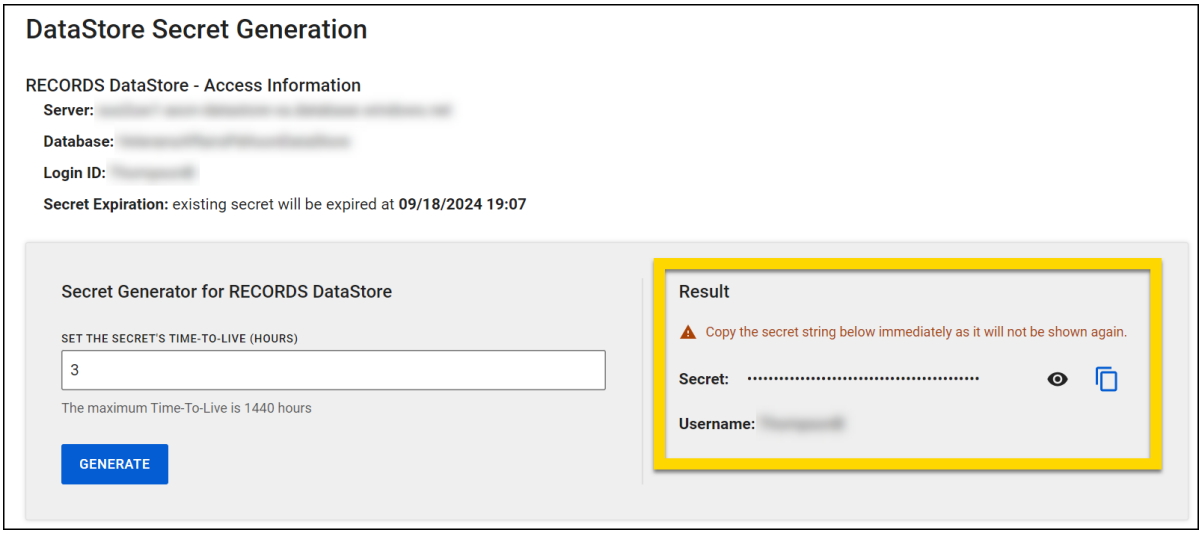
Each section provides the following information:

- Server name
- Database name
- Login ID: The username of the user who is viewing and using the tool

To generate a new secret:

1. Go to either the Axon Records or Axon Standards section and enter the time-to-live (TTL) of the secret in hours.
   - You can't enter a longer TTL than the maximum set by administrators.
   - To generate a secret that never expires, select the **This secret never expires** checkbox.

2. Select **Generate**.
3. The secret will generate and appear on the right side of the gray box. **This secret must be copied immediately, as it will not be displayed again.**
4. The secret will be added to the DataStore Settings tool where users with the appropriate privileges can revoke it if necessary.



Privileges

The privileges related to the Axon Records DataStore appear in the **DataStore - Records** category, and the privileges related to the Axon Standards DataStore appear in the **DataStore - Standards** category, as shown below:

Name	Description
DataStore - Records	
Manage the Records DataStore using the DataStore Settings tool	Lets users access the DataStore Settings tool in the Administrator Console and manage configurations for the Axon Records DataStore.
Use the Records DataStore Secret Generation tool to create DataStore secrets	Lets users access the DataStore Secret Generation tool in the Administrator Console and generate secrets for the Axon Records DataStore.
DataStore - Standards	
Manage the Standards DataStore using the DataStore Settings tool	Lets users access the DataStore Settings tool in the Administrator Console and manage configurations for the Axon Standards DataStore.

Name	Description
Use the Standards DataStore Secret Generation tool to create DataStore secrets	Lets users access the DataStore Secret Generation tool in the Administrator Console and generate secrets for the Axon Standards DataStore.

ODBC server

To connect to the Axon DataStore, you can create a Microsoft Open Database Connectivity (ODBC) server. ODBC is a C programming language interface that lets applications access data from a variety of database management systems (DBMSs). ODBC is a low-level, high-performance interface designed specifically for relational datastores.

After creating an ODBC server, you can link it to Microsoft Excel or Microsoft Access to view the data.

Before creating an ODBC server, contact your Axon representative to receive the following information:

- Server name
- Database name
- Login information (username and password)

Once you have the above information, take these steps:

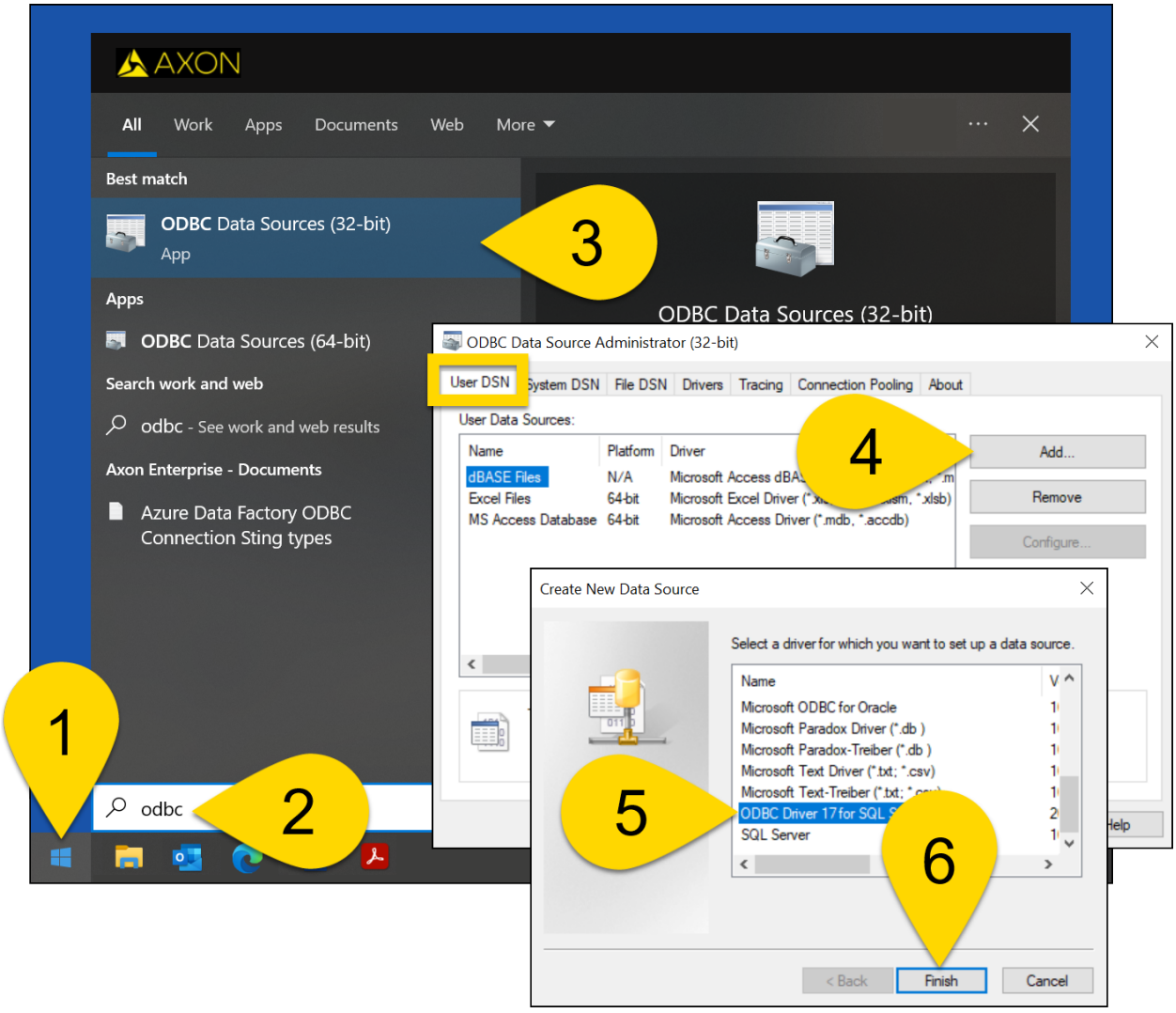
1. Create a data source
2. Connect to SQL server
3. Test the data source
4. After creating and connecting to the server, you can link it to Microsoft Excel or Microsoft Access.

Create a data source

The first step in creating an ODBC server is to create a data source. To do this from a Windows machine, take these steps:

1. Select the Windows icon in your start/task bar.
2. Type "ODBC."
   - After selecting the Windows icon, you will not see the search bar, but it will appear once you begin typing.
3. Select **ODBC Data source (32 bit)**.
   - This program is part of the Windows operating system and does not need to be downloaded.
4. On the **User DSN** tab, select **Add**.
5. Select **ODBC Driver 17 for SQL Server**.
6. Select **Finish**.
   - You must use version 17 or greater. If you do not see this option, download and

install the driver here.



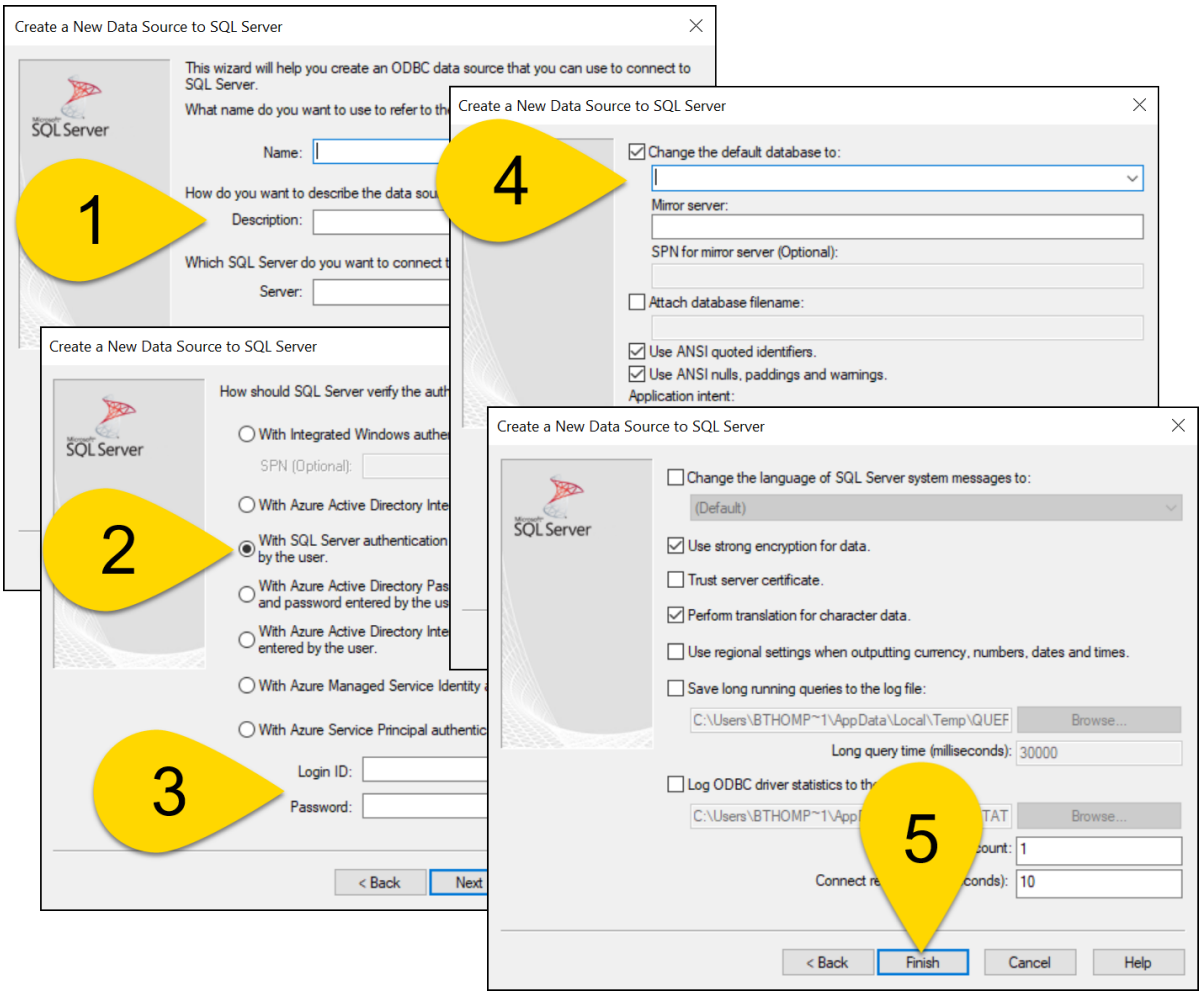
Connect to the SQL server

Once you create a data source as explained above, you need to connect the data source to the SQL server.

1. When you select **Finish** after creating the data source, a new window will appear. Enter a name for the data source (free-form text), description (free-form text), and the server name. Select **Next**.
   - Contact your Axon representative to receive the server name for your agency.
2. Select the **With SQL Server authentication using a login ID and password entered by the user**
3. Enter your username and password.
   - Contact your Axon representative to receive this login information.

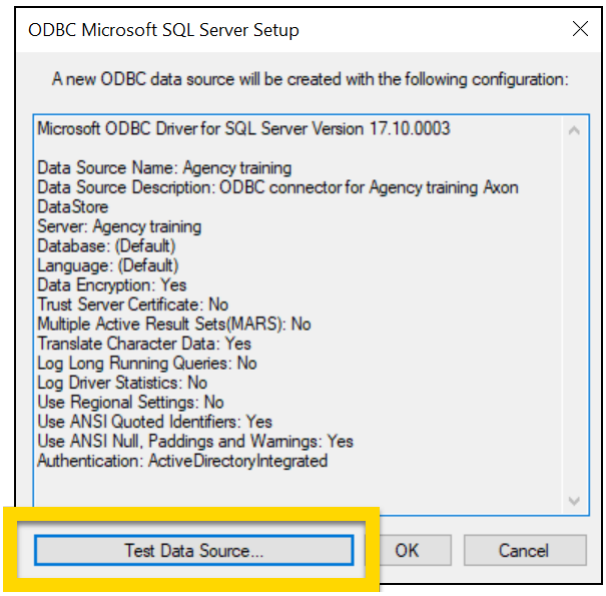
4. Select the **Change the default database to option,** enter your database name, and select **Next**. The rest of the entries in this section do not need to be adjusted.
   - Contact your Axon representative to receive the database name for your agency.
5. Nothing on this screen should be adjusted. Select **Finish**.

> **Warning**
>
> **Always** use strong encryption.
>
> **Do NOT** blindly trust the server certificate (this forces the client to verify the identity of the TLS certificate received from the server)



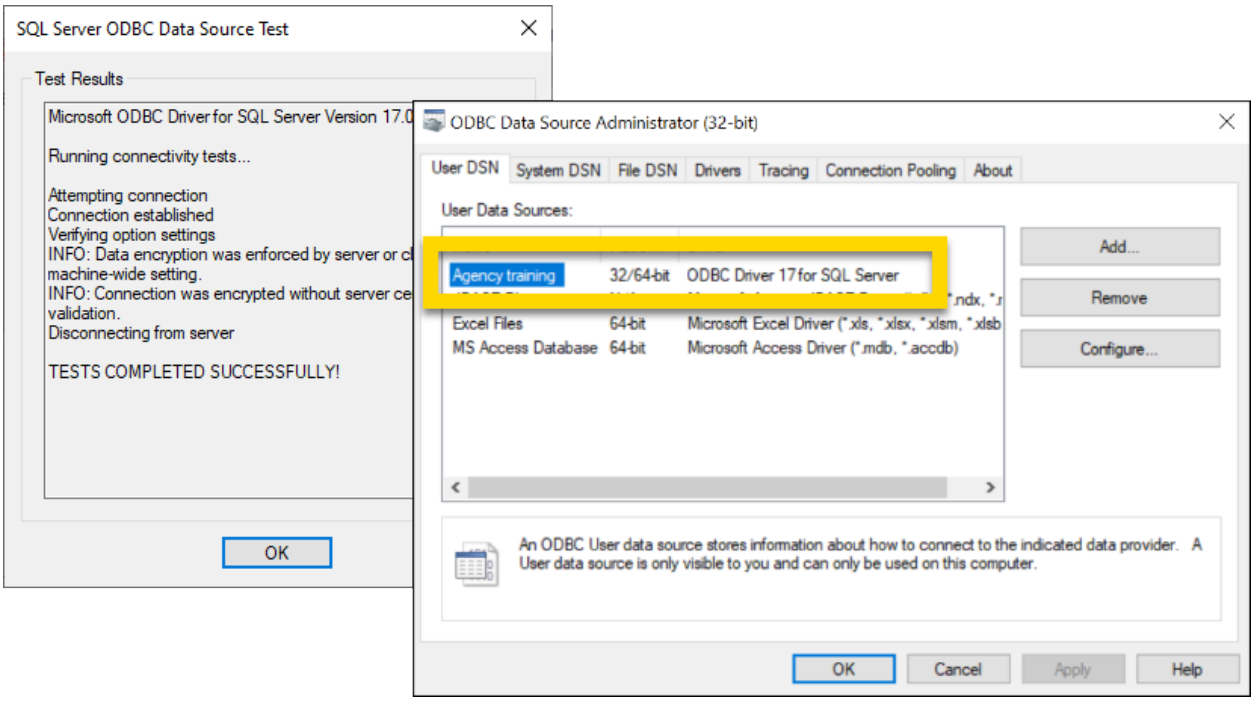
Test the data source

When you select **Finish** after setting up the SQL server connection, a new window will appear and display a configuration summary. Select **Test Data Source**.



Test completed successfully

If your test passes, select **OK**. The data source testing window will close, and you will again see the ODBC Data Administrator application. Your new data source will now display on the **User DSN** tab.

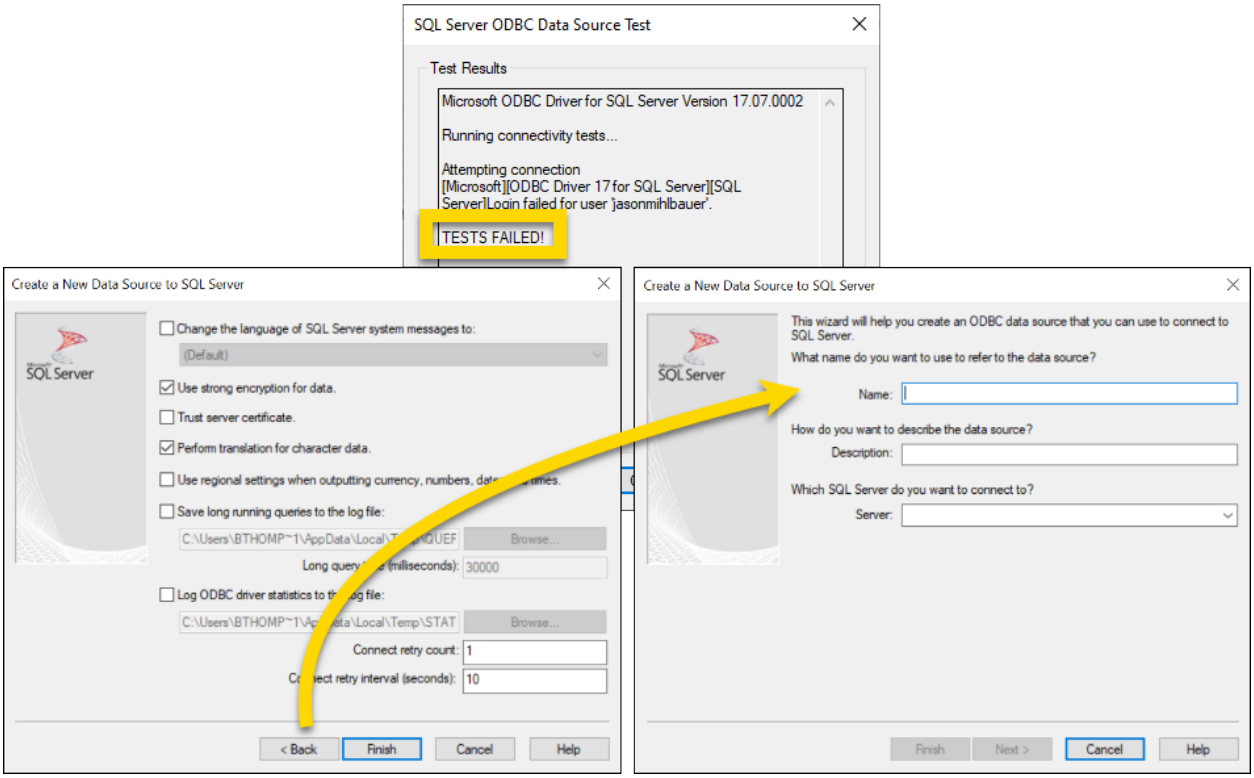


Test failed

If your test fails, select **OK**.

You will return to the final window in the SQL server connection workflow (step 5 in the previous section). Select **Back** three times until you reach the first window in this workflow (step 1 in the previous section).

Repeat steps 1-5 from the previous section and again test the data source until the test passes and your new data source appears in the ODBC Data Administrator application.

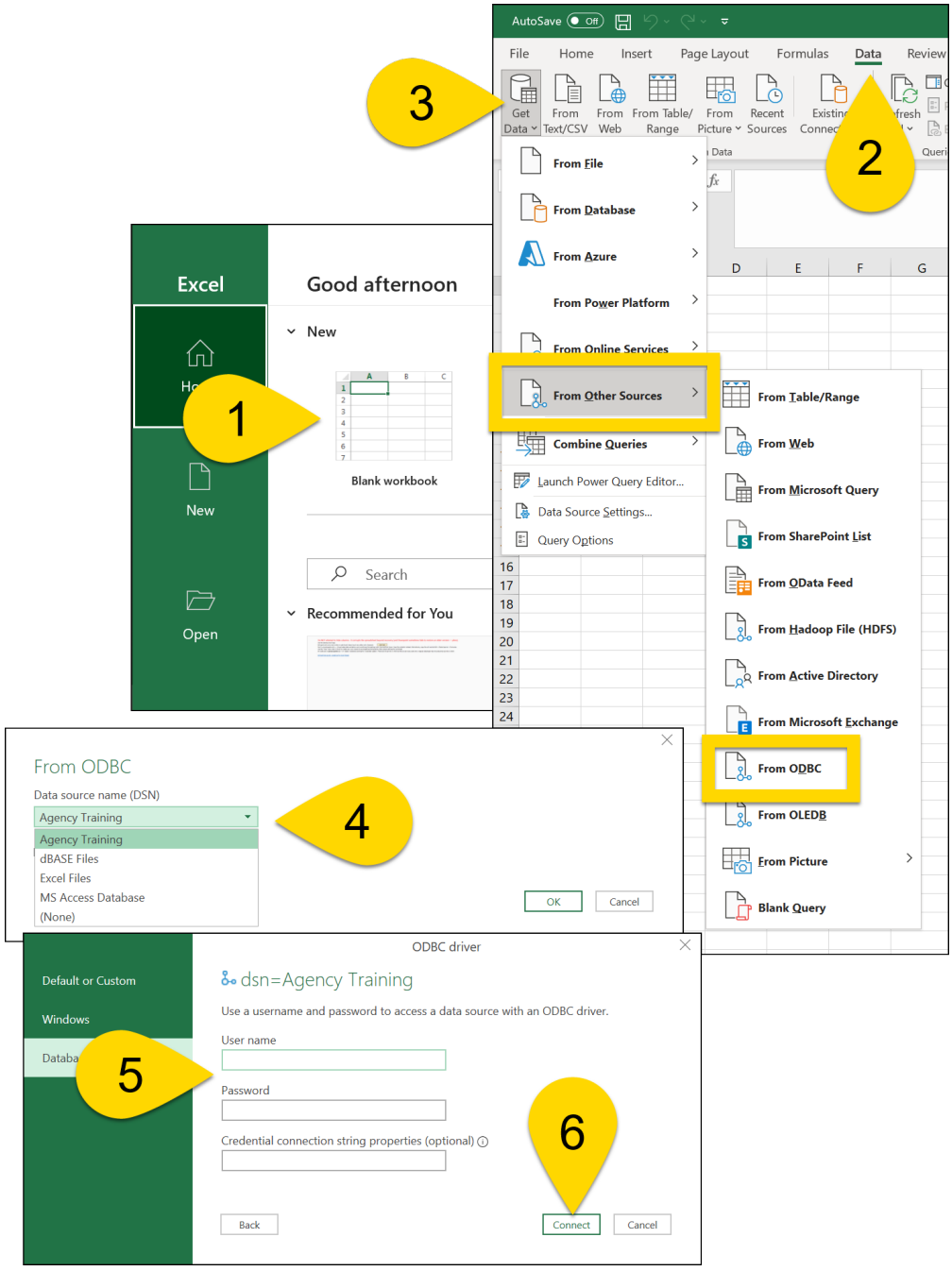


Link ODBC server to Microsoft Excel

Once you have created an ODBC server, you can link it to Microsoft Excel and access the Axon DataStore. To create this link, follow these steps:

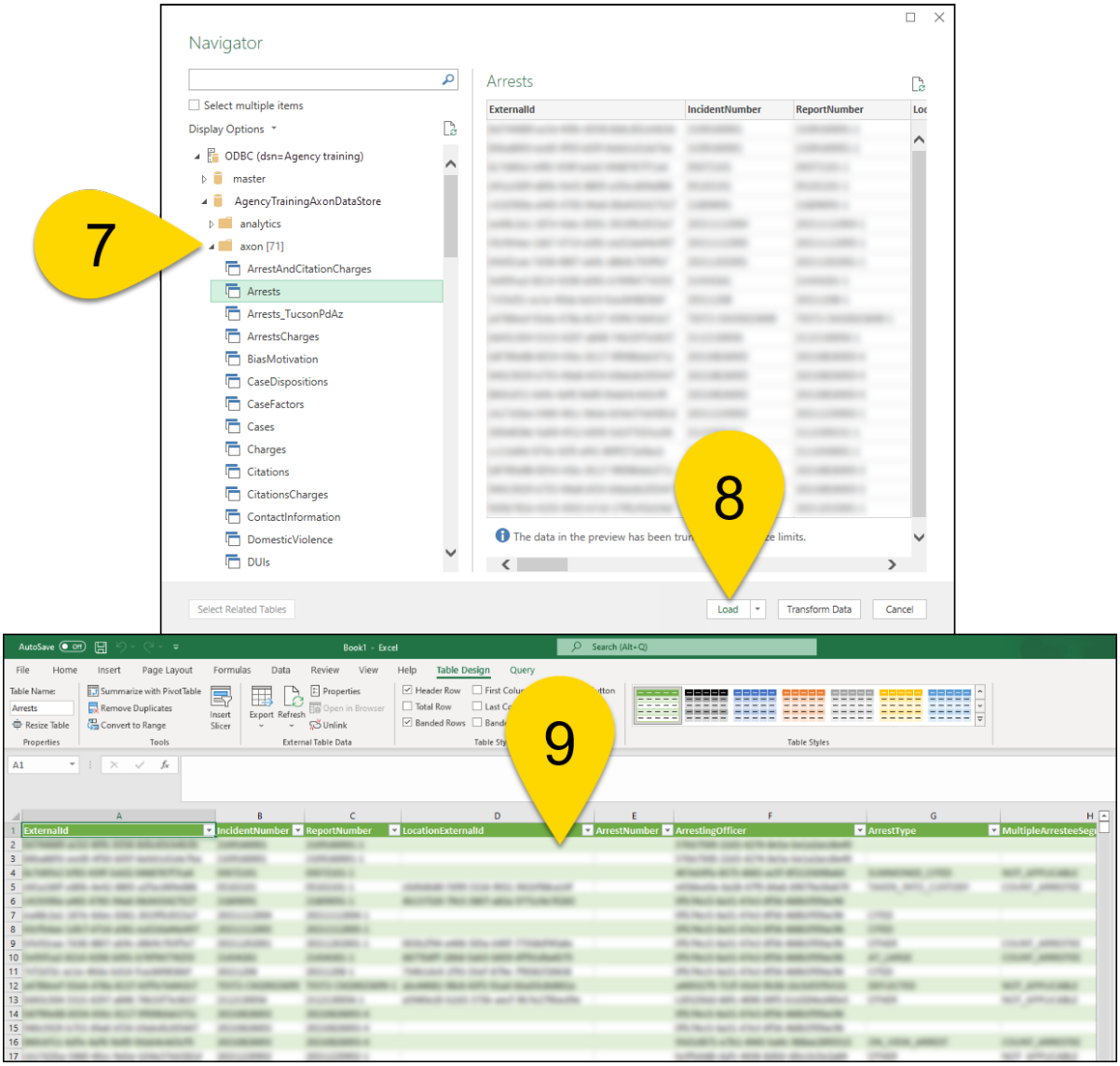
1. Open a blank workbook in Microsoft Excel.
2. Go to the **Data** tab.
3. Select **Get Data > From Other Sources > From ODBC**.
4. Reveal the list of data sources, choose your ODBC server, and select **OK**.
5. Enter your username and password associated with that Axon DataStore. You can leave the **Credential connection string properties (optional)** field blank.

6. Select **Connect**.



7. The navigator will open where you can select the arrow beside *axon*, *dw* or *raw*, depending on which data you want to populate in Excel.
8. After a data preview appears, select **Load**.

9. The data will populate in Excel.

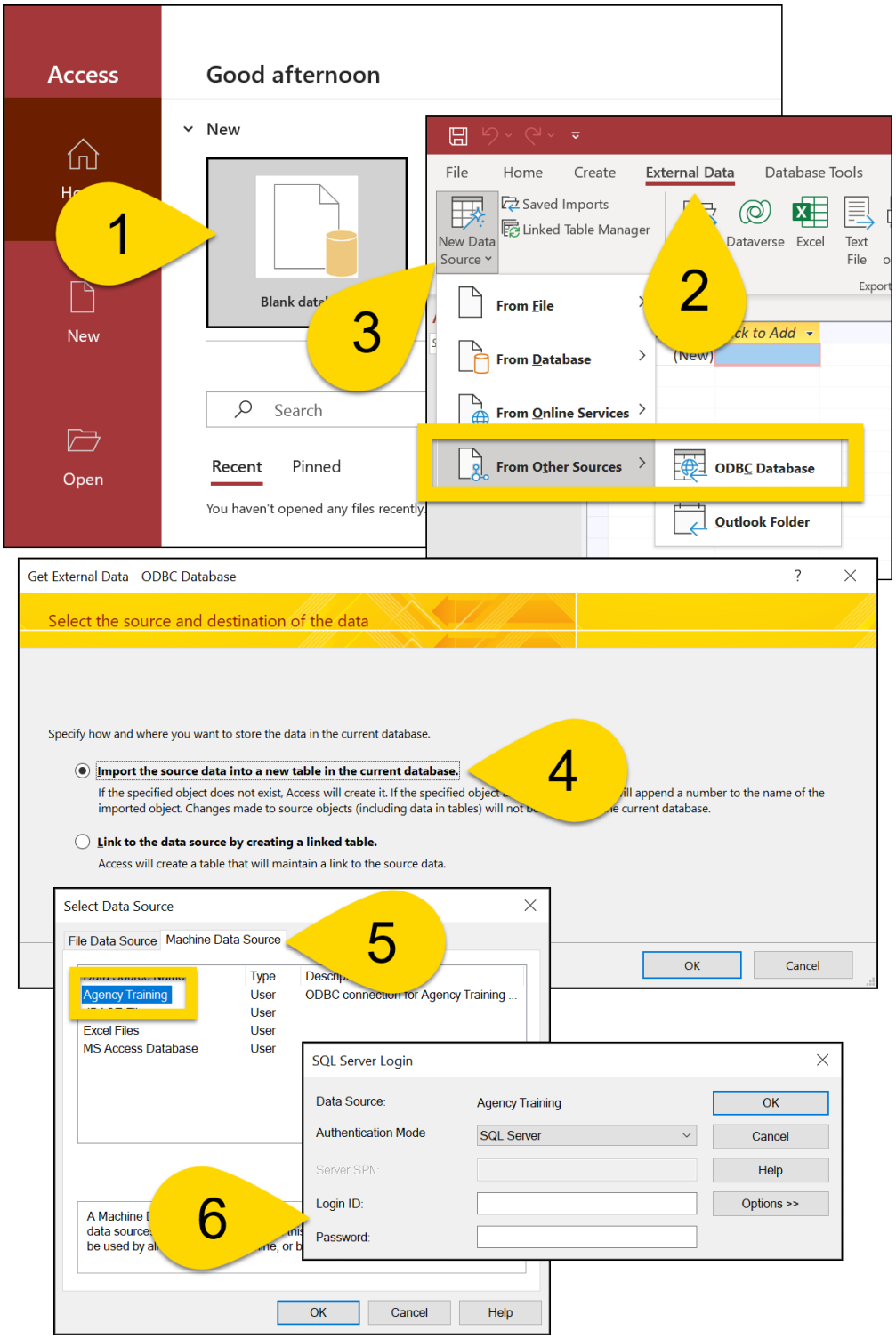


Link ODBC server to Microsoft Access

Once you have created an ODBC server, you can link it to Microsoft Access and access the Axon DataStore. To create this link, follow these steps:

1. Open a blank database in Microsoft Access.
2. Go to the **External Data** tab.
3. Select **New Data Source > From Other Sources > From ODBC**.
4. Choose **Import the source data into a new table in the current database** and select **OK**.

5. Select the **Machine Data Source** tab in the explorer window that appears and choose the ODBC server connection.
6. Enter your username and password for with that Axon DataStore and select **OK**.



7. Select which views to import into the Microsoft Access database.

- You can also save your password for future ease.
- Once you have selected all views you want to import, select **OK**.

8. It may take some time for the views to import. Once the import is complete, you can save the import steps and close out of the import process.
9. Double-click a view in the left panel to display the data for that view.