Version 4 Release 3

IBM i2 Analyze
Information Store Data Ingestion Guide

Note

Before you use this information and the product that it supports, read the information in "Notices" on page 51.

This edition applies to version 4, release 3, modification 2 of IBM® i2® Analyze and to all subsequent releases and modifications until otherwise indicated in new editions. Ensure that you are reading the appropriate document for the version of the product that you are using. To find a specific version of this document, access the Configuring section of the IBM Knowledge Center, and ensure that you select the correct version.

Contents

Information Store data ingestion.....1
About this guide.....3
Contacting IBM Support.....3
Information Store ingestion and the production deployment process.....4
Preparing for ingestion.....4
Identifying the data to be ingested.....6
Creating the staging tables.....6
Preparing the external data.....8
Populating the staging tables.....10
Defining an ingestion source.....11
Creating an ingestion mapping file.....12
Validating the ingestion configuration.....13
Adding data to the Information Store.....13
Using bulk import mode.....14
Using standard import mode.....19
Updating the Information Store for changed data.....20
Updating the Information Store for deleted data.....20
Ingestion resources.....24
Understanding the architecture.....24
Information Store staging tables.....28
Information Store property value ranges.....31
Ingestion mapping files.....32
Origin identifiers.....39
The **`ingestInformationStoreRecords`** toolkit task.....42
Understanding ingestion reports.....43
Troubleshooting the ingestion process.....46

Notices.....51
Trademarks.....52

Information Store data ingestion

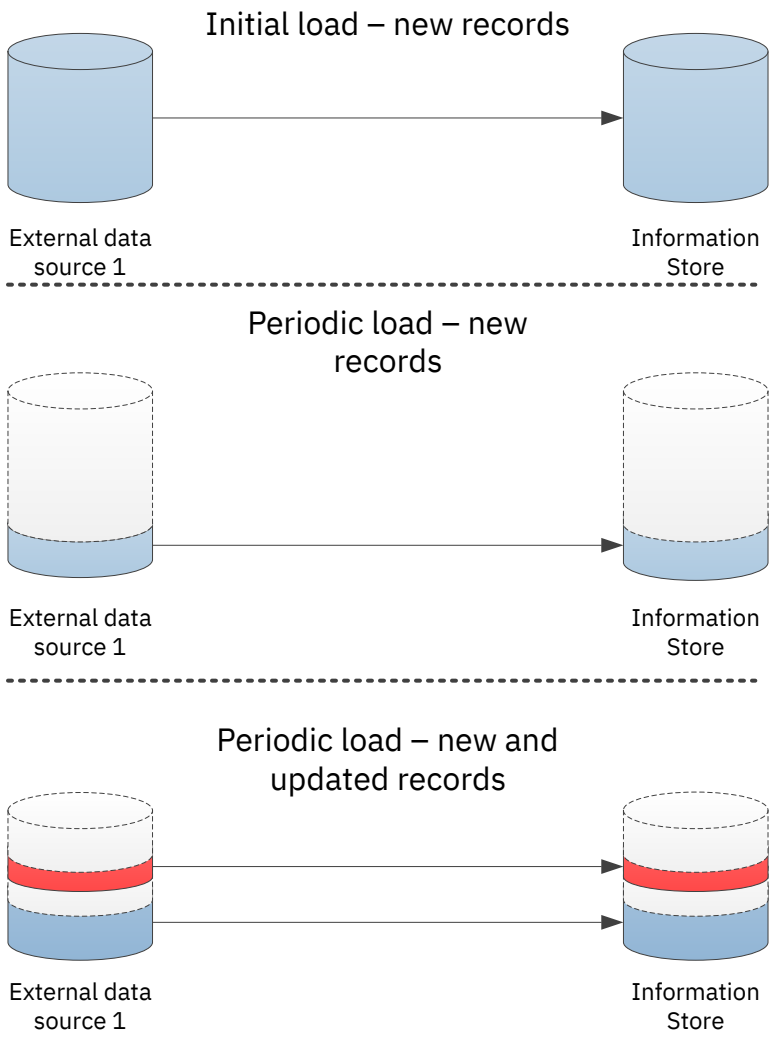
To enable its analytical capabilities, i2 Analyze places requirements on the data records that it processes. i2 Analyze records must conform to the general data model and a deployment-specific i2 Analyze schema, and they need information that might not be present in external sources. Before you can add data from your source to the Information Store, you must transform it to meet the same requirements.

i2 Analyze provides two mechanisms that make it easier to align your data with an i2 Analyze schema and prepare it for ingestion:

- The i2 Analyze deployment toolkit includes a command for creating staging tables that match the structure of the schema, in the same database as the Information Store. If you can load your data into the staging tables successfully, the Information Store can ingest your data from those tables.
- During data ingestion, i2 Analyze uses ingestion mappings that describe how to create records in the Information Store from the data in the staging tables. You can define these mappings in one or more files.

There are two scenarios for ingesting data from an external source. The first time that you load data of a particular item type from an external source, you perform an *initial load*. In an initial load, you are presenting the data to the Information Store for the first time. After you perform the initial load, you might want to perform *periodic loads* that update the Information Store as data is added to the external source or the data is changed. The same staging tables and mapping files can be used in both scenarios.

The following diagram shows the types of data that can be included in each type of load that you might perform. The data in blue is presented to the Information Store for the first time, and the data in red contains updates to data that exists in the Information Store.



The i2 Analyze deployment toolkit provides two *import mode*s that you can use to ingest data:

Bulk import mode

The *bulk* import mode can be used in the following situations:

- Initial load, new records, no correlation
- Periodic load, new records, no correlation

The bulk import mode is significantly faster for ingesting new, uncorrelated records.

Standard import mode

The *standard* import mode can be used in the following situations:

- Initial load, with or without correlation
- Periodic load, new and updated records, with or without correlation

Standard import mode trades some performance for increased data validation and additional operations during ingestion.

For more information about correlation, see Overview of correlation.

Because ingestion is completed on a per-item-type basis, you can use the bulk import mode for some item types and use standard mode for others from the external source. You can also use the bulk import mode to complete an initial load, and then use the standard import mode to complete periodic loads to update the data.

About this guide

This documentation explains how to add data from external sources to the IBM i2 Analyze Information Store, and how to keep the Information Store up-to-date as the contents of the external source changes. The requirements are mostly the same in both scenarios, but there are some differences depending on the data that you are updating the Information Store with.

Intended audience

This guide is intended for users who understand both the i2 Analyze data model and the i2 Analyze schema that defines the data structure of their deployment. Populating the Information Store staging tables also requires you to be familiar with your database management system.

Contacting IBM Support

IBM Support provides assistance with product defects, answers FAQs, and helps users to resolve problems with the product.

About this task

After trying to find your answer or solution by using other self-help options such as technotes, you can contact IBM Support. Before contacting IBM Support, your company or organization must have an active IBM software subscription and support contract, and you must be authorized to submit problems to IBM. For information about the types of available support, see the Support portfolio topic in the *Software Support Handbook*.

Procedure

To contact IBM Support about a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For more information, see the Getting IBM Support topic in the *Software Support Handbook*.
2. Gather diagnostic information.
3. Submit the problem to IBM Support in one of the following ways:
   - Online through the IBM Support Portal at Support Portal. You can open, update, and view all of your service requests from the Service Request portlet on the Service Request page.
   - By phone. For the phone number to call in your region, see the Directory of worldwide contacts web page at https://www.ibm.com/planetwide/

Results

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Support

website daily, so that other users who experience the same problem can benefit from the same resolution.

Information Store ingestion and the production deployment process

Before you deploy i2 Analyze into production you develop your configuration in a number of environments. The same is true for the ingestion process. You use the same environments to develop the Information Store ingestion process for each of your external data sources.

In the *configuration development environment*, ingest small amounts of test data that is representative of the entity and link types in your existing data sources. By doing this, you can learn how to ingest your data into the Information Store database and ensure that your i2 Analyze schema and security schema are correct for your data. If you discover that the schemas cannot represent your data, you can update your schema files.

You are likely to complete many test ingestions during this development phase. As you understand the process more, you might change how the data is loaded into the staging tables, the data that you use to generate identifiers, or change the ingestion mappings. If you modify the i2 Analyze schema, you must re-create your staging tables to match.

The ingestion mappings that you use are slightly different for each external data source that you ingest data from. It is recommended that you keep any ingestion mapping files in a source control system to track changes and so that you can use them in later development and production environments.

In your *pre-production environment* or *test environment*, use the process that you developed to ingest larger quantities of data and note the time that is taken to ingest the data. You can complete representative initial and periodic loads from each of your external data sources. The time that each ingestion takes allows you to plan when to ingest data in your production environment more accurately.

In your *production environment*, complete your initial loads by using the developed and tested process. According to your schedule, perform your periodic loads on an ongoing basis.

Your deployment architecture can impact the ingestion process in general, and any logic or tools used for transformation of data. For more information, see “Understanding the architecture” on page 24.

The instructions that describe the ingestion process are separated into preparing for ingestion and adding data to the Information Store.

Preparing for ingestion

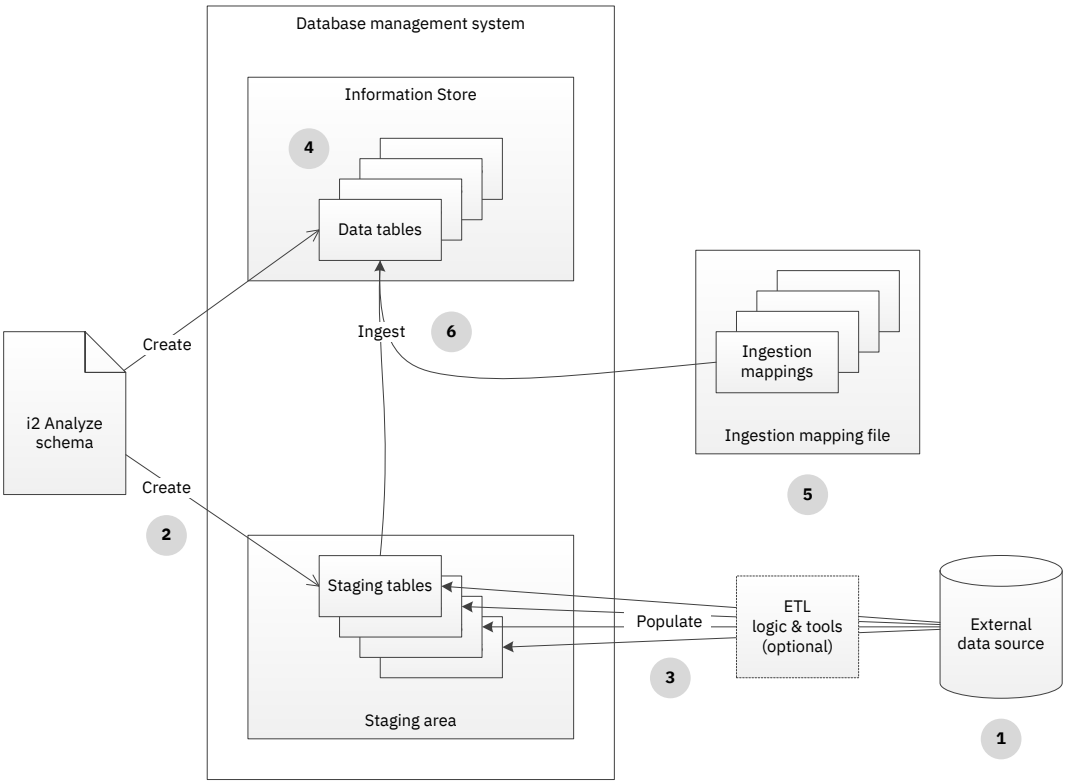
You must complete three tasks before the Information Store can ingest data from an external source. You must identify exactly which data to load, transform the data to align with the active i2 Analyze schema, and augment the data with extra information that the Information Store requires.

About this task

The only way to add and modify large volumes of data in the i2 Analyze Information Store is to enable and then instruct the Information Store to ingest it. The enablement process involves creating and populating staging tables for the data, and then supplying the metadata that is crucial to the analytical capabilities of i2 Analyze.

Procedure

You can plan and run the Information Store data ingestion process in a series of discrete steps. This diagram illustrates the approach.



1. Decide which entity types and link types in the active i2 Analyze schema best represent the data that you want the Information Store to ingest.
2. Create staging tables in the database for the types that you identified. Create more than one staging table for some link types.
3. Use external tools, or any other appropriate technique, to transform your data and load the staging tables with the data for ingestion.
4. Add information about your data source to the list of ingestion sources that the Information Store maintains.
5. Write the ingestion mappings that govern the ingestion process and provide additional information that the Information Store requires.
6. Run the ingestion command separately for each of the ingestion mappings that you wrote.

Example

The `examples\data\law-enforcement-data-set-1` and `\signal-intelligence-data-set-1` directories in the deployment toolkit contain files that i2 Analyze uses when you run the `setup -t ingestExampleData` command to populate the Information Store during deployment. These files provide demonstrations of many of the steps in the standard approach to ingestion. The following topics describe those steps in more depth as they detail the Information Store ingestion process.

Identifying the data to be ingested

The detail of how you arrange for the Information Store to ingest your data varies according to how that data is stored in its source. However, the start of the process is always to consider what data you have, and work out how you can shape it to fit the i2 Analyze schema.

About this task

Usually, when you start thinking about adding data from an external source into the Information Store, an i2 Analyze deployment is already in place. That deployment necessarily has an i2 Analyze schema that defines all of the entity types, link types, and property types that data in the system can have. Before you go any further, you must have a clear idea of how your data becomes i2 Analyze entity records and link records in the Information Store.

It is unlikely that the data in your external source has a one-to-one mapping with the entity types and link types in the i2 Analyze schema:

- Probably, your source does not contain data for all the entity types in the schema. As a result, you do not usually need to create a staging table for every possible entity type.
- The schema can define link types that connect several different entity types. In that case, each entity-link-entity type combination for which your source contains data requires a separate staging table.

For example, imagine an i2 Analyze schema that defines the entity types "Person", "Vehicle", and "Account", and the link type "Access to". In this situation, you might decide to create a staging table for each of the entity types. However, the data requires two staging tables for "Access to" links: one for links between people and vehicles, and the other for links between people and accounts.

Procedure

1. Open the schema for the i2 Analyze deployment in Schema Designer.
2. Go through the list of entity types, and determine which of them represent data in your source.
3. Make a note of the identifier of each entity type that represents your data.
4. Repeat steps 2 and 3 for the list of link types. Check the Link Ends tab, and make a note of all the combinations for which your source contains data.

Results

When you complete the steps, you have a list of all the i2 Analyze schema types that your data contains. You also have a list of all the staging tables that you need to create.

Creating the staging tables

The Information Store does not ingest data directly from your data source. Instead, ingestion takes place from staging tables that you create and populate. This abstraction makes it easier for you to align your data with the Information Store, and allows i2 Analyze to validate your data before ingestion.

Before you begin

The staging tables that you use to ingest data into the Information Store must conform to a specific structure. For more information about the staging table structure, see “Information Store staging tables” on page 28.

About this task

The simplest approach to Information Store ingestion is to create a staging table for every entity type, and every entity-link-entity type combination, that you identified in your data. The i2 Analyze deployment toolkit and the ETL toolkit both have a command for creating one staging table at a time.

The deployment toolkit command looks like this:

```
setup -t createInformationStoreStagingTable
      -p schemaTypeId=type_identifier
      -p databaseSchemaName=staging_schema
      -p tableName=staging_table_name
```

While the ETL toolkit command looks like this:

```
createInformationStoreStagingTable
      -stid type_identifier
      -sn staging_schema
      -tn staging_table_name
```

In both cases, *type_identifier* is the identifier of one of the entity types or link types from the i2 Analyze schema that is represented in your data source. *staging_schema* is the name of the database schema to contain the staging tables. (If you are using Db2, the command creates the database schema if it does not exist. If you are using SQL Server, the schema must exist.) *staging_table_name* is the name of the staging table itself, which must be unique, and must not exceed 21 characters in length.

Important: Many of the commands that are associated with the ingestion process modify the database that hosts the Information Store. By default, the commands use the database credentials that you specified during deployment in the `credentials.properties` file.

To use different credentials in the deployment toolkit, add `importName` and `importPassword` parameters to the list that you pass to the command. To use different credentials in the ETL toolkit, modify the `DBUsername` and `DBPassword` settings in the `Connection.properties` file.

Procedure

1. If you are using the deployment toolkit, open a command prompt and navigate to the `toolkit\scripts` directory. If you are using the ETL toolkit, navigate to the `etltoolkit` directory.
2. For each entity type or link type that you identified for ingestion, run the **`createInformationStoreStagingTable`** command.
   For example:

   ```
   setup -t createInformationStoreStagingTable
         -p schemaTypeId=ET5 -p databaseSchemaName=IS_Staging
         -p tableName=E_Person
   ```

   By convention, you create all of the staging tables for the same source in the same database schema, which has the name `IS_Staging` in this example. It is also conventional to name the staging table itself similarly to the display name of the entity type or link type to which the table corresponds. In this case, the staging table is for the Person entity type.

   Note: When the i2 Analyze schema allows the same link type between several different entity types, create separate staging tables for each combination:

```
setup -t createInformationStoreStagingTable
      -p schemaTypeId=LAC1 -p databaseSchemaName=IS_Staging
      -p tableName=L_Access_To_Per_Acc
```

```
setup -t createInformationStoreStagingTable
      -p schemaTypeId=LAC1 -p databaseSchemaName=IS_Staging
      -p tableName=L_Access_To_Per_Veh
```

This example illustrates an Access To link type (with identifier LAC1) that can make connections from Person entities to Account entities, or from Person entities to Vehicle entities. The commands create staging tables with different names based on the same link type.

Results

At the end of this procedure, you have a set of staging tables that are ready to receive your data before ingestion takes place. The next task is to make your data ready to populate the staging tables.

Preparing the external data

The staging tables that you create during the ingestion process have data structures that are similar to, but simpler than, the Information Store data tables. Whatever your source is, you must find a way to shape the data that it contains into a form that is compatible with the staging tables.

About this task

After you create the staging tables, you can view them in IBM Data Studio (or similar software) to see the definitions of their columns. You must make your data matches these definitions before you can go on to populate the staging tables.

Procedure

Because all data sources and many i2 Analyze schemas are different, there is no single procedure that you can follow to prepare your data for ingestion. However, there are a number of common considerations.

- Each staging table can contain data that maps to only one entity type or link type. If your source data has rows or records that contain data for more than one of the types that you identified, then you must separate them during preparation or population.

  For data in a relational source, this preparation might mean creating views on the original tables. If the data is in CSV files, then you might need to wait until you populate the staging tables to change its shape in this way.
- The Information Store does not support storing properties with multiple values in the same i2 Analyze record. The records that you create must contain values for a maximum of one property with each permitted property type.
- If you are dealing with date and time data, that data must meet extra requirements before the Information Store can ingest it. To retain information unambiguously, the staging tables use four columns to represent date and time data.

  Even if you know that your date and time data was recorded in Coordinated Universal Time, you must make that fact explicit in the data to be ingested. For example, if your source contains information about an event that started at 9 AM on October 6 2002, then the values you need to prepare are:

  `2002-10-06 09:00:00` (the data and time originally entered)

UTC (the timezone)
0 (Daylight Saving Time is not in effect)
2002-10-06 09:00:00 (the date and time in Coordinated Universal Time)

- The source_id, origin_id_type, origin_id_keys columns of the staging table are used to store values that reference the data in its original source and can be used to make up the origin identifier of the resulting record.

  Note: If the staging table definition was for a link type, it would also contain from_ and to_ variations of each of the columns.

  For more information about generating origin identifiers during ingestion, see "Origin identifiers" on page 39.
- If your external source is a relational database, you might find that the only data for some links is the presence of a foreign key relationship between two tables. In that case, you must synthesize a reproducible reference for the link from the other data that you have available.

  For example, you might be able to create a unique reference for a link by combining the identifiers of the entity records at its ends.
- All staging tables contain a source_last_updated column that you can use to store information about when the data to be ingested was modified in its source.
- All staging tables contain columns for each of the access dimensions that the security schema defines. If your external source includes security information, then you can map that information to the security schema of your target deployment, and populate the staging table columns accordingly.

  Alternatively, you can leave the security columns blank, and provide security dimension values on a mapping- or source-wide basis later in the ingestion process.
- All staging tables contain correlation_id_type and correlation_id_key columns. To correlate data that is ingested into the Information Store, use these columns to store the values that comprise the correlation identifier for each row of data. If you do not want to use correlation, leave the columns blank.

  If you specify values for a correlation identifier, then also specify a value for the source_last_updated column, which is used during the correlation process.

  For more information about correlation, correlation identifiers, and the impact of the source_last_updated value, see Overview of correlation.
- The columns named source_ref_source_type, source_ref_source_location, and source_ref_source_image_url are used to populate the source reference that is generated when the data is ingested.

  For more information about implementing source references in your deployment, see Configuring source references.
- The staging tables for link types contain a column for the direction of the link.

  The Information Store considers links to go "from" one entity "to" another. The direction of a link can be WITH or AGAINST that flow, or it can run in BOTH directions, or NONE.

  – If your link data includes direction information, then you can add it to the staging table during the population process, and then refer to it from the mapping file.
  – If your link data does not include direction information, then you can specify a value in the mapping file directly.

  By default, if you have no direction information and you do nothing in the mapping file, the Information Store sets the direction of an ingested link to NONE.

Important: The Information Store places limits on the ranges of values that properties with different logical types can contain. If you attempt to use values outside these ranges, failures can occur during or after ingestion. For more information, see “Information Store property value ranges” on page 31.

Example

The `examples\data\law-enforcement-data-set-1` directory of the deployment toolkit contains a set of CSV files that were exported from a relational database.

In files like `event.csv`, you can see date and time data that meets the requirements of the staging tables. You can also see multiple files for "Access to" links, and how some staged link rows contain little more than a set of identifiers.

Populating the staging tables

The i2 Analyze deployment toolkit and the ETL toolkit create the staging tables for data ingestion in the same database as the Information Store data tables. After you prepare your data, but before you can instruct the Information Store to ingest it, you must populate the staging tables.

About this task

The approach that you take to populate the staging tables depends on the database management system that you are using, the form that your source data is in, and the tools that you have available.

Db2 provides the ingest, import, and load utilities:

- If your data is in comma-separated value (CSV) files, then you can use the **IMPORT** or **INGEST** commands.
- If your data is in the tables or views of another database, then you can use the **IMPORT**, **INGEST**, or **LOAD** commands.

SQL Server provides the bulk insert and insert utilities:

- If your data is in comma-separated value (CSV) files, then you can use the **BULK INSERT** command.

  Note: To use the **BULK INSERT** command, the user that you run the command as must be a member of the *bulkadmin* server role.
- If your data is in the tables or views of another database, then you can use the **INSERT** command.
- You can use SQL Server Integration Services as a tool to extract and transform data from various sources, and then load it into the staging tables.

Alternatively, regardless of your database management system, you can use IBM InfoSphere DataStage as a tool for transforming your data and loading it into the staging tables. You can specify the database schema that contains the staging tables as the target location for the ETL output.

Example

The subdirectories of the `examples\data` directory in the deployment toolkit all contain a db2 and a `sqlserver` directory.

If you are using Db2, inspect the `LoadCSVDataCommands.db2` file in the db2 directory. In each case, this file is a Db2 script that populates the example staging tables from the prepared CSV files. The script calls the **IMPORT** command repeatedly to do its work. In most instances, the command just takes data from columns in a CSV file and adds it to a staging table in a Db2 database schema.

If you are using SQL Server, inspect the `LoadCSVDataCommands.sql` file in the `sqlserver` directory. In each case, this file is an SQL script that populates the example tables from the prepared

CSV files. The script calls the **BULK INSERT** command repeatedly to do its work. The **BULK INSERT** command uses `.fmt` format files, which are also in the `sqlserver` directory, to instruct SQL Server how to process the CSV files into the staging tables. For more information about format files, see Non-XML Format Files.

Defining an ingestion source

The Information Store keeps a list of all the sources from which it has ingested data. Before it can ingest data, you must tell the Information Store about your source. In the ingestion mapping file, you then specify the data source name in the mapping definition for each entity type and link type.

About this task

The i2 Analyze deployment toolkit and the ETL toolkit both have a command for adding information about an ingestion source to the Information Store.

The deployment toolkit command looks like this:

```
setup -t addInformationStoreIngestionSource
      -p ingestionSourceName=src_name
      -p ingestionSourceDescription=src_display_name
```

While the ETL toolkit command looks like this:

```
addInformationStoreIngestionSource
      -n src_name
      -d src_display_name
```

In both cases, *src_name* is a unique name for the ingestion source, which also appears in the mapping file. *src_display_name* is a friendlier name for the ingestion source that might appear in the user interface of applications that display records from the Information Store.

Important: The value that you provide for *src_name* must be 30 characters or fewer in length. Also, do not use the word ANALYST as the name of your ingestion source. That name is reserved for records that analysts create in the Information Store through a user interface.

Procedure

1. If you are using the deployment toolkit, open a command prompt and navigate to the `toolkit\scripts` directory. If you are using the ETL toolkit, navigate to the `etltoolkit` directory.
2. Run the **addInformationStoreIngestionSource** command, specifying the short and display names of your ingestion source.
   For example:

   ```
   setup -t addInformationStoreIngestionSource
         -p ingestionSourceName=EXAMPLE
         -p ingestionSourceDescription="Example data source"
   ```

   If the Information Store already contains information about an ingestion source with the name EXAMPLE, this command has no effect.

Results

After you complete this task, you have performed all the necessary actions, and gathered all the necessary information, to be able to write ingestion mapping files. The next task is to create the ingestion mapping file for your ingestion source.

Creating an ingestion mapping file

The mappings in an ingestion mapping file define how rows in staging tables become i2 Analyze records in the Information Store during the ingestion process. Each mapping that you write describes how to construct the origin identifiers for data of a particular type, and specifies the security dimension values that apply to records.

Before you begin

Before you create your ingestion mappings, review the information about origin identifiers and security dimension values in i2 Analyze.

About this task

The Information Store ingestion mechanism makes it possible for you to develop and extend your ingestion mappings over time. You can test your approach to ingestion by writing and using a single (entity type) mapping, and then adding more entity type and link type mappings later. You can put all your mappings in one file, or put each mapping in a separate file, or anything between those two extremes.

Procedure

If you populated the staging tables successfully, then writing ingestion mappings can be straightforward. Eventually, you need a mapping for each staging table that you created, but you can approach the problem one mapping at a time.

1. Choose a populated staging table for an entity type that has links to other entity records in the data model.
2. Create an ingestion mapping file that contains an ingestion mapping for the staging table that you chose in step 1.

   If you prefer to start from an existing file, look at `mapping.xml` in the `examples\data\law-enforcement-data-set-1` directory of the deployment toolkit.
3. Run the ingestion command to validate the mapping.

   If you are unhappy with the outcome, edit the ingestion mapping and run the command again.
4. Repeat all of the preceding steps for all the other staging tables that you populated.

Example

The `examples\data\law-enforcement-data-set-1` directory of the deployment toolkit contains an ingestion mapping file named `mapping.xml`. This file contains ingestion mappings for all the staging tables that the ingestion example creates. You can use `mapping.xml` as the basis for the ingestion mappings that you need for your data.

Validating the ingestion configuration

After you create and populate your staging tables and write your ingestion mappings, the final part of the process is to run the command to validate your work.

About this task

When you instruct the Information Store to ingest the data that you loaded into the staging tables, you do it one ingestion mapping (and one staging table) at a time. The i2 Analyze deployment toolkit and the ETL toolkit both have a command for validating your staging tables and a particular ingestion mapping in a particular mapping file.

Procedure

The procedure for instructing the Information Store to ingest your data is similar to many others in this process. You start with one type or one staging table, and build from there.

1. Choose an entity staging table that you populated with data and provided with an ingestion mapping.
2. Run the **`ingestInformationStoreRecords`** command in `VALIDATE` mode.
   For example:

   ```
   setup -t ingestInformationStoreRecords -p importMappingsFile=mapping.xml
         -p importMappingId=Person -p importMode=VALIDATE
   ```

   When you specify `importMode` and set it to `VALIDATE`, the command checks the validity of the specified mapping, but no ingestion takes place. For more information about the running the command and any arguments, see "The ingestInformationStoreRecords toolkit task" on page 42.

   The output to the console indicates whether the mapping you identified is valid, provides guidance when it is not valid, and gives a full list of column mappings. The command sends the same information to a log file that you can find at `toolkit\configuration\logs\importer \IBM_i2_Importer.log`.
3. Complete this validation step for each staging table and ingestion mapping that you plan to use.

   The ingestion process for links verifies that the entities at each end of the link are already ingested. If it fails to find them, the process fails. When you are developing your ingestion process, ingest a small amount of entity data before you validate your links.

What to do next

Correct any problems in the ingestion mappings file (or any ingestion properties file that you specified) before you proceed to "Adding data to the Information Store" on page 13.

Adding data to the Information Store

After you populate the staging tables and write ingestion mappings, you can use toolkit commands to instruct the Information Store to ingest or update the records that represent external data. The Information Store keeps a log of all such instructions that you can review to determine the success or failure of each one.

About this task

The commands in the i2 Analyze deployment and ETL toolkits make it possible for you to create, update, and delete records in the Information Store. All three operation types are controlled by the data in the staging tables and the mappings in the ingestion mapping files.

After any operation that uses toolkit commands to change the contents of the Information Store, you can examine ingestion reports to determine how successful the operation was.

As described in "Information Store data ingestion" on page 1, there are 2 different import modes that you can use to ingest your data. Before you run the ingestion commands, ensure that you use the correct import mode for the data that you want to ingest. Remember that this might differ depending on the item type that you are ingesting or ingestion mapping that you are using.

Using bulk import mode

Bulk import mode can be used for improved ingestion performance when you are populating the Information Store with new records as part of an initial or periodic load.

Before you begin

- If you are using Microsoft SQL Server, the bulk import mode conforms to the requirements and conditions that are described in Using bulk mode for faster ingestion.
- Before you run the ingestion commands, ensure that you complete all of the tasks in "Preparing for ingestion" on page 4.
- The mechanism that bulk import mode uses to load the data from the staging tables to the Information Store can require changes to the Db2 configuration. Before you ingest data, ensure that your database is configured correctly. For more information, see "Database configuration" on page 15.
- You can drop and re-create the database indexes during the ingestion process, which can reduce the total time that it takes to ingest data. For more information about when it is beneficial to drop and re-create the database indexes, see "Database index management" on page 15.

About this task

When you instruct the Information Store to ingest the data that you loaded into the staging tables, you do it one ingestion mapping (and one staging table) at a time. The i2 Analyze deployment toolkit and the ETL toolkit both have a command for ingesting the data that is associated with a particular ingestion mapping in a particular mapping file.

When you use bulk import mode, take the following points into consideration:

- For best performance, ensure that there are no analysts using the system during the ingestion process.
- Do not include correlation identifiers in the data that you ingest by using bulk import mode. If any correlation identifiers are included, the ingestion fails.
- You cannot run concurrent `ingestInformationStoreRecords` commands when using bulk import mode.

Procedure

After you stage your data and create your configuration files, you can instruct the Information Store to ingest your data by using bulk import mode. The procedure to ingest your data is similar to many others in this process. You start with one type or one staging table, and build from there.

1. If you have decided to drop the database indexes, do so now.
2. Use the following command to ingest your data.

```
setup -t ingestInformationStoreRecords
      -p importMappingsFile=ingestion_mapping_file
      -p importMappingId=ingestion_mapping_id
      -p importLabel=ingestion_label
      -p importConfigFile=ingestion_settings_file
      -p importMode=BULK
```

For more information about the running the command and any arguments, see "The ingestInformationStoreRecords toolkit task" on page 42.

3. If any errors occur, refer to "Troubleshooting the ingestion process" on page 46.
4. If you need to re-create the database indexes, do so now.

Database configuration

When you use bulk import mode, the way that the data is inserted into the Information Store from the staging table requires more transaction log space and database locks available than standard import mode.

Transaction log size

For entity ingestion, allocate at least 1.5 GB of transaction log space.

For link ingestion, allocate at least 22 GB of transaction log space.

For more information about changing the Db2 log file size, see logfilsiz - Size of log files configuration parameter, logprimary - Number of primary log files configuration parameter, and logsecond - Number of secondary log files configuration parameter.

Lock list size

For entity ingestion, the formula for calculating the required lock list size is:

```
1200000 * (lock_record_size / 4000)
```

If the lock record size for your deployment is 256 bytes, you need a lock list size of 75,000.

For link ingestion, the formula for calculating the required lock list size is:

```
3600000 * (lock_record_size / 4000)
```

If the lock record size for your deployment is 256 bytes, you need a lock list size of 225,000. Link ingestion requires an increased lock list size because more tables in the Information Store are written to.

For information about lock lists, see locklist - Maximum storage for lock list configuration parameter.

Database index management

It can be beneficial for performance to drop and re-create indexes in the Information Store database during bulk mode ingestion. The situations when dropping and creating the database indexes might help change over the lifetime of a deployment as your Information Store contains more data.

Decide when to drop and create indexes

The situations where you might want to drop and re-create the database indexes include:

- Completing an initial load for an item type or ingesting data into an empty Information Store
- When ingesting large amounts of data (For example, more than 20 million rows in the staging table)

The situation where you do not want to drop and re-create the database indexes include:

- When the Information Store contains a large amount of data for the item type you are ingesting, the time that it takes to re-create the indexes can make the total ingestion time longer. If you are ingesting into an Information Store that already contains about 1 billion records, do not drop and re-create the database indexes.
- If you are ingesting links between entities of the same type and some of those entities might be correlated, do not drop and re-create the database indexes.

By default, the indexes are kept in place and not dropped during data ingestion.

To help determine what is best for your data, you can test some ingestions that drop and re-create indexes and some that don't. When you complete the test ingestions, record the time that it takes to complete. You can use these times to inform you whether to manage the database indexes during ingestion or not. As the amount of data in the Information Store increases, the time it takes to create the indexes also increases.

Dropping and re-creating indexes during a bulk import ingestion process

If you decide to drop and re-create the database indexes during large ingestions that use multiple staging tables across both entity and link types, the high-level process consists of the following steps:

1. Stop Liberty and Solr
2. Drop the database indexes for the entity types that you are ingesting data for
3. Ingest all entity data
4. Create the database indexes for the entity types that you ingested data for
5. Drop the database indexes for the link types that you are ingesting data for
6. Ingest all link data
7. Create the database indexes for the link types that you ingested data for
8. Start Liberty and Solr

You can use two methods to drop and create indexes:

- Use the i2 Analyze deployment toolkit to generate scripts that you run against the database manually.
- Use the import configuration properties file to specify whether the indexes must be dropped or created during a bulk import mode ingestion.

If you are ingesting data for multiple link types or you want to review the database scripts and run them manually, it is best to use the generated scripts to drop and re-create the database indexes. If you do not want to, or cannot, run scripts against the Information Store database, you can use the import configuration file and allow the ingestion process to drop and re-create the database indexes.

If you are completing an ingestion that spans multiple ingestion periods where the system is in use between ingestion periods, ensure that all of the indexes are created at the end of an ingestion period and that you start Liberty and Solr before analysts use the system.

For information about creating the import configuration file, see References and system properties.

Using the generated scripts method

1. In your import configuration file, set both the `dropIndexes` and `createIndexes` settings to `FALSE`.

For example:

```
dropIndexes=FALSE
createIndexes=FALSE
```

2. Generate the *create* and *drop* index scripts for each item type that you are ingesting data for.

   For example, to generate the scripts for the item type with identifier ET5, run the following commands:

   ```
   setup -t generateInformationStoreIndexCreationScripts -p schemaTypeId=ET5
   setup -t generateInformationStoreIndexDropScripts -p schemaTypeId=ET5
   ```

   The scripts are output in the `toolkit\scripts\database\db2\InfoStore\generated` directory, in the `createIndexes` and `dropIndexes` directories.

3. Stop Liberty and Solr.

   On the Liberty server, run:

   ```
   setup -t stopLiberty
   ```

   On each Solr server, run:

   ```
   setup -t stopSolrNodes --hostname solr.host-name
   ```

   Where *solr.host-name* is the host name of the Solr server where you are running the command, and matches the value for the `host-name` attribute of a `<solr-node>` element in the `topology.xml` file.

4. Run the scripts that you generated in step 2 to drop the indexes for each entity type that you plan to ingest data for. For example:

   ```
   db2 -tvf ET5-drop-indexes.db2
   ```

   Where *ET5* is the item type identifier.

5. Complete the process to ingest the entity data using the bulk import mode. For more information, see “Using bulk import mode” on page 14.

6. Run the scripts that you generated in step 2 to create the indexes for each entity type that you ingested. For example:

   ```
   db2 -tvf ET5-create-indexes.db2
   ```

7. Run the scripts that you generated in step 2 to drop the indexes for each link type that you plan to ingest data for. For example:

   ```
   db2 -tvf LT1-drop-indexes.db2
   ```

   Where *LT1* is the item type identifier.

8. Complete the process to ingest the link data using the bulk import mode. For more information, see “Using bulk import mode” on page 14.

   Only re-create the database indexes after you ingest the link data for every link type that you plan to ingest.

9. Run the scripts that you generated in step 2 to create the indexes for each link type that you ingested. For example:

```
db2 -tvf LT1-create-indexes.db2
```

10. Start Liberty and Solr.

    On each Solr server, run:

```
setup -t startSolrNodes --hostname solr.host-name
```

    Where *solr.host-name* is the host name of the Solr server where you are running the command, and matches the value for the `host-name` attribute of a `<solr-node>` element in the `topology.xml` file.

    On the Liberty server, run:

```
setup -t startLiberty
```

Using the import configuration properties file method

When you use the import configuration settings to drop and re-create the database indexes, the toolkit creates and drops the indexes for you as part of the ingestion process instead of running scripts against the database manually. However, you must modify the import configuration file a number of times throughout the ingestion of multiple item types. You must still stop Liberty and Solr before you run the ingestion command, and start them again after the indexes are created.

If all of the data for a single item type is ingested with a single ingestion command, in your import configuration file set the `dropIndexes` and `createIndexes` settings as follows:

```
dropIndexes=TRUE
createIndexes=TRUE
```

If the data for a single item type must be ingested by using multiple ingestion commands, you need to modify the import configuration file before the first ingestion command, for the intermediate commands, and before the final command for each item type. For example, if your data for a single entity type is in more than one staging table.

1. The first time you call the ingestion command, set the `dropIndexes` and `createIndexes` settings as follows:

```
dropIndexes=TRUE
createIndexes=FALSE
```

2. For the intermediate times that you call the ingestion command, set the `dropIndexes` and `createIndexes` settings as follows:

```
dropIndexes=FALSE
createIndexes=FALSE
```

3. The final time you call the ingestion command, set the `dropIndexes` and `createIndexes` settings as follows:

```
dropIndexes=FALSE
createIndexes=TRUE
```

After you ingest all the entity data, repeat the process for the link data.

Using standard import mode

After you create and populate your staging tables, write your ingestion mappings, and validate your work, instruct the Information Store to ingest your data.

Before you begin

If you are using correlation, you must ensure that the application server is started before you run the ingestion commands.

About this task

When you instruct the Information Store to ingest the data that you loaded into the staging tables, you do it one ingestion mapping (and one staging table) at a time. The i2 Analyze deployment toolkit and the ETL toolkit both have a command for ingesting the data that is associated with a particular ingestion mapping in a particular mapping file.

Procedure

The procedure for instructing the Information Store to ingest your data is similar to many others in this process. You start with one type or one staging table, and build from there.

1. Choose an entity staging table that you populated with data and provided with an ingestion mapping.
2. Run the **ingestInformationStoreRecords** command in STANDARD mode to instruct the Information Store to ingest your data.
   For example:

   ```
   setup -t ingestInformationStoreRecords
         -p importMappingsFile=ingestion_mapping_file
         -p importMappingId=ingestion_mapping_id
         -p importLabel=ingestion_label
         -p importConfigFile=ingestion_settings_file
         -p importMode=STANDARD
   ```

   For more information about the running the command and any arguments, see "The ingestInformationStoreRecords toolkit task" on page 42.

   Note:

   - You can improve the performance of entity ingestion by running **ingestInformationStoreRecords** for different entity types at the same time when you use the standard import mode. It is recommended that you use the ETL toolkit to run concurrent **ingestInformationStoreRecords** commands. *Do not* attempt to run the command for data of the same type at the same time.
   - Due to the increased number of operations and comparisons that are required, using correlation can make the ingestion process take longer.
3. Repeat steps 1 and 2 for the other staging table and ingestion mapping combinations that you created. Take care to run the command for entity types before you run it for link types.

   The ingestion process for links verifies that the entities at each end of the link are already ingested. If it fails to find them, the process fails.

Results

At the end of this procedure, all the external data that you populated the staging tables with is in the Information Store. To add or update records, you can repopulate the staging tables and rerun the **`ingestInformationStoreRecords`** command.

Updating the Information Store for changed data

The data that the Information Store ingests is fixed at the moment of ingestion, and changes to the data in its source do not automatically update the Information Store. However, you can update the Information Store to reflect changes in an external source by running through the ingestion process again.

About this task

For most changes to the data in an external source, it is likely that you can reuse the work that you did to enable initial ingestion. If the changes to an external source are not significant enough to affect your method for generating reproducible origin identifiers, repeat ingestion follows the same process as initial ingestion.

Procedure

1. Examine the new or changed data in the external source, and your ingestion mappings. Confirm that your configuration still generates origin identifiers that the Information Store can compare with their equivalents in existing ingested data.
2. Delete the contents of each staging table that you know to be affected by changes to the external data.
3. Populate the affected staging tables with the latest data from your external source.
4. Run the ingestion command specifying the standard import mode for each ingestion mapping that refers to an affected staging table, taking care to process entity data before link data, as usual.

   The Information Store uses the origin identifier of each row that it attempts to ingest to determine whether the data is new:

   - If the origin identifier does not match the origin identifier of any data that is already in the Information Store, then the data is new to the Information Store. It is ingested in the usual way.
   - If the origin identifier does match the origin identifier of any data that is already in the Information Store, then the staging table contains updated information. The Information Store clears its existing data and refills it with the new data.

   Note: If the correlation identifier changed, additional merge and unmerge operations might occur.

Results

After you follow this procedure, the Information Store contains new data that was added to an external source since the last ingestion. It also contains updated data that was changed in an external source since the last ingestion.

Updating the Information Store for deleted data

The data that the Information Store ingests is fixed at the moment of ingestion, and removal of the data in the external source does not automatically delete it from the Information Store. However, you

can update the Information Store to reflect the deletion of data in the external source by using staging tables and the deployment toolkit.

Before you begin

When data changes in its original source, you can use the same pipeline that you used for initial ingestion to update the records in the Information Store. If data is deleted from its source, you can use the staging tables and the deployment toolkit to reflect that fact in the Information Store as well.

A single i2 Analyze record can represent data from multiple sources, which results in a record that contains multiple pieces of provenance. As a consequence, responding to source data deletion does not necessarily mean deleting records from the Information Store. When you use the toolkit to reflect deleted source data, the effect is to remove the provenance associated with that data. If the process removes a record's only provenance, the record is deleted. If not, the record remains in the Information Store.

Note: To delete records from the Information Store explicitly, use the deletion-by-rule approach. You can write conditions to determine which records are deleted. For more information about deleting records in this way, see Deleting records by rule.

About this task

The commands to update the Information Store for deleted data use the same mapping file and staging tables as the commands for ingesting data, and you call them in a similar way. However, the only information that *must* be in the staging table is what the mapping file requires to generate the origin identifiers of the data that is no longer in the external source.

When you run the commands to update the Information Store for deleted data, the rules that apply differ from the rules for adding and updating data:

- Links do not have to be processed before entities, or vice versa.
- Links can be processed without specifying the origin identifiers of their ends.
- Deleting a piece of provenance from an entity record also deletes all the link provenance that is connected to it.
- The process silently ignores any origin identifiers that are not in the Information Store.

Because this process might cause significant numbers of i2 Analyze records to be deleted, two commands are provided. The first command previews the effect of running the second command before you commit to doing so. In the deployment toolkit, the two commands have different names but the same syntax:

```
setup -t previewDeleteProvenance
      -p importMappingsFile=ingestion_mapping_file
      -p importMappingId=ingestion_mapping_id
```

```
setup -t deleteProvenance
      -p importMappingsFile=ingestion_mapping_file
      -p importMappingId=ingestion_mapping_id
      -p importLabel=ingestion_label
      -p logConnectedLinks
```

In the ETL toolkit, you reuse the `ingestInformationStoreRecords` command with two new mode parameters:

```
ingestInformationStoreRecords
      -imf ingestion_mapping_file
      -imid ingestion_mapping_id
      -im DELETE_PREVIEW
```

```
ingestInformationStoreRecords
      -imf ingestion_mapping_file
      -imid ingestion_mapping_id
      -il ingestion_label
      -lcl true
      -im DELETE
```

In all cases, *ingestion_mapping_file* is the path to the XML file that contains the mapping that you want to use, and *ingestion_mapping_id* is the identifier of the mapping within that file. The latter is mandatory unless the file contains only one mapping.

When `logConnectedLinks` is specified, any links that are removed as part of an entity delete operation are logged in `IS_Public.D<import identifier><entity type id>_<link type id>_Links` tables. For example, `IS_Public.D20180803090624143563ET5_LAC1_Links`.

Previewing the delete operation does not create an entry in the `Ingestion_Deletion_Reports` view, so you do not need to specify a label in that case. The delete operation does populate that view, and *ingestion_label* is then an optional parameter.

Procedure

The procedure for updating the Information Store in this way starts with a staging table that contains information about the data that you no longer want to represent in the Information Store.

1. Ensure that the application server that hosts i2 Analyze is running.
2. Run the `previewDeleteProvenance` command to discover what the effect of running `deleteProvenance` is.
   For example:

   ```
   setup -t previewDeleteProvenance -p importMappingsFile=mapping.xml
         -p importMappingId=Person
   ```

   The output to the console window describes the outcome of a delete operation with these settings. High counts or a long list of types might indicate that the operation is going to delete more records than you expected.

```
>INFO [DeleteLogger] - Delete preview requested at 2017.12.08 11:05:32
>INFO [DeleteLogger] - Item type: Person
>INFO [DeleteLogger] - Number of 'Person' provenance pieces to be deleted: 324
>INFO [DeleteLogger] - Number of 'Person' i2 Analyze records to be deleted: 320
>INFO [DeleteLogger] - Number of 'Access To' provenance pieces to be deleted:
187
>INFO [DeleteLogger] - Number of 'Access To' i2 Analyze records to be deleted:
187
>INFO [DeleteLogger] - Number of 'Associate' provenance pieces to be deleted:
27
>INFO [DeleteLogger] - Number of 'Associate' i2 Analyze records to be deleted:
27
>INFO [DeleteLogger] - Number of 'Employment' provenance pieces to be deleted:
54
>INFO [DeleteLogger] - Number of 'Employment' i2 Analyze records to be
deleted: 54
>INFO [DeleteLogger] - Number of 'Involved In' provenance pieces to be
deleted: 33
>INFO [DeleteLogger] - Number of 'Involved In' i2 Analyze records to be
deleted: 33
>INFO [DeleteLogger] - Duration: 1 s
```

Note: When you run the command for entity records, the output can exaggerate the impact of the operation. If the staging table identifies the entities at both ends of a link, the preview might count the link record twice in its report.

3. Correct any reported problems, and verify that the statistics are in line with your expectations for the operation. If they are not, change the contents of the staging table, and run the preview command again.
4. Run the `deleteProvenance` command with the same parameters to update the Information Store.
   For example:

```
setup -t deleteProvenance -p importMappingsFile=mapping.xml
      -p importMappingId=Person -p importLabel=DeletePeople
      -p logConnectedLinks
```

   Note: Do not run multiple `deleteProvenance` commands at the same time, or while data is being ingested into the Information Store.
5. Repeat steps 1, 2, and 3 for the types of any other records that you want to process.

Results

At the end of this procedure, the Information Store no longer contains the provenance (or any connected link provenance) for the data that you identified though the mapping files and staging tables. Any records that lose all of their provenance, and any connected link records, are deleted as a result. Deleting data is permanent, and the only way to restore it to the Information Store is to add it again through the `ingestInformationStoreRecords` command.

Ingestion resources

The ingestion resources section contains information that is referenced elsewhere in the ingestion section.

Understanding the architecture

The physical architecture of your i2 Analyze deployment both affects and is affected by how you acquire and transform external data for the Information Store. Depending on the architecture, you might need to perform more deployment tasks before you can run data ingestion commands.

About this task

A complete solution for loading and ingesting data into the Information Store has four mandatory architectural components:

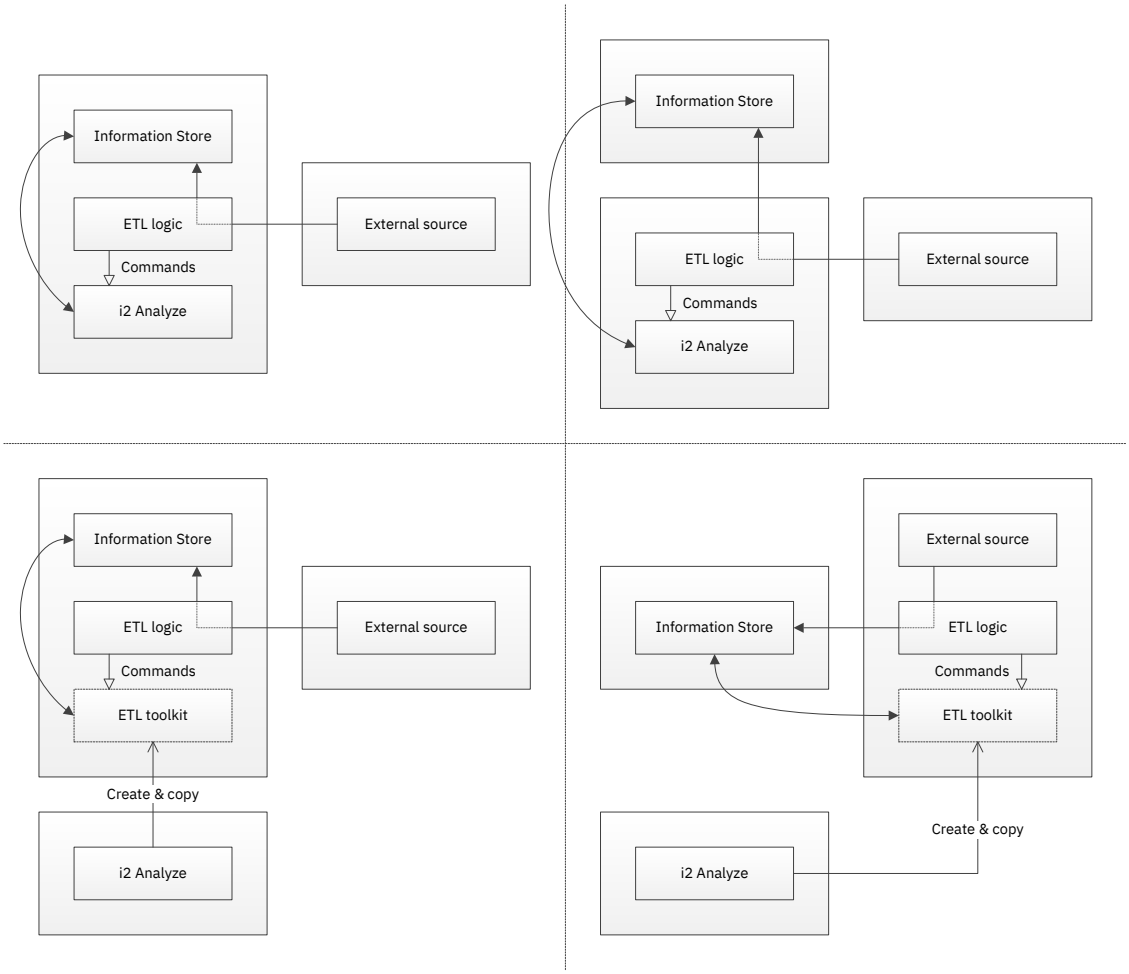
- The i2 Analyze server, and in particular the deployment toolkit that it contains
- The database management system that contains the Information Store and the staging tables
- An external data source
- The ETL logic that transforms the source data and loads it into the staging tables

Note: In addition to the mandatory components, you can opt to include a component that prepares your data for correlation, such as a matching engine or a context computing platform. These components can help you to determine when data in multiple different sources represents the same real-world "thing". This preparation of your data might occur as part of your ETL logic, or between an external data source and the ETL logic.

i2 Analyze supports physical architectures in which the database is hosted on the same server as the application, or on a different one. You can also choose to locate your ETL logic on the same server as the i2 Analyze application, or on the same server as the database, or on an entirely separate server.

The process of transforming source data can be demanding, especially if the requirements are complex or the volume is high. There are also scenarios in which you might want to automate the process of loading and then ingesting the external data. Ultimately, the architecture that you decide upon depends on the needs and constraints of your deployment.

The following diagram shows some of the permutations. The examples in the upper-left and upper-right quadrants represent deployments in which the ETL logic (implemented by a tool like IBM DataStage, for example) is co-hosted with the i2 Analyze application. The database can be on the same or a separate server; the solid arrows show data flow between the components during data load and ingestion.



The examples in the lower-left and lower-right quadrants represent deployments in which the ETL logic is on a separate server from the i2 Analyze application. (Typically, the ETL logic is hosted alongside the database or the external data source.) To enable the architecture, those deployments include the *ETL toolkit*, which is a cut-down version of the main deployment toolkit that targets only data ingestion.

When you need the ETL toolkit, you can generate it on the i2 Analyze server, and copy it to the server that hosts the ETL logic. When the ETL toolkit is properly configured, your ETL logic can run toolkit commands without reference to the rest of the deployment. If you decide to use the ETL toolkit, the next step is to deploy it. If not, you can move on to creating staging tables in the database.

Procedure

As the diagrams show, the ETL toolkit is most likely to be useful in deployments where the i2 Analyze application and the ETL logic are on separate servers. As you plan your approach to ingestion, consider the following:

- If the ETL logic is relatively simple and data volumes are low, there are benefits to colocating as many components as you can, especially in a new deployment.

- If your deployment requires separate servers from the start, or as it evolves over time, determine where the bottlenecks are. Is it limited by server speed or network speed?
- If the ETL logic is taxing a server that hosts other components, consider moving the logic, but be aware of the increase in network traffic.
- If the volume of data is taxing the network, consider colocating components when you are able. (You might not have permission to deploy components to some servers, for example.)

Results

By acting as a proxy for the i2 Analyze deployment toolkit, the ETL toolkit provides for more flexibility in your choice of architecture. In some circumstances you can separate the database, the ETL logic, and the i2 Analyze application without incurring a networking penalty.

Deploying the ETL toolkit

If your deployment includes logic that extracts, transforms, and loads data on a different server from the i2 Analyze application or the Information Store, consider deploying the ETL toolkit. The ETL logic can then run ETL toolkit commands to automate loading and ingesting data into the Information Store.

About this task

In an i2 Analyze deployment that uses data from an external source, the ETL logic is the processing that transforms source data for loading into the Information Store staging tables. In mature deployments, it is common for the ETL process to be automated so that loading and ingesting data happen in sequence, on a schedule.

When your ETL logic is colocated with the standard i2 Analyze deployment toolkit, the logic can use that toolkit to drive the ingestion process automatically. When those components are on separate servers, you can deploy the ETL toolkit to the server that hosts the ETL logic. The ETL toolkit provides the ingestion functions of the deployment toolkit in a stand-alone package.

Procedure

The ETL toolkit must be able to communicate with the Information Store with all the same credentials as the deployment toolkit. To enable this behavior, you use the deployment toolkit to create the ETL toolkit, and then copy it to the ETL logic server.

1. On the server that has the deployment toolkit, open a command prompt and navigate to the `toolkit\scripts` directory.
2. Run the `createEtlToolkit` command to generate the ETL toolkit:

   ```
   setup -t createEtlToolkit -p outputPath=output_path
   ```

   This command creates the ETL toolkit in a directory that is named `etltoolkit` in the output path that you specify.
3. Copy the ETL toolkit to the server that hosts the ETL logic.

If the ETL logic and toolkit are on the same server as the database management system that hosts the Information Store, you do not need to modify the connection configuration. If the database management system is on a different server, then you must ensure that the ETL toolkit can communicate with the remote database.

4. Dependent on your database management system, install either Db2® client software or Microsoft Command Line Utilities for SQL Server on the server that hosts the ETL toolkit.

   For more information, see Software Prerequisites.

5. Navigate to the `classes` directory of the ETL toolkit and open the `Connection.properties` file in a text editor.
6. Ensure that the value for the `db.installation.dir` setting is correct for the path to the Db2 client or Microsoft Command Line Utilities for SQL Server on the server that hosts this ETL toolkit. For example:

   ```
   db.installation.dir=C:/Program Files/IBM/SQLLIB
   ```

7. If you are using Db2 to host the Information Store, you must catalog the remote Db2 database. Run the following commands to enable the ETL toolkit to communicate with the Information Store:

   ```
   db2 catalog tcpip node node-name host-name server port-number
   db2 catalog database instance-name at node node-name
   ```

   Here, *host-name*, *port-number*, and *instance-name* are the values that are specified in the `topology.xml` file. *node-name* can be any value that you choose, but you must use the same value in both commands.

If the database management system that hosts the Information Store is not using SSL, then the process is complete.

If the database management system is configured to use SSL, you must also enable the ETL toolkit to communicate by using SSL.

8. Register the `i2-database_management_system-certificate.cer` certificate that you exported from the database management system when you configured SSL on the server that hosts the ETL toolkit.
   - On Windows, import the certificate into the Trusted Root Certification Authorities store for the current user.
   - On Linux, copy the certificate to the `/etc/pki/ca-trust/source/anchors` directory and use **update-ca-trust** to enable it as a system CA certificate.
9. Create a truststore and import into the truststore the certificate that you exported from the database management system when you configured SSL.
   For example, run the following command:

   ```
   keytool -importcert -alias "dbKey"
           -file C:\IBM\etltoolkit\i2-database_management_system-certificate.cer
           -keystore "C:\IBM\etltoolkit\i2-etl-truststore.jks"
           -storepass "password"
   ```

   Enter `yes` in response to the query, `Trust this certificate?`
10. Navigate to the `classes` directory of the ETL toolkit and open the `TrustStore.properties` file in a text editor.
11. Populate the `DBTrustStoreLocation` and `DBTrustStorePassword` properties with the full path to the truststore that you created, and the password that is required to access it.
    For example:

    ```
    DBTrustStoreLocation=C:/IBM/etltoolkit/i2-etl-truststore.jks
    DBTrustStorePassword=password
    ```

12. You can use the Liberty profile **securityUtility** command to encode the password for the truststore.

a) Navigate to the `bin` directory of the WebSphere® Application Server Liberty profile deployment that was configured by the deployment toolkit.

b) In a command prompt, run `securityUtility encode` *`password`*, which generates and displays the encoded password.

   Use the entire value, including the `{xor}` prefix, for the `DBTrustStorePassword` property value. For more information about using the security utility, see securityUtility command.

Results

The ETL toolkit is ready for use by your ETL logic to modify the Information Store. At key points in the processes of preparing for and performing ingestion, you can use commands in the ETL toolkit in place of deployment toolkit functions.

Information Store staging tables

At your request, i2 Analyze generates an Information Store staging table that can contain data for i2 Analyze records of a single entity type or link type. To generate the staging table, it uses information from the i2 Analyze schema, which is the same starting point from which it generates the main data tables during deployment.

An entity type staging table contains:

- At least one column for each property type in the schema.
- Columns for storing information that uniquely identifies data in its source. During ingestion, i2 Analyze can use this information to construct an origin identifier for the ingested data.
- Two columns to record when the data was created and updated in the source.
- Three columns for storing information that describes the source of the data. During ingestion, i2 Analyze uses this information to populate a source reference for the ingested data.
- Two columns to record the *correlation identifier type* and *correlation identifier key* of the data. During ingestion, i2 Analyze uses the information in these columns to construct the correlation identifier for the ingested data.
- A column for each security dimension that the security schema defines. During ingestion, i2 Analyze can use the information in these columns to implement per-record security.

For example, if the i2 Analyze schema contains this simplified entity type definition:

```
<EntityType Id="ET5" DisplayName="Person">
    <PropertyTypes>
        <PropertyType DisplayName="First (Given) Name"
                      LogicalType="SINGLE_LINE_STRING" Id="PER4"/>
        <PropertyType DisplayName="Birth place location"
                      LogicalType="GEOSPATIAL" Id="PER5"/>
        <PropertyType DisplayName="Date of Birth"
                      LogicalType="DATE" Id="PER9"/>
        <PropertyType DisplayName="Date and Time of Death"
                      LogicalType="DATE_AND_TIME" Id="PER10"/>
    </PropertyTypes>
</EntityType>
```

This SQL statement is then the definition of a corresponding staging table in Db2:

```
CREATE TABLE "IS_Staging"."E_Person" (
        "source_id" VARCHAR(50),
        "origin_id_type" VARCHAR(100),
        "origin_id_keys" VARCHAR(1000),
        "source_created" TIMESTAMP,
        "source_last_updated" TIMESTAMP,
        "correlation_id_type" VARCHAR(100),
        "correlation_id_key" VARCHAR(1000),
        "p_first_given_name" VARCHAR(250),
        "p_birth_place_location" VARCHAR(250),
        "p_date_of_birth" DATE,
        "p0_date_and_time_of_deat" TIMESTAMP,
        "p1_date_and_time_of_deat" VARCHAR(250),
        "p2_date_and_time_of_deat" SMALLINT,
        "p3_date_and_time_of_deat" TIMESTAMP,
        "security_level" VARCHAR(50),
        "security_compartment" VARCHAR(50),
        "source_ref_source_type" VARGRAPHIC(200),
        "source_ref_source_location" DBCLOB(2000),
        "source_ref_source_image_url" DBCLOB(2000)
    );
```

And this SQL statement is then the definition of a corresponding staging table in SQL Server:

```
CREATE TABLE IS_Staging.E_Person(
        source_id nvarchar(50) NULL,
        origin_id_type nvarchar(100),
        origin_id_keys nvarchar(1000),
        source_created datetime2(6) NULL,
        source_last_updated datetime2(6) NULL,
        correlation_id_type nvarchar(100) NULL,
        correlation_id_key nvarchar(1000) NULL,
        p_first_given_name varchar(250) NULL,
        p_birth_place_location varchar(max) NULL,
        p0_date_and_time_of_deat datetime2(4) NULL,
        p1_date_and_time_of_deat nvarchar(250) NULL,
        p2_date_and_time_of_deat smallint NULL,
        p3_date_and_time_of_deat datetime2(6) NULL,
        security_level nvarchar(50) NULL,
        security_compartment nvarchar(50) NULL,
        source_ref_source_type nvarchar(200) NULL,
        source_ref_source_location nvarchar(2000) NULL,
        source_ref_source_image_url nvarchar(2000) NULL
    );
```

Note: Additionally, staging tables for link types contain a column for the direction of the link, and further columns for the information that uniquely identifies the link end data in the source.

The statements create the staging table in a separate schema from the Information Store data tables. Many of the columns in the staging table have names that are derived from the display names of the

property types in the i2 Analyze schema. In most cases, the relationship between the schema and the staging table is obvious, but there are a number of extra columns and differences:

- The `source_id`, `origin_id_type`, `origin_id_keys` columns of the staging table can be used to store values that reference the rest of the data in its original source and can be used to make up the origin identifier of the resulting record.

  Note: If the staging table definition was for a link type, it would also contain `from_` and `to_` variations of each of the columns.

  For more information about generating origin identifiers during ingestion, see “Origin identifiers” on page 39.
- The next two columns of the staging table are `source_created` and `source_last_updated`. You can use these columns to store information about when the data to be ingested was created and modified in its source.
- The next two columns of the staging table are `correlation_id_type` and `correlation_id_key`. If you want to correlate data during ingestion into the Information Store, you can use these columns to store values that i2 Analyze uses to generate correlation identifiers. For more information, see Overview of correlation.

  Note: Although populating the correlation identifier columns is not mandatory, doing so acts like a switch. The presence of correlation identifier values in any row of a staging table causes i2 Analyze to perform correlation for all the rows in that table.
- Any property type in the i2 Analyze schema that has the logical type `DATE_AND_TIME` occupies four columns in the staging table. These columns always appear in the same order:
  - The "`P0`" column is for the local date and time as originally recorded, as a `DATE_AND_TIME`.
  - The "`P1`" column is for the time zone of the local date and time, as listed in the IANA database. For example, `Europe/London`.
  - The "`P2`" column is for an indicator of whether Daylight Saving Time is (1) or is not (0) in effect.

    Note: i2 Analyze considers this value only when the time is ambiguous because it occurs during the hour that is "repeated" when Daylight Saving Time ends.
  - The "`P3`" column is for the date and time as expressed in Coordinated Universal Time (UTC), as another `DATE_AND_TIME`.

  For more information about the permitted values for `DATE_AND_TIME` columns in your database management system, see “Information Store property value ranges” on page 31.
- The next columns derive from the security schema rather than the i2 Analyze schema. One column exists for each security dimension that the security schema defines. You can use these columns if you want to give different dimension values to each i2 Analyze record that is created or updated as a result of ingestion.
- In link tables, there is also a `direction` column to store the direction of links.
- The final three columns are named `source_ref_source_type`, `source_ref_source_location`, and `source_ref_source_image_url`. These columns are used to populate the source reference that is generated when the data is ingested.

  For more information about implementing source references in your system, see Configuring source references.

The staging tables contain some, but never all, of the data for i2 Analyze records. They do not contain the type identifiers that Information Store records must have, and it is not mandatory to populate the columns for timestamps, security dimension values, or correlation identifiers. You can supply the remainder of the information in an ingestion mapping.

Information Store property value ranges

The Information Store places limits on the ranges of values that properties can contain. Different logical types in the i2 Analyze schema imply different limits, which are not always the same as the restrictions on the underlying database. It is important to consider the limits when you prepare data for ingestion.

Logical type	Db2 values	SQL Server values
`SINGLE_LINE_STRING`	Up to 8000 bytes of UTF-8 characters. The default value is 250.*	Up to 4000 bytes of UTF-16 characters. The default value is 250.*
`MULTI_LINE_STRING`	Up to 32700 bytes of UTF-8 characters	Up to 2^{31}-1 bytes of UTF-16 characters.
`SELECTED_FROM`	Same as `SINGLE_LINE_STRING`	Same as `SINGLE_LINE_STRING`
`SUGGESTED_FROM`	Same as `SINGLE_LINE_STRING`	Same as `SINGLE_LINE_STRING`
`DATE`	From 1753-01-01 to 9999-12-31	From 1753-01-01 to 9999-12-31
`TIME`	From 00:00:00 to 23:59:59 **	From 00:00:00 to 23:59:59
`DATE_AND_TIME`	From 1753-01-01T00:00:00Z to 9999-12-31T23:59:59Z **	From 0001-01-01 00:00:00.00Z to 9999-12-31 23:59:59.99Z
`BOOLEAN`	`true` or `false`	`true` or `false`
`INTEGER`	From -2^{31} to 2^{31} - 1	From -2^{31} to 2^{31} - 1
`DOUBLE`	From 4.9 x 10^{-324} to 1.79769313486231 x 10^{308} (Equivalent range for negative values. Maximum 15 digits of precision.)	From - 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308 (Maximum 17 digits of precision.)
`DECIMAL`	From -999999999999999999.9999 to 999999999999999999.9999 (Maximum 18 digits before the decimal mark. Maximum 4 digits after it.)	From -999999999999999999.9999 to 999999999999999999.9999 (Maximum 18 digits before the decimal mark. Maximum 4 digits after it.)
`GEOSPATIAL`	From -180.0000 to 180.0000 for longitude values, and -90.0000 to 90.0000 for latitude values. The data must be in the Well-known text representation, `POINT(`*`longitude latitude`*`)`. The longitude and latitude values must conform to the WGS84 Bounds. For more information about the values, see Spatial reference - WGS84.	From -180.0000 to 180.0000 for longitude values, and -90.0000 to 90.0000 for latitude values. The data must be in the Well-known text representation, `POINT(`*`longitude latitude`*`)`. The longitude and latitude values must conform to the WGS84 Bounds. For more information about the values, see Spatial reference - WGS84.

* The value for SINGLE_LINE_STRING depends on the value that is specified for each property with that type in the i2 Analyze schema.

** If a Db2 database underlies the Information Store you can load time values that represent midnight as 24:00:00. When it stores such values, the database converts them to fit the ranges in the table.

In addition to the values in the table, you can set the value of any non-mandatory property to null. In the staging table for an item type that has a DATE_AND_TIME property type, all four columns that the value is spread across must be null in that case.

Ingestion mapping files

An ingestion mapping file is an XML document whose structure is validated during the ingestion process. Every time that you instruct the Information Store to ingest data, you specify both the mapping file to use, and the ingestion mapping within it. You can choose to put all your ingestion mappings in one file, or to spread them across several files.

Ingestion mappings have two complementary purposes. First, they make the association between an entity type or a link type in the i2 Analyze schema and a staging table in the database. Second, they provide any extra information that the Information Store requires but the staging tables do not contain.

For all record types, the extra information that an ingestion mapping can provide includes:

- The type identifier of the entity or link type that the mapping applies to
- The name of the data source that the data to be ingested comes from
- How to create an origin identifier for data of this type
- The security dimension values that all records of this type receive, if you do not use per-record security

Link type ingestion mappings provide further information that addresses the requirements of link records:

- The Information Store must be able to test that it already contains the entity records at the ends of an incoming link. The link type mapping must describe how to create the origin identifiers that are associated with those records so that the Information Store can look them up.
- To make the look-up more efficient, the link type mapping also contains the type identifiers of the entity records that appear at the "from" and "to" ends of the incoming links. A link type that can connect entities of several different types requires a separate mapping for each valid combination of end types.

Ingestion mapping syntax

The root element of an ingestion mapping file is an <ingestionMappings> element from the defined namespace. For example:

```
<ns2:ingestionMappings
    xmlns:ns2="http://www.i2group.com/Schemas/2016-08-12/IngestionMappings"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
</ns2:ingestionMappings>
```

Within the ingestion mapping file, you use an <ingestionMapping> element to define a mapping for a particular entity type or link type. Each <ingestionMapping> element has a mandatory id

attribute that must be unique within the mapping file. You use the value to identify the mapping when you start ingestion. For example:

```
<ingestionMapping id="Person">
    ...
</ingestionMapping>
```

Note: For examples of complete ingestion mapping files, search for files with the name `mapping.xml` in the i2 Analyze deployment toolkit. All of those files contain definitions that are similar to the definitions here.

Entity type ingestion mappings

When the mapping is for an entity type, the `<ingestionMapping>` element has the following children:

`stagingArea`

The `<stagingArea>` element specifies where the mapping gets its staged data from. In this version of i2 Analyze, the staged data is always in a staging table, and `<stagingArea>` always has a `<tableName>` child.

`tableName`

The value of `<tableName>` is the name of the staging table that contains the data to be ingested.

For example:

```
...
<stagingArea xsi:type="ns2:databaseIngestionSource">
    <tableName>IS_Staging.E_Person</tableName>
</stagingArea>
...
```

`itemTypeId`

The value of the `<itemTypeId>` element is the identifier of the entity type (or the link type) to which the mapping applies, as defined in the i2 Analyze schema.

For example:

```
...
<itemTypeId>ET5</itemTypeId>
...
```

`originId`

The `<originId>` element contains a template for creating the origin identifier of each ingested row. `<originId>` has two mandatory child elements: `<type>` and `<keys>`.

For example:

```
...
<originId>
  <type>$(origin_id_type)</type>
  <keys>
    <key>$(origin_id_keys)</key>
  </keys>
</originId>
...
```

Here, `$(origin_id_type)` and `$(origin_id_keys)` are references to the columns named `origin_id_type` and `origin_id_keys` in the staging table to which this ingestion mapping applies. When the Information Store ingests the data, the values from the staging table become the origin identifier in the Information Store.

For more information about generating origin identifiers during ingestion, see "Origin identifiers" on page 39.

dataSourceName

The value of the `<dataSourceName>` element identifies the data source from which the data in the staging table came. It must match the name of an ingestion source that you provide to the Information Store during the ingestion process.

For example:

```
...
<dataSourceName>EXAMPLE</dataSourceName>
...
```

createdSource and lastUpdatedSource

By default, the ingestion process automatically puts the values from the `source_created` and `source_last_updated` columns of the staging tables into the Information Store. If you want to use the same values for all ingested data, you can override that behavior by including the non-mandatory `<createdSource>` and `<lastUpdatedSource>` elements and specifying values in the date-time string format for your database management system.

For example:

```
...
<createdSource>2002-10-04 09:21:33</createdSource>
<lastUpdatedSource>2002-10-05 09:34:45</lastUpdatedSource>
...
```

securityDimensionValues

Every row that the Information Store ingests must have at least one security dimension value from each dimension in the security schema. The Information Store staging tables contain a column for each access security dimension that the security schema defines.

In your ingestion process, you can use the staging table columns to store dimension values on a per-row basis. Alternatively, you can specify that all the data that the Information Store ingests through the same mapping get the same security dimension values.

In the ingestion mapping file, the `<securityDimensionValues>` element has `<securityDimensionValue>` children. For per-row security, use the value of each `<securityDimensionValue>` element to reference a security dimension column.

For example:

```
...
<securityDimensionValues>
  <securityDimensionValue>$(security_level)</securityDimensionValue>
  <securityDimensionValue>$(security_compartment)</securityDimensionValue>
</securityDimensionValues>
...
```

In the staging table, the referenced columns can contain either a single dimension value, or a comma-separated list of dimension values.

For per-mapping security, set the value of each <securityDimensionValue> element to a security dimension value.

For example:

```
...
<securityDimensionValues>
    <securityDimensionValue>HI</securityDimensionValue>
    <securityDimensionValue>UC</securityDimensionValue>
    <securityDimensionValue>OSI</securityDimensionValue>
</securityDimensionValues>
...
```

In either approach, the values that you specify must be present in the i2 Analyze security schema.

Link type ingestion mappings

When the ingestion mapping is for a link type, the <ingestionMapping> element has the same children that entity types require, plus the following ones:

fromItemTypeId

The value of the <fromItemTypeId> element is the type identifier of entity records that the schema permits at the "from" end of the link type to which this mapping applies.

For example:

```
...
<fromEntityTypeId>ET5</fromEntityTypeId>
...
```

fromOriginId

The <fromOriginId> element contains a template for creating the origin identifier of the entity record at the "from" end of each ingested link row. Its syntax is identical to the <originId> element.

The origin identifiers that result from <fromOriginId> must match the origin identifiers that result from the <originId> element for the entity type in question. The ingestion process uses this information to verify that the Information Store already ingested an entity record that has this origin identifier.

For example:

```
...
<fromOriginId>
  <type>$(from_origin_id_type)</type>
  <keys>
    <key>$(from_origin_id_keys)</key>
  </keys>
</fromOriginId>
...
```

For more information about generating origin identifiers during ingestion, see “Origin identifiers” on page 39.

toItemTypeId

The value of the <toItemTypeId> element is the type identifier of entity records that the schema permits at the "to" end of the link type to which this mapping applies.

For example:

```
...
<toEntityTypeId>ET10</toEntityTypeId>
...
```

toOriginId

The <toOriginId> element behaves identically to the <fromOriginId> element, except that it applies to the entity record at the "to" end of each ingested link row.

For example:

```
...
<toOriginId>
  <type>$(to_origin_id_type)</type>
  <keys>
    <key>$(to_origin_id_keys)</key>
  </keys>
</toOriginId>
...
```

For more information about generating origin identifiers during ingestion, see “Origin identifiers” on page 39.

linkDirection

The <linkDirection> element is a non-mandatory child of the <ingestionMapping> element. When you include a <linkDirection> element in an ingestion mapping, you can either provide the same value for all links, or refer to the direction column of the staging table. Legal values for the element or the column are WITH, AGAINST, BOTH, and NONE.

For example, to use a fixed value:

```
...
<linkDirection>WITH</linkDirection>
...
```

Or, to use the value in the `direction` column:

```
...
<linkDirection>$(direction)</linkDirection>
...
```

If an ingestion mapping for a link type does not contain a `<linkDirection>` element, then any links that the Information Store ingests through the mapping have no direction.

References and system properties

In an ingestion mapping, you can use constants or references to specify values for i2 Analyze records. When you use a reference, the ingestion process retrieves a value from a staging table column or a property in a settings file. The settings file can also set system properties that control some aspects of ingestion into the Information Store.

By default, the settings file does not exist. You must create the settings file and specify the file when you run the ingestion command.

For example, you might call your settings file `ingestion_settings.properties` and it might contain the following name-value pairs :

```
SEC_LEVEL_VALUE=UC
SEC_COMPARTMENT_VALUE=HI,OSI
IngestionFailureMode=MAPPING
```

When you run the ingestion command, you reference your settings file as follows:

```
setup -t ingestInformationStoreRecords
      ...
      -p importConfigFile=ingestion_settings.properties
```


- "System properties" on page 37
- "References" on page 38

System properties

As well as providing values for ingestion mappings, you can use the settings file to configure the behavior of the ingestion process. The file supports a handful of system properties that you can set in the same way as you create and set custom properties.

`IngestionFailureMode [ RECORD | MAPPING ]`

When the Information Store encounters a problem with a record during ingestion, its default behavior is to log the error and move on to the next record. Failure is *record-based*. Instead, you can specify that a problem with one record causes the Information Store not to ingest any of the records from that staging table. Failure then is *mapping-based*.

In the settings file, the possible values for the `IngestionFailureMode` setting are `RECORD` or `MAPPING`. The default value is `RECORD`.

For example, to change the failure mode to mapping, add the following line to your settings file:

```
IngestionFailureMode=MAPPING
```

RecordFailureThreshold

During the ingestion process, if the number of errors that occur is greater than the value for the `RecordFailureThreshold` property the process stops and no data is ingested into the Information Store. By default, the value for this property is 1000.

For example:

```
RecordFailureThreshold=500
```

IngestionRunstats [TRUE | FALSE]

If you are using Db2, during the ingestion process the **RUNSTATS** command updates statistics in the system catalog about the characteristics of a table, associated indexes, or statistical views.

In a table that contains many records, it can take the **RUNSTATS** command a long time to complete. If the number of records that you are ingesting is a small percentage of the total number of records in the table, it is recommended that you configure the ingestion process not to call the RUNSTATS command. You can then manually run the **RUNSTATS** command after a significant number of records are ingested.

To prevent the ingestion process from calling the RUNSTATS command, set the value of `IngestionRunstats` to FALSE. For example:

```
IngestionRunstats=FALSE
```

The following command is the **RUNSTATS** command to run manually after an ingestion. The command generates query output, and saves the output to the file named `ISStatisticsCollection.sql` that you can run.

```
db2 connect to ISTORE user <user name> using <password>

db2 -x
 "SELECT
      'RUNSTATS ON TABLE ' || TRIM(TABSCHEMA) ||'.'|| TRIM(TABNAME) || '
      WITH DISTRIBUTION ON ALL COLUMNS
      AND INDEXES ALL
      ALLOW WRITE ACCESS
      TABLESAMPLE BERNOULLI(10)
      INDEXSAMPLE BERNOULLI(10);'
 FROM SYSCAT.TABLES
 WHERE TYPE = 'T' AND TABSCHEMA = 'IS_DATA'"
> ISStatisticsCollection.SQL
```

References

Many of the pieces of information that you provide in an ingestion mapping are fixed for that mapping. Item types, end types, and some parts of the origin identifier do not change between the i2 Analyze records that one mapping is responsible for. The most appropriate way to specify this kind of information is to use constant values on a per-mapping basis.

The two main reasons for preferring references to constant values lie at opposite ends of the spectrum:

- To give different values for the same field in records that are ingested through the same mapping, you can refer to a staging table column. This approach is appropriate for many non-property values that change from one record to the next.
- To use the same values across multiple ingestion mappings, refer to a property in a settings file. This approach might be appropriate when you want all the data from a source to get the same security dimension values. You can refer to the same property from every mapping that you write.

  A settings file that defines properties for the ingestion process is just a text file that contains a set of *name=value* pairs, with one pair on each line:

  ```
  SEC_LEVEL_VALUE=UC
  SEC_COMPARTMENT_VALUE=HI,OSI
  ```

  When you run one of the ingestion commands, you can supply it with the name of the properties file whose values you want to use.

To use a value by reference in an ingestion mapping, you use the `$(`*name*`)` syntax. *name* is the name of either a column in the staging table or a property in a settings file. For example, `$(SOURCE_ID)` and `$(DIRECTION)` refer to staging table columns, while in the previous example `$(SEC_LEVEL_VALUE)` and `$(SEC_COMPARTMENT_VALUE)` refer to properties.

Note: Since referring to columns and properties uses the same syntax, a clash can happen if a column and a property have the same name. In that case, the value of the property takes precedence.

Origin identifiers

The role of a source identifier is to reference the data for a record reproducibly in its original source. The source identifiers that records receive during ingestion are unique within i2 Analyze, and they have a special name in this context. They are called origin identifiers.

The nature of a origin identifier depends on the source and the creation method, and sometimes on whether the record is a link or an entity. When you ingest data into the Information Store, i2 Analyze compares the incoming origin identifier with existing records. If it finds a match, i2 Analyze updates a record instead of creating one.

After you develop your process for creating origin identifiers, you must continue to use that process. If you change the way that your origin identifiers are created and ingest the same data again, the Information Store creates new records for the data instead of updating the existing records. To ensure that changes to data are processed as updates, you must create your origin identifiers consistently.

For more information about different identifiers in i2 Analyze, see Identifiers in i2 Analyze records.

The structure of an origin identifier

During the ingestion process, you specify the data for your identifiers in the staging table and ingestion mapping file. An origin identifier is constructed of a "type" and "keys".

`type`

The "type" of an origin identifier allows the services in an i2 Analyze deployment to determine quickly whether they are interested in (or how to process) a particular row of data. The value of the type element does not have to be meaningful, but data from different sources generally have different values.

The length of the origin identifier type must not exceed 100 bytes. This is equivalent to 100 ASCII characters.

keys

The "keys" of an origin identifier contain the information necessary to reference the data in its original source. The pieces of information that you use to make up the keys differs depending on the source of the data. For data that originates in relational sources, you might use keys whose values include the source name, the table name, and the unique identifier of the data within that table.

The length of the origin identifier keys must not exceed 1000 bytes. This is equivalent to 1000 ASCII characters.

It is recommended that your origin identifiers are as short as possible in length, and that any common values are at the end of the key.

Creating origin identifiers for your data

There are two mechanisms for specifying the data for your origin identifiers. You can populate the staging table with all the information required to create the origin identifiers, or you can use a combination of information in the staging table and the ingestion mapping.

When you can provide all the information in the staging tables, there is less processing of the data during ingestion, which can improve ingestion performance.

All information in the staging table

If you can populate the staging table with all the information for your origin identifiers, you can use the `origin_id_type` and `origin_id_keys` columns to store this information. Populate the `origin_id_type` column with the type of your origin identifier. Populate the `origin_id_keys` column with a unique value that is already a composite of key values including the unique identifier from the source. When you use these columns, you must specify them in the ingestion mapping file that you use.

To ingest links, you must specify the origin identifiers at the end of the link. You specify the "to" end of the link in the `to_origin_id_type` and `to_origin_id_keys` columns, and the "from" end in `from_origin_id_type` and `from_origin_id_keys`.

When you specify all the information in the staging table, the origin identifier section of your ingestion mapping is more simple. For example:

```
...
<originId>
  <type>$(origin_id_type)</type>
  <keys>
    <key>$(origin_id_keys)</key>
  </keys>
</originId>
...
```

To specify the origin identifiers at the link ends:

```
...
<fromOriginId>
  <type>$(from_origin_id_type)</type>
  <keys>
    <key>$(from_origin_id_keys)</key>
  </keys>
</fromOriginId>

<toOriginId>
  <type>$(to_origin_id_type)</type>
  <keys>
    <key>$(to_origin_id_keys)</key>
  </keys>
</toOriginId>
...
```

Combination of information in the staging table and the ingestion mapping

If you cannot populate the staging table with all the information for your origin identifiers, you can use the `source_id` column in the staging table to contain the unique identifier from the source and provide any other keys and the origin identifier type in the ingestion mapping.

To ingest links, you must specify the origin identifiers at the end of the link. You specify the unique identifier of the "to" end of the link in the `to_source_id` column, and the "from" end in `from_source_id`. In the ingestion mapping, you specify the other keys that make up the origin identifiers of the link ends.

When you provide the information in both the staging table and the ingestion mapping, the mapping file is more complex. For example:

```
...
<originId>
  <type>OI.EXAMPLE</type>
  <keys>
    <key>$(source_id)</key>
    <key>PERSON</key>
  </keys>
</originId>
...
```

To specify the origin identifiers at the link ends, if the to end is an "Account" entity type:

```
...
<fromOriginId>
  <type>OI.EXAMPLE</type>
  <keys>
    <key>$(from_source_id)</key>
    <key>PERSON</key>
  </keys>
</fromOriginId>

<toOriginId>
  <type>OI.EXAMPLE</type>
  <keys>
    <key>$(to_source_id)</key>
    <key>ACCOUNT</key>
  </keys>
</toOriginId>
...
```

The `ingestInformationStoreRecords` toolkit task

The deployment toolkit command for ingesting records looks like this:

```
setup -t ingestInformationStoreRecords
      -p importMappingsFile=ingestion_mapping_file
      -p importMappingId=ingestion_mapping_id
      -p importLabel=ingestion_label
      -p importConfigFile=ingestion_settings_file
      -p importMode=STANDARD|VALIDATE|BULK
```

While the ETL toolkit command looks like this:

```
ingestInformationStoreRecords
      -imf ingestion_mapping_file
      -imid ingestion_mapping_id
      -il ingestion_label
      -icf ingestion_settings_file
      -im STANDARD|VALIDATE|BULK
```

Here, *ingestion_mapping_file* is the path to the XML file that contains the mapping that you want to use, and *ingestion_mapping_id* is the identifier of the mapping within that file. The latter is mandatory unless the file contains only one mapping.

The `importLabel`, `importConfigFile`, and `importMode` parameters are optional:

- When you specify `importLabel`, *ingestion_label* is a name that identifies a particular use of the ingestion command in the Information Store's `IS_Public.Ingestion_Deletion_Reports` view.
- When you specify `importConfigFile`, *ingestion_settings_file* is the path to a settings file that contains *name=value* pairs. You can refer to names in the settings file from references in the ingestion mapping file to use their values when you run the `ingestInformationStoreRecords` command.

- When you specify `importMode` you can set it to `STANDARD`, `VALIDATE`, or `BULK`.
  - `STANDARD` mode can be used to ingest new and updated records, with or without correlation. If you do not specify `importMode`, `STANDARD` mode is used.
  - `VALIDATE` mode checks the validity of the specified mapping, but no ingestion takes place.
  - `BULK` mode can be used to ingest new records without correlation.

Understanding ingestion reports

Every attempt to add, update, or delete data in the Information Store through the deployment or ETL toolkit adds rows to the `IS_Public.Ingestion_Deletion_Reports` view. You can use the contents of this view to track the history of all such operations, and to examine the impact of a particular operation.

About this task

Each time that you run a command that might change the contents of the Information Store, you create a job in the database. Each job acts on one or more batches of i2 Analyze records. There is always one batch per item type that the command affects, but there can also be several batches for the same type if the number of affected records is large.

For example, consider a command that processes updates for deleted Person entity data. The first batch in the resulting job is for Person records, and there might be more such batches if there are many records to be deleted. If the Person data has links, then the job has further batches for each type of link that might get deleted as a result of the entity deletion.

The `IS_Public.Ingestion_Deletion_Reports` view contains information about every batch from every toolkit operation to create or update data in the Information Store.

Note: Deletion-by-rule operations also result in job and batch creation, and view population, according to the same rules. For more information, see the *Deletion Guide*.

The first few columns in the view have the same value for all batches within a job:

Column name	Description
`label`	The value that you passed in the `importLabel` parameter of a toolkit command, or the value that a deletion-by-rule operation generates, or null.
`job_id`	The server-assigned identifier for this ingestion or deletion job. This identifier is also a cross-reference to the `Deletion_By_Rule_Log` view if the job originated from a deletion-by-rule operation.
`ingestion_mode`	The value that you passed in the `importMode` parameter, or **Delete** for all deletion-by-rule operations.
`validation_mode`	A description of how the job was configured to react to errors during the operation.
`error_threshold`	The threshold that applies to some of the validation modes.
`primary_item_type`	The i2 Analyze schema ID of the item type that was specified at job creation.
`primary_record_count`	The number of records of the primary item type that were affected by the job. (Deleting entity data can affect link records too.)
`start_time`	The start time of the job as a whole.

Column name	Description
end_time	The end time of the job as a whole.

The remaining columns can have different values for different batches of records:

Column name	Description
batch_item_type	The i2 Analyze schema ID of the item type that was acted on in this batch. For at least one batch, the batch_item_type is the same as the primary_item_type.
batch_start_time	The start time of this batch, which is always later than the start time of the job.
batch_end_time	The end time of this batch, which is always earlier than the end time of the job.
insert_count	The number of rows of data from this batch that were inserted to the Information Store, resulting in new i2 Analyze records.
update_count	The number of rows of data from this batch that updated existing records in the Information Store.
merge_count	The number of *merge* operations that occurred in the Information Store from this batch.
unmerge_count	The number of *unmerge* operations that occurred in the Information Store from this batch.
delete_count	The number of pieces of provenance that were deleted from the Information Store as a result of this batch.
delete_record_count	The number of records that were deleted from the Information Store as a result of this batch.
reject_count	The number of rows that were rejected from this batch during processing because they are invalid.
status	An indicator of the result of this batch, from success (all rows processed correctly) through partial success to failure (no rows processed).
reject_view	The full name of the view that contains details of any rejected rows.
stack_trace	If i2 Analyze generated a stack trace as a result of errors during ingestion or deletion, this column contains it.

Example

Ingest example

The (abbreviated) report for successful ingestion operations might look like this:

job_id	1	2
ingestion_mode	Standard	Standard
primary_item_type	ET10	ET4
primary_record_count	62	8

batch_item_type	ET10	ET4
batch_start_time	2017-11-30 15:27:06.76	2017-11-30 15:27:09.45
batch_end_time	2017-11-30 15:27:09.87	2017-11-30 15:27:09.63
insert_count	57	7
update_count	0	0
merge_count	5	1
unmerge_count	0	6
delete_count	0	0
delete_record_count	0	0
reject_count	0	0
status	Succeeded	Succeeded

In this example, several commands to ingest entity records resulted in the creation of several jobs. Each job demonstrates different behavior that is possible during ingestion, including correlation operations:

JOB_ID 1

This job demonstrates what the ingestion report can look like when data in the staging table causes merge operations. In this example, five merge operations are completed on the incoming rows of data, as shown in the `merge_count` column. This results in 57 i2 Analyze records created from the 62 rows of data, as shown in the `insert_count` and `primary_record_count` columns. This includes merging five rows of data with existing i2 Analyze records in the Information Store.

JOB_ID 2

This job demonstrates what the ingestion report can look like when the data in the staging table causes unmerge and merge operations. In this example, six unmerge operations are completed on the incoming rows of data, as shown in the `unmerge_count` column. One merge operation is completed on the incoming rows, as shown in the `merge_count` column. This results in 7 i2 Analyze records created, from eight rows of data as shown in the `insert_count` and `primary_record_count` columns. The `primary_record_count` value does not include the `unmerge_count`.

Delete example

The (abbreviated) report for a successful delete operation might look like this:

job_id	26	26	26	26	26
ingestion_mode	Delete	Delete	Delete	Delete	Delete
primary_item_type	ET5	ET5	ET5	ET5	ET5
primary_record_count	324	324	324	324	324
batch_item_type	ET5	LAC1	LAS1	LEM1	LIN1
batch_start_time	2017-11-30	2017-11-30	2017-11-30	2017-11-30	2017-11-30

	15:27:06.76	15:27:08.60	15:27:08.60	15:27:09.43	15:27:09.45
batch_end_time	2017-11-30 15:27:09.87	2017-11-30 15:27:09.30	2017-11-30 15:27:09.29	2017-11-30 15:27:09.62	2017-11-30 15:27:09.63
insert_count	0	0	0	0	0
update_count	0	0	0	0	0
merge_count	0	0	0	0	0
unmerge_count	0	0	0	0	0
delete_count	324	187	27	54	33
delete_record_count	320	187	27	54	33
reject_count	0	0	0	0	0
status	Succeeded	Succeeded	Succeeded	Succeeded	Succeeded

In this example, a command to update the Information Store for deleted entity data (with item type ET5) resulted in the creation of a job with five batches. The first few columns of the `Ingestion_Deletion_Reports` view contain the same values for all batches in the same job. Later columns reveal how deleting entity records results in the deletion of connected link records (with item types LAC1, LAS1, LEM1, LIN1).

In one case, the `delete_record_count` value is less than the `delete_count` value. This is because some of the provenance to be deleted was associated with an i2 Analyze record that had more than one piece of provenance. An i2 Analyze record is deleted only when the last associated provenance is deleted.

Troubleshooting the ingestion process

The commands that you run during the ingestion process send information about their progress to the command line and a log file. If any command encounters errors or does not run to completion, you can read the output to help you to diagnose the problem.

When an ingestion process runs to completion, the final output from the command is a report of what happened to the Information Store. The reports appear on the command line and in the ingestion log at `toolkit\configuration\logs\importer\IBM_i2_Importer.log`. The three possible end states are success, partial success, and failure.

Success

If the ingestion command processed all of the rows in the staging table without error, then the Information Store reflects the contents of the staging table. The command reports success like this example:

```
> INFO  [IImportLogger] - Total number of rows processed: 54
> INFO  [IImportLogger] - Number of records inserted: 0
> INFO  [IImportLogger] - Number of records updated: 54
> INFO  [IImportLogger] - Number of merges: 0
> INFO  [IImportLogger] - Number of unmerges: 0
> INFO  [IImportLogger] - Number of rows rejected: 0
> INFO  [IImportLogger] - Duration: 5 s
> INFO  [IImportLogger] -
> INFO  [IImportLogger] - Result: SUCCESS
```

Partial success

If you ran the command in record-based failure mode, and it processed some of the rows in the staging table without error, then it reports partial success like this example:

```
> INFO  [IImportLogger] - Total number of rows processed: 34
> INFO  [IImportLogger] - Number of records inserted: 0
> INFO  [IImportLogger] - Number of records updated: 30
> INFO  [IImportLogger] - Number of merges: 0
> INFO  [IImportLogger] - Number of unmerges: 0
> INFO  [IImportLogger] - Number of rows rejected: 4
> INFO  [IImportLogger] - Duration: 4 s
> INFO  [IImportLogger] -
> INFO  [IImportLogger] - Result: PARTIAL SUCCESS
> INFO  [IImportLogger] -
> INFO  [IImportLogger] - Total number of errors: 4
> INFO  [IImportLogger] - Error categories:
> INFO  [IImportLogger] - ABSENT_VALUE: 4
> INFO  [IImportLogger] -
> INFO  [IImportLogger] - The rejected records and errors are recorded in
the database. For details, use the following view:
> INFO  [IImportLogger] - IS_Staging.S20171204122426717092ET5_Rejects_V
```

The records in the Information Store reflect the rows from the staging table that the command successfully processed. The report includes the name of a database view that you can examine to discover what went wrong with each failed row.

Failure

If you ran the command in mapping-based failure mode, then any error you see is the first one that it encountered, and the report is of failure:

```
> INFO  [IImportLogger] - Total number of rows processed: 1
> INFO  [IImportLogger] - Number of records inserted: 0
> INFO  [IImportLogger] - Number of records updated: 0
> INFO  [IImportLogger] - Number of merges: 0
> INFO  [IImportLogger] - Number of unmerges: 0
> INFO  [IImportLogger] - Number of rows rejected: 0
> INFO  [IImportLogger] - Duration: 0 s
> INFO  [IImportLogger] -
> INFO  [IImportLogger] - Result: FAILURE
```

When the process fails in this fashion, the next lines of output describe the error in more detail. In this event, the command does not change the contents of the Information Store.

Note: If a serious error occurs, it is possible for the ingestion command not to run to completion. When that happens, it is harder to be certain of the state of the Information Store. The ingestion process uses batching, and the records in the store reflect the most recently completed batch. If you are using the bulk import mode with Db2, see "Db2 bulk import mode error" on page 50 for more information about recovering from errors at this time.

If the command reports partial success, you might be able to clean up the staging table by removing the rows that were ingested and fixing the rows that failed. However, the main benefit of record-based failure is that you can find out about multiple problems at the same time.

The most consistent approach to addressing failures of all types is to fix up the problems in the staging table and run the ingestion command again. The following sections describe how to react to some of the more common failures.

Link rows in the staging table refer to missing entity records

When the Information Store ingests link data, you might see the following error message in the console output:

```
Link data in the staging table refers to missing entity records
```

This message is displayed if the entity record at either end of a link is not present in the Information Store. To resolve the error:

- Examine the console output for your earlier operations to check that the Information Store ingested all the entity records properly.
- Ensure that the link end origin identifiers are constructed correctly, and exist for each row in the staging table.
- Ensure that the link type and the entity types at the end of the links are valid according to the i2 Analyze schema.

Then, rerun the ingestion command.

Rows in the staging table have duplicate origin identifiers

During any ingestion procedure, but especially when a staging table is large, you might see the following error message in the console output:

```
Rows in the staging table have duplicate origin identifiers
```

This message is displayed when several rows in a staging table generate the same origin identifier. For example, more than one row might have the same value in the `source_id` column.

If more than one row in the staging table contains the same provenance information, you must resolve the issue and repopulate the staging table. Alternatively, you can separate the rows so that they are not in the same staging table at the same time.

This problem is most likely to occur during an update to the Information Store that attempts to change the same record (with the same provenance) twice in the same batch. It might be appropriate to combine the changes, or to process only the last change. After you resolve the problem, repopulate the staging table and rerun the ingestion command.

Geospatial data is in the incorrect format

During an ingestion procedure that contains geospatial data, you might see the following error messages in the console output:

On Db2:

```
SQLERRMC=GSEGEOMFROMWKT;;GSE3052N  Unknown type "FOO(33.3" in WKT.
```

On SQL Server:

```
System.FormatException: 24114: The label FOO(33.3 44.0) in the input well-known text (WKT) is
not valid.
```

This message is displayed when data in a geospatial property column is not in the correct format.

Data in geospatial property columns must be in the POINT(*longitude latitude*) format. For more information, see "Information Store property value ranges" on page 31.

Error occurred during a correlation operation

During an ingestion procedure with correlated data, you might see the following error message in the console output:

```
An error occurred during a correlation operation. There might be some data in an unusable
state.
```

This message is displayed if the connection to the database or Solr is interrupted during a correlation operation.

To resolve the problem, you must repair the connection that caused the error, and then run the `syncInformationStoreCorrelation` toolkit task. This task synchronizes the data in the Information Store with the data in the Solr index so that the data returns to a usable state.

After you run the `syncInformationStoreCorrelation` task, reingest the data that you were ingesting when the failure occurred. Any attempt to run an ingestion or a deletion command before you run `syncInformationStoreCorrelation` will fail.

Ingestion with correlated data is still in progress

During an ingestion procedure, you might see the following error message in the console output:

```
You cannot ingest data because an ingestion with correlated data is still in progress,
or because an error occurred during a correlation operation in a previous ingestion.
```

If another ingestion is still in progress, you must wait until it finishes. If a previous ingestion failed during a correlation operation, you must run the `syncInformationStoreCorrelation` toolkit task.

For more information about running the `syncInformationStoreCorrelation` toolkit task, see "Error occurred during a correlation operation" on page 49.

Ingestion of the same item type is still in progress

During an ingestion procedure, you might see the following error message in the console output:

```
You cannot ingest data for item type <ET5> because an ingestion is still in progress.
You must wait until the process is finished before you can start another ingestion for this
item type.
```

If another ingestion of the same item type is still in progress, you must wait until it finishes.

Db2 bulk import mode error

The symptoms of this type of failure are a stack trace and failure message in the console and importer log. To recover from a failure at this time:

1. Identify the cause of the failure. You must use the SQL error codes to determine the cause of the failure.

   You might see error messages from Db2 about the following issues:

   - Transaction log size or connectivity issues.
   - Invalid data in the staging table.
2. Fix the problem that caused the failure. This might include ensuring connectivity to the database or increasing the transaction log size.
3. After you resolve the problem that caused the error, you can attempt the ingestion again. If any of the rows in the staging table were already ingested into the Information Store, you must remove them from the staging table before you can ingest in bulk mode.
   - In the console or importer log, if the value for `Number of rows accepted` is 0 then run the ingestion command again.
   - In the console or importer log, if the value for `Number of rows accepted` is greater than 0, you must ensure that these records are not ingested again.
     - If you are using only the `origin_id_keys` and `origin_id_type` columns to contain data for your origin identifiers, the Information Store database contains a stored procedure that you can use to remove the ingested rows from the specified staging table:

       ```
       CALL IS_Public.Delete_Imported_Rows('<staging schema and staging table
       name>', '<item type id>');
       ```

       For example:

       ```
       CALL IS_Public.Delete_Imported_Rows('IS_Staging.E_Person', 'ET5');
       ```

       Before you run the stored procedure, make a copy of the staging table to prevent any data loss.

       The stored procedure compares the records in the Information Store for the specified item type against the rows in the specified staging table. Any rows that were already ingested are removed from the staging table. After you remove the rows from the staging table, you can run the ingestion command again.
     - If you are using the `source_id` columns, or your origin identifiers contain data from multiple columns and the ingestion mapping, you can manually remove the ingested rows from the staging table and run the ingestion command again, or use the standard import mode to ingest data from the staging table.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk
NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Limited
Hursley House
Hursley Park
Winchester
Hants SO21 2JN
U.K.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, i2, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.

Product Number: 5725-G22