Structure Data Crawler Documentation

Introduction

The structured Data crawler allows structured data as found in databases or web services to be crawled and assembled into yaml documents inside SimSage which then in turn can be searched and processed as any other document style asset within SimSage.
The structured Data crawler is configured via a yaml based configuration file. Code completion and validation is provided by a Json schema definition, when adding the configuration into the editor inside the Structure Data Crawler.

The schema file can also be downloaded from: *SIMSAGE_URL/api/crawler/sdc_schema*

Configuration Overview

The configuration has two core elements:

1. A list of data provider configurations (dataProviders)
   This list contains the connection details for a data provider to connect to it’s data source
   See Data Providers below for configuration details
2. A structure definition defining how individual records (and child records) are to be assembled
   To allow for hierarchical data structures, the definition presents a nested view of record definitions, starting with the root definition.
   Each record has the following properties available:

- **Provider** - The name of the provider configured in the provider list to use to fetch the data
- **Primary Key Template** - A template string defining how the primary key for the uploaded asset in SimSage is to be constructed (see below for template strings)
- **Title Template** - A template string defining how the title of the uploaded asset in SimSage is to be constructed (see below for template strings)
- **Fields** - The list of singular data items fields for the constructed asset.
  Each field is represented as an object, keyed as the field name and with the following sub properties:
  - **Data Type**
    The data type for the field (String, Int, Date, Decimal)
  - **format (Optional)**
    Formatting options depending on the data type such as
    format: "dd/MM/yyyy"
    to get a date formatted in UK format
  - **Enum (Optional)**
    If the value is a key to an *Enum* value, a mapping between the value and a human readable text can be added here, e.g.
    enum:

1: Pending
2: Processing
3: Rejected
4: Completed

- **Meta** - List of mappinga between a Simsage Meta data item for the asset to be created and a string template for the value, e.g. customer-id: ${customerId} to create a meta data item for the record's customerId field
- **Record Action** – optional for root record (has to be DOCUMENT)
  This field defines for child collections how they are stored inside SimSage.
  Available options are:
  - **DOCUMENT**
    The child records become assets in their own right and are only linked via SimSage's attachment mechanism to the parent record
  - **CHILD_COLLECTION**
    The child records become an array of items inside the parent record themselves
  - **NONE**
    The child record is ignored. Useful if a level of the data structure is purely technical and should not be uploaded
- **Collections** - a list of nested record definitions for one to many relations of the current record
- **Config** - provider specific details how to fetch the data and map it to the record. See Data Providers below for specific details

Template Strings

Certain fields such as `primaryKeyTemplate`, `titleTemplate` and some of the provider configuration fields
will need to contain values from the actual fetched record. This can be achieved by adding the name of the field inside the value inside substitution brackets (${}), e.g. "`primaryKeyTemplate: Customer-${customerId}`" In nested structures the field name can be proceeded by one or more "`../`" steps to climb up the hierarchy to fields of a parent record, e.g. `${../customerId}` to use the customer Id field of the parent record.

Data Providers

JDBC Provider

Connection properties:

- **Name** - The name of the provider
- **Type** - 'jdbc'
- **Connection String** - jdbc connection string for the database
- **Username** - The username for the DB login
- **Password** - The password for the DB login

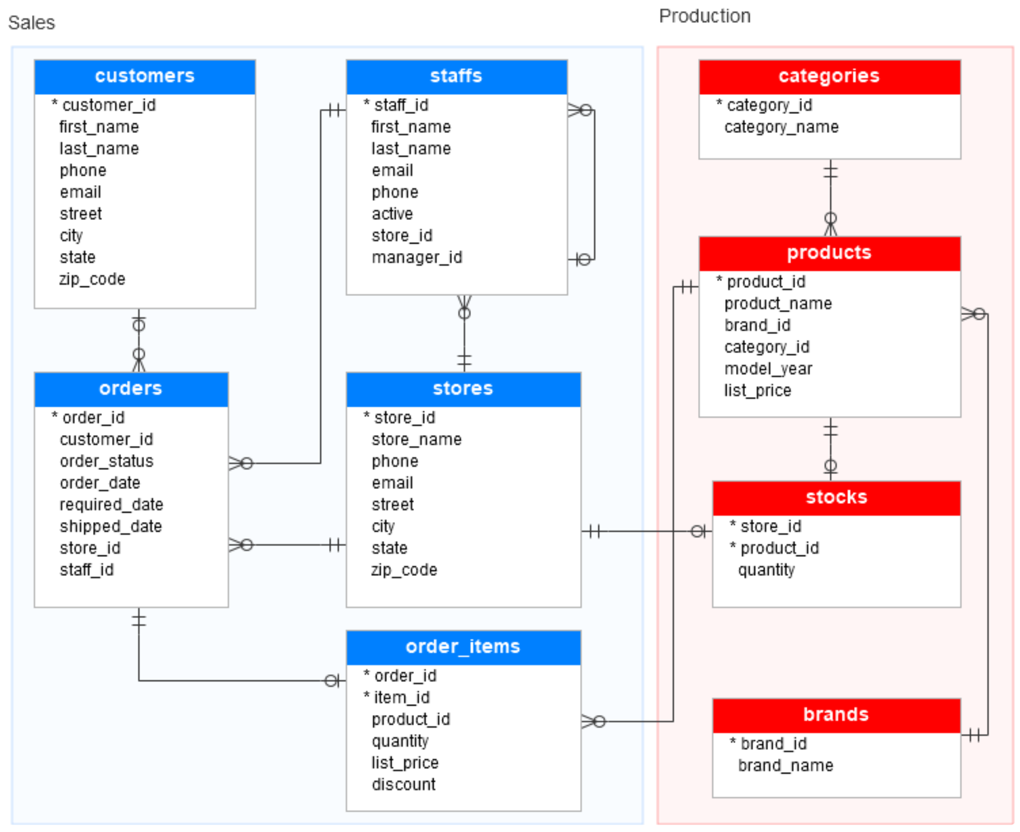
Record configuration:

- columns - List off mappings from record field name to column name as per from clause in sql query below
- query - from clause of the sql query to fetch the records. Template syntax can be used.

Example Configuration

The below is an example configuration to crawl a fictional store database as described in:

SQL Server Sample Database



Configuration Yml

dataProviders:

- **name** - OrderDB
  type - jdbc
  Connection String - `jdbc:sqlserver://mysqlserverhost:1433;database=orders`
  userName: username
  password: password
- root:
  provider: OrderDB
  primaryKeyTemplate: Customer-${customerId}
  titleTemplate: ${firstName} ${lastName} (${customerId})
  fields:
  customerId:
  dataType: IntfirstName:
  dataType: String
  lastName:
  dataType: String
  email:
  dataType: String
  street:
  dataType: String

city:
dataType: String
state:
dataType: String
zip:
dataType: String

- config:
  - columns:
    customerId: customer_id
    firstName: first_name
    lastName: last_name
    email: email
    street: street
    city: city
    state: state
    zip: zip_code
    query: from sales.customers
- meta:
  customer-id: ${customerId}
  fullname: ${firstName} ${lastName}
- collections:
  - orders:
    provider: OrderDB
    primaryKeyTemplate: Order-${../customerId}:${orderId}
    titleTemplate: Order ${orderId} for ${../firstName} ${../lastName} (${../customerId})
    recordAction: DOCUMENT
    - fields:
      - orderId:
        dataType: Int
      - orderStatus:
        dataType: Int
      - enum:
        1: Pending
        2: Processing
        3: Rejected
        4: Completed
      - orderDate:
        dataType: Date
        format: "dd/MM/yyyy"
      - shippedDate:
        dataType: Date
        format: "dd/MM/yyyy"
    - config:
      - columns:
        orderId: order_id
        orderStatus: order_status
        orderDate: order_date
        shippedDate: shipped_date
        query: from sales.orders where customer_id=${../customerId}

- meta:
  order-id: ${orderId}
- collections:
  - items:
    provider: OrderDB
    recordAction: CHILD_COLLECTION
  - fields:
    - productId:
      dataType: Int
    - brand:
      dataType: String
    - productName:
      dataType: String
    - quantity:
      dataType: Int
    - listPrice:
      dataType: Decimal
    - discount:
      dataType: Decimal
      format: Percent
  - config:
    - columns:
      productId: items.product_id
      productName: products.product_name
      quantity: items.quantity
      listPrice: items.list_price
      discount: items.discount
      brand: brands.brand_name
    - query: |
      from sales.order_items items
      join production.products as products on products.product_id = items.product_id
      join production.brands as brands on products.brand_id = brands.brand_id
      where order_id=${../orderId}