Smarten
Augmented Analytics

Self-Serve Data Preparation (SSDP) Best Practices

Document Version: 1.0
Product Version: 5.2

Document Information	
Document ID	Smarten-SSDP-Best-Practices
Document Version	1.0
Product Version	5.2
Date	09-February-2022
Recipient	NA
Author	EMTPL

Statement of Confidentiality, Disclaimer and Copyright

This document contains information that is proprietary and confidential to EMTPL, which shall not be disclosed, transmitted, or duplicated, used in whole or in part for any purpose other than its intended purpose. Any use or disclosure in whole or in part of this information without the express written permission of EMTPL is prohibited.

Any other company and product names mentioned are used for identification purpose only, may be trademarks of their respective owners and are duly acknowledged.

Disclaimer

This document is intended to support administrators, technology managers or developers using and implementing Smarten. The business needs of each organization will vary and this document is expected to provide guidelines and not rules for making any decisions related to Smarten. The overall performance of Smarten depends on many factors, including but not limited to hardware configuration and network throughput.

Contents

1 Loading Data from Data Source

Loading data from the data sources to datasets.

1.1 Select only required Columns

- Select only the required number of columns. This will help reduce load time and also require less storage and data space.
    - For example, if for reporting purposes we are only interested in students' merits data, then we may skip personal details, such as their address and contact number.

1.2 Filter Data from Data Source

1. **SQL Data Source**:
    - Using SQL Query—Filter out unnecessary data from the source query itself. Again, this will help in reducing storage space, increasing performance, and optimizing the data loading process.
        - For example, if we are to create front-end objects for monthly comparison up to a maximum of 12 months, then we do not have to pull 5 years of data from the source.
    - Using the step-by-step wizard for SQL data sources—In the case of extracting data from a database source by the step-by-step wizard, filtering the data from the selection screen itself should be done if and when possible. In a query, for example, we can add a "where" clause in extracting source data by adding filters; in the same way, when using the step-by-step wizard for data extraction in SSDP, you should filter data from the selection screen itself.
        - For example, if we have data for all employees from all cities in our source but our front-end objects are mainly focused on the north-eastern states, then we may add a filter for the zones/states in the selection screen of the step-by-step wizard.
2. **Other Data Sources:**
    - When loading data from sources other than any database type, unnecessary columns should be avoided. Select only the required columns from the source for optimizing the data loading performance.

2 Dataset Metadata Management

2.1 Meaningful, Standardized Data Column Names

- Rename column names to Camel case or Capital case for better visibility regarding, for example, page filters.
- Avoid confusing column names that may be misleading when creating a report. For example, Unit Price is better than Sales Price; Sales Price and Sales Amount may be misleading when used together.

2.2 Delete Unwanted Columns

- It is best to delete columns that are no longer required rather than marking them as inactive.
    - For example, the Maiden Name column can be removed from Employee Details when creating front-end objects for employee-wise sales.

3 Aggregation

3.1 Aggregating Over Time Period

- Aggregate the data for the required time period wherever possible. This optimizes report performance and decreases data operation time.
    - For example, if our data is on a daily basis and we are going to create front-end objects that are on a month-to-month basis, then data should be aggregated to the monthly level in the dataset.

3.2 Aggregating Over Group of Dimensions

- Aggregation is also advisable for the group of dimensions needed for reporting purposes.
    - For example, we have data from all the zones, states, and cities for all Product Categories and Product Names. If, however, our front-end objects are mainly focused on state-wise performance for all product categories, then we can remove the cities and product names from the data and roll up the aggregation level of the dataset to the States and Product Categories level.

3.3 Aggregating Over a Combination of Time Period and Descriptive Dimensions

- A combination of descriptive and periodic aggregation is required on datasets being used for final analysis, especially for trend comparisons.
    - For example, a combination from the above two aggregations wherein we will need state-wise, product category-wise monthly comparative analysis, we will be aggregating our final dataset to the combination of all three dimensions.

4 Blend/Join Data

4.1 Avoid Cross-Joins

- When joining datasets, the join must be based on columns that may result in a 1-1 or 1-N relationship only. Avoid cross-joins.
    - A master and transaction table join is a classic example here. Sales for a product will have sales transaction data joined with product master with a 1-N relationship depicting one product sold in multiple transactions.
    - Department information data and employee data will have a 1-1 join depicting that an employee belongs to a particular department in an organization.

4.2 Publish Child Dataset before Parent Dataset

- In case of a parent/child dataset used in JOINs, be sure to schedule publish for the child dataset before the parent dataset.
    - For example, considering the join scenarios mentioned above, if updated, product data needs to be refreshed and published before joined with sales data so as to have final updated product details in sales.

4.3 Publish only Datasets that are used in Front-end Objects

- Publish only datasets that are needed for report or front-end BI objects building. If the dataset is supposed to be used as a child dataset to be joined/appended in other dataset, then that doesn't need to be published.
    - For example, in the earlier scenario, if one dataset is for Products and another is Sales Data and the front-end objects are mainly focused on Sales Data analysis and we are joining Products Data in Sales Data for getting Product Information with Sales Details, then we may avoid publishing the Products Master dataset. This reduces storage space for published data and helps avoid any unnecessary dataset listing for reporting users.

4.4 Other points to be considered when joining Datasets

- Joins on integers will work faster than strings.
- Single column joins work better than multicolumn joins.
- Create a copy of the dataset after join/append only when you wish to retain the disjoined dataset for any other purpose.
    - For example, we have a dataset for details of students from the state board and another dataset for students from the central board, and we are appending this data with each other to create a combined report. We still have a few front-end objects, however, that are necessarily different for both boards. In that case, while performing the append operation, we can create a copy of the original dataset, and then we will have three separate datasets for different front-end objects.
- In case of extracting data from a database source, if there is a need to pull data from more than one table, then there is no need to create a separate dataset of each table. A better approach is to get the data joined from the query itself combined in one dataset.
    - For example, if there is a database table for the Products Category and another table for Product Details, then we shall join both tables in a query and pull combined data into one single dataset, i.e., we are not required to create a product category based on front-end objects without product information. There is no need to create two separate datasets for Products Category and Product Details.

5 Transformations

5.1 Data Cleansing and Sampling

- Be sure column markings for measures and dimensions are accurate, especially numeric dimension columns, such as Ids, which may be considered as a measure by default. Such columns should be explicitly marked as dimensions.
- Use TRANSFORM instead of a custom formula update wherever possible, for example, in case of transforming date formats. This is especially important when we need to change the display format of dates and numeric columns. In such cases, creating a new column is also a waste of storage space. Transforming the same column will be sufficient. You can later change display formats as per the will in front-end objects as well.
    - For example, if data from a source is in mm-dd-yyyy format but we need it in dd-mm-yyyy format, then we should use the transform option on the same date column.
    - If names from sources are in uppercase and we need them capitalized, then we can use transformation on the same column.
- Create required dimension hierarchies.
- Avoid switching to full data mode when working with transformations. Transformations work faster on sample data since volume is less. The reports will display all the data anyway.
- In case of a date-time column, if the time stamp part is empty, convert the datatype to date.
- For a column with double datatype and having values up to 5–6 decimal places, show data till 2 decimal places if it is not required to have more than 2 decimal digits in your front-end objects.
- Use "Find and Replace" (pattern-based replace) wherever applicable instead of Unique Value Edit. This will help reduce the number of steps and in turn increase the loading performance of a dataset.
    - For example, in the Product column, if we want to update “Tea & Coffee” with “Tea and Coffee” and “Biscuits & Cookies” with “Biscuits and Cookies,” we may use Find and Replace to replace “&” with “and” instead of running two separate steps for updating the two values.

5.2 Datatype Transformations

- Datatype identification is based on the source data scan before pulling data from the data source. Before this step, a user needs to be sure the data in each column is consistent, as a date column contains all the date values that too in same format or a numeric column with all numeric values and not alphanumeric or special characters.
    - For example, if a date column contains a few values in dd-mm-yyyy format but a few others in dd-mmm-yyy format, Smarten will not be able to identify its datatype as a date due to an inconsistent format and will mark it as String. So be sure to cleanse and streamline the source data before pulling it in Smarten.
- Inconsistent values may result in wrong datatype identification or sometimes even data loss.
    - For example, if by chance in a source with a million records with dates in a format of 01-Jan-2000 and a few records with dates are in a format of January 01, 2000, then these dates will not be parsed and changed to nulls.

- If a huge decimal point number column in Excel mistakingly contains a string or date value, then the whole column will be detected as string, and the numbers will be pulled with exponential update with no way of being able to transform it back to digits. Again, these kinds of anomalies or wrong entries need to be handled at the source before the extraction process. Be sure to cleanse or streamline the source data before loading into Smarten.

6 Balancing Load between Data Sources and Dataset Actions

6.1 Data Load Balancing

Balancing processing between data loading from data sources and transformation at the dataset level:

- Reduce the number of steps in the action editor in case of a very large amount of data (if possible).
- All derived calculations that can be taken care of at the query or data extraction level should be avoided in a dataset.
- Avoid creating copies of datasets unnecessarily, as this will result in replication of the same data.
- The unique value Replace or find/replace value should be done after all the join/append operations to avoid repeating the same operation many times.

7 Optimizing performance for Real-Time Datasets

Real-time datasets query a data source in real time, and it can affect the data source performance, so it is important to optimize the SQL query in case of database data sources and selection/filtration of source data columns for a database and all other data source types.

7.1 Deciding Granular Levels

- As in the case of cached datasets, for real-time datasets, also be sure the query is at its lowest granular level required for the report.
  - For example, an hourly trend report for 24 hours needs only one day's data to be pulled. We should avoid pulling all the data of the entire duration from the source.

7.2 Other points to be considered

- Avoid unnecessary columns and transformations
  - For example, our data has a date column with a format of dd-mmm-yyyy and date entries such as 01-Nov-2000 and 02-Nov-2000, and we need to display it as Month-Year only in mmm-yy format, i.e., 01-Nov-2000 -> Nov-20; 02-Nov-2000 -> Nov-20. We can do it by changing the display format by right-clicking on the date column in SSDP. We do NOT need to create a new column to extract Month and Year from the dates and then a third column with merged output.
- All the derived calculations that can be taken care of at the query level should be avoided in a dataset.

- For improved performance, a real-time dataset should only be report/dashboard specific instead of a generic dataset that can be used for numerous front-end objects at once.

8 Dataset Designing

8.1 Report-Based Data Building

- Datasets should be designed based on front-end objects, such as report requirements. This will improve and optimize report and dataset performance and loading time.
    - A report with State-Wise Product Category-Wise Monthly Trend can be created from an aggregated dataset.
    - A detailed report for the underlying Citi-wise Product-wise Daily Trend can be created from another transaction-level dataset.
    - These front-end objects can then be made interactive based on linked front-end objects or subviews from the front-end designing.

8.2 Building Generic Data Pool to be used by Multiple Front-end Objects

- Generic data that can be reused across multiple front-end objects by joining it in different datasets can be saved as a separate dataset. This dataset can be joined with other datasets as and when required. This will help avoid reloading the same data multiple times.
    - For example, Zone, State, City, area, office, and its exact geo-coordinates data are used for product-wise sales front-end objects as well as employee transfer and tenure front-end objects. So this regional data can be a separate dataset that is joined with multiple datasets as required.

9 Data Partitioning Guidelines

9.1 Data Partitioning Guidelines

- Data partitioning is the technique of distributing data across many files, tables, or disks in order to improve query processing performance or increase data manageability. Query processing performance can be improved depending on how the data is partitioned. Data partitioning enables parallel data processing. When data is partitioned across multiple disks or files, I/O parallelism and in some cases query parallelism can be attained, as different partitions can be accessed in parallel.
- Please refer to the Smarten Data Partitioning Guidelines document for more details.

10 Product and Support Information

Find more information about Smarten and its features at www.smarten.com
Support: support@smarten.com
Sales: sales@smarten.com
Feedback & Suggestions: support@smarten.com
Support & Knowledgebase Portal: support.smarten.com