Performance Optimization Guide

■ SAP BusinessObjects Data Services 4.1 Support Package 1 (14.1.1.0)

2012-12-12

Contents

Chapter 1 Welcome to SAP BusinessObjects Data Services....7

1.1 Welcome....7
1.2 Documentation set for SAP BusinessObjects Data Services....7
1.3 Accessing documentation....9
1.3.1 Accessing documentation on Windows....10
1.3.2 Accessing documentation on UNIX....10
1.3.3 Accessing documentation from the Web....10
1.4 SAP BusinessObjects information resources....10

Chapter 2 Environment Test Strategy....13

2.1 The source OS and database server....13
2.1.1 Operating system....13
2.1.2 Database....14
2.2 The target OS and database server....14
2.2.1 Operating system....14
2.2.2 Database....14
2.3 The network....15
2.4 Job Server OS and job options....15
2.4.1 Operating system....15
2.4.2 Jobs....16

Chapter 3 Measuring Performance....19

3.1 Data Services processes and threads....19
3.1.1 Processes....19
3.1.2 Threads....20
3.2 Measuring performance of jobs....20
3.2.1 Checking system utilization....20
3.2.2 Analyzing log files for task duration....24
3.2.3 Reading the Monitor Log for execution statistics....25
3.2.4 Reading the Performance Monitor for execution statistics....26
3.2.5 Reading Operational Dashboards for execution statistics....26

Chapter 4 Tuning Overview.....29

4.1 Strategies to execute jobs.....29
4.1.1 Maximizing push-down operations to the database server.....29
4.1.2 Improving throughput.....30
4.1.3 Using advanced tuning options.....30

Chapter 5 Maximizing Push-Down Operations.....33

5.1 Push-down operations.....33
5.1.1 Full push-down operations.....33
5.1.2 Partial push-down operations.....34
5.1.3 Operations that cannot be pushed down.....34
5.2 Push-down examples.....35
5.2.1 Collapsing transforms to push down operations example.....35
5.2.2 Full push down from the source to the target example.....36
5.2.3 Full push down for auto correct load to the target example.....36
5.2.4 Partial push down to the source example.....37
5.2.5 Push-down of SQL join example.....37
5.3 To view SQL.....38
5.4 Data_Transfer transform for push-down operations.....39
5.4.1 Push down an operation after a blocking operation example.....39
5.4.2 Using Data_Transfer tables to speed up auto correct loads example.....41
5.5 Database link support for push-down operations across datastores.....42
5.5.1 Software support.....42
5.5.2 Example of push-down with linked datastores.....43
5.5.3 Generated SQL statements.....45
5.5.4 Tuning performance at the data flow or Job Server level.....45

Chapter 6 Using Caches.....47

6.1 Caching data.....47
6.1.1 Caching sources.....47
6.1.2 Caching joins.....48
6.1.3 Changing cache type for a data flow.....49
6.1.4 Caching lookups.....50
6.1.5 Caching table comparisons.....51
6.1.6 Specifying a pageable cache directory.....51
6.2 Using persistent cache.....52
6.2.1 Using persistent cache tables as sources.....53
6.3 Monitoring and tuning caches.....53
6.3.1 Using statistics for cache self-tuning.....53

6.3.2 To monitor and tune in-memory and pageable caches....54

Chapter 7 Using Parallel Execution....57

7.1 Parallel data flows and work flows....57
7.2 Parallel execution in data flows....58
7.2.1 Table partitioning....58
7.2.2 Degree of parallelism....62
7.2.3 Combining table partitioning and a degree of parallelism....68
7.2.4 File multi-threading....70

Chapter 8 Distributing Data Flow Execution....75

8.1 Splitting a data flow into sub data flows....75
8.1.1 Run as a separate process option....75
8.1.2 Examples of multiple processes for a data flow....76
8.1.3 Data_Transfer transform....81
8.1.4 Examples of multiple processes with Data_Transfer....81
8.2 Using grid computing to distribute data flow execution....84
8.2.1 Server Group....84
8.2.2 Distribution levels for data flow execution....84

Chapter 9 Bulk Loading and Reading....89

9.1 Bulk loading in DB2 Universal Database....89
9.1.1 When to use each DB2 bulk-loading method....89
9.1.2 Using the DB2 CLI load method....90
9.1.3 Using the DB2 bulk load utility....92
9.1.4 Using the import utility....95
9.2 Bulk loading in HP Neoview....95
9.2.1 How Data Services and HP Neoview use the file options to load....96
9.2.2 Using the UPSERT bulk operation....97
9.3 Bulk loading in Informix....97
9.3.1 To set Informix server variables....97
9.4 Bulk loading in Microsoft SQL Server....98
9.4.1 To use the SQL Server ODBC bulk copy API....98
9.4.2 Network packet size option....98
9.4.3 Maximum rejects option....99
9.5 Bulk loading in Netezza....99
9.5.1 Netezza bulk-loading process....99
9.5.2 Options overview....100
9.5.3 Configuring bulk loading for Netezza....100
9.5.4 Netezza log files....101

9.6 Bulk loading in Oracle....102
9.6.1 Bulk-loading methods....102
9.6.2 Bulk-loading modes....102
9.6.3 Bulk-loading parallel-execution options....103
9.6.4 Bulk-loading scenarios....103
9.6.5 Using bulk-loading options....104
9.7 Bulk loading in SAP HANA....106
9.8 Bulk loading in Sybase ASE....106
9.9 Bulk loading in Sybase IQ....107
9.9.1 Configuring bulk loading for Sybase IQ....107
9.9.2 Sybase IQ log files....108
9.10 Bulk loading and reading in Teradata....108
9.10.1 Bulk loader file options....109
9.10.2 When to use each Teradata bulk-loading method....110
9.10.3 Parallel Transporter method....114
9.10.4 Teradata standalone utilities....117
9.10.5 Using the UPSERT bulk-loading operation....125
9.11 Bulk loading using DataDirect's Wire Protocol SQL Server ODBC driver....126
9.11.1 Enabling the DataDirect bulk load feature in Windows....126
9.11.2 Enabling the DataDirect bulk load feature in UNIX....127

Chapter 10 Other Tuning Techniques....129

10.1 Source-based performance options....129
10.1.1 Join ordering....130
10.1.2 Minimizing extracted data....136
10.1.3 Using array fetch size....136
10.2 Target-based performance options....137
10.2.1 Loading method....138
10.2.2 Rows per commit....139
10.3 Job design performance options....139
10.3.1 Loading only changed data....139
10.3.2 Minimizing data type conversion....140
10.3.3 Minimizing locale conversion....140
10.3.4 Precision in operations....140

Index 141

Welcome to SAP BusinessObjects Data Services

1.1 Welcome

SAP BusinessObjects Data Services delivers a single enterprise-class solution for data integration, data quality, data profiling, and text data processing that allows you to integrate, transform, improve, and deliver trusted data to critical business processes. It provides one development UI, metadata repository, data connectivity layer, run-time environment, and management console—enabling IT organizations to lower total cost of ownership and accelerate time to value. With SAP BusinessObjects Data Services, IT organizations can maximize operational efficiency with a single solution to improve data quality and gain access to heterogeneous sources and applications.

1.2 Documentation set for SAP BusinessObjects Data Services

You should become familiar with all the pieces of documentation that relate to your SAP BusinessObjects Data Services product.

Document	What this document provides
Administrator's Guide	Information about administrative tasks such as monitoring, lifecycle management, security, and so on.
Customer Issues Fixed	Information about customer issues fixed in this release.
Designer Guide	Information about how to use SAP BusinessObjects Data Services Designer.
Documentation Map	Information about available SAP BusinessObjects Data Services books, languages, and locations.
Installation Guide for Windows	Information about and procedures for installing SAP BusinessObjects Data Services in a Windows environment.
Installation Guide for UNIX	Information about and procedures for installing SAP BusinessObjects Data Services in a UNIX environment.
Integrator's Guide	Information for third-party developers to access SAP BusinessObjects Data Services functionality using web services and APIs.

Document	What this document provides
Master Guide	Information about the application, its components and scenarios for planning and designing your system landscape. Information about SAP BusinessObjects Information Steward is also provided in this guide.
Management Console Guide	Information about how to use SAP BusinessObjects Data Services Administrator and SAP BusinessObjects Data Services Metadata Reports.
Performance Optimization Guide	Information about how to improve the performance of SAP BusinessObjects Data Services.
Reference Guide	Detailed reference material for SAP BusinessObjects Data Services Designer.
Release Notes	Important information you need before installing and deploying this version of SAP BusinessObjects Data Services.
Technical Manuals	A compiled “master” PDF of core SAP BusinessObjects Data Services books containing a searchable master table of contents and index: • *Administrator's Guide* • *Designer Guide* • *Reference Guide* • *Management Console Guide* • *Performance Optimization Guide* • *Supplement for J.D. Edwards* • *Supplement for Oracle Applications* • *Supplement for PeopleSoft* • *Supplement for Salesforce.com* • *Supplement for Siebel* • *Supplement for SAP* • *Workbench Guide*
Text Data Processing Extraction Customization Guide	Information about building dictionaries and extraction rules to create your own extraction patterns to use with Text Data Processing transforms.
Text Data Processing Language Reference Guide	Information about the linguistic analysis and extraction processing features that the Text Data Processing component provides, as well as a reference section for each language supported.
Tutorial	A step-by-step introduction to using SAP BusinessObjects Data Services.
Upgrade Guide	Release-specific product behavior changes from earlier versions of SAP BusinessObjects Data Services to the latest release. This manual also contains information about how to migrate from SAP BusinessObjects Data Quality Management to SAP BusinessObjects Data Services.
What's New	Highlights of new key features in this SAP BusinessObjects Data Services release. This document is not updated for support package or patch releases.

Document	What this document provides
Workbench Guide	Provides users with information about how to use the Workbench to migrate data and database schema information between different database systems.

In addition, you may need to refer to several Supplemental Guides.

Document	What this document provides
Supplement for J.D. Edwards	Information about interfaces between SAP BusinessObjects Data Services and J.D. Edwards World and J.D. Edwards OneWorld.
Supplement for Oracle Applications	Information about the interface between SAP BusinessObjects Data Services and Oracle Applications.
Supplement for PeopleSoft	Information about interfaces between SAP BusinessObjects Data Services and PeopleSoft.
Supplement for Salesforce.com	Information about how to install, configure, and use the SAP BusinessObjects Data Services Salesforce.com Adapter Interface.
Supplement for SAP	Information about interfaces between SAP BusinessObjects Data Services, SAP Applications, SAP Master Data Services, and SAP NetWeaver BW.
Supplement for Siebel	Information about the interface between SAP BusinessObjects Data Services and Siebel.

We also include these manuals for information about SAP BusinessObjects Information platform services.

Document	What this document provides
Information Platform Services Administrator's Guide	Information for administrators who are responsible for configuring, managing, and maintaining an Information platform services installation.
Information Platform Services Installation Guide for UNIX	Installation procedures for SAP BusinessObjects Information platform services on a UNIX environment.
Information Platform Services Installation Guide for Windows	Installation procedures for SAP BusinessObjects Information platform services on a Windows environment.

1.3 Accessing documentation

You can access the complete documentation set for SAP BusinessObjects Data Services in several places.

1.3.1 Accessing documentation on Windows

After you install SAP BusinessObjects Data Services, you can access the documentation from the Start menu.

1. Choose **Start** > **Programs** > **SAP BusinessObjects Data Services 4.1** > **Data Services Documentation** > **All Guides**.
2. Click the appropriate shortcut for the document that you want to view.

1.3.2 Accessing documentation on UNIX

After you install SAP BusinessObjects Data Services, you can access the documentation by going to the directory where the printable PDF files were installed.

1. Go to `<LINK_DIR>/doc/book/en/`.
2. Using Adobe Reader, open the PDF file of the document that you want to view.

1.3.3 Accessing documentation from the Web

You can access the complete documentation set for SAP BusinessObjects Data Services from the SAP BusinessObjects Business Users Support site.

To do this, go to http://help.sap.com/bods.

You can view the PDFs online or save them to your computer.

1.4 SAP BusinessObjects information resources

A global network of SAP BusinessObjects technology experts provides customer support, education, and consulting to ensure maximum information management benefit to your business.

Useful addresses at a glance:

Address	Content
Customer Support, Consulting, and Education services http://service.sap.com/	Information about SAP Business User Support programs, as well as links to technical articles, downloads, and online forums. Consulting services can provide you with information about how SAP BusinessObjects can help maximize your information management investment. Education services can provide information about training options and modules. From traditional classroom learning to targeted e-learning seminars, SAP BusinessObjects can offer a training package to suit your learning needs and preferred learning style.
Product documentation http://help.sap.com/bods/	SAP BusinessObjects product documentation.
Supported Platforms (Product Availability Matrix) https://service.sap.com/PAM	Get information about supported platforms for SAP BusinessObjects Data Services. Use the search function to search for Data Services. Click the link for the version of Data Services you are searching for.

Environment Test Strategy

This section covers suggested methods of tuning source and target database applications, their operating systems, and the network used by your SAP BusinessObjects Data Services environment. It also introduces key job execution options.

This section contains the following topics:

- The source OS and database server
- The target OS and database server
- The network
- Job Server OS and job options

To test and tune jobs, work with all four of these components in the order shown above.

In addition to the information in this section, you can use your UNIX or Windows operating system and database server documentation for specific techniques, commands, and utilities that can help you measure and tune the SAP BusinessObjects Data Services environment.

2.1 The source OS and database server

Tune the source operating system and database to quickly read data from disks.

2.1.1 Operating system

Make the input and output (I/O) operations as fast as possible. The read-ahead protocol, offered by most operating systems, can greatly improve performance. This protocol allows you to set the size of each I/O operation. Usually its default value is 4 to 8 kilobytes which is too small. Set it to at least 64K on most platforms.

2.1.2 Database

Tune your database on the source side to perform SELECTs as quickly as possible.

In the database layer, you can improve the performance of SELECTs in several ways, such as the following:

- Create indexes on appropriate columns, based on your data flows.
- Increase the size of each I/O from the database server to match the OS read-ahead I/O size.
- Increase the size of the shared buffer to allow more data to be cached in the database server.
- Cache tables that are small enough to fit in the shared buffer. For example, if jobs access the same piece of data on a database server, then cache that data. Caching data on database servers will reduce the number of I/O operations and speed up access to database tables.

See your database server documentation for more information about techniques, commands, and utilities that can help you measure and tune the the source databases in your jobs.

2.2 The target OS and database server

Tune the target operating system and database to quickly write data to disks.

2.2.1 Operating system

Make the input and output operations as fast as possible. For example, the asynchronous I/O, offered by most operating systems, can greatly improve performance. Turn on the asynchronous I/O.

2.2.2 Database

Tune your database on the target side to perform INSERTs and UPDATES as quickly as possible.

In the database layer, there are several ways to improve the performance of these operations.

Here are some examples from Oracle:

- Turn off archive logging
- Turn off redo logging for all tables
- Tune rollback segments for better performance
- Place redo log files and data files on a raw device if possible
- Increase the size of the shared buffer

See your database server documentation for more information about techniques, commands, and utilities that can help you measure and tune the the target databases in your jobs.

2.3 The network

When reading and writing data involves going through your network, its ability to efficiently move large amounts of data with minimal overhead is very important. Do not underestimate the importance of network tuning (even if you have a very fast network with lots of bandwidth).

Set network buffers to reduce the number of round trips to the database servers across the network. For example, adjust the size of the network buffer in the database client so that each client request completely fills a small number of network packets.

2.4 Job Server OS and job options

Tune the Job Server operating system and set job execution options to improve performance and take advantage of self-tuning features of SAP BusinessObjects Data Services.

2.4.1 Operating system

SAP BusinessObjects Data Services jobs are multi-threaded applications. Typically a single data flow in a job initiates one `al_engine` process that in turn initiates at least 4 threads.

For maximum performance benefits:

- Consider a design that will run one `al_engine` process per CPU at a time.
- Tune the Job Server OS so that threads spread to all available CPUs.

For more information, see Checking system utilization.

2.4.2 Jobs

You can tune job execution options after:

- Tuning the database and operating system on the source and the target computers
- Adjusting the size of the network buffer
- Your data flow design seems optimal

You can tune the following execution options to improve the performance of your jobs:

- **Monitor sample rate**
- **Collect statistics for optimization** and **Use collected statistics**

2.4.2.1 Setting Monitor sample rate

During job execution, the SAP BusinessObjects Data Services writes information to the monitor log file and updates job events after the number of seconds specified in **Monitor sample rate** has elapsed. The default value is 30. Increase **Monitor sample rate** to reduce the number of calls to the operating system to write to the log file.

When setting **Monitor sample rate**, you must evaluate performance improvements gained by making fewer calls to the operating system against your ability to view more detailed statistics during job execution. With a higher **Monitor sample rate**, the software collects more data before calling the operating system to open the file, and performance improves. However, with a higher monitor rate, more time passes before you can view statistics during job execution.

Note:
If you use a virus scanner on your files, exclude the SAP BusinessObjects Data Services log from the virus scan. Otherwise, the virus scan analyzes the log repeatedly during the job execution, which causes a performance degradation.

2.4.2.2 Collecting statistics for self-tuning

SAP BusinessObjects Data Services provides a self-tuning feature to determine the optimal cache type (in-memory or pageable) to use for a data flow.

2.4.2.3 To take advantage of this self-tuning feature

1. When you first execute a job, select the option **Collect statistics for optimization** to collect statistics which include number of rows and width of each row. Ensure that you collect statistics with data volumes that represent your production environment. This option is not selected by default.
2. The next time you execute the job, this option is selected by default.
3. When changes occur in data volumes, re-run your job with **Collect statistics for optimization** to ensure that the software has the most current statistics to optimize cache types.

Related Topics

- Using Caches

Measuring Performance

This section contains the following topics:

- Data Services processes and threads
- Measuring performance of jobs

3.1 Data Services processes and threads

Data Services uses processes and threads to execute jobs that extract data from sources, transform the data, and load data into a data warehouse. The number of concurrently executing processes and threads affects the performance of Data Services jobs.

3.1.1 Processes

The processes used to run jobs are:

- al_jobserver

  The al_jobserver initiates one process for each Job Server configured on a computer. This process does not use much CPU power because it is only responsible for launching each job and monitoring the job's execution.

- al_engine

  For batch jobs, an al_engine process runs when a job starts and for each of its data flows. Real-time jobs run as a single process.

  The number of processes a batch job initiates also depends upon the number of:

  - parallel work flows
  - parallel data flows
  - sub data flows

For an example of the monitor log that displays the processes, see Analyzing log files for task duration.

3.1.2 Threads

A data flow typically initiates one `al_engine` process, which creates one thread per data flow object. A data flow object can be a source, transform, or target. For example, two sources, a query, and a target could initiate four threads.

If you are using parallel objects in data flows, the thread count will increase to approximately one thread for each source or target table partition. If you set the **Degree of parallelism** (DOP) option for your data flow to a value greater than one, the thread count per transform will increase. For example, a DOP of `5` allows five concurrent threads for a Query transform. To run objects within data flows in parallel, use the following features:

- Table partitioning
- File multithreading
- Degree of parallelism for data flows

Related Topics

- Using Parallel Execution

3.2 Measuring performance of jobs

You can use several techniques to measure performance of SAP BusinessObjects Data Services jobs:

- Checking system utilization
- Analyzing log files for task duration
- Reading the Monitor Log for execution statistics
- Reading the Performance Monitor for execution statistics
- Reading Operational Dashboards for execution statistics

3.2.1 Checking system utilization

The number of processes and threads concurrently executing affects the utilization of system resources (see Data Services processes and threads).

Check the utilization of the following system resources:

- CPU
- Memory
- Disk
- Network

To monitor these system resources, use the following tools:

For UNIX:

- top or a third party utility (such as glance for HPUX)

For Windows:

- Performance tab on the Task Manager

Depending on the performance of your jobs and the utilization of system resources, you might want to adjust the number of processes and threads. The following sections describe different situations and suggests features to adjust the number of processes and threads for each situation.

3.2.1.1 CPU utilization

SAP BusinessObjects Data Services is designed to maximize the use of CPUs and memory available to run the job.

The total number of concurrent threads a job can run depends upon job design and environment. Test your job while watching multi-threaded processes to see how much CPU and memory the job requires. Make needed adjustments to your job design and environment and test again to confirm improvements.

For example, if you run a job and see that the CPU utilization is very high, you might decrease the DOP value or run less parallel jobs or data flows. Otherwise, CPU thrashing might occur.

For another example, if you run a job and see that only half a CPU is being used, or if you run eight jobs on an eight-way computer and CPU usage is only 50%, you can be interpret this CPU utilization in several ways:

- One interpretation might be that the software is able to push most of the processing down to source and/or target databases.
- Another interpretation might be that there are bottlenecks in the database server or the network connection. Bottlenecks on database servers do not allow readers or loaders in jobs to use Job Server CPUs efficiently.

  To determine bottlenecks, examine:

  - Disk service time on database server computers

  Disk service time typically should be below 15 milliseconds. Consult your server documentation for methods of improving performance. For example, having a fast disk controller, moving database server log files to a raw device, and increasing log size could improve disk service time.

  - Number of threads per process allowed on each database server operating system. For example:
    - On HPUX, the number of kernel threads per process is configurable. The CPU to thread ratio defaults to one-to-one. It is recommended that you set the number of kernel threads per CPU to between 512 and 1024.
    - On Solaris and AIX, the number of threads per process is not configurable. The number of threads per process depends on system resources. If a process terminates with a message like "Cannot create threads," you should consider tuning the job.

      For example, use the **Run as a separate process** option to split a data flow or use the Da ta_Transfer transform to create two sub data flows to execute sequentially. Since each sub data flow is executed by a different al_engine process, the number of threads needed for each will be 50% less than in your previous job design.

      If you are using the **Degree of parallelism** option in your data flow, reduce the number for this option in the data flow Properties window.
  - Network connection speed

    Determine the rate that your data is being transferred across your network.
    - If the network is a bottle neck, you might change your job execution distribution level from sub data flow to data flow or job to execute the entire data flow on the local Job Server.
    - If the capacity of your network is much larger, you might retrieve multiple rows from source databases using fewer requests.

- Yet another interpretation might be that the system is under-utilized. In this case, you might increase the value for the **Degree of parallelism** option and increase the number of parallel jobs and data flows.

Related Topics
- Using Parallel Execution
- Using grid computing to distribute data flow execution
- Using array fetch size

3.2.1.2 Memory

For memory utilization, you might have one of the following different cases:

- Low amount of physical memory.

  In this case, you might take one of the following actions:

  - Add more memory to the Job Server.
  - Redesign your data flow to run memory-consuming operations in separate sub data flows that each use a smaller amount of memory, and distribute the sub data flows over different Job Servers to access memory on multiple machines. For more information, see Splitting a data flow into sub data flows.
  - Redesign your data flow to push down memory-consuming operations to the database. For more information, see Push-down operations.

  For example, if your data flow reads data from a table, joins it to a file, and then groups it to calculate an average, the group by operation might be occurring in memory. If you stage the data after the join and before the group by into a database on a different computer, then when a sub data flow reads the staged data and continues with the group processing, it can utilize memory from the database server on a different computer. This situation optimizes your system as a whole.

  For information about how to stage your data, see Data_Transfer transform. For more information about distributing sub data flows to different computers, see Using grid computing to distribute data flow execution.

- Large amount of memory but it is under-utilized.

  In this case, you might cache more data. Caching data can improve the performance of data transformations because it reduces the number of times the system must access the database.

  There are two types of caches available: in-memory and pageable. For more information, see Caching data.

- Paging occurs.

  Pageable cache is the default cache type for data flows. On Windows, UNIX, and Linux, the virtual memory available to the al_engine process is 3.5 gigabytes (500 megabytes of virtual memory is reserved for other engine operations, totaling 4GB). You can change this default limit by increasing the value of the `MAX_64BIT_PROCESS_VM_IN_MB` parameter in the `DSConfig.txt` file.

  If more memory is needed than these virtual memory limits, the software starts paging to continue executing the data flow.

  If your job or data flow requires more memory than these limits, you can take advantage of one of the following features to avoid paging:

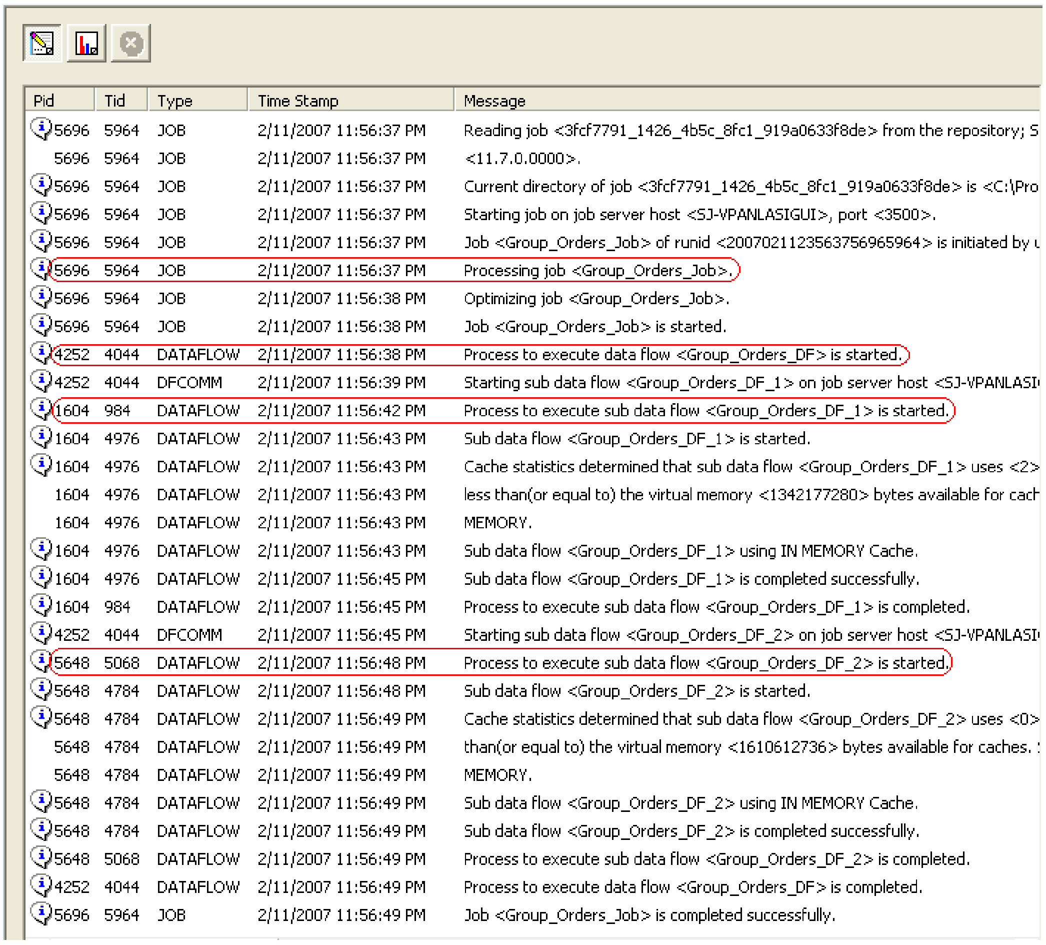
  - Split the data flow into sub data flows that can each use the amount of memory set by the virtual memory limits.

    Each data flow or each memory-intensive operation within a data flow can run as a separate process that uses separate memory from each other to improve performance and throughput. For more information, see Splitting a data flow into sub data flows.

  - Push-down memory-intensive operations to the database server so that less memory is used on the Job Server computer. For more information, see Push-down operations.

3.2.2 Analyzing log files for task duration

The trace log shows the progress of an execution through each component (object) of a job. The following sample Trace log shows a separate Process ID (Pid) for the Job, data flow, and each of the two sub data flows.



```
Pid   Tid   Type       Time Stamp              Message
5696  5964  JOB        2/11/2007 11:56:37 PM   Reading job <3fcf7791_1426_4b5c_8fc1_919a0633f8de> from the repository; S
5696  5964  JOB        2/11/2007 11:56:37 PM   <11.7.0.0000>.
5696  5964  JOB        2/11/2007 11:56:37 PM   Current directory of job <3fcf7791_1426_4b5c_8fc1_919a0633f8de> is <C:\Pro
5696  5964  JOB        2/11/2007 11:56:37 PM   Starting job on job server host <SJ-VPANLASIGUI>, port <3500>.
5696  5964  JOB        2/11/2007 11:56:37 PM   Job <Group_Orders_Job> of runid <2007021123563756965964> is initiated by u
5696  5964  JOB        2/11/2007 11:56:37 PM   Processing job <Group_Orders_Job>.
5696  5964  JOB        2/11/2007 11:56:38 PM   Optimizing job <Group_Orders_Job>.
5696  5964  JOB        2/11/2007 11:56:38 PM   Job <Group_Orders_Job> is started.
4252  4044  DATAFLOW   2/11/2007 11:56:38 PM   Process to execute data flow <Group_Orders_DF> is started.
4252  4044  DFCOMM     2/11/2007 11:56:39 PM   Starting sub data flow <Group_Orders_DF_1> on job server host <SJ-VPANLASI
1604  984   DATAFLOW   2/11/2007 11:56:42 PM   Process to execute sub data flow <Group_Orders_DF_1> is started.
1604  4976  DATAFLOW   2/11/2007 11:56:43 PM   Sub data flow <Group_Orders_DF_1> is started.
1604  4976  DATAFLOW   2/11/2007 11:56:43 PM   Cache statistics determined that sub data flow <Group_Orders_DF_1> uses <2>
1604  4976  DATAFLOW   2/11/2007 11:56:43 PM   less than(or equal to) the virtual memory <1342177280> bytes available for cach
1604  4976  DATAFLOW   2/11/2007 11:56:43 PM   MEMORY.
1604  4976  DATAFLOW   2/11/2007 11:56:43 PM   Sub data flow <Group_Orders_DF_1> using IN MEMORY Cache.
1604  4976  DATAFLOW   2/11/2007 11:56:45 PM   Sub data flow <Group_Orders_DF_1> is completed successfully.
1604  984   DATAFLOW   2/11/2007 11:56:45 PM   Process to execute sub data flow <Group_Orders_DF_1> is completed.
4252  4044  DFCOMM     2/11/2007 11:56:45 PM   Starting sub data flow <Group_Orders_DF_2> on job server host <SJ-VPANLASI
5648  5068  DATAFLOW   2/11/2007 11:56:48 PM   Process to execute sub data flow <Group_Orders_DF_2> is started.
5648  4784  DATAFLOW   2/11/2007 11:56:48 PM   Sub data flow <Group_Orders_DF_2> is started.
5648  4784  DATAFLOW   2/11/2007 11:56:49 PM   Cache statistics determined that sub data flow <Group_Orders_DF_2> uses <0>
5648  4784  DATAFLOW   2/11/2007 11:56:49 PM   than(or equal to) the virtual memory <1610612736> bytes available for caches.
5648  4784  DATAFLOW   2/11/2007 11:56:49 PM   MEMORY.
5648  4784  DATAFLOW   2/11/2007 11:56:49 PM   Sub data flow <Group_Orders_DF_2> using IN MEMORY Cache.
5648  4784  DATAFLOW   2/11/2007 11:56:49 PM   Sub data flow <Group_Orders_DF_2> is completed successfully.
5648  5068  DATAFLOW   2/11/2007 11:56:49 PM   Process to execute sub data flow <Group_Orders_DF_2> is completed.
4252  4044  DATAFLOW   2/11/2007 11:56:49 PM   Process to execute data flow <Group_Orders_DF> is completed.
5696  5964  JOB        2/11/2007 11:56:49 PM   Job <Group_Orders_Job> is completed successfully.
```

This sample log contains messages about sub data flows, caches, and statistics.

Related Topics

- Splitting a data flow into sub data flows
- Caching data
- Reference Guide: Objects, Log

3.2.3 Reading the Monitor Log for execution statistics

The Monitor log file indicates how many rows SAP BusinessObjects Data Services produces or loads for a job. By viewing this log during job execution, you can observe the progress of row-counts to determine the location of bottlenecks. You can use the Monitor log to answer questions such as the following:

- What transform is running at any moment?
- How many rows have been processed so far?

  The frequency that the Monitor log refreshes the statistics is based on Monitor sample rate.
- How long does it take to build the cache for a lookup or comparison table? How long does it take to process the cache?

  If take long time to build the cache, use persistent cache.
- How long does it take to sort?

  If take long time to sort, you can redesign your data flow to push down the sort operation to the database.
- How much time elapses before a blocking operation sends out the first row?

  If your data flow contains resource-intensive operations after the blocking operation, you can add Data_Transfer transforms to push-down the resource-intensive operations.

You can view the Monitor log from the following tools:
- The Designer, as the job executes, when you click the Monitor icon.
- The Administrator of the Management Console, when you click the Monitor link for a job from the Batch Job Status page.

The Monitor log in the Designer shows the path for each object in the job, the number of rows processed, and the elapsed time for each object. The Absolute time column displays the total time from the start of the job to when the software completes the execution of the data flow object.

Related Topics
- Setting Monitor sample rate
- Using persistent cache
- Push-down operations
- Data_Transfer transform for push-down operations
- Reference Guide: Objects, Log

3.2.4 Reading the Performance Monitor for execution statistics

The Performance Monitor displays execution information for each work flow, data flow, and sub data flow within a job. You can display the execution times in a table format. You can use the Performance Monitor to answer questions such as the following:

- Which data flows might be bottlenecks?
- How much time did a a data flow or sub data flow take to execute?
- How many rows did the data flow or sub data flow process?
- How much memory did a specific data flow use?

Note:
Memory statistics (Cache Size column) display in the Performance Monitor only if you select the **Collect statistics for monitoring** option when you execute the job.

3.2.4.1 To view the Performance Monitor

1. Access the Management Console with one of the following methods:
   - In the Designer top menu bar, click **Tools** and select **Management Console**.
   - Click **Start** > **Programs** > **SAP BusinessObjects Data Services** > **Data Services Management Console**.
2. On the launch page, click **Administrator**.
3. Select **Batch** > ***repository***
4. On the Batch Job Status page, find a job execution instance.
5. Under **Job Information** for an instance, click **Performance Monitor**.

Related Topics
• To monitor and tune in-memory and pageable caches

3.2.5 Reading Operational Dashboards for execution statistics

Operational dashboard reports contain job and data flow execution information for one or more repositories over a given time period (for example the last day or week). You can use operational statistics reports to answer some of the following questions:

- Are jobs executing within the allotted time frames?
- How many jobs succeeded or failed over a given execution period?
- How is the execution time for a job evolving over time?
- How many rows did the data flow process?

3.2.5.1 To compare execution times for the same job over time

1. Open the Management Console via one of the following methods:
   - In the Designer top menu bar, choose **Tools** > **Management Console**.
   - Choose **Start** > **Programs** > **SAP BusinessObjects Data Services** ***x.x*** > **Data Services Management Console**.
2. On the launch page, click **Operational Dashboard**.
3. Look at the graphs in Job Execution Statistic History or Job Execution Duration History to see if performance is increasing or decreasing.
4. On the Job Execution Duration History page, if there is a specific day that looks high or low compared to the other execution times, click that point on the graph to view the Job Execution Duration graph for all of the jobs that ran that day.
5. Click **View all history** to compare different executions of a specific job or data flow.
6. On the Job Execution History tab, you can select a specific job and number of days.
7. On the Data Flow Execution History tab, you can select a specific job and number of days, as well as search for a specific data flow.

Related Topics

- Management Console Guide: Operational Dashboard Reports

Tuning Overview

This section presents an overview of the different Data Services tuning options, with cross-references to subsequent chapters for more details.

4.1 Strategies to execute jobs

4.1.1 Maximizing push-down operations to the database server

SAP BusinessObjects Data Services generates SQL SELECT statements to retrieve the data from source databases. The software automatically distributes the processing workload by pushing down as much as possible to the source database server.

Pushing down operations provides the following advantages:

- Use the power of the database server to execute SELECT operations (such as joins, Group By, and common functions such as decode and string functions). Often the database is optimized for these operations.
- Minimize the amount of data sent over the network. Fewer rows can be retrieved when the SQL statements include filters or aggregations.

You can also do a full push down from the source to the target, which means the software sends SQL INSERT INTO... SELECT statements to the target database. The following features enable a full push down:

- Data_Transfer transform
- Database links and linked datastores

Related Topics

- Maximizing Push-Down Operations

4.1.2 Improving throughput

Use the following features to improve throughput:

- Using caches for faster access to data

  You can improve the performance of data transformations by caching as much data as possible. By caching data in memory, you limit the number of times the system must access the database.
- Bulk loading to the target

  The software supports database bulk loading engines including the Oracle bulk load API. You can have multiple bulk load processes running in parallel.
- Other tuning techniques
  - Source-based performance options
    - Join ordering
    - Minimizing extracted data
    - Using array fetch size
  - Target-based performance options
    - Loading method
    - Rows per commit
  - Job design performance options
    - Loading only changed data
    - Minimizing data type conversion
    - Minimizing locale conversion
    - Precision in operations

4.1.3 Using advanced tuning options

If your jobs have CPU-intensive and memory-intensive operations, you can use the following advanced tuning features to improve performance:

- Parallel processes—Individual work flows and data flows can execute in parallel if you do not connect them in the Designer workspace.

- Parallel threads—The software supports partitioned source tables, partitioned target tables, and degree of parallelism. These options allow you to control the number of instances for a source, target, and transform that can run in parallel within a data flow. Each instance runs as a separate thread and can run on a separate CPU.
- Server groups and distribution levels—You can group Job Servers on different computers into a logical component called a server group. A server group automatically measures resource availability on each Job Server in the group and distributes scheduled batch jobs to the computer with the lightest load at runtime. This functionality also provides a hot backup method. If one Job Server in a server group is down, another Job Server in the group processes the job.

  You can distribute the execution of data flows or sub data flows within a batch job across multiple Job Servers within a Server Group to better balance resource-intensive operations.

Related Topics

- Using Parallel Execution
- Management Console Guide: Server Groups
- Using grid computing to distribute data flow execution

Maximizing Push-Down Operations

For SQL sources and targets, SAP BusinessObjects Data Services creates database-specific SQL statements based on the data flow diagrams in a job. The software generates SQL SELECT statements to retrieve the data from source databases. To optimize performance, the software pushes down as many SELECT operations as possible to the source database and combines as many operations as possible into one request to the database. It can push down SELECT operations such as joins, Group By, and common functions such as decode and string functions.

Data flow design influences the number of operations that the software can push to the database. Before running a job, you can view the SQL that is generated and adjust your design to maximize the SQL that is pushed down to improve performance.

You can use database links and the Data_Transfer transform to pushdown more operations.

5.1 Push-down operations

By pushing down operations to the source database, Data Services reduces the number of rows and operations that the engine must retrieve and process, which improves performance. When determining which operations to push to the database, Data Services examines the database and its environment.

5.1.1 Full push-down operations

The Optimizer always first tries to do a full push-down operation. A full push-down operation is when all transform operations can be pushed down to the databases and the data streams directly from the source database to the target database. SAP BusinessObjects Data Services sends SQL INSERT INTO... SELECT statements to the target database where SELECT retrieves data from the source.

The software does a full push-down operation to the source and target databases when the following conditions are met:

- All of the operations between the source table and target table can be pushed down.
- The source and target tables are from the same datastore or they are in datastores that have a database link defined between them.

To enable a full push-down from the source to the target, you can also use the following features:

- Data_Transfer transform
- Database links

For database targets that support the **Allow merge or upsert** option, when all other operations in the data flow can be pushed down to the source database, the auto-correct loading operation may also be pushed down for a full push-down operation to the target. The software sends an SQL MERGE INTO *target* statement that implements the **Ignore columns with value** and **Ignore columns with null** options.

5.1.2 Partial push-down operations

When a full push-down operation is not possible, SAP BusinessObjects Data Services still pushes down the SELECT statement to the source database. Operations within the SELECT statement that the software can push to the database include:

- Aggregations — Aggregate functions, typically used with a **Group by** statement, always produce a data set smaller than or the same size as the original data set.
- Distinct rows — When you select **Distinct rows** from the **Select** tab in the query editor, the software will only output unique rows.
- Filtering — Filtering can produce a data set smaller than or equal to the original data set.
- Joins — Joins typically produce a data set smaller than or similar in size to the original tables. The software can push down joins when either of the following conditions exist:
  - The source tables are in the same datastore
  - The source tables are in datastores that have a database link defined between them
- Ordering — Ordering does not affect data-set size. The software can efficiently sort data sets that fit in memory. It is recommended that you push down the Order By for very large data sets.
- Projection — Projection is the subset of columns that you map on the **Mapping** tab in the query editor. Projection normally produces a smaller data set because it only returns columns needed by subsequent operations in a data flow.
- Functions — Most functions that have equivalents in the underlying database are appropriately translated. These functions include decode, aggregation, and string functions.

5.1.3 Operations that cannot be pushed down

SAP BusinessObjects Data Services cannot push some transform operations to the database. For example:

- Expressions that include functions that do not have database correspondents
- Load operations that contain triggers
- Transforms other than Query
- Joins between sources that are on different database servers that do not have database links defined between them.

Similarly, the software cannot always combine operations into single requests. For example, when a stored procedure contains a COMMIT statement or does not return a value, the software cannot combine the stored procedure SQL with the SQL for other operations in a query.

The software can only push operations supported by the DBMS down to that DBMS. Therefore, for best performance, try not to intersperse SAP BusinessObjects Data Services transforms among operations that can be pushed down to the database.

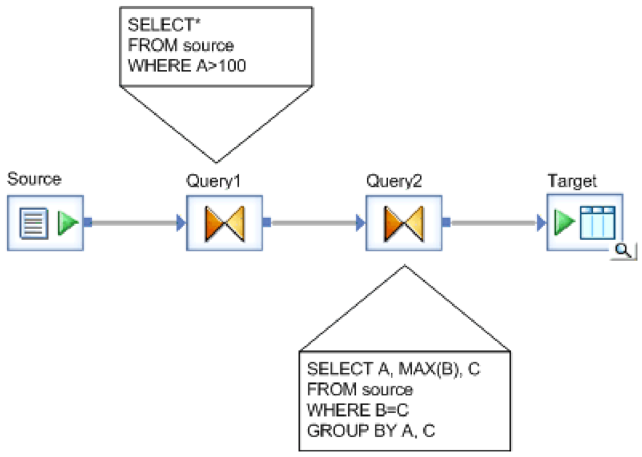
5.2 Push-down examples

The following are typical push-down scenarios.

5.2.1 Collapsing transforms to push down operations example

When determining how to push operations to the database, SAP BusinessObjects Data Services first collapses all the transforms into the minimum set of transformations expressed in terms of the source table columns. Next, the software pushes all possible operations on tables of the same database down to that DBMS.

For example, the following data flow extracts rows from a single source table.



```
SELECT*
FROM source
WHERE A>100
```

```
SELECT A, MAX(B), C
FROM source
WHERE B=C
GROUP BY A, C
```

The first query selects only the rows in the source where column `A` contains a value greater than `100`. The second query refines the extraction further, reducing the number of columns returned and further reducing the qualifying rows.

The software collapses the two queries into a single command for the DBMS to execute. The following command uses AND to combine the WHERE clauses from the two queries:

```
SELECT A, MAX(B), C
      FROM source
      WHERE A > 100 AND B = C
      GROUP BY A, C
```

The software can push down all the operations in this SELECT statement to the source DBMS.

5.2.2 Full push down from the source to the target example

If the source and target are in the same datastore, the software can do a full push-down operation where the INSERT into the target uses a SELECT from the source. In the sample data flow in scenario 1, a full push down passes the following statement to the database:

```
INSERT INTO target (A, B, C)
   SELECT A, MAX(B), C
      FROM source
      WHERE A > 100 AND B = C
      GROUP BY A, C
```

If the source and target are not in the same datastore, the software can also do a full push-down operation if you use one of the following features:

- Add a Data _Transfer transform before the target.
- Define a database link between the two datastores.

5.2.3 Full push down for auto correct load to the target example

For supported databases, if you enable the **Auto correct load** and **Allow merge or upsert** options, the Optimizer may be able to do a full push-down operation where the SQL statement is a MERGE into the target with a SELECT from the source.

In order for the **Allow merge or upsert** option to generate a MERGE statement, the primary key of the source table must be a subset of the primary key of the target table and the source row must be unique on the target key. In other words, there cannot be duplicate rows in the source data. If this condition is not met, the Optimizer pushes down the operation using a database-specific method to identify, update, and insert rows into the target table.

For example, suppose you have a data flow where the source and target tables are in the same datastore and the **Auto correct load** and **Allow merge or upsert** options are set to **Yes**.

The push-down operation passes the following statement to an Oracle database:

```
MERGE INTO "ODS"."TARGET" s
USING
SELECT "SOURCE"."A" A , "SOURCE"."B" B , "SOURCE"."C" C
    FROM "ODS"."SOURCE" "SOURCE"
    ) n
ON ((s.A = n.A))
WHEN MATCHED THEN
UPDATE SET s."B" = n.B,
    s."C" = n.C
WHEN NOT MATCHED THEN
INSERT /*+ APPEND */ (s."A", s."B", s."C" )
VALUES (n.A , n.B , n.C)
```

Similar statements are used for other supported databases.

5.2.4 Partial push down to the source example

If the data flow contains operations that cannot be passed to the DBMS, the software optimizes the transformation differently than the previous two scenarios. For example, if Query1 called `func(A) > 100`, where `func` is a SAP BusinessObjects Data Services custom function, then the software generates two commands:

- The first query becomes the following command which the source DBMS executes:

  ```
  SELECT A, B, C
      FROM source
       WHERE B = C
  ```

- The second query becomes the following command which SAP BusinessObjects Data Services executes because `func` cannot be pushed to the database:

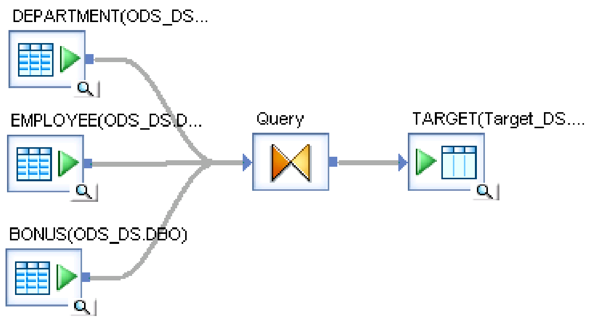
  ```
  SELECT A, MAX(B), C
        FROM Query1
        WHERE func(A) > 100
      GROUP BY A, C
  ```

5.2.5 Push-down of SQL join example

If the tables to be joined in a query meet the requirements for a push-down operation, then the entire query is pushed down to the DBMS.

To confirm that the query will be pushed down, look at the Optimized SQL. If the query shows a single SELECT statement, then it will be pushed down.

For example, in the data flow shown below, the Department and Employee tables are joined with a inner join and then the result of that join is joined with left outer join to the Bonus table.



The resulting Optimized SQL contains a single select statement and the entire query is pushed down to the DBMS:

```
SELECT  DEPARTMENT.DEPTID, DEPARTMENT.DEPARTMENT, EMPLOYEE.LASTNAME,
BONUS.BONUS
FROM (DEPARTMENT INNER JOIN EMPLOYEE
(ON DEPARTMENT.DEPTID=EMPLOYEE.DEPTID))
LEFT OUTER JOIN BONUS
ON  (EMPLOYEE.EMPID = BONUS.EMPID)
```

Related Topics

- To view SQL
- Maximizing Push-Down Operations
- Reference Guide: Transforms, Query, Joins in the Query transform

5.3 To view SQL

Before running a job, you can view the SQL code that SAP BusinessObjects Data Services generates for table sources in data flows. By examining the SQL code, you can verify that the software generates the commands you expect. If necessary, you can alter your design to improve the data flow.

1. Validate and save data flows.
2. Open a data flow in the workspace.
3. Select **Display Optimized SQL** from the **Validation** menu.

   Alternately, you can right-click a data flow in the object library and select Display Optimized SQL.

   The "Optimized SQL" window opens and shows a list of datastores and the optimized SQL code for the selected datastore. By default, the "Optimized SQL" window selects the first datastore.

   The software only shows the SELECT generated for table sources and INSERT INTO... SELECT for targets. It does not show the SQL generated for SQL sources that are not table sources, such as:

   - Lookup function

- Key_generation function
- Key_Generation transform
- Table_Comparison transform

4. Select a name from the list of datastores on the left to view the SQL that this data flow applies against the corresponding database or application.

   The following example shows the optimized SQL for the second datastore which illustrates a full push-down operation (INSERT INTO... SELECT). This data flows uses a Data_Transfer transform to create a transfer table that the software loads directly into the target.

   ```
   INSERT INTO "DBO"."ORDER_AGG" ("SHIPCOUNTRY","SHIPREGION", "SALES_AGG")
   SELECT "TS_Query_Lookup"."SHIPCOUNTRY" , "TS_Query_Lookup"."SHIPREGION" ,sum("TS_Query_Lookup"."SALES")
   FROM"DBO"."TRANS2""TS_Query_Lookup"
   GROUP BY "TS_Query_Lookup"."SHIPCOUNTRY" , "TS_Query_Lookup"."SHIPREGION"
   ```

   In the "Optimized SQL" window you can:

   - Use the **Find** button to perform a search on the SQL displayed.
   - Use the **Save As** button to save the text as a .sql file.

If you try to use the **Display Optimized SQL** command when there are no SQL sources in your data flow, the software alerts you. Examples of non-SQL sources include:

- Message sources
- File sources
- IDoc sources

If a data flow is not valid when you click the **Display Optimized SQL** option, the software alerts you.

Note:
The "Optimized SQL" window displays the existing SQL statement in the repository. If you changed your data flow, save it so that the "Optimized SQL" window displays your current SQL statement.

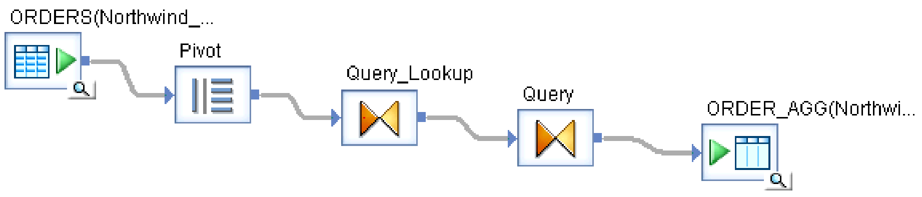
5.4 Data_Transfer transform for push-down operations

Use the Data_Transfer transform to move data from a source or from another transform into the target datastore and enable a full push-down operation (INSERT INTO... SELECT) to the target. You can use the Data_Transfer transform to push-down resource-intensive operations that occur anywhere within a data flow to the database. Resource-intensive operations include joins, GROUP BY, ORDER BY, and DISTINCT.

5.4.1 Push down an operation after a blocking operation example

You can place a Data_Transfer transform after a blocking operation to enable Data Services to push down a subsequent operation. A blocking operation is an operation that the software cannot push down to the database, and prevents ("blocks") operations after it from being pushed down.

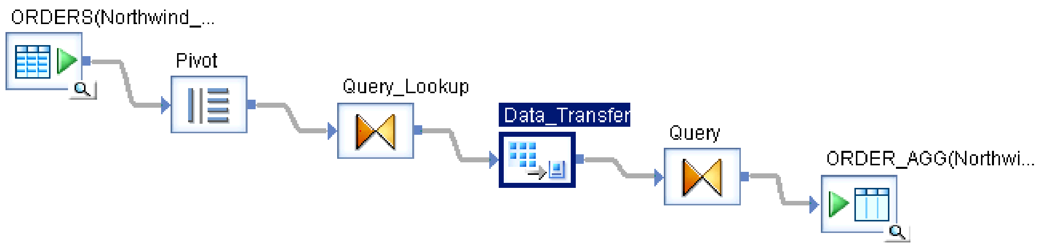
For example, you might have a data flow that groups sales order records by country and region, and sums the sales amounts to find which regions are generating the most revenue. The following diagram shows that the data flow contains a Pivot transform to obtain orders by Customer ID, a Query transform that contains a lookup_ext function to obtain sales subtotals, and another Query transform to group the results by country and region.



Because the Pivot transform and the lookup_ext function are before the query with the GROUP BY clause, the software cannot push down the GROUP BY operation. Here is how the "Optimized SQL" window would show the SELECT statement that the software pushes down to the source database:

```
SELECT "ORDERID", "CUSTOMERID", "EMPLOYEEID", "ORDERDATE", "REQUIREDDATE", "SHIPPEDDATE",, "SHIPVIA"
"FREIGHT", "SHIPNAME", "SHIPADDRESS", "SHIPCITY", "SHIPREGION", "SHIPPOSTALCODE", "SHIPCOUNTRY"
FROM "DBO"."ORDERS"
```

However, if you add a Data_Transfer transform before the second Query transform and specify a transfer table in the same datastore as the target table, the software can push down the GROUP BY operation.



The Data_Transfer Editor window shows that the transfer type is **Table** and the transfer table is in the same datastore as the target table (Northwind_DS.DBO.TRANS2).

Here's how the "Optimized SQL" window would show that the software pushed down the GROUP BY to the transfer table TRANS2.

```
INSTER INTO "DBO"."ORDER_AGG" ("SHIPCOUTNRY", "SHIPREGION", "SALES_AGG")
SELECT "TS_Query_Lookup"."SHIPCOUNTRY" , "TS_Query_Lookup"."SHIPREGION" , sum("TS_Query_Lookup"."SALES")
FROM "DBO"."TRANS2""TS_Query_Lookup"
GROUP BY "TS_Query_Lookup"."SHIPCOUNTRY" , "TS_Query_Lookup"."SHIPREGION"
```

Related Topics

• Operations that cannot be pushed down

5.4.2 Using Data_Transfer tables to speed up auto correct loads example

Auto correct loading ensures that the same row is not duplicated in a target table, which is useful for data recovery operations. However, an auto correct load prevents a full push-down operation from the source to the target when the source and target are in different datastores.

For large loads using database targets that support the **Allow merge or upsert** option for auto correct load, you can add a Data_Transfer transform before the target to enable a full push-down from the source to the target. In order for the **Allow merge or upsert** option to generate a MERGE statement:

- the primary key of the source table must be a subset of the primary key of the target table
- the source row must be unique on the target key

In other words, there cannot be duplicate rows in the source data. If this condition is not met, the Optimizer pushes down the operation using a database-specific method to identify, update, and insert rows into the target table.

If the MERGE statement can be used, SAP BusinessObjects Data Services generates an SQL MERGE INTO *`target`* statement that implements the **Ignore columns with value** value (if a value is specified in the target transform editor) and the **Ignore columns with null** Yes/No setting.

For example, suppose you create a data flow that loads sales orders into an Oracle target table which is in a different datastore from the source.

For this data flow, the **Auto correct load** option is active and set to Yes, and the **Ignore columns with null** and **Allow merge or upsert** options are also active.

The SELECT statement that the software pushes down to the source database would look like the following (as it would appear in the "Optimized SQL" window).

```
SELECT "ODS_SALESORDER"."SALES_ORDER_NUMBER" , "ODS_SALESORDER"."ORDER_DATE" , "ODS_SALESORDER"."CUST_ID"
FROM "ODS"."ODS_SALESORDER""ODS_SALESORDER"
```

When you add a Data_Transfer transform before the target and specify a transfer table in the same datastore as the target, the software can push down the auto correct load operation.

The following MERGE statement is what the software pushes down to the Oracle target (as it appears in the "Optimized SQL" window).

```
MERGE INTO "TARGET"."AUTO_CORRECT_LOAD2_TARGET" s
USING
(SELECT "AUTOLOADTRANSFER"."SALES_ORDER_NUMBER" SALES_ORDER_NUMBER,
"AUTOLOADTRANSFER"."ORDER_DATE" ORDER_DATE, "AUTOLOADTRANSFER"."CUST_ID" CUST_ID
FROM "TARGET"."AUTOLOADTRANSFER" "AUTOLOADTRANSFER") n
ON ((s.SALES_ORDER_NUMBER=n.SALES_ORDRE_NUMBER00
WHEN MATCHED THEN
UPDATE SET s."ORDER_DATE"=nvl(n.ORDER_DATE,s."ORDER_DATE"),
s."CUST_ID"=nbl(n.CUST_ID,s."CUST_ID"
WHEN NOT MATCHED THEN
INSERT(s."SALES_ORDER_NUMBER",s."ORDER_DATE",s."CUST_ID")
VALUES(n.SALES_ORDRE_NUMBER,n.ORDRE_DATE,n.CUSTID)
```

Similar statements are used for other supported databases.

5.5 Database link support for push-down operations across datastores

Various database vendors support one-way communication paths from one database server to another. SAP BusinessObjects Data Services refers to communication paths between databases as database links. The datastores in a database link relationship are called linked datastores.

The software uses linked datastores to enhance its performance by pushing down operations to a target database using a target datastore. Pushing down operations to a database not only reduces the amount of information that needs to be transferred between the databases and SAP BusinessObjects Data Services but also allows the software to take advantage of the various DMBS capabilities, such as various join algorithms.

With support for database links, the software pushes processing down from different datastores, which can also refer to the same or different database type. Linked datastores allow a one-way path for data. For example, if you import a database link from target database B and link datastore B to datastore A, the software pushes the load operation down to database B, not to database A.

This section contains the following topics:

- Software support
- Example of push-down with linked datastores
- Generated SQL statements
- Tuning performance at the data flow or Job Server level

Related Topics

- Designer Guide: Datastores, Linked datastores

5.5.1 Software support

SAP BusinessObjects Data Services supports push-down operations using linked datastores on all Windows and Unix platforms. It supports DB2, Oracle, and MS SQL server databases.

5.5.1.1 To take advantage of linked datastores

1. Create a database link on a database server that you intend to use as a target in a job.

The following database software is required. See the Supported Platforms document for specific version numbers.

- For DB2, use the DB2 Information Services (previously known as Relational Connect) software and make sure that the database user has privileges to create and drop a nickname.

  To end users and client applications, data sources appear as a single collective database in DB2. Users and applications interface with the database managed by the information server. Therefore, configure an information server and then add the external data sources. DB2 uses nicknames to identify remote tables and views.

  See the DB2 database manuals for more information about how to create links for DB2 and non-DB2 servers.

- For Oracle, use the Transparent Gateway for DB2 and MS SQL Server.

  See the Oracle database manuals for more information about how to create database links for Oracle and non-Oracle servers.

- For MS SQL Server, no special software is required.

  Microsoft SQL Server supports access to distributed data stored in multiple instances of SQL Server and heterogeneous data stored in various relational and non-relational data sources using an OLE database provider. SQL Server supports access to distributed or heterogeneous database sources in Transact-SQL statements by qualifying the data sources with the names of the linked server where the data sources exist.

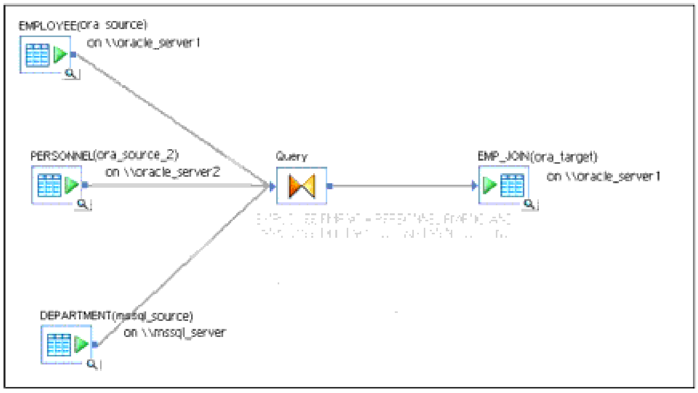
  See the MS SQL Server database manuals for more information.

2. Create a database datastore connection to your target database.

5.5.2 Example of push-down with linked datastores

Linked datastores enable a full push-down operation (INSERT INTO... SELECT) to the target if all the sources are linked with the target. The sources and target can be in datastores that use the same database type or different database types.

The following diagram shows an example of a data flow that will take advantage of linked datastores:



The dataflow joins three source tables from different database types:

- ora_source.HRUSER1.EMPLOYEE on \\oracle_server1
- ora_source_2.HRUSER2.PERSONNEL on \\oracle_server2
- mssql_source.DBO.DEPARTMENT on \\mssql_server3.

The software loads the join result into the target table ora_target.HRUSER3.EMP_JOIN on \\oracle_server1.

In this data flow, the user (HRUSER3) created the following database links in the Oracle database oracle_server1.

Database Link Name	Local (to database link location) Connection Name	Remote (to database link location) Connection Name	Remote User
orasvr2	oracle_server1	oracle_server2	HRUSER2
tg4msql	oracle_server1	mssql_server	DBO

To enable a full push-down operation, database links must exist from the target database to all source databases and links must exist between the following datastores:

- ora_target and ora_source
- ora_target and ora_source2
- ora_target and mssql_source

The software executes this data flow query as one SQL statement in oracle_server1:

```
INSERT INTO HR_USER3.EMP_JOIN (FNAME, ENAME, DEPTNO, SAL, COMM)
SELECT psnl.FNAME, emp.ENAME, dept.DEPTNO, emp.SAL, emp.COMM
FROM HR_USER1.EMPLOYEE emp, HR_USER2.PERSONNEL@orasvr2 psnl,
oracle_server1.mssql_server.DBO.DEPARTMENT@tg4msql dept;
```

5.5.3 Generated SQL statements

To see how SAP BusinessObjects Data Services optimizes SQL statements, use **Display Optimized SQL** from the **Validation** menu when a data flow is open in the workspace.

- For DB2, it uses nicknames to refer to remote table references in the SQL display.
- For Oracle, it uses the following syntax to refer to remote table references: `<remote_ta ble>@<dblink_name>`.
- For SQL Server, it uses the following syntax to refer to remote table references: `<liked_server >.<remote_database >.<remote_user >.<remote_table>`.

5.5.4 Tuning performance at the data flow or Job Server level

You might want to turn off linked-datastore push downs in cases where you do not notice performance improvements.

For example, the underlying database might not process operations from different data sources well. Data Services pushes down Oracle stored procedures and external functions. If these are in a job that uses database links, it will not impact expected performance gains. However, Data Services does not push down functions imported from other databases (such as DB2). In this case, although you may be using database links, Data Services cannot push the processing down.

Test your assumptions about individual job designs before committing to a large development effort using database links.

5.5.4.1 For a data flow

On the data flow properties dialog, this product enables the **Use database links** option by default to allow push-down operations using linked datastores. If you do not want to use linked datastores in a data flow to push down processing, deselect the check box.

This product can perform push downs using datastore links if the tables involved share the same database type and database connection name, or datasource name, even if the tables have different schema names. However, problems with enabling this feature could arise, for example, if the user of one datastore does not have access privileges to the tables of another datastore, causing a data access problem. In such a case, you can disable this feature.

5.5.4.2 For a Job Server

You can also disable linked datastores at the Job Server level. However, the **Use database links** option, at the data flow level, takes precedence.

Related Topics

• Designer Guide: Executing Jobs, Changing Job Server options

Using Caches

This section contains the following topics:

- Caching data
- Using persistent cache
- Monitoring and tuning caches

6.1 Caching data

You can improve the performance of data transformations that occur in memory by caching as much data as possible. By caching data, you limit the number of times the system must access the database.

SAP BusinessObjects Data Services provides the following types of caches that your data flow can use for all of the operations it contains:

- In-memory

  Use in-memory cache when your data flow processes a small amount of data that fits in memory.

- Pageable cache

  Use pageable cache when your data flow processes a very large amount of data that does not fit in memory. When memory-intensive operations (such as Group By and Order By) exceed available memory, the software uses pageable cache to complete the operation.

Pageable cache is the default cache type. To change the cache type, use the **Cache type** option on the data flow Properties window.

Note:
If your data fits in memory, it is recommended that you use in-memory cache because pageable cache incurs an overhead cost.

6.1.1 Caching sources

By default, the **Cache** option is set to `Yes` in a source table or file editor to specify that data from the source is cached using memory on the Job Server computer. When sources are joined using the Query transform, the cache setting in the Query transform takes precedence over the setting in the source.

The default value for **Cache type** for data flows is `Pageable`.

It is recommended that you cache small tables in memory. Calculate the approximate size of a table with the following formula to determine if you should use a cache type of `Pageable` or `In-memory`.

`table size = (in bytes)`	`# of rows *`
	`    # of columns *`
	`    20 bytes (average column size) *`
	`    1.3 (30% overhead)`

Compute row count and table size on a regular basis, especially when:

- You are aware that a table has significantly changed in size.
- You experience decreased system performance.

If the table fits in memory, change the value of the **Cache type** option to `In-memory` in the Properties window of the data flow.

Related Topics

- Caching joins

6.1.2 Caching joins

The Cache setting indicates whether the software should read the required data from the source and load it into memory or pageable cache.

When sources are joined using the Query transform, the cache setting in the Query transform takes precedence over the setting in the source. In the Query editor, the cache setting is set to Automatic by default. The automatic setting carries forward the setting from the source table. The following table shows the relationship between cache settings in the source, Query editor, and whether the software will load the data in the source table into cache.

Cache Setting in Source	Cache Setting in Query Editor	Effective Cache Setting
Yes	Automatic	Yes
No	Automatic	No
Yes	Yes	Yes
No	Yes	Yes

Cache Setting in Source	Cache Setting in Query Editor	Effective Cache Setting
Yes	No	No
No	No	No

Note:

- If any one input schema has a cache setting other than Automatic specified in the Query editor, the Data Services Optimizer considers only Query editor cache settings and ignores all source editor cache settings.
- Best practice is to define the join rank and cache settings in the Query editor.

In the Query editor, cache a source only if it is being used as an inner source in a join.

When the cache setting is such that data will be cached if possible, a source is used as an inner source in a join under the following conditions:

- The source is specified as the inner source of a left outer join.
- In an inner join between two tables, the source has a lower join rank.

Caching does not affect the order in which tables are joined.

If optimization conditions are such that the software is pushing down operations to the underlying database, it ignores your cache setting.

If a table becomes too large to fit in the cache, ensure that the cache type is pageable.

Related Topics

- About join ordering

6.1.3 Changing cache type for a data flow

You can improve the performance of data transformations that occur in memory by caching as much data as possible. By caching data, you limit the number of times the system must access the database.

To change the cache type for a data flow:

1. In the object library, select the data flow name.
2. Right-click and choose **Properties**.
3. On the General tab of the Properties window, select the desired cache type in the drop-down list for the **Cache type** option.

6.1.4 Caching lookups

You can also improve performance by caching data when looking up individual values from tables and files.

6.1.4.1 Using a Lookup function in a query

SAP BusinessObjects Data Services has three Lookup functions: `lookup`, `lookup_seq`, and `lookup_ext`. The `lookup` and `lookup_ext` functions have cache options. Caching lookup sources improves performance because the software avoids the expensive task of creating a database query or full file scan on each row.

You can set cache options when you specify a lookup function. There are three caching options:

- NO_CACHE — Does not cache any values.
- PRE_LOAD_CACHE — Preloads the result column and compare column into memory (it loads the values before executing the lookup).
- DEMAND_LOAD_CACHE — Loads the result column and compare column into memory as the function executes.

  Use this option when looking up highly repetitive values that are a small subset of the data and when missing values are unlikely.

  Demand-load caching of lookup values is helpful when the lookup result is the same value multiple times. Each time the software cannot find the value in the cache, it must make a new request to the database for that value. Even if the value is invalid, the software has no way of knowing if it is missing or just has not been cached yet.

  When there are many values and some values might be missing, demand-load caching is significantly less efficient than caching the entire source.

6.1.4.2 Using a source table and setting it as the outer join

Although you can use lookup functions inside SAP BusinessObjects Data Services queries, an alternative is to expose the translate (lookup) table as a source table in the data flow diagram, and use an outer join (if necessary) in the query to look up the required data. This technique has some advantages:

- You can graphically see the table the job will search on the diagram, making the data flow easier to maintain
- The software can push the execution of the join down to the underlying RDBMS (even if you need an outer join)

This technique also has some disadvantages:

- You cannot specify default values in an outer join (default is always null), but you can specify a default value in lookup_ext.
- If an outer join returns multiple rows, you cannot specify what to return, (you can specify MIN or MAX in lookup_ext).
- The workspace can become cluttered if there are too many objects in the data flow.
- There is no option to use DEMAND_LOAD caching, which is useful when looking up only a few repetitive values in a very large table.

Tip:
If you use the lookup table in multiple jobs, you can create a persistent cache that multiple data flows can access. For more information, see Using persistent cache.

6.1.5 Caching table comparisons

You can improve the performance of a Table_Comparison transform by caching the comparison table. There are three modes of comparisons:

- Row-by-row select
- Cached comparison table
- Sorted input

Of the three, **Row-by-row select** will likely be the slowest and **Sorted input** the fastest.

Tip:

- If you want to sort the input to the table comparison transform, then choose the **Sorted input** option for comparison.
- If the input is not sorted, then choose the **Cached comparison table** option.

6.1.6 Specifying a pageable cache directory

If the memory-consuming operations in your data flow exceed the available memory, SAP Business Objects Data Services uses pageable cache to complete the operation. Memory-intensive operations include the following operations:

- Distinct
- Functions such as count_distinct and lookup_ext
- Group By
- Hierarchy_Flattening
- Order By

Note:
The default pageable cache directory is %LINKDIR\Log\PCache. If your data flows contain memory-consuming operations, change this value to a pageable cache directory that:

- Contains enough disk space for the amount of data you plan to profile.
- Is on a separate disk or file system from the SAP BusinessObjects Data Services system.

Change the directory in the **Specify a directory with enough disk space for pageable cache** option in the Server Manager, under Runtime resources configured for this computer.

6.2 Using persistent cache

Persistent cache datastores provide the following benefits for data flows that process large volumes of data.

- You can store a large amount of data in persistent cache which SAP BusinessObjects Data Services quickly pages into memory each time the job executes. For example, you can access a lookup table or comparison table locally (instead of reading from a remote database).
- You can create cache tables that multiple data flows can share (unlike a memory table which cannot be shared between different real-time jobs). For example, if a large lookup table used in a lookup_ext function rarely changes, you can create a cache once and subsequent jobs can use this cache instead of creating it each time.

Persistent cache tables can cache data from relational database tables and files.

Note:
You cannot cache data from hierarchical data files such as XML messages and SAP IDocs (both of which contain nested schemas). You cannot perform incremental inserts, deletes, or updates on a persistent cache table.

You create a persistent cache table by loading data into the persistent cache target table using one data flow. You can then subsequently read from the cache table in another data flow. When you load data into a persistent cache table, SAP BusinessObjects Data Services always truncates and recreates the table.

6.2.1 Using persistent cache tables as sources

After you create a persistent cache table as a target in one data flow, you can use the persistent cache table as a source in any data flow. You can also use it as a lookup table or comparison table.

Related Topics

- Reference Guide: Objects, Persistent cache source

6.3 Monitoring and tuning caches

This section describes the following topics:

Related Topics

- Using statistics for cache self-tuning
- To monitor and tune in-memory and pageable caches

6.3.1 Using statistics for cache self-tuning

SAP BusinessObjects Data Services uses cache statistics collected from previous job runs to automatically determine which cache type to use for a data flow. Cache statistics include the number of rows processed.

The default cache type is pageable. the software can switch to in-memory cache when it determines that your data flow processes a small amount of data that fits in memory.

6.3.1.1 To automatically choose the cache type

1. Run your job with options **Collect statistics for optimization**.
2. Run your job again with option **Use collected statistics** (this option is selected by default).

6.3.2 To monitor and tune in-memory and pageable caches

You can also monitor and choose the cache type to use for the data flow.

1. Test run your job with options **Collect statistics for optimization** and **Collect statistics for monitoring**.

   Note:
   The option Collect statistics for monitoring is very costly to run because it determines the cache size for each row processed.

2. Run your job again with option **Use collected statistics** (this option is selected by default).
3. Look in the Trace Log to determine which cache type was used.
   - The first time you run the job or if you have not previously collected statistics, the following messages indicate that cache statistics are not available and the sub data flows use the default cache type, pageable.

     ```
     Cache statistics for sub data flow <GroupBy_DF_1_1> are not available to be used for optimization and
      need to be collected before they can be used.
     ```

     ```
     Sub data flow <GroupBy_DF_1_1> using PAGEABLE Cache with <1280 MB> buffer pool.
     ```

   - You might see the following message that indicates that the software is switching to In-memory cache:

     ```
     Cache statistics determined that sub data flow <GroupBy_DOP2_DF_1_4> uses <1> caches with a total
     size of <1920> bytes. This is less than (or equal to) the virtual memory <1342177280> bytes available
      for caches. Statistics is switching the cache type to IN MEMORY.
     ```

     ```
     Sub data flow <GroupBy_DOP2_DF_1_4> using IN MEMORY Cache.
     ```

     Because pageable cache is the default cache type for a data flow, you might want to permanently change the **Cache type** to In-Memory in the data flow "Properties" window.

   - You might see the following messages that indicate on sub data flow uses IN MEMORY cache and the other sub data flow uses PAGEABLE cache:

     ```
     Sub data flow <Orders_Group_DF_1> using IN MEMORY Cache.
     ```

     ```
     ...
     ```

     ```
     Sub data flow <Orders_Group_DF_2> using PAGEABLE Cache with <1536 MB> buffer pool.
     ```

4. Look in the Administrator Performance Monitor to view data flow statistics and see the cache size.
   a. On the Administrator, select **Batch** > ***repository name***
   b. On the Batch Job Status page, find a job execution instance.
   c. Under **Job Information** for an instance, click **Performance Monitor**. The Administrator opens the Table tab of the Performance Monitor page. This tab shows a tabular view of the start time, stop time, and execution time for each work flow, data flow, and sub data flow within the job.
   d. To display statistics for each object within a data flow or sub data flow, click one of the data flow names on the Table tab. The Transform tab displays the following statistics.

Statistic	Description
Name	Name that you gave the object (source, transform, or target) in the Designer.
Type	Type of object within the data flow. Possible values include Source, Mapping, Target.
Start time	Date and time this object instance started execution.
End time	Date and time this object instance stopped execution.
Execution time (sec)	Time (in seconds) the object took to complete execution.
Row Count	Number of rows that this object processed.
Cache Size (KB)	Size (in kilobytes) of the cache that was used to process this object. **Note:** This statistics displays only if you selected **Collect statistics for monitoring** for the job execution.

5. If the value in Cache Size is approaching the physical memory limit on the job server, consider changing the **Cache type** of a data flow from **In-memory** to **Pageable**.

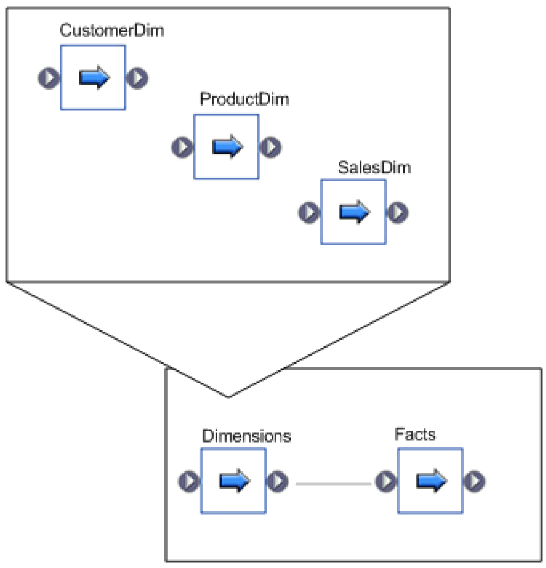
Using Parallel Execution

You can set SAP BusinessObjects Data Services to perform data extraction, transformation, and loads in parallel by setting parallel options for sources, transforms, and targets. In addition, you can set individual data flows and work flows to run in parallel by simply not connecting them in the workspace. If the Job Server is running on a multi-processor computer, it takes full advantage of available CPUs.

7.1 Parallel data flows and work flows

You can explicitly execute different data flows and work flows in parallel by not connecting them in a work flow or job. SAP BusinessObjects Data Services coordinates the parallel steps, then waits for all steps to complete before starting the next sequential step.

For example, use parallel processing to load dimension tables by calling work flows in parallel. Then specify that your job creates dimension tables before the fact table by moving it to the left of a second (parent) work flow and connecting the flows.



Parallel engine processes execute the parallel data flow processes. Note that if you have more than eight CPUs on your Job Server computer, you can increase **Maximum number of engine processes** to improve performance. To change the maximum number of parallel engine processes, use the Job Server options (**Tools** > **Options> Job Server** > **Environment**).

7.2 Parallel execution in data flows

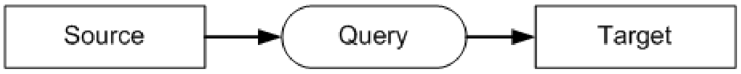
For batch jobs, SAP BusinessObjects Data Services allows you to execute parallel threads in data flows.

7.2.1 Table partitioning

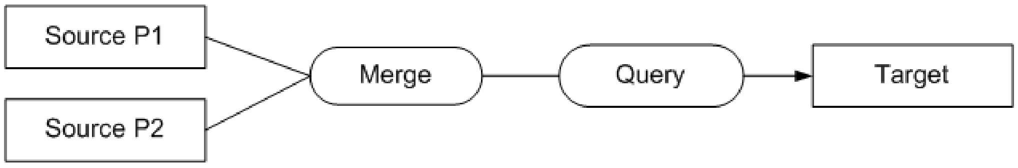
SAP BusinessObjects Data Services processes data flows with partitioned tables based on the amount of partitioning defined.

7.2.1.1 Data flow with source partitions only

If you have a data flow with a source that has two partitions connected to a query and a target, it appears in the workspace as shown in the following diagram:



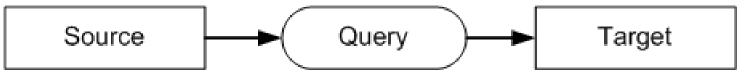
At runtime, the software translates this data flow to:



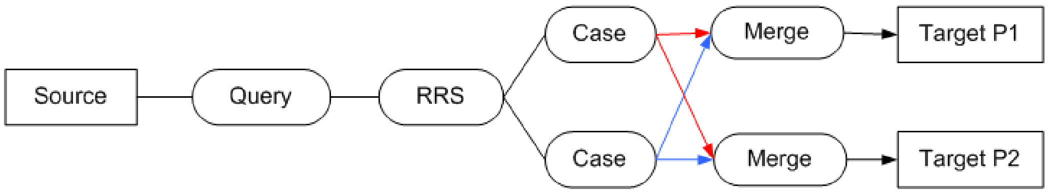
The software instantiates a source thread for each partition, and these threads run in parallel. The data from these threads later merges into a single stream by an internal merge transform before processing the query.

7.2.1.2 Data flow with target partitions only

If you have a data flow with a target that has two partitions connected to a query and a source, it appears in the workspace as shown in the following diagram:



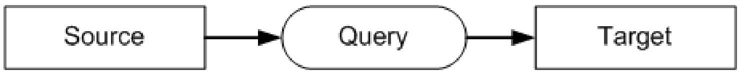
At runtime, the software translates this data flow to:



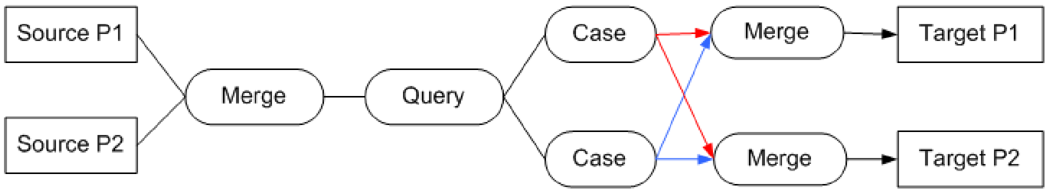
The software inserts an internal Round Robin Split (RRS) transform after the Query transform, which routes incoming rows in a round-robin fashion to internal Case transforms. The Case transforms evaluate the rows to determine the partition ranges. Finally, an internal Merge transform collects the incoming rows from different Case transforms and outputs a single stream of rows to the target threads. The Case , Merge , and the target threads execute in parallel.

7.2.1.3 Dataflow with source and target partitions

If you have a data flow with a source that has two partitions connected to a query and a target that has two partitions, it appears in the workspace as shown in the following diagram:



At runtime, the software translates this data flow to:



The source threads execute in parallel and the Case , Merge , and targets execute in parallel.

7.2.1.4 Viewing, creating, and enabling table partitions

Oracle databases support range, list, and hash partitioning. You can import this information as table metadata and use it to extract data in parallel. You can use range and list partitions to load data to Oracle targets. You can also specify logical range and list partitions using SAP BusinessObjects Data Services metadata for Oracle tables.

In addition, it provides the ability to specify logical range partitions for DB2, Microsoft SQL Server, Sybase ASE, and Sybase IQ tables by modifying imported table metadata.

SAP BusinessObjects Data Services uses partition information by instantiating a thread at runtime for each partition. These threads execute in parallel. To maximize performance benefits, use a multi-processor environment.

7.2.1.4.1 To view partition information

1. Import a table into SAP BusinessObjects Data Services.
2. In the **Datastores** tab of the object library, right-click the table name and select **Properties**.
3. Click the **Partitions** tab.

   When you import partitioned tables from your database, you will find these partitions displayed on the Partitions tab of the table's Properties window. The partition name appears in the first column. The columns that are used for partitioning appear as column headings in the second row.

   If you import a table that does not have partitions, you can create logical partitions using the Partitions tab of the table's Properties window.

7.2.1.4.2 To create or edit table partition information

1. In the **Datastores** tab of the object library, right-click the table name and select **Properties**.
2. In the Properties window, click the **Partitions** tab.
3. Select a partition type.

Partition Type	Description
None	This table is not partitioned.
Range	Each partition contains a set of rows with column values less than those specified. For example, if the value of column one is `100,000`, then the data set for partition one will include rows with values less than `100,000` in column one.
List	Each partition contains a set of rows that contain the specified column values.

Note:
If you imported an Oracle table with hash partitions, you cannot edit the hash settings in SAP BusinessObjects Data Services. The Partitions tab displays the hash partition name and ID as read-only information. However, you can change the partition type to Range or List to create logical range or list partitions for an Oracle table imported with hash partitions.

4. Add, insert, or remove partitions and columns using the tool bar. (See table at the end of this procedure.)
5. Select the name of a column from each column list box.
6. Enter column values for each partition.

   SAP BusinessObjects Data Services validates the column values entered for each partition according to the following rules:
   - Values can be literal numbers and strings or datetime types.
   - Column values must match column data types.
   - Literal strings must include single quotes: `'Director'`.
   - For range partitions, the values for a partition must be greater than the values for the previous partition.
   - For the last partition, you can enter the value `MAXVALUE` to include all values.
7. Click **OK**.

   If the validation rules described in the previous step are not met, you will see an error message.

Icon	Description
	Add Partition
	Insert Partition
	Remove Partition
	Add Column
	Insert Column
	Remove Column

The number of partitions in a table equals the maximum number of parallel instances that the software can process for a source or target created from this table.

In addition to importing partitions or creating and editing partition metadata, enable the partition settings when you configure sources and targets.

7.2.1.4.3 To enable partition settings in a source or target table

1. Drop a table into a data flow and select **Make Source** or **Make Target**.

2. Click the name of the table to open the source or target table editor.
3. Enable partitioning for the source or target:
   a. For a source table, click the **Enable Partitioning** check box.
   b. For a target table, click the **Options** tab, then click the **Enable Partitioning** check box.
4. Click **OK**.

   When the job executes, the softwarre generates parallel instances based on the partition information.

   Note:
   If you are loading to partitioned tables, a job will execute the load in parallel according to the number of partitions in the table. If you set **Enable Partitioning** to Yes and **Include in transaction** to No, the **Include in transaction** setting overrides the **Enable Partitioning** option. For example, if your job is designed to load to a partitioned table but you set **Include in transaction** to Yes and enter a value for **Transaction order**, when the job executes, the software will include the table in a transaction load and does not parallel load to the partitioned table.

7.2.1.4.4 Tip

If the underlying database does not support range partitioning and if you are aware of a natural distribution of ranges, for example using an Employee Key column in an Employee table, then you can edit the imported table metadata and define table ranges. The software would then instantiate multiple reader threads, one for each defined range, and execute them in parallel to extract the data.

Note:
Table metadata editing for partitioning is designed for source tables. If you use a partitioned table as a target, the physical table partitions in the database must match the metadata table partitions in SAP BusinessObjects Data Services. If there is a mismatch, the software will not use the partition name to load partitions. Consequently, the whole table updates.

7.2.2 Degree of parallelism

Degree of Parallelism (DOP) is a property of a data flow that defines how many times each transform defined in the data flow replicates for use on a parallel subset of data. If there are multiple transforms in a data flow, SAP BusinessObjects Data Services chains them together until it reaches a merge point.

You can run transforms in parallel by entering a number in the **Degree of Parallelism** box on a data flow's Properties window. The number is used to replicate transforms in the data flow which run as separate threads when the Job Server processes the data flow.

7.2.2.1 Degree of parallelism and transforms

The Query transform always replicates when you set DOP to a value greater than 1. SAP BusinessObjects Data Services also replicates query operations such as Order By, Group By, join, and functions such as lookup_ext.

The Table Comparison replicates when you use the Row-by-row select and Cached comparison table comparison methods.

- Map_Operation
- History_Preserving
- Pivot

There are two basic scenarios:

- DOP and a data flow with a single transform
- DOP and a data flow with multiple transforms

DOP and a data flow with a single transform

The following figures show runtime instances of a data flow with a DOP of 1, and the same data flow with a DOP of 2.

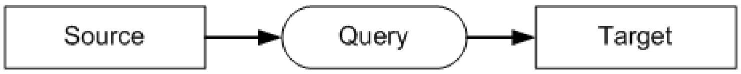


Figure 7-1: Runtime instance of a data flow where DOP =1

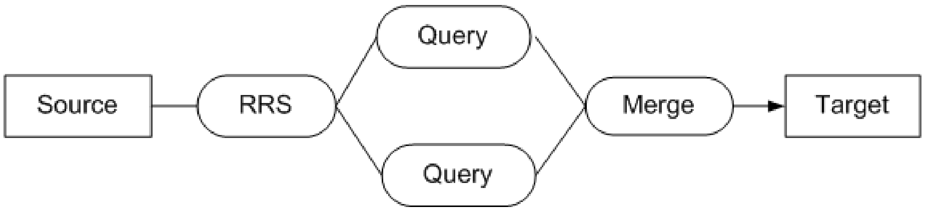


Figure 7-2: Runtime instance of a data flow where DOP = 2

With a DOP greater than 1, the software inserts an internal Round Robin Split (RRS) that transfers data to each of the replicated queries. The replicated queries execute in parallel, and the results merge into a single stream by an internal Merge transform.

DOP and a data flow with multiple transforms

The following figures show runtime instances of a data flow with a DOP of 1, and the same data flow with a DOP of 2. Notice multiple transforms in a data flow replicate and chain when the DOP is greater than 1.

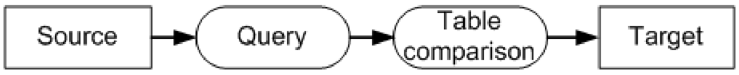


Figure 7-3: Runtime instance of a data flow where DOP =1

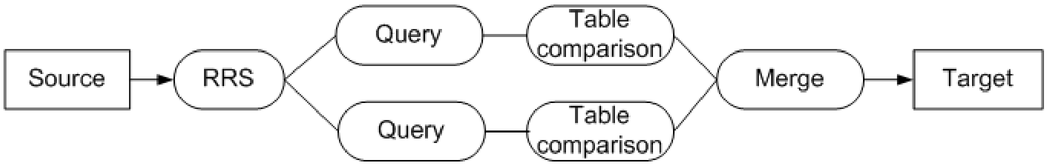


Figure 7-4: Runtime instance of a data flow where DOP = 2

When there are multiple transforms in a data flow and the DOP is greater than 1, the software carries the replicated stream as far as possible, then merges the data into a single stream.

7.2.2.2 To set the Degree of Parallelism for a data flow

The degree of parallelism (DOP) is a data flow property that acts on transforms added to the data flow.

1. In the object library, select the Data Flow tab.
2. Right-click the data flow icon and select **Properties**.
3. Enter a number in the **Degree of parallelism** option.

   The default value for degree of parallelism is 0. If you set an individual data flow's degree of parallelism to this default value, then you can control it using a Global_DOP value which affects all data flows run by a given Job Server. If you use any other value for a data flow's degree of parallelism, it overrides the Global_DOP value.

   You can use the local and global DOP options in different ways. For example:

   - If you want to globally set all data flow DOP values to 4, but one data flow is too complex and you do not want it to run in parallel, you can set the **Degree of parallelism** for this data flow locally. From the data flow's Properties window, set this data flow's **Degree of parallelism** to 1. All other data flows will replicate and run transforms in parallel after you set the Global_DOP value to 4. The default for the Global_DOP value is 2.
   - If you want to set the DOP on a case-by-case basis for each data flow, set the value for each data flow's **Degree of parallelism** to any value except zero.

   You set the Global_DOP value in the Job Server options.
4. Click **OK**.

Related Topics

• Designer Guide: Executing Jobs, Changing Job Server options

7.2.2.3 Degree of parallelism and joins

If your Query transform joins sources, DOP determines the number of times the join replicates to process a parallel subset of data.

This section describes two scenarios:
- DOP and executing a join as a single process
- DOP and executing a join as multiple processes

DOP and executing a join as a single process

The following figures show runtime instances of a data flow that contains a join with a DOP of 1 and the same data flow with a DOP of 2. You use join ranks to define the outer source and inner source. In both data flows, the inner source is cached in memory.

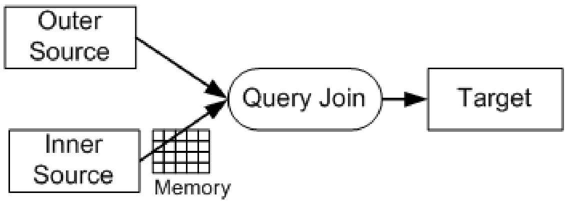


Figure 7-5: Runtime instance of a join where DOP =1

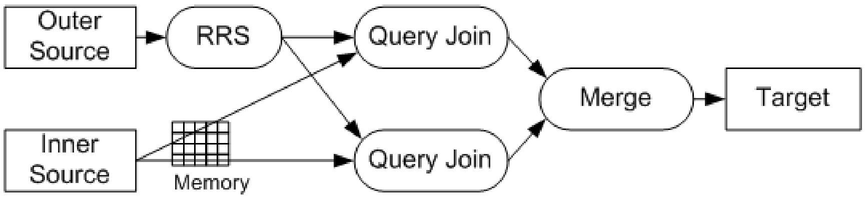


Figure 7-6: Runtime instance of a join where DOP = 2

With a DOP greater than one, the software inserts an internal Round Robin Split (RRS) that transfers data to each of the replicated joins. The inner source is cached once, and each half of the outer source joins with the cached data in the replicated joins. The replicated joins execute in parallel, and the results merge into a single stream by an internal Merge transform.

DOP and executing a join as multiple processes

When you select the **Run JOIN as a separate process** in the Query transform, you can split the execution of a join among multiple processes. the software creates a sub data flow for each separate process.

The following figure shows a runtime instance of a data flow that contains a join with a DOP of 2 and the **Run JOIN as a separate process** option selected.

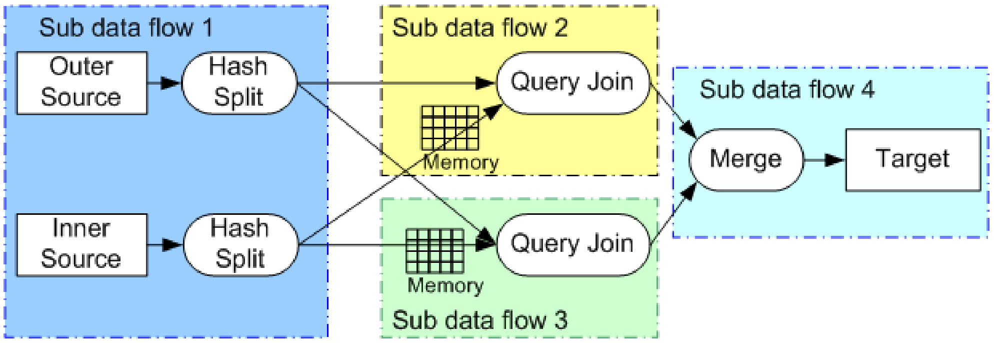


Figure 7-7: Runtime instance of a join that runs as multiple processes and DOP = 2

The data flow becomes four sub data flows (indicated by the blue dotted and dashed line in the figure):

- The first sub data flow uses an internal hash algorithm to split the data.
- The next two sub data flows are the replicated joins that run as separate processes.
- The last sub data flow merges the data and loads the target.

Tip:
If DOP is greater than one, select either **job** or **data flow** for the **Distribution level** option when you execute the job. If you execute the job with the value **sub data flow** for **Distribution level**, the Hash Split sends data to the replicated queries that might be executing on different Job Servers. Because the data is sent on the network between different Job Servers, the entire data flow might be slower.

Related Topics
- About join ordering
- Caching joins
- Using grid computing to distribute data flow execution

7.2.2.4 Degree of parallelism and functions

You can set stored procedures and custom functions to replicate with the transforms in which they are used. To specify this option, select the **Enable parallel execution** check box on the function's Properties window. If this option is not selected and you add the function to a transform, the transform will not replicate and run in parallel even if its parent data flow has a value greater than `1` set for **Degree of parallelism**.

When enabling functions to run in parallel, verify that:

- Your database will allow a stored procedure to run in parallel
- A custom function set to run in parallel will improve performance

All built-in functions, except the following, replicate if the transform they are used in replicates due to the DOP value:

• `avg()`	• `min()`
• `count()`	• `previous_row_value()`
• `count_distinct()`	• `print()`
• `double_metaphone()`	• `raise_exception()`
• `exec()`	• `raise_exception_ext()`
• `get_domain_description()`	• `set_env()`
• `gen_row_num()`	• `sleep()`
• `gen_row_num_by_group()`	• `smtp_to()`
• `is_group_changed()`	• `soundex()`
• `key_generation()`	• `sql()`
• `mail_to()`	• `sum()`
• `max()`	• `total_rows()`

7.2.2.5 To enable stored procedures to run in parallel

Use the **Enable parallel execution** option to set functions to run in parallel when the transforms in which they are used execute in parallel.

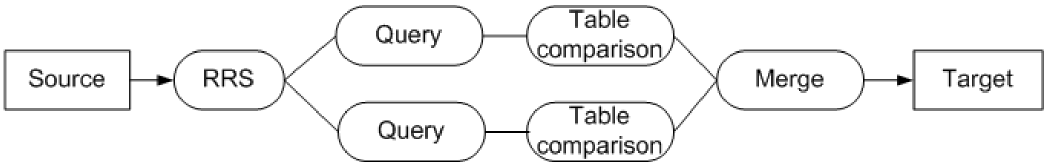
1. In the **Datastores** tab of the object library, expand a **Datastore** node.
2. Expand its **Function** node.
3. Right-click a function and select **Properties**.
4. In the Properties window, click the **Function** tab.
5. Click the **Enable Parallel Execution** check box.
6. Click **OK**.

7.2.2.5.1 To enable custom functions to run in parallel

1. In the **Custom Functions** tab of the object library, right-click a function name and select **Properties**.
2. In the Properties window, click the **Function** tab.
3. Click the **Enable Parallel Execution** check box.
4. Click **OK**.

7.2.2.6 Tips

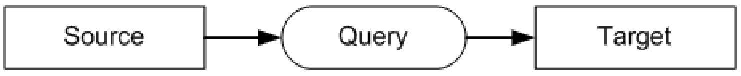
DOP can degrade performance if you do not use it judiciously. The best value to choose depends on the complexity of the flow and number of CPUs available. For example, on a computer with four CPUs, setting a DOP greater than two for the following data flow will not improve performance but can potentially degrade it due to thread contention.



If your data flow contains an Order By or a Group By that is not pushed down to the database, put them at the end of a data flow. A sort node (Order By, Group By) is always a merge point, after which the engine proceeds as if the DOP value is 1. For information on viewing the SQL statements pushed down to the database, see To view SQL.

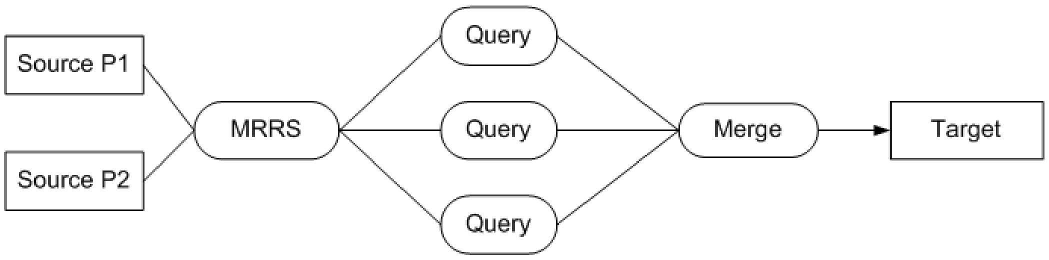
7.2.3 Combining table partitioning and a degree of parallelism

Different settings for source and target partitions and the degree of parallelism result in different behaviors in the SAP BusinessObjects Data Services engine. The sections that follow show some examples. For all the following scenarios, the data flow appears as follows:



7.2.3.1 Two source partitions and a DOP of three

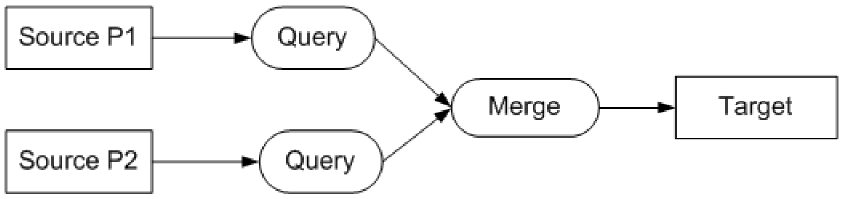
When a source has two partitions, it replicates twice. The input feeds into a merge-round-robin splitter (MRRS) that merges the input streams and splits them into a number equal to the value for DOP (in this case, three outputs to the query transform). The stream then merges and feeds into the target.



Tip:
If the target is not partitioned, set the **Number of loaders** option equal to the DOP value. Depending on the number of CPUs available, set the DOP value equal to the number of source partitions as a general rule. This produces a data flow without the Merge Round Robin Split and each partition pipes the data directly into the consuming transform.

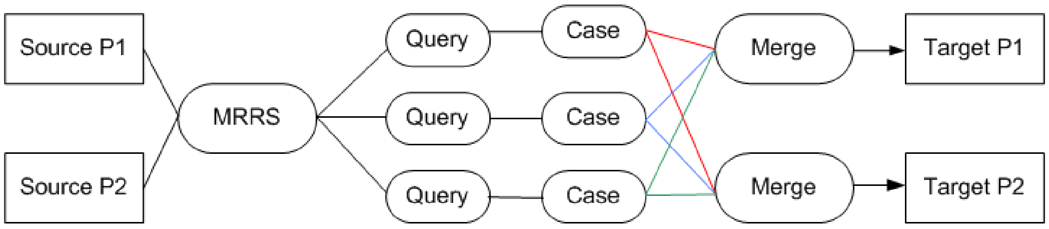
7.2.3.2 Two source partitions and a DOP of two

When the number of source partitions is the same as the value for DOP, the engine merges before the target (or before any operation that requires a merge, such as aggregation operations) and proceeds in a single stream to complete the flow.



7.2.3.3 Two source partitions, DOP of three, two target partitions

When the number of source partitions is less then the value for DOP, the input feeds into a merge-round-robin splitter (MRRS) that merges the input streams and splits them into a number equal to the value for DOP. The engine then merges the data before the target to equal the number of target partitions, then proceeds to complete the flow.



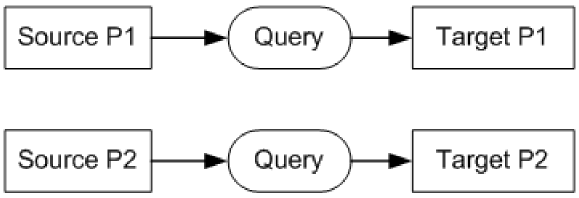
Tip:
If the number of target partitions is not equal to the number of source partitions, set the **Number of loaders** option equal to the DOP value and do not enable partitioning for the target. Depending on the number of CPUs available, set the DOP value equal to the number of source partitions as a general rule. This produces a data flow without the Merge Round Robin Split and each partition pipes the data directly into the consuming transform.

7.2.3.4 Two source partitions, DOP of two, and two target partitions

The best case situation is when the following conditions exist:

- The source and target are partitioned the same way.
- The source and target have the same number of partitions.
- DOP is equal to the same number of partitions.

When a source has two partitions, it replicates twice. Because the DOP value is two, the query transform replicates twice. When a target has two partitions, it replicates twice. The following figure shows that each source partition feeds directly into a replicated query transform, and the output from each query feeds directly into a replicated target.



7.2.4 File multi-threading

You can set the number of threads used to process some sources and targets. The **Parallel process threads** option is available on the:

- File format editor
- Source file editor

- Target file editor
- Properties window of an ABAP data flow

Without multi-threading:

- With delimited file reading, the SAP BusinessObjects Data Services reads a block of data from the file system and then scans each character to determine if it is a column delimiter, a row delimiter, or a text delimiter. Then it builds a row using an internal format.
- For positional file reading, the software does not scan character by character, but it still builds a row using an internal format.
- For file loading, processing involves building a character-based row from the internal row format.

You can set these time-consuming operations to run in parallel. You can use the **Parallel process threads** option to specify how many threads to execute in parallel to process the I/O blocks.

Note:
Enabling CPU hyperthreading can negatively affect the performance of servers and is therefore not supported.

Related Topics
- Designer Guide: File Formats
- Reference Guide: Objects, Source
- Reference Guide: Objects, Target

7.2.4.1 Flat file sources

To use the **Parallel process threads** option, the following conditions must be met:

- In the file format editor:
  - For delimited files, no text delimiters are defined.

    For fixed-width files, having a text delimiter defined does not prevent the file from being read by parallel process threads.

    You can set SAP BusinessObjects Data Services to read flat file data in parallel in most cases because the majority of jobs use fixed-width or column-delimited source files that do not have text delimiters specified.
  - An end-of-file (EOF) marker for the file's input/output style is not specified.
  - The value of the row delimiter is not set to `{none}`. A row delimiter can be `{none}` only if the file is a fixed-width file.
  - If the file has a multi-byte locale and you want to take advantage of parallel process threads, set the row delimiter as follows:
    - The length of the row delimiter must be 1. If the codepage of the file is UTF-16, the length of the row delimiter can be 2.
    - The row delimiter hex value must be less than 0x40.

- In the Source File Editor, no number has been entered for **Rows to read**.

  The Rows to read option indicates the maximum number of rows that the software reads. It is normally used for debugging. Its default value is none.
- The maximum row size does not exceed 128 KB.

If a file source needs to read more than one file, for example, *.txt is specified for the **File(s)** option in the file format editor, the software processes the data in the first file before the data in the next file. It performs file multi-threading one file at a time.

7.2.4.2 Flat file targets

If you enter a positive value for **Parallel process threads**, Data Services parallel processes flat file targets when the maximum row size does not exceed 128KB.

7.2.4.3 Tuning performance

The **Parallel process threads** option is a performance enhancement for some sources and targets. Performance is defined as the total elapsed time used to read a file source.

A multi-threaded file source or target achieves high performance by maximizing the utilization of the CPUs on your Job Server computer. You will notice higher CPU usage when you use this feature. You might also notice higher memory usage because the number of process threads you set (each consisting of blocks of rows that use 128 kilobytes) reside in memory at the same time.

To tune performance, adjust the value for **Parallel process threads**. Ideally, have at least as many CPUs as process threads. For example, if you enter the value 4 for **Parallel process threads**, have at least four CPUs on your Job Server computer.

However, increasing the value for process threads does not necessarily improve performance. The file reads and loads achieve their best performance when the work load is distributed evenly among all the CPUs and the speed of the file's input/output (I/O) thread is comparable with the speed of the process threads.

The I/O thread for a file source reads data from a file and feeds it to process threads. The I/O thread for a file target takes data from process threads and loads it to a file. Therefore, if a source file's I/O thread is too slow to keep the process threads busy, there is no need to increase the number of process threads.

If there is more than one process thread on one CPU, that CPU will need to switch between the threads. There is an overhead incurred in creating these threads and switching the CPU between them.

7.2.4.4 Tips

The best value for **Parallel process threads** depends on the complexity of your data flow and the number of available processes. If your Job Server is on a computer with multiple CPUs, the values for file sources and targets should be set to at least two.

After that, experiment with different values to determine the best value for your environment.

Here are some additional guidelines:

- If **Parallel process threads** is set to `none`, then flat file reads and loads are not processed in parallel.
- If **Parallel process threads** is set to `1`, (meaning that one process thread will spawn) and your Job Server computer has one CPU, then reads and loads can occur faster than single-threaded file reads and loads because SAP BusinessObjects Data Services reads the I/O thread separately and concurrently with the process thread.
- If **Parallel process threads** is set to `4`, four process threads will spawn. You can run these threads on a single CPU. However, using four CPUs would more likely maximize the performance of flat file reads and loads.

Distributing Data Flow Execution

The previous chapter describes how SAP BusinessObjects Data Services can run a single process as multiple threads that run in parallel on a multiprocessor computer. Using Degree of Parallelism (DOP), it can execute each thread on a separate CPU on the computer.

This section describes how the software can split a process (data flow) into multiple processes (sub data flows) that can take advantage of more memory across multiple computers or on the same computer that has more than two gigabytes of memory. For example, if your computer has eight gigabytes of memory, you can have four sub data flows that each can use up to two gigabytes.

With this capability, the software can distribute CPU-intensive and memory-intensive operations (such as join, grouping, table comparison and lookups). This distribution of data flow execution provides the following potential benefits:

- Better memory management by taking advantage of more CPU power and physical memory
- Better job performance and scalability by taking advantage of grid computing

You can create sub data flows so that the software does not need to process the entire data flow in memory at one time. You can also distribute the sub data flows to different job servers within a server group to use additional memory and CPU resources.

This section contains the following topics:

- Splitting a data flow into sub data flows
- Using grid computing to distribute data flow execution

8.1 Splitting a data flow into sub data flows

8.1.1 Run as a separate process option

If your data flow contains multiple resource-intensive operations, you can run each operation as a separate process (sub data flow) that uses separate resources (memory and computer) from each other to improve performance and throughput. When you specify multiple **Run as separate process** options in objects in a data flow, SAP BusinessObjects Data Services splits the data flow into sub data flows that run in parallel.

The **Run as a separate process** option is available on resource-intensive operations that including the following:

- Hierarchy_Flattening transform
- Associate transform
- Country ID transform
- Global Address Cleanse transform
- Global Suggestion Lists transform
- Match Transform
- United States Regulatory Address Cleanse transform
- User-Defined transform
- Query operations that are CPU-intensive and memory-intensive:
  - Join
  - GROUP BY
  - ORDER BY
  - DISTINCT
- Table_Comparison transform
- Lookup_ext function
- Count_distinct function
- Search_replace function

8.1.2 Examples of multiple processes for a data flow

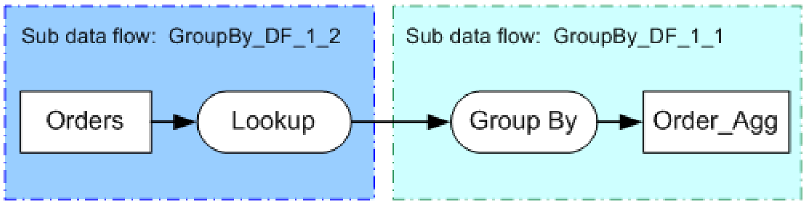
A data flow can contain multiple resource-intensive operations that each require large amounts of memory or CPU utilization. You can run each resource-intensive operation as a separate process that can use more memory on a different computer or on the same computer that has more than two gigabytes of memory. For example, you might have a data flow that sums sales amounts from a lookup table and groups the sales by country and region to find which regions are generating the most revenue. Other than the source and target, the data flow contains a Query transform for the lookup_ext function to obtains sales subtotals and another Query transform to group the results by country and region.

To define separate processes in this sample data flow, take one of the following actions:

- When you define the Lookup_ext function in the first query transform, select the **Run as a separate process** option.
- When you define the Group By operation in the second query transform, select the **Run GROUP BY as a separate process** option on the **Advanced** tab.

8.1.2.1 Scenario 1: Run multiple sub data flows with DOP set to 1

The following diagram shows how SAP BusinessObjects Data Services splits this data flow into two sub data flows when you specify the **Run as a separate process** option for either the Lookup_ext function or the Group By.

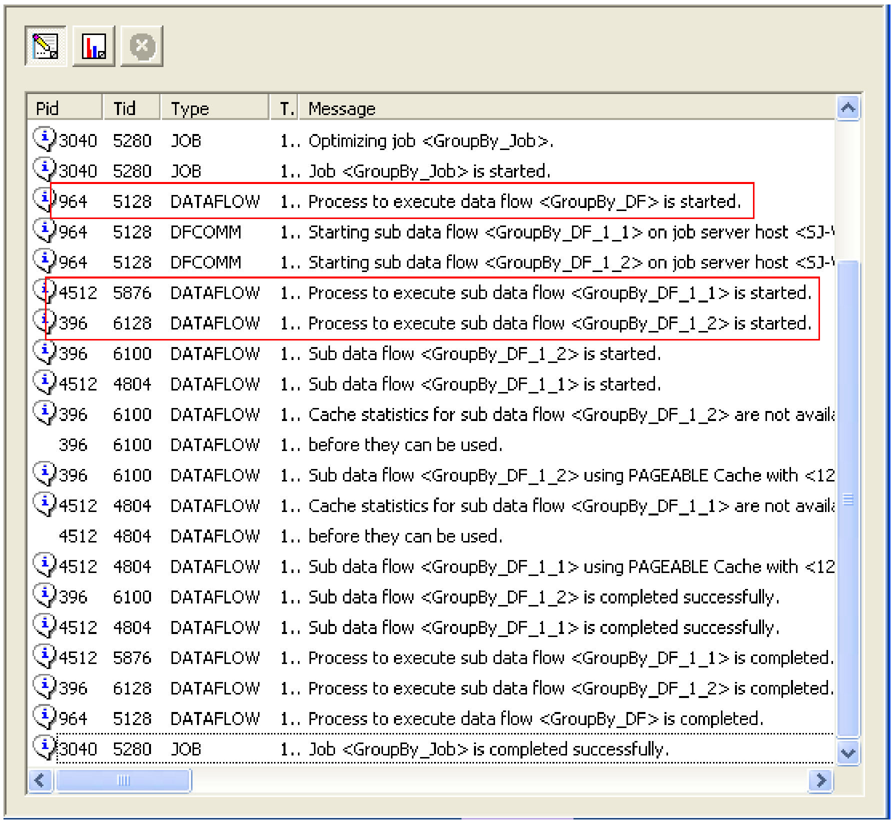


The software generates sub data flow names that follow this format:

```
DFName_executionGroupNumber_indexInExecutionGroup
```

- *`DFName`* is the name of the data flow.
- *`executionGroupNumber`* is the order that the software executes a group of sub data flows
- *`indexInExecutionGroup`* is the sub data flow within an execution group.

When you execute the job, the Trace Log shows that the software creates two sub data flows that execute in parallel and have different process IDs (Pids). For example, the following trace log shows two sub data flows GroupBy_DF_1_1 and GroupBy_DF_1_2 that each start at the same time and have a different Pid than the parent data flow GroupBy_DF.



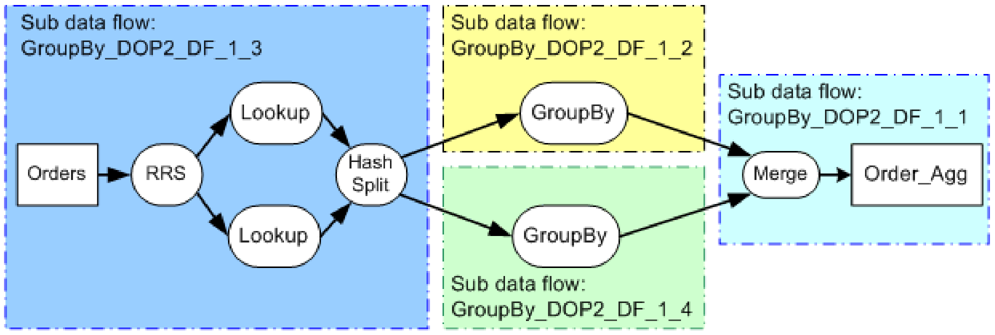
```
Pid    Tid    Type       T.  Message
3040   5280   JOB        1.. Optimizing job <GroupBy_Job>.
3040   5280   JOB        1.. Job <GroupBy_Job> is started.
964    5128   DATAFLOW   1.. Process to execute data flow <GroupBy_DF> is started.
964    5128   DFCOMM     1.. Starting sub data flow <GroupBy_DF_1_1> on job server host <SJ-\
964    5128   DFCOMM     1.. Starting sub data flow <GroupBy_DF_1_2> on job server host <SJ-\
4512   5876   DATAFLOW   1.. Process to execute sub data flow <GroupBy_DF_1_1> is started.
396    6128   DATAFLOW   1.. Process to execute sub data flow <GroupBy_DF_1_2> is started.
396    6100   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_2> is started.
4512   4804   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_1> is started.
396    6100   DATAFLOW   1.. Cache statistics for sub data flow <GroupBy_DF_1_2> are not availa
396    6100   DATAFLOW   1.. before they can be used.
396    6100   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_2> using PAGEABLE Cache with <12
4512   4804   DATAFLOW   1.. Cache statistics for sub data flow <GroupBy_DF_1_1> are not availa
4512   4804   DATAFLOW   1.. before they can be used.
4512   4804   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_1> using PAGEABLE Cache with <12
396    6100   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_2> is completed successfully.
4512   4804   DATAFLOW   1.. Sub data flow <GroupBy_DF_1_1> is completed successfully.
4512   5876   DATAFLOW   1.. Process to execute sub data flow <GroupBy_DF_1_1> is completed.
396    6128   DATAFLOW   1.. Process to execute sub data flow <GroupBy_DF_1_2> is completed.
964    5128   DATAFLOW   1.. Process to execute data flow <GroupBy_DF> is completed.
3040   5280   JOB        1.. Job <GroupBy_Job> is completed successfully.
```

8.1.2.2 Scenario 2: Run multiple sub data flows with DOP greater than 1

When Degree Of Parallelism (DOP) is set to a value greater than 1, each transform defined in the data flow replicates for use on a parallel subset of data.

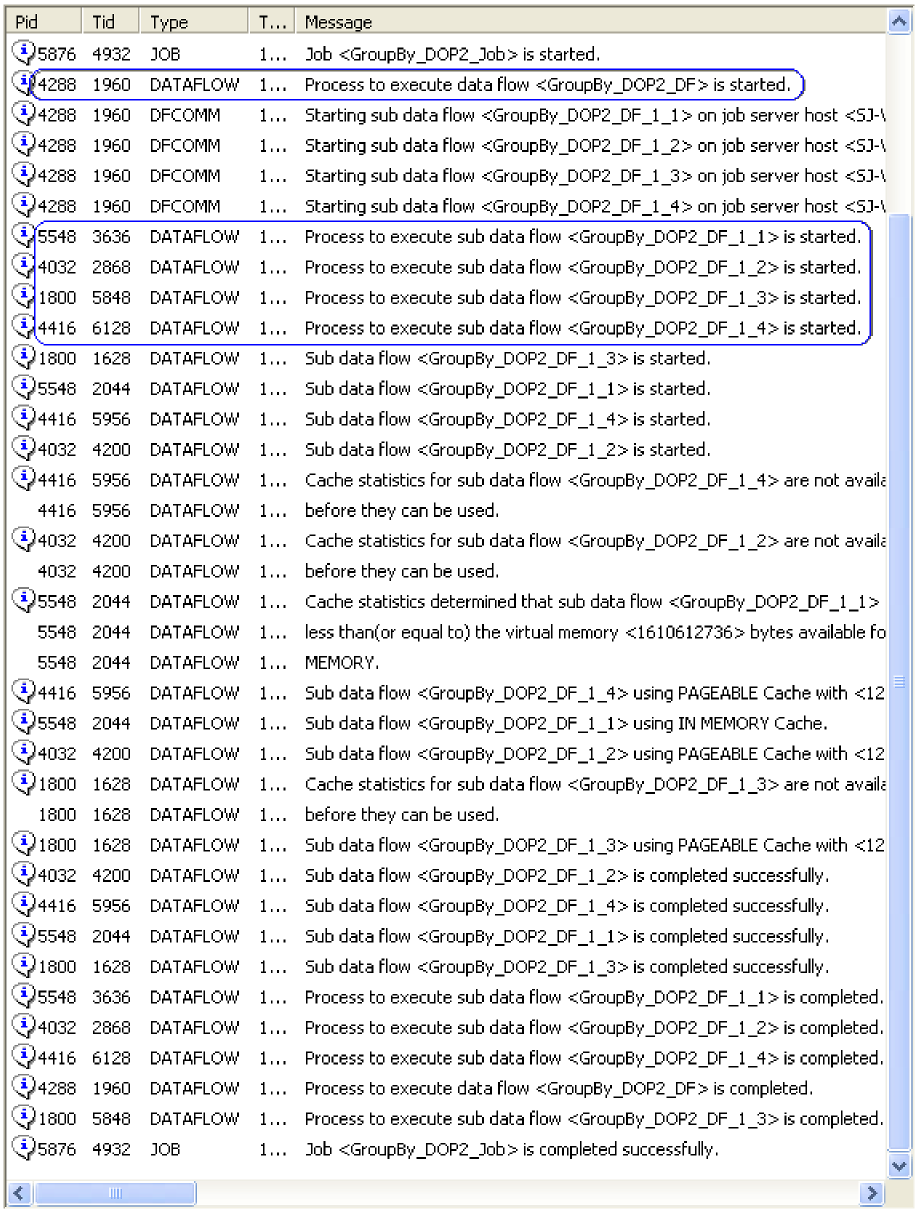
Set DOP to a value greater than 1 on the data flow Properties window.

The following diagram shows the sub data flows that Data Services generates for GroupBy_DOP2_Job when the **Run GROUP BY as a separate process** is selected and DOP set to 2.



When you execute the job, the Trace Log shows that the software creates sub data flows that execute in parallel with different process IDs (Pids). For example, the following trace log shows the following four sub data flows that start concurrently and that each have a different Pid than the parent data flow GroupBy_DOP2_DF:

- GroupBy_DOP2_DF_1_1
- GroupBy_DOP2_DF_1_2
- GroupBy_DOP2_DF_1_3
- GroupBy_DOP2_DF_1_4



```
Pid   Tid   Type      T...  Message
5876  4932  JOB       1...  Job <GroupBy_DOP2_Job> is started.
4288  1960  DATAFLOW  1...  Process to execute data flow <GroupBy_DOP2_DF> is started.
4288  1960  DFCOMM    1...  Starting sub data flow <GroupBy_DOP2_DF_1_1> on job server host <SJ-\
4288  1960  DFCOMM    1...  Starting sub data flow <GroupBy_DOP2_DF_1_2> on job server host <SJ-\
4288  1960  DFCOMM    1...  Starting sub data flow <GroupBy_DOP2_DF_1_3> on job server host <SJ-\
4288  1960  DFCOMM    1...  Starting sub data flow <GroupBy_DOP2_DF_1_4> on job server host <SJ-\
5548  3636  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_1> is started.
4032  2868  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_2> is started.
1800  5848  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_3> is started.
4416  6128  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_4> is started.
1800  1628  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_3> is started.
5548  2044  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_1> is started.
4416  5956  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_4> is started.
4032  4200  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_2> is started.
4416  5956  DATAFLOW  1...  Cache statistics for sub data flow <GroupBy_DOP2_DF_1_4> are not availa
4416  5956  DATAFLOW  1...  before they can be used.
4032  4200  DATAFLOW  1...  Cache statistics for sub data flow <GroupBy_DOP2_DF_1_2> are not availa
4032  4200  DATAFLOW  1...  before they can be used.
5548  2044  DATAFLOW  1...  Cache statistics determined that sub data flow <GroupBy_DOP2_DF_1_1>
5548  2044  DATAFLOW  1...  less than(or equal to) the virtual memory <1610612736> bytes available fo
5548  2044  DATAFLOW  1...  MEMORY.
4416  5956  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_4> using PAGEABLE Cache with <12
5548  2044  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_1> using IN MEMORY Cache.
4032  4200  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_2> using PAGEABLE Cache with <12
1800  1628  DATAFLOW  1...  Cache statistics for sub data flow <GroupBy_DOP2_DF_1_3> are not availa
1800  1628  DATAFLOW  1...  before they can be used.
1800  1628  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_3> using PAGEABLE Cache with <12
4032  4200  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_2> is completed successfully.
4416  5956  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_4> is completed successfully.
5548  2044  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_1> is completed successfully.
1800  1628  DATAFLOW  1...  Sub data flow <GroupBy_DOP2_DF_1_3> is completed successfully.
5548  3636  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_1> is completed.
4032  2868  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_2> is completed.
4416  6128  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_4> is completed.
4288  1960  DATAFLOW  1...  Process to execute data flow <GroupBy_DOP2_DF> is completed.
1800  5848  DATAFLOW  1...  Process to execute sub data flow <GroupBy_DOP2_DF_1_3> is completed.
5876  4932  JOB       1...  Job <GroupBy_DOP2_Job> is completed successfully.
```

Tip:
When your data flow has DOP is greater than one, select either `job` or `data flow` for the **Distribution level** option when you execute the job. If you execute the job with the value `sub data flow` for **Distribution level**, the Round-Robin Split or Hash Split sends data to the replicated queries that might be executing on different job servers. Because the data is sent on the network between different job servers, the entire data flow might be slower.

Related Topics

- Degree of parallelism
- Using grid computing to distribute data flow execution

8.1.3 Data_Transfer transform

The Data_Transfer transform creates transfer tables in datastores to enable the software to push down operations to the database server. The Data_Transfer transform creates two sub data flows and uses the transfer table to distribute the data from one sub data flow to the other sub data flow. The sub data flows execute serially.

Related Topics

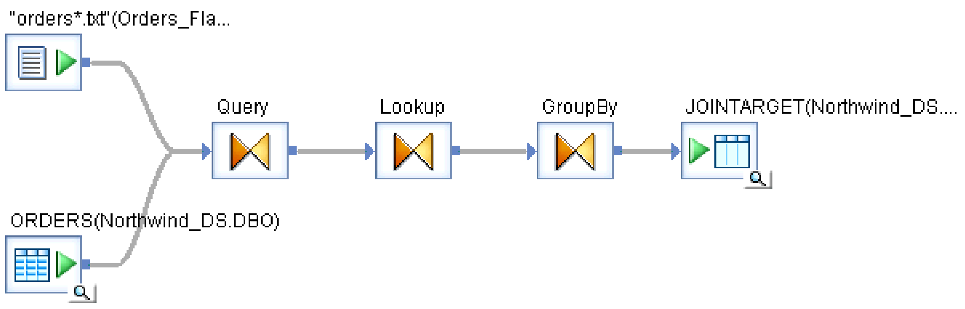
- Reference Guide: Transforms, Data_Transfer

8.1.4 Examples of multiple processes with Data_Transfer

The following are typical scenarios of when you might use the Data_Transfer transform to split a data flow into sub data flows to push down operations to the database server.

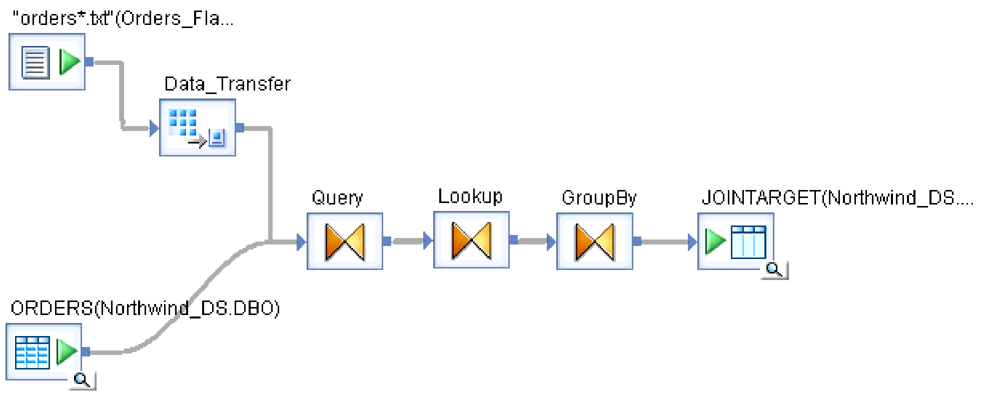
8.1.4.1 Scenario 1: Sub data flow to push down join of file and table sources

Your data flow might join an Orders flat file and a Orders table, perform a lookup_ext function to obtain sales subtotals, and another Query transform to group the results by country and region.



8.1.4.1.1 To define sub data flows to push down a join of a file and table

1. Add a Data_Transfer transform between the Orders file source and the Query transform.



2. Select the value `Table` from the drop-down list in the **Transfer type** option in the Data_Transfer editor.
3. For **Table name** in the Table options area, browse to the datastore that contains the source table that the Query joins to this file. Double-click the datastore name and enter a name for the transfer table on the Input table for Data_Transfer window.

   In this example, browse to the same datastore that contains the Orders table and enter Orders_FromFile in Table name.

4. After you save the data flow and click **ValidationDisplay Optimized SQL...**, the Optimized SQL window shows that the join between the transfer table and source Orders table is pushed down to the database.

```
SELECT "Data_Transfer_Orders_Flatfile"."PRODUCTID" , "ORDERS"."SHIPCOUNTRY" , "ORDERS"."SHIPREGION" ,
"Data_Transfer_Orders_Flatfile"."ORDERID"
FROM "DBO"."ORDERS_FROMFILE" "Data_Transfer_Orders_Flatfile","DBO"."ORDERS""ORDERS"
WHERE ("Data_Transfer_Orders_Flatfile"."ORDERID" = "ORDERS"."ORDERID")
```

   SAP BusinessObjects Data Services can push down many operations without using the Data_Transfer transform.

5. When you execute the job, the Trace Log shows messages that indicate that the software created two sub data flows with different Pids to run the different operations serially.

Related Topics
• Push-down operations

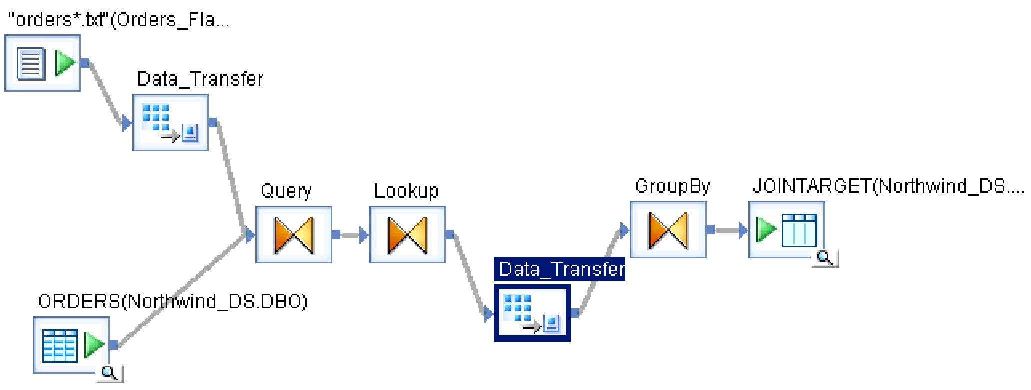
8.1.4.2 Scenario 2: Sub data flow to push down memory-intensive operations

You can use the Data_Transfer transform to push down memory-intensive operations such as Group By or Order By.

For the sample data flow in Scenario 1: Sub data flow to push down join of file and table sources, you might want to push down the Group By operation.

8.1.4.2.1 To define sub data flows to push down another operation

1. Add a Data_Transfer transform between the Lookup and GroupBy query transforms, as the following diagram shows.



2. Select the value `Table` from the drop-down list in the **Transfer type** option in the Data_Transfer editor.
3. For **Table name** in the Table options area, browse to the datastore that contains the target table. Double-click the datastore name and enter a name for the transfer table on the Input table for Da ta_Transfer window.
4. After you save the data flow and click **Validation** > **Display Optimized SQL**, the Optimized SQL window shows that the software pushes the GroupBy down to the target database.

```
INSERT INTO "DBO"."JOINTARGET"("PRODUCTID","SHIPCOUNTRY","SHIPREGION","SALES")

SELECT "Data_Transfer_1_Lookup"."PRODUCTID", "Data_Transform_1_Lookup"."SHIPCOUNTRY", "Data_Trans
fer_1_Lookup"."SHIPREGION",sum("Data_Transfer_1_Lookup"."SALES")
FROM "DBO"."GROUPTRANS""Data_Transfer_1_Lookup"
GROUP BY "Data_Transfer_1_Lookup"."PRODUCTID","Data_Transfer_1_Lookup"."SHIPCOUNTRY", "Data_Trans
fer_1_Lookup"."SHIPREGION"
```

   The software can push down many operations without using the Data_Transfer transform.

5. When you execute the job, the messages indicate that the software creates three sub data flows to run the different operations serially.

Related Topics

• Push-down operations

8.2 Using grid computing to distribute data flow execution

SAP BusinessObjects Data Services takes advantage of grid computing when you:

- Define a group of Job Servers (called a Server Group) that acts as a server grid. The software leverages available CPU and memory on the computers where the Job Servers execute.
- Specify Distribution levels for data flow execution to process smaller data sets or fewer transforms on different Job Servers in a Server Group. Each data flow or sub data flow consumes less virtual memory.

8.2.1 Server Group

You can distribute the execution of a job or a part of a job across multiple Job Servers within a Server Group to better balance resource-intensive operations. A server group automatically measures resource availability on each Job Server in the group and distributes scheduled batch jobs to the Job Server with the lightest load at runtime.

Related Topics

- Management Console Guide: Server Groups

8.2.2 Distribution levels for data flow execution

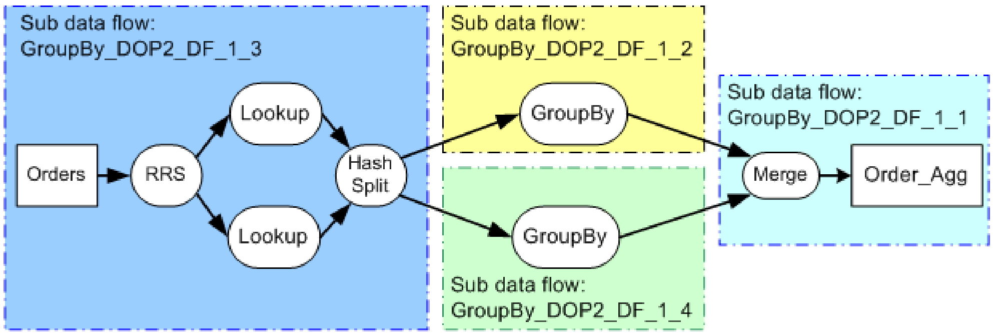
When you execute a job, you can specify the following values on the **Distribution level** option:

- Job level - An entire job can execute on an available Job Server.
- Data flow level - Each data flow within a job can execute on an available Job Server and can take advantage of additional memory (up to two gigabytes) for both in-memory and pageable cache on another computer.
- Sub data flow level - A resource-intensive operation (such as a sort, table comparison, or table lookup) within a data flow can execute on an available Job Server. Each operation can take advantage of up to two gigabytes additional memory for both in-memory and pageable cache on another computer.

8.2.2.1 Job level

When you choose a Server Group to execute your job, the default distribution level is Job.

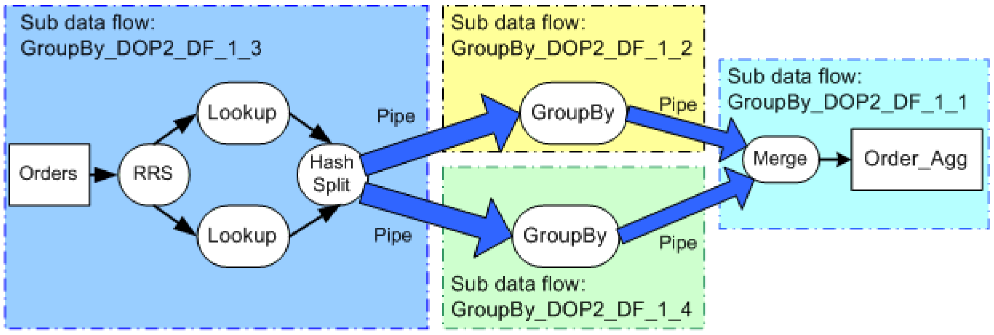
When **Distribution level** has the value `Job`, all of the processes that belong to the job execute on the same computer. For example, section Scenario 2: Run multiple sub data flows with DOP greater than 1 describes the data flow GroupBy_DOP2_DF which is designed to generate four sub data flows as follows.



When you execute the job, the following Trace log messages indicate the distribution level for each sub data flow:

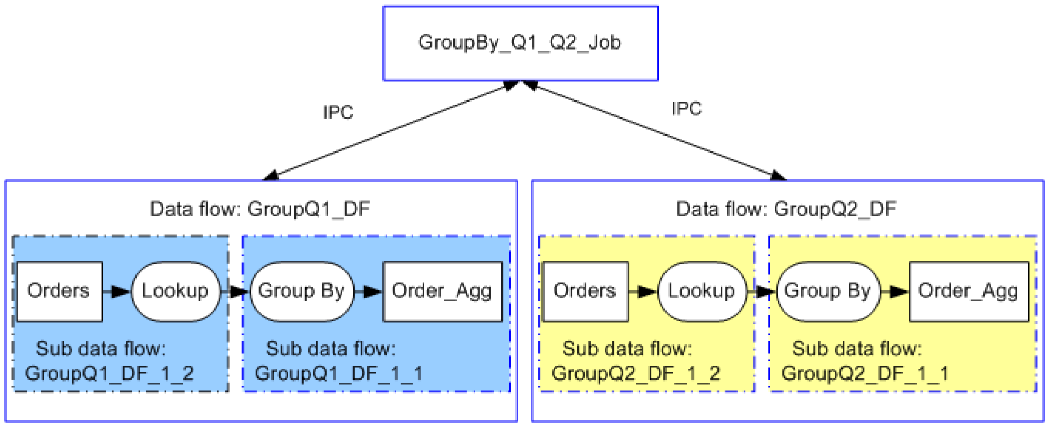
```
Starting sub data flow <GroupBy_DOP2_DF_1_1> on job server host <SJ-C>, port <3502>. Distribution level
<Job>.
Starting sub data flow <GroupBy_DOP2_DF_1_2> on job server host <SJ-C>, port <3502>. Distribution level
<Job>.
Starting sub data flow <GroupBy_DOP2_DF_1_3> on job server host <SJ-C>, port <3502>. Distribution level
<Job>.
Starting sub data flow <GroupBy_DOP2_DF_1_4> on job server host <SJ-C>, port <3502>. Distribution level
<Job>.
```

When **Distribution level** is `Job`, the software uses named pipes to send data between the sub data flow processes on the same computer, as the following diagram indicates with the blue arrows.



8.2.2.2 Data flow level

When **Distribution level** has the value `Data flow`, all of the processes that belong to each data flow can execute on a different computer. For example, the following GroupBy_Q1_Q2_Job has two data flows: GroupQ1_DF and GroupQ1_DF that process orders for the first quarter and second quarter, respectively.



- The solid blue lines enclose each process that can execute on a separate Job Server. In this example, each data flow can execute on a different computer than the computer where the job started.
- SAP BusinessObjects Data Services uses Inter-Process Communications (IPC) to send data between the job and data flows on the different computers. IPC uses the peer-to-peer port numbers specified on the **Start port** and **End port** options in the Server Manager.

Note:
The default values for **Start port** and **End port** are `1025` and `32767`, respectively. Change these values if you want to restrict the number of ports or if some of the ports are already in use.

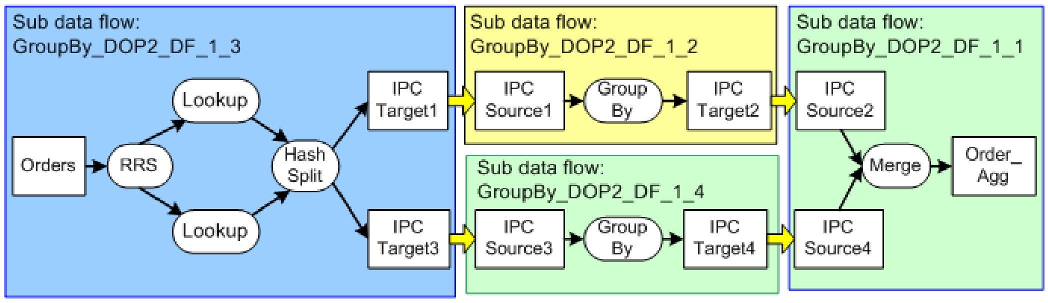
When you execute the job, the Trace log displays messages such as the following that indicate the communication port for the data flow and the distribution level for each data flow. All of the sub data flows within a data flow run on the same computer.

```
Data flow communication using peer-to-peer method with the port range <1025> to <32767>.
...
Peer-to-peer connection server for session process is listening at host <SJ-C>, port <1025>.
Job <GroupBy_Q1_Q2_Job> is started.
Starting data flow </GroupBy_Q1_Q2_Job/GroupBy_Q1_DF> on job server host <SJ-C>, port <3502>. Distribution
level <Data
flow>. Data flow submitted to server group <sg_direpo>. Load balancing algorithm <Least load>. Server group
 load statistics
from job server <mssql_lap_js SJ-C 3502>:
<mssql_lap_js SJ-C 3502> System Load <47%> Number of CPUs <1>
<MSSQL2005_JS SJ-W-C 3500> System Load <70%> Number of CPUs <2>
Process to execute data flow <GroupBy_Q1_DF> is started.
Starting sub data flow <GroupBy_Q1_DF_1_1> on job server host <SJ-C>, port <3502>. Distribution level <Data
 flow>.
Starting sub data flow <GroupBy_Q1_DF_1_2> on job server host <SJ-C>, port <3502>. Distribution level <Data
 flow>.
Starting sub data flow <GroupBy_Q1_DF_1_3> on job server host <SJ-C>, port <3502>. Distribution level <Data
 flow>.
```

```
Starting sub data flow <GroupBy_Q1_DF_1_4> on job server host <SJ-C>, port <3502>. Distribution level <Data
 flow>.
```

8.2.2.3 Sub data flow level

When **Distribution level** has the value `Sub data flow`, each sub data flow within a data flow can execute on a different computer. In the example that section Scenario 2: Run multiple sub data flows with DOP greater than 1 describes, the GroupBy_DOP2_Job has four sub data flows as follows.



- The solid blue lines enclose each process that can execute on a separate Job Server. In this example, each sub data flow can execute on a different computer than the computer where the job started.
- The yellow arrows indicate the Inter-Process Communications (IPC) that SAP BusinessObjects Data Services uses to send data between the job and sub data flows on the different computers. IPC the peer-to-peer port numbers specified on the **Start port** and **End port** options in the Server Manager.

  The default values for Start port and End port are 1025 and 32767, respectively. Change these values if you want to restrict the number of ports or if some of the ports are already in use.

Note:
If you find that sending data across the network is causing your data flow to execute longer, you might want to change **Distribution level** from `Sub data flow` to `Data flow` or `Job`.

When you execute the job, the Trace log displays messages such as the following that indicate that the software selects a job server for each sub data flow based on the system load on each computer:

```
Starting sub data flow <GroupBy_DOP2_DF_1_1> on job server host <SJ-C>, port <3502>. Distribution level <Sub
 data flow>. Sub data
flow submitted to server group <sg_direpo>. Load balancing algorithm <Least load>. Server group load
statistics
from job server <mssql_lap_js SJ-C 3502>:
<mssql_lap_js SJ-C 3502> System Load <21%> Number of CPUs <1>
<MSSQL2005_JS SJ-W-C 3500> System Load <70> Number of CPUs <1>
Starting sub data flow <GroupBy_DOP2_DF_1_2> on job server host <SJ-C>, port <3502>. Distribution level <Sub
 data
flow>. Sub data flow submitted to server group <sg_direpo>. Load balancing algorithm <Least load>. Server
group load statistics
from job server <mssql_lap_js SJ-C 3502>:
<mssql_lap_js SJ-C 3502> System Load <21%> Number of CPUs <1>
<MSSQL2005_JS SJ-W-C 3500> System Load <70> Number of CPUs <2>
```

The following messages show the communication port that each sub data flow uses:

```
Peer-to-peer connection server for sub data flow <GroupBy_DOP2_DF_1_1> is listening at host <SJ-C>, port
<1027>.
```

```
Process to execute sub data flow <GroupBy_DOP2_DF_1_4> is started.
Peer-to-peer connection server for sub data flow <GroupBy_DOP2_DF_1_2> is listening at host <SJ-C>, port
<1028>.
Peer-to-peer connection server for sub data flow <GroupBy_DOP2_DF_1_3> is listening at host <SJ-C>, port
<1029>.
Peer-to-peer connection server for sub data flow <GroupBy_DOP2_DF_1_4> is listening at host <SJ-C>, port
<1030>.
```

Bulk Loading and Reading

SAP BusinessObjects Data Services supports capabilities present in most supported databases that enable you to load and in some cases read data in bulk rather than using SQL statements. Some general considerations when using bulk loading and reading are:

- Specify bulk-loading options on the Data Services target table editor on the **Options** and **Bulk Loader Options** tabs.
- Specify Teradata reading options on the source table editor **Teradata options** tab.
- Most databases do not support bulk loading with a template table.

For details on the options for each database type, see the *Reference Guide*.

9.1 Bulk loading in DB2 Universal Database

SAP BusinessObjects Data Services supports bulk loading to the DB2 Universal Database.

9.1.1 When to use each DB2 bulk-loading method

SAP BusinessObjects Data Services supports multiple bulk-loading methods for DB2 Universal Database (UDB) on Windows and UNIX. The following table lists the methods that you can select depending on your requirements.

Note:
You cannot bulk load data to DB2 databases that run on AS/400 or z/OS (MVS) systems.

Load method	Description	Advantages	Restrictions
CLI Load	Loads a large volume of data at high speed by passing it directly from memory to the table on the DB2 UDB server.	• Provides the fastest way to bulk load. • Eliminates some parameters because no intermediate data file is required. • Can put rows that violate the unique key constraint into an exception table.	• Must specify **Recoverable** and **Copy target directory** options to enable recovery because DB2 logging is not enabled for CLI Load. • The DB2 UDB server and client must be Version 8.0 or later. • Stops loading when it encounters the first rejected row.
Load	Loads a large volume of data by writing to a data file that it passes to the DB2 UDB server.	• This method is faster than the import utility. • Puts rejected rows into a "dump" file. • Can put rows that violate unique key constraint into an exception table.	• Must have disk space for intermediate data file. • Must specify **Recoverable** and **Copy target directory** options to enable recovery because DB2 logging is not enabled for DB2 Load. • The DB2 UDB server and client must be Version 8.0 or later.
Import	Loads a large volume of data by using a SQL INSERT statement to write data from an input file into a table or view.	• Recovery is enabled automatically because DB2 logging occurs during import. • Performs referential integrity or table constraint checking in addition to unique key constraint checking.	• Because DB2 logs each INSERT statement, this method is the slowest way to bulk load data. • The Data Services Job Server and DB2 UDB server must be on the same computer.

9.1.2 Using the DB2 CLI load method

The DB2 Call Level Interface (CLI) load method performs faster than the bulk load or import utilities because it does not write the data to an intermediate file. Instead, the CLI load method writes the data from memory (where SAP BusinessObjects Data Services extracted or transformed the data) directly to the table on the DB2 server.

9.1.2.1 To configure your system to use the CLI load method

1. Enter the appropriate information in the datastore editor, on the **DB2 Properties** tab.

   Fields include:

   - **Bulk loader user name**: The user name SAP BusinessObjects Data Services uses when loading data with the CLI load option. For bulk loading, you might specify a different user name, for example one with import and load permissions.
   - **Bulk loader password**: The password SAP BusinessObjects Data Services uses when loading with the CLI load option.

2. To use a different bulk loader working directory than the default (`<DS_COMMON_DIR>\log\bulk loader`), specify the directory name in the datastore editor on the **Connections** tab.

9.1.2.2 To use the CLI load method in a job

1. Open the DB2 target table editor in the Designer workspace.
2. Select the **Bulk Loader Options** tab below the table schema area.
3. In the **Bulk loader** list, select **CLI load**.

   The window updates to show all CLI load options. CLI load options include these existing bulk load options:

   - Mode
   - Warning row count
   - Exception table name
   - Recoverable
   - Copy target directory

Additional or changed CLI load options include:

- **Maximum bind array**: Defines the maximum number of rows extracted or transformed before the software sends the data to the DB2 table or view. If you do not enter a value, Data Services uses the CLI load default value of 10000 rows.
- **Clean up bulk loader directory after load**: If you select this option, the software deletes the message file when the CLI load completes successfully. Because the CLI load obtains the data from memory, Data Services creates no control or data files.

Related Topics

- Reference Guide: Objects, Target tables

9.1.3 Using the DB2 bulk load utility

The DB2 load utility performs faster than the import utility because it writes data directly into the data file.

9.1.3.1 To configure your system to use the load utility

1. Connect to the DB2 Version 8.x target database that uses the following:
   - For all platforms, run:

   ```
   bind <LINK_DIR>/bin/db2bulkload.bnd blocking all grant public
   ```

2. Determine how Data Services will transmit data files to DB2. Depending on your configuration, there are different ways to transmit data files.

   For example, if your Data Services Job Server and the DB2 server reside on different computers, you can choose one of the following methods:
   - FTP: Data Services generates the data file and uses FTP to send the file to the DB2 server. To use the FTP option, you must define the FTP host name, user name, and password in the DB2 datastore you create in the Designer.
   - Data file on DB2 client computer: SAP BusinessObjects Data Services writes to the data file on the DB2 client computer, and the DB2 client transfers the data directly to the DB2 server during the load process. To use this option, you must select **Data file on client machine** on the **Bulk Loader Options** tab when you define the target in your job.

   The following matrix outlines supported data file transmission methods.

Configuration	Data file transmission methods
Data Services and DB2 server on same computer	Automatic data transmission
Data Services and DB2 server on different computer	FTP or data file on DB2 client computer.

3. If Data Services and DB2 are on different computers, you must provide a working directory for Data Services on the DB2 server. Data Services instructs the DB2 load process to generate the file for rejected rows on the DB2 server at this location.
4. Enter the appropriate information in the datastore editor, on the **DB2 Properties** tab. Fields include:

- **Bulk loader user name**—The user name the software uses when loading data with the bulk loader option. For bulk loading, you might specify a different user name—one who has import and load permissions, for example.
- **Bulk loader password**—The password the software uses when loading with the bulk loader option.
- **FTP host name**—If this field is left blank or contains the name of the computer where the Job Server resides, the software assumes that DB2 and SAP BusinessObjects Data Services share the same computer and that FTP is unnecessary. When FTP is unnecessary, all other FTP-related fields can remain blank.
- **FTP user name**—Must be defined to use FTP.
- **FTP password**—Must be defined to use FTP.
- **Server working directory**—The working directory for the load utility on the computer that runs the DB2 server. You must complete this field whenever the DB2 server and the Job Server run on separate computers. Data Services instructs the DB2 load process to generate the file for rejected rows on the DB2 server at this location.

5. If Data Services will use FTP and the DB2 server runs on Windows NT, verify connectivity with the FTP server.

   If your Job Server runs on Windows NT:

   - Connect to the FTP server using the command:

     ```
     ftp <ServerName>
     ```

   - Type the command:

     put `<`*`DS_COMMON_DIR`*`>\conf\DSConfig.txt<`*`Server working directory`*`>\conf\DSConfig.txt`

     where `<`*`DS_COMMON_DIR`*`>` is the common SAP BusinessObjects Data Services program data directory and `<`*`Server working directory`*`>` is the working directory entered on the datastore's DB2 Properties tab.

     You can only use the load utility if this command succeeds.

   If your Job Server runs on UNIX:

   - Connect to the FTP server.
   - Change directories to the **Server working directory** entered on the **DB2 Properties** tab on the datastore editor.

     For example, if the directory is `c:\temp`, type: `cd c:\temp`

     You can only use the load utility if this command succeeds.

6. To use a different bulk loader working directory than the default (`<`*`DS_COMMON_DIR`*`>\log\bulk loader`), specify the directory name in the datastore editor on the Connections tab.

9.1.3.2 To use the load utility in a job

1. Open the DB2 target table editor in the Designer workspace.
2. Select the **Bulk Loader Options** tab below the table schema area.
3. In the **Bulk loader** list, select **load**.

The window updates to show all load options. Load options include these existing import bulk loader options:

- **Generate files only**
- **Clean up bulk loader directory**
- **Text delimiter**
- **Column delimiter**

Additional load options include:

- **Mode**: Determines load mode. Valid values are:
  - **Insert**: Appends the new records into the target table
  - **Replace**: Deletes the existing records, then inserts the loaded data
- **Save count**:Determines the consistency point while loading data into tables.
- **Warning row count**: Defines the number of warnings allowed for each load operation.
- **Exception table name**: Defines the table into which the DB2 server loads rows that violate a table defined with constraints. Rows that violate those constraints are deleted from the target table and inserted into the exception table.
- **Recoverable**: Enables or disables data recovery.
  - When this option is not selected (disabled), the load operation is not recoverable because DB2 does not log the loading of data.
  - When selected (enabled), DB2 makes a copy of the loaded portion of the table. DB2 uses this copy in the roll-forward phase of database recovery. You must define the path in the **Copy target directory** for this copy.
- **Copy target directory**: Defines the directory of the copy files when you enable both the database forward log recovery and select the **Recoverable** option. SAP BusinessObjects Data Services supports only the copy files option for the DB2 CLI load method.
- **Data file on client machine**: When you select this option, the software writes to the data file on the DB2 client machine. SAP BusinessObjects Data Services does not need to FTP the data file because the DB2 client transfers the data directly to the DB2 server during the load process. To use this option:
  - You must use DB2 Version 8.x or later.
  - The target DB2 cannot be a DB2 enterprise (extended edition environment).
  - The target table and database must not be partitioned.
  - This option is only applicable if SAP BusinessObjects Data Services and DB2 are on different servers.

When you execute the DB2 bulk load utility, DB2 automatically generates the following files:

- Local message file (named `.log`) in the bulk loader working directory. DB2 writes output messages into this log file.
- "Dump" file (named `.bad`) in the DB2 server working directory. DB2 writes rejected input rows into this `.bad` file. If you clear the **Data file on client machine** option, SAP BusinessObjects Data

Services uses FTP to send the `.bad` file to the bulk loader working directory and deletes it after the load completes successfully.

Check the trace log to find either of these files.

9.1.4 Using the import utility

SAP BusinessObjects Data Services also supports bulk loading in the DB2 Universal Database 5.2 environment using the import utility. For the software to initiate DB2 bulk loading by this method directly, the Job Server and DB2 must be located on the same system. If they are not, use the following procedure to initiate bulk loading:

1. Generate a control file and data file. Check **Generate files only** in the target table editor on the Bulk Loader Options tab.
2. Manually move the control file and data file to the system where the target database is located.
3. Start the execution of the bulk loader manually.

9.2 Bulk loading in HP Neoview

SAP BusinessObjects Data Services supports bulk loading to HP Neoview via Neoview Transporter.

For detailed information about HP Neoview loading options and their behavior, see the relevant HP Neoview product documentation.

To use Neoview Transporter, you must also install the following components:
- Neoview Transporter Java Client
- Java JRE version 1.5 or newer
- Neoview JDBC Type 4 Driver
- Neoview ODBC Windows Driver (for Windows)
- Neoview UNIX Drivers (for connecting to a database on UNIX)
- Neoview Command Interface

Note:

- If you are using multibyte data on Windows, you must change the Windows regional settings to the multibyte language, for example, Japanese.
- When you install the Java Client, an environment variable called NVTHOME is created and will point to the location of the Neoview Transporter base directory. You may receive an error in SAP BusinessObjects Data Services if NVTHOME is not defined.

HP Neoview recommends that you use the bulk-loading method to load data for faster performance. The SAP BusinessObjects Data Services bulk loader for HP Neoview supports UPDATE and UPSERT as well as INSERT operations, which allows for more flexibility and performance.

SAP BusinessObjects Data Services generates a control file as input into Neoview Transporter. The control file specifies the data files and the target tables to be loaded. Being in UTF-8, the control file supports multibyte data.

By default, HP Neoview uses the SQL insert operation. For SQL update and upsert options, the control file specifies the columns used in the WHERE clause and the columns to be updated in the UPDATE clause. By default, SAP BusinessObjects Data Services uses the primary key columns in the WHERE clause.

To bulk load to a HP Neoview target table, the software:

- Creates a control file in UTF-8
- Loads data from the source into the file or named pipe in UTF-8
- Invokes Neoview Transporter

9.2.1 How Data Services and HP Neoview use the file options to load

You can choose to use either named pipes or data files as staging for loading the data. Choose from the following file options:

- Data file (for Windows and UNIX)
- Named pipe (for UNIX only)

Data file

SAP BusinessObjects Data Services runs bulk-loading jobs using a staging data file as follows:

1. The software generates staging data file(s) containing data to be loaded into a HP Neoview table.
2. The software generates a control file to be used by Neoview Transporter.

Named pipe

SAP BusinessObjects Data Services runs bulk-loading jobs using named pipes as follows:

1. The software generates a control file that Neoview Transporter uses to manipulate the database.
2. The software creates a pipe to contain the data to apply into an HP Neoview table.

   On UNIX, the pipe is a FIFO (first in, first out) file that has name of this format:

   ```
   /temp/filename.dat
   ```
3. The software invokes Neoview Transporter with the control file as input.
4. The software writes data to the pipes.

5. Neoview Transporter reads data from the pip and applies data to the HP Neoview table.

9.2.2 Using the UPSERT bulk operation

The purpose of the HP Neoview Upsert operation is to update a row, but if no row matches the update, the row is inserted.

In SAP BusinessObjects Data Services, you enable Upsert on the target table editor's **Bulk Loader Options** tab. In the **Data Services options** section, for **Bulk Operation**, select **Upsert** (the default is **Insert**) in the SQL Operation list.

After selecting **Upsert**, notice you can also enable the **Use input keys** option on the target editor's **Options** tab. The **Use input keys option** will assign the input primary keys as primary keys in the target table.

Related Topics

• Reference Guide: Objects, Target tables

9.3 Bulk loading in Informix

SAP BusinessObjects Data Services supports Informix bulk loading. For detailed information about Informix bulk-loading utility options and their behavior in the Informix DBMS environment, see the relevant Informix product documentation.

Setting up Informix for bulk-loading requires that you set the `INFORMIXDIR`, `INFORMIXSERVER`, and `PATH` environment variables.

For the software to initiate Informix bulk loading directly, the Job Server and the target database must be located on the same system.

Note:
SAP BusinessObjects Data Services provides Informix bulk-loading support only for single-byte character ASCII delimited files (not for fixed-width files).

9.3.1 To set Informix server variables

For Windows platforms, configure the environment variables in the `$LINK_DIR\bin\dbloadIfmx.bat` script.

```
set INFORMIXDIR=C:\path\to\informix\installation
set INFORMIXSERVER=ol_svr_custom
set PATH=%INFORMIXDIR%\bin;%PATH%
```

For UNIX platforms, configure the environment variables in the `$LINK_DIR/bin/dbloadIfmx.sh` script.

```
export INFORMIXDIR=/path/to/informix/installation
export INFORMIXSERVER=ol_svr_custom
export PATH=$INFORMIXDIR/bin:$PATH
```

9.4 Bulk loading in Microsoft SQL Server

SAP BusinessObjects Data Services supports Microsoft SQL Server bulk loading through the SQL Server ODBC bulk copy API. For detailed information about the SQL Server ODBC bulk copy API options and their behavior in the Microsoft SQL Server DBMS environment, see the relevant Microsoft SQL Server product documentation.

9.4.1 To use the SQL Server ODBC bulk copy API

1. From the **Tools** menu, select **Options** > **Job Server** > **General**.
2. For **Section**, enter `al_engine`.
3. For **Key**, enter `UseSQLServerBulkCopy`.
4. Select TRUE (default) or FALSE. If you leave the default, the software uses the SQL Server ODBC bulk copy API. If you set this parameter to FALSE, the software overrides the default and uses the SQLBulkOperations API.

9.4.2 Network packet size option

When loading to SQL Server, the client caches rows until it either fills a network packet or reaches the commit size (regardless of whether the packet is full). Then the client sends the packet to the server. You can affect performance by tuning commit size and network packet size. You can change these sizes on the Bulk Loader Options tab for SQL Server:

- Rows per commit

  This option lets you specify the number of rows to put in the cache before issuing a commit.

- Network packet size

  This option lets you specify network packet size in kilobytes. The default packet size is 4 kilobytes.

Note:
It is recommended that you set the **Rows per commit** and **Network packet size** parameters to avoid sending many partially filled packets over the network and ensure that the packet size contains all rows in the commit.

9.4.3 Maximum rejects option

The **Maximum rejects** parameter (on the Bulk Loader Options page) can also affect your SQL Server bulk-loading performance. When you set **Maximum rejects** to 0, SAP BusinessObjects Data Services stops at the first error it encounters and does not cache rows in the transaction (caching rows in a transaction allows the software to process each row even if an error occurs during the transaction commit process.)

When you do not specify a value for **Maximum rejects** , the software ignores the rejected rows, logs warnings, and continues processing.

9.5 Bulk loading in Netezza

SAP BusinessObjects Data Services supports bulk loading to Netezza Performance Servers.

For detailed information about Netezza loading options and their behavior in the Netezza environment, see the relevant Netezza product documentation.

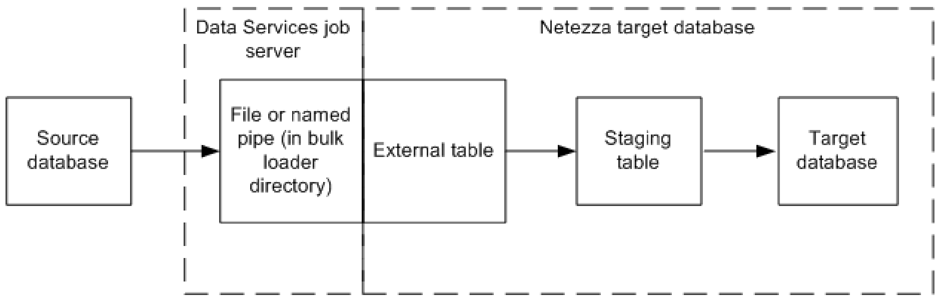
Netezza recommends using the bulk-loading method to load data for faster performance. Unlike some other bulk loaders, the SAP BusinessObjects Data Services bulk loader for Netezza supports UPDATE and DELETE as well as INSERT operations, which allows for more flexibility and performance.

9.5.1 Netezza bulk-loading process

To bulk load to a Netezza target table, SAP BusinessObjects Data Services:

- Creates an external table that is associated with a local file or named pipe
- Loads data from the source into the file or named pipe
- Loads data from the external table into a staging table by executing an INSERT statement

- Loads data from the staging table to the target table by executing a set of INSERT/UPDATE/DELETE statements



9.5.2 Options overview

From the **Bulk Loader Options** tab of your Netezza target table editor, select one of these methods depending on your Netezza environment:

- **Named pipe**— SAP BusinessObjects Data Services streams data as it is written to the named pipe through the external table to the staging table. For files that are larger than 4 GB in size, select this option for faster performance.
- **File**— SAP BusinessObjects Data Services writes the data to a file before loading through the external table to the staging table. For files that are smaller than 4 GB in size, select this option for faster performance.
- **None**— SAP BusinessObjects Data Services does not use bulk loading.

Because the bulk loader for Netezza also supports UPDATE and DELETE operations, the following options (on the target table editor **Options** tab) are also available for Netezza bulk loading.

- Column comparison
- Number of loaders
- Use input keys
- Update key columns
- Auto correct load

Related Topics

- Reference Guide: Objects, Target tables

9.5.3 Configuring bulk loading for Netezza

First configure the bulk loader and log directories in the datastore editor, then enable and configure bulk loading in the target table editor:

1. In the Netezza datastore editor, click **Advanced**.
2. Click in the field to the right of **Bulk loader directory** and type the directory path or click **Browse** to where the software should write SQL and data files for bulk loading.
3. In the **FTP** category, enter the FTP host name, login user name, login password, and host working directory.

   These options are used to transfer the Netezza nzlog and nzbad files.

   Note:
   If this datastore is not being used specifically for Netezza bulk loading, the software ignores any FTP option entries.

4. If you are loading non-ASCII character data, set the **Code page** to **latin-9**.

   If you are loading mulitbyte data, set the **Code page** to **utf-8**.

5. Click **OK** or **Apply**.
6. Open the data flow and open the target table editor by clicking its name.
7. On the **Bulk Loader Options** tab, select a bulk-loading method and configure the remaining options there and on the **Options** tab.
8. Save the data flow.

Related Topics

- Reference Guide: Objects, Database datastores (ODBC)
- Designer Guide: Datastores, Defining a database datastore

9.5.4 Netezza log files

When writing from the external table to the staging table, Netezza generates logs (nzlog and nzbad files) and writes them to a database server working directory. You can use these logs to troubleshoot your jobs. (If you do not enter a **Database server working directory** in the datastore editor, Netezza uses the temp directory on its server, /tmp, to store the nzlog and nzbad files.)

For SAP BusinessObjects Data Services to access and manage these logs, configure the FTP parameters in the datastore editor. After a load, trhe software copies these files from the specified Netezza **Database server working directory** to the specified **Bulk loader directory** and deletes them from the Netezza server.

For successful loads, SAP BusinessObjects Data Services then deletes these log files from the **Bulk loader directory** (assuming the **Clean up bulk loader directory after load** option is checked in the target table editor).

For failed loads, the software does not delete the log files from the **Bulk loader directory** even if the **Clean up bulk loader directory after load** option is checked in the target table editor.

9.6 Bulk loading in Oracle

SAP BusinessObjects Data Services supports Oracle bulk loading.

9.6.1 Bulk-loading methods

You can bulk load to Oracle using an API or a staging file:

- If you select the **API** method, SAP BusinessObjects Data Services accesses the direct path engine of Oracle's database server associated with the target table and connected to the target database. Using Oracle's Direct-Path Load API, input data feeds directly into database files. To use this option, you must have Oracle version 8.1 or later.
- If you select the **File** method, Data Services writes an intermediate staging file, control file, and log files to the local disk and invokes the Oracle SQL*Loader. This method requires more processing time than the API method.

  For detailed information about the Oracle SQL*Loader options, see the relevant Oracle product documentation.

9.6.2 Bulk-loading modes

Bulk loading in Oracle supports two modes of data loading: conventional-path and direct-path. Conventional-path loading is implicit for the **File** option if you do not select **Direct-path** on the **Bulk LoaderOptions** tab in the target table editor. SAP BusinessObjects Data Services always uses direct-path loading for the **API** option.

- Conventional-path loading

  Conventional-path loads use the SQL INSERT statements to load data to tables.
- Direct-path loading

  Direct-path loads use multiple buffers for a number of formatted blocks that load data directly to database files associated with tables.

9.6.3 Bulk-loading parallel-execution options

Parallel-execution options for bulk loading are on the **Options** tab.

For the API method, you can choose to select the **Enable partitioning** check box. If selected, SAP BusinessObjects Data Services generates the number of target parallel instances based on the number of partitions in the target table. If not selected or if your table target is not partitioned, Data Services uses one loader by default.

For the File method, enter a value in the **Number of loaders** box or select the **Enable partitioning** check box.

Note:
The **Enable partitioning** check box does not appear on the **Options** tab if the target table is not partitioned.

9.6.4 Bulk-loading scenarios

With two bulk-loading methods, two load modes, and two parallel load options, there are several scenarios you can configure:

Scenario	Method	Load Mode	Parallel Load Options
1	API	Direct-path	Enable partitions is not selected (One loader is used by default)
2	API	Direct-path	Enable partitions is selected
3	File	Direct-path	Number of loaders = 1
4	File	Direct-path	Number of loaders > 1
5	File	Direct-path	Enable partitions is selected
6	File	Conventional	Number of loaders = 1

Scenario	Method	Load Mode	Parallel Load Options
7	File	Conventional	Number of loaders > 1
8	File	Conventional	Enable partitions is selected

Here are some tips for using these scenarios:

- The API method always uses the direct-path load type, and when it is used with a partitioned target table, SAP BusinessObjects Data Services processes loads in parallel. The software instantiates multiple loaders based on the number of partitions in a table. Each loader receives rows that meet the conditions specified by the partition.
- With the File method, direct-path is faster than conventional load, but the File method is slower than using an API because of the need to generate a staging file, logs, and invoke Oracle's SQL*Loader.
- With the File method, when you use a value of greater than one for either the **Number of Loaders** or the **Enable partitioning** option, loads cannot truly run in parallel. The creation of a staging file and log for each loader is serialized.

9.6.5 Using bulk-loading options

As seen in the table on the previous page, there are many ways to set up bulk loading for an Oracle database. The following sections describe two scenarios in detail.

9.6.5.1 Direct-path loads using Number of Loaders and File method

In the **Options** tab of the target table editor, when you enter a value for **Number of loaders**, SAP BusinessObjects Data Services instantiates multiple loaders. Each loader receives rows equal to the amount specified in the **Rows per commit** box on the **Bulk Loader Options** tab. The loaders pipe rows to a staging file, then call the SQL*Loader to load the staging file contents into the table.

This process occurs in "round-robin" fashion. For example, if you set **Rows per commit** to `5000` and **Number of loaders** to `2`, then the first loader receives 5000 rows, writes them to the staging file, and then invokes the SQL*Loader to load the data into the table.

Meanwhile, the second loader receives the second batch of 5000 rows, writes them to a staging file, and then waits for the first loader to complete the loading. When the first loader completes the bulk load, the second loader starts, and while the second loader is loading, the first loader receives the third batch of 5000 rows. This process continues until all the data loads.

The SQL*Loader uses a control file to read staging files and load data. The software either creates this control file at runtime or uses one that is specified on the **Bulk Loader Options** tab at design time.

For parallel loading, the generated control files, data files, and log files are named as follows:

```
TableNameTIDPID_LDNUM_BATCHNUM
```

Where:

`TableName`: The name of the table into which data loads.

`TID`: The thread ID.

`PID`: The process ID.

`LDNUM`: The loader number, which ranges from `0` to number of loaders minus 1. For single loaders, `LDNUM` is always `0`.

`BATCHNUM`: The batch number the loader is processing. For single loaders the `BATCHNUM` is always `0`.

Note:
Product performance during this type of parallel loading depends on a number of factors such as distribution of incoming data and underlying DBMS capabilities. Under some circumstances it is possible that specifying parallel loaders can be detrimental to performance. Always test the parallel loading process before moving to production.

9.6.5.2 Direct-path loads using partitioned tables and API method

You can import partitioned tables as SAP BusinessObjects Data Services metadata.

In the **Options** tab of the target table editor, when you select **Enable partitioning**, the software instantiates multiple loaders based on the number of partitions in a table. Each loader receives rows that meet the conditions specified by the partition. In addition, commits occur based on the number specified in the **Rows per commit** box on the **Bulk Loader Options** tab.

For example:

- If you **Rows per commit** to 5000, the number of partitions is set `2`, and your first partition includes 2500 rows, then the first loader commits after receiving all possible rows (2500) while concurrently processing the second loader.
- If you **Rows per commit** to 5000, the number of partitions is set `2`, and your first partition includes 10,000 rows, then the first loader commits twice. Once after receiving 5000 rows and again after receiving the second batch of 5000 rows. Meanwhile, the second loader is processing its rows.

The loaders pipe rows directly to Oracle database files by using Oracle direct-path load APIs (installed with the Oracle client) that are associated with the target database.

The API method allows the software to bypass the use of the SQL* Loader (and the control and staging files it needs). In addition, by using table partitioning, bulk loaders can pass data to different partitions in the same target table at the same time. Using the API method with partitioned tables fully optimizes performance.

Note:
If you plan to use a partitioned table as a target, the physical table partitions in the database must match the metadata table partitions in SAP BusinessObjects Data Services. If there is a mismatch, Data Services will not use the partition name to load partitions, which impacts processing time.

For the API method, the software records and displays error and trace logs as it does for any job. A monitor log records connection activity between components; however, it does not record activity while the API is handling the data.

9.7 Bulk loading in SAP HANA

SAP BusinessObjects Data Services supports bulk loading to the SAP HANA database.

For improved performance when using changed-data capture or auto correct load, Data Services uses a temporary staging table to load the target table. Data Services first loads the data to the staging table, then it applies the operation codes (INSERT, UPDATE, and DELETE) to update the target table. With the **Bulk load** option selected in the target table editor, any one of the following conditions triggers the staging mechanism:

- The data flow contains a Map_CDC_Operation transform
- The data flow contains a Map_Operation transform that outputs UPDATE or DELETE rows
- The data flow contains a Table_Comparison transform
- The **Auto correct load** option in the target table editor is set to **Yes**

If none of these conditions are met, that means the input data contains only INSERT rows. Therefore Data Services does only a bulk insert operation, which does not require a staging table or the need to execute any additional SQL.

By default, Data Services automatically detects the SAP HANA target table type and updates the table accordingly for optimal performance.

Because the bulk loader for SAP HANA is scalable and supports UPDATE and DELETE operations, the following options (target table editor**Options** > **Advanced** > **Update control**) are also available for bulk loading:

- Use input keys
- Auto correct load

9.8 Bulk loading in Sybase ASE

SAP BusinessObjects Data Services supports bulk loading of Sybase ASE databases through the Sybase ASE bulk copy utility. For detailed information about the Sybase ASE bulk loader options and their behavior in the Sybase ASE DBMS environment, see the relevant Sybase ASE product documentation.

9.9 Bulk loading in Sybase IQ

SAP BusinessObjects Data Services supports bulk loading to Sybase IQ databases via the Sybase IQ LOAD TABLE SQL command. For detailed information about the Sybase IQ LOAD TABLE parameters and their behavior in the Sybase IQ database environment, see the relevant Sybase IQ product documentation.

For improved performance when using changed-data capture or auto correct load, Data Services uses a termporary staging table to load the target table. Data Services first loads the data to the staging table, then it applies the operation codes (INSERT, UPDATE, and DELETE) to update the target table. With the **Bulk load** option selected in the target table editor, any one of the following conditions triggers the staging mechanism:

- The data flow contains a Map_CDC_Operation transform
- The data flow contains a Map_Operation transform that outputs UPDATE or DELETE rows
- The **Auto correct load** option in the target table editor is set to **Yes**

If none of these conditions are met, that means the input data contains only INSERT rows. Therefore, Data Services does only a bulk INSERT operation, which does not require a staging table or the need to execute any additional SQL.

Note that because the bulk loader for Sybase IQ also supports UPDATE and DELETE operations, the following options (target table editor **Options** > **Advanced** > **Update control**) are also available for bulk loading:

- Use input keys
- Auto correct load

9.9.1 Configuring bulk loading for Sybase IQ

First configure the bulk loader and log directories in the datastore editor, then enable and configure bulk loading in the target table editor:

1. In the Sybase IQ datastore editor, click **Advanced**.
2. Click in the field next to **Bulk loader directory** and type the directory path or click **Browse** to navigate to where Data Services should write command and data files for bulk loading.
3. Depending on the version of Sybase IQ to which this datastore connects, configure "Bulk Loader" and/or "FTP" options.

4. Click **OK** or **Apply**.
5. Open the data flow and open the target table editor by clicking its name.
6. On the **Bulk Loader Options** tab, select a bulk-loading method and configure the remaining options there and on the **Options** tab.
7. Save the data flow.

9.9.2 Sybase IQ log files

After a job executes, Data Services stores the Sybase IQ message and row logs in the **Bulk loader directory** specified in the datastore editor (regardless of the setting for the **JS and DB on same machine** option). A data file will also be present if you do not use the named pipe option. If you do not specify a Bulk loader directory, Data Services by default writes the files to the directory `<DS_COMMON_DIR>\log\bulkloader`.

The logs include:

- message log—Records constraint violations specified in the **Error handling** section of the target table **Bulk Loader Options** tab.
- row log—Contains the data from the violating row. The data in the row log is delimited by the **Field delimiter** character specified on the **Bulk Loader Options** tab.

If you select **Clean up bulk loader directory after load**, Data Services deletes the data file and log files after loading completes. If you choose not to clean up the bulk loader directory or if your job results in errors captured in the logs, the software does not delete the data file and log files.

9.10 Bulk loading and reading in Teradata

SAP BusinessObjects Data Services supports the following bulk loading and reading tools:

- Parallel Transporter
- FastLoad
- MultiLoad
- TPump
- Load Utility
- None (use ODBC)

Note:
If your Job Server is on a UNIX platform, to take advantage of bulk loading on Teradata 13 databases you must set the required environment parameters in the file $LINK_DIR/bin/td_env.config. Instructions are documented inside the file.

For detailed information about Teradata options and their behavior in the Teradata environment, see the relevant Teradata product documentation.

9.10.1 Bulk loader file options

For all bulk loader methods, you can use staging data files or named pipes. These **File option** types are on the **Bulk Loader Options** tab of the target table editor. This section describes how each file option works.

9.10.1.1 Data file

The Data file option loads a large volume of data by writing to a data file that it passes to the Teradata server. SAP BusinessObjects Data Services runs bulk-loading jobs using a staging data file as follows:

1. It generates staging data file(s) containing data to be loaded into a Teradata table.
2. It generates a loading script to be used by Teradata Parallel Transporter. The script defines read and load operators.
3. If you use Teradata Parallel Transporter, the read operator reads the staging data file, then passes the data to the load operator, which loads data into the Teradata table.

9.10.1.2 Generic named pipe

The **Generic named pipe** file option loads a large volume of data by writing to a pipe from which Teradata reads. SAP BusinessObjects Data Services runs bulk-loading jobs using a generic named pipe as follows:

1. It generates a script that Teradata Parallel Transporter uses to load the database.
2. It creates a pipe to contain the data to load into a Teradata table.

   On UNIX, the pipe is a FIFO (first in, first out) file that has a name of this format:

   ```
   /temp/filename.dat
   ```

   On Windows, the name has this format:

   ```
   \\.\pipe\datastorename_ownername_tablename_loadernum.dat
   ```

3. It executes the loading script. If you use Teradata Parallel Transporter, the script starts Teradata Parallel Transporter and defines read and load operators.
4. It writes data to the pipes.
5. Teradata Parallel Transporter connects to the pipes. Then the read operator reads the named pipe and passes the data to the load operator, which loads the data into the Teradata table.

9.10.1.3 Named pipes access module

The **Named pipes access module** file option loads a large volume of data by writing to a pipe from which Teradata reads. SAP BusinessObjects Data Services runs bulk-loading jobs using a named pipe access module as follows:

1. Data Services generates a script that Teradata Parallel Transporter uses to load the database. The script starts Teradata Parallel Transporter and defines read and load operators.
2. Teradata (Parallel Transporter or non-Parallel Transporter utility) creates named pipes to contain the data to load into a Teradata table.

   On UNIX, the pipe is a FIFO (first in, first out) file that has name of this format:

   ```
   /temp/filename.dat
   ```

   On Windows, the name has this format:

   ```
   \\.\pipe\datastorename_ownername_tablename_loadernum.dat
   ```

3. Data Services connects to the pipes and writes data to them.

   Note:
   When Data Services tries to connect to the pipes, Teradata Parallel Transporter might not have yet created them. Data Services tries to connect every second for up to 30 seconds. You can increase the 30-second connection time to up to 100 seconds as follows: In the Designer, select **Tools** > **Options** > **Job Server** > **General** and enter the following:

   Section: al_engine

   Key: NamedPipeWaitTime

   Value: *nn*

   (*nn* is from 30 to 100)

4. The Teradata Parallel Transporter read operator reads the named pipe and passes the data to the load operator, which loads the data into the Teradata table.

9.10.2 When to use each Teradata bulk-loading method

SAP BusinessObjects Data Services supports multiple bulk-loading methods for Teradata on Windows and UNIX. The following table lists the methods and file options that you can select, depending on your requirements.

Bulk loader method	File Option	Advantages	Restrictions
Parallel Transporter	Data file	• Can use Data Services parallel processing. • Data Services creates the loading script.	• The Teradata Server Tools and Utilities must be Version 7.0 or later. • If you use TTU 7.0 or 7.1, see the *Release Notes*.
	Generic named pipe	• Provides a fast way to bulk load because: ◦ As soon as Data Services writes to a pipe, Teradata can read from the pipe. ◦ Can use Data Services parallel processing. ◦ On Windows, no I/O to an intermediate data file occurs because a pipe is in memory • Data Services creates the loading script.	• A job that uses a generic pipe is not restartable. • The Teradata Server Tools and Utilities must be Version 7.0 or later. • If you use TTU 7.0 or 7.1, see the *Release Notes*.
	Named pipe access module	• The job is restartable. • Provides a fast way to bulk load because: ◦ As soon as Data Services writes to a pipe, Teradata can read from the pipe. ◦ Can use Data Services parallel processing. ◦ On Windows, no I/O to an intermediate data file occurs because a pipe is in memory. • Data Services creates the loading script.	• The Teradata Server Tools and Utilities must be Version 7.0 or later. • If you use TTU 7.0 or 7.1, see the *Release Notes*.

Bulk loader method	File Option	Advantages	Restrictions
Load utility	Data file	Load utilities are faster than INSERT statements through the ODBC driver.	• User must provide the loading script. • Cannot use Data Services parallel processing
	Generic named pipe	• Load utilities are faster than INSERT statements through the ODBC driver. • Named pipes are faster than data files because: ◦ As soon as Data Services writes to a pipe, Teradata can read from the pipe. ◦ On Windows, no I/O to an intermediate data file occurs because a pipe is in memory.	• User must provide the loading script. • Cannot use Data Services parallel processing • A job that uses a generic pipe is not restartable.
	Named pipes access module	• Load utilities are faster than INSERT statements through the ODBC driver. • Named pipes should be faster than data files because: ◦ As soon as Data Services writes to a pipe, Teradata can read from the pipe. ◦ On Windows, no I/O to an intermediate data file occurs because a pipe is in memory • The job is restartable.	• User must provide the loading script. • Cannot use Data Services parallel processing.

Bulk loader method	File Option	Advantages	Restrictions
FastLoad, Multi-Load, and TPump	Data file	Load utilities are faster than INSERT or UPSERT statements through the ODBC driver. Data Services creates the loading script.	Cannot use Data Services parallel processing
	Generic named pipe	• Load utilities are faster than INSERT or UPSERT statements through the ODBC driver. • Named pipes are faster than data files because: • As soon as Data Services writes to a pipe, Teradata can read from the pipe. • On Windows, no I/O to an intermediate data file occurs because a pipe is in memory. • Data Services creates the loading script.	• Cannot use Data Services parallel processing • A job that uses a generic pipe is not restartable.
	Named pipes access module	• Load utilities are faster than INSERT or UPSERT statements through the ODBC driver. • Named pipes should be faster than data files because: • As soon as Data Services writes to a pipe, Teradata can read from the pipe. • On Windows, no I/O to an intermediate data file occurs because a pipe is in memory. • The job is restartable. • Data Services creates the loading script.	Cannot use Data Services parallel processing.

Bulk loader method	File Option	Advantages	Restrictions
None (use ODBC)	Uses Teradata ODBC driver to send separate SQL INSERT statements to load data.	INSERT statements through the ODBC driver are simpler to use than a data file or pipe.	This method does not bulk-load data.

9.10.3 Parallel Transporter method

SAP BusinessObjects Data Services supports Teradata's Parallel Transporter, an ETL tool that consolidates bulk-loading utilities into a single interface.

When you use the Parallel Transporter method, you can leverage Data Services' powerful parallel processing capabilities to specify a number of source and target options including the number of files or pipes for the software to use when processing large quantities of data.

9.10.3.1 Source performance tuning

Teradata source options are on the **Teradata options** tab in the source table editor. Here you can select the reading mode plus a number of advanced options.

You can tune the following options in the Teradata source editor to improve performance:

- **Maximum number of sessions**: For large volumes of data, more sessions allows Data Services to read more data parallel. Ideally this number should equal the number of AMPs.
- **Number of export operator instances**: When reading data in parallel, it can be consumed by multiple export instances for better performance. Ideally this value should equal the number of CPUs.
- **Parallel process threads**: These internal threads break buffered data into rows and columns, which can improve performance by maximizing CPU usage on the Job Server computer. Ideally this number should equal the number of CPUs.

9.10.3.1.1 Special considerations

Be aware of the following limitations when using the Teradata Parallel Transporter.

- It is not always possible to use the Parallel Transporter method to read from a source. In certain situations, Teradata does not support certain SQL constructs and you must use the ODBC method instead. In the following scenarios, Data Services will automatically switch the source mode to ODBC regardless of the actual mode selected on the **Teradata options** tab in the source editor:

  - The WHERE clause of a Query contains `<primary key>=<value>` predicate(s). Such a primary key can be a single column or a composite key.
  - The input schema contains columns of the LOB (CLOB or BLOB) data type.
- Unique secondary index columns are not allowed in the WHERE clause of a Query. However, because Data Services does not have the information as to whether a WHERE clause predicate is part of a unique secondary index, the WHERE clause gets pushed down to the Parallel Transporter reader. In this situation, a run-time error will occur, and you can manually set the source mode to **None** (ODBC).
- Database functions that are pushed down for ODBC readers are also pushed down for Parallel Transporter except for the functions Year and Month.
- Parallel Transporter does not accept parameterized SQL. As a result of this restriction, if a Teradata table is the inner loop of a join, the table will always be cached.
- Readers generated from Table Comparison transform and Lookup function families do not use Parallel Transporter.
- When multiple Teradata readers are optimized by Data Services by collapsing into one, Data Services uses Parallel Transporter whenever possible. When not possible, ODBC is used instead.

9.10.3.2 Target performance tuning

SAP BusinessObjects Data Services provides the option for parallel processing when you bulk load data using the Parallel Transporter method.

In the target table editor using a combination of choices from the **Options** and **Bulk Loader Options** tabs, you can specify the number of data files or named pipes as well as the number of read and load Operator Instances. The **Number of Loaders** option distributes the workload while the Operators perform parallel processing.

In the target table **Options** tab, specify the **Number of Loaders** to control the number of data files or named pipes that Data Services or Parallel Transporter generates. Data Services writes data to these files in batches of 999 rows. For example, if you set **Number of Loaders** to 2, the software would generate two data files, writing 999 rows to the first file, then writing the next 999 rows to the second file. If there are more rows to process, the software continues, writing to the first file again, then the second, and so forth.

On the **Bulk Loader Options** tab, specify the number of instances in the loading scripts. If you set **Number of DataConnector instances** to 2 and **Number of instances** to 2, Parallel Transporter will assign the first read operator instance to read one data file and the other instance to read another data file in parallel. The DataConnector (read operator) instances then pass the data to the load operator instances for parallel loading into Teradata.

The Parallel Transporter uses a control file to read staging files or pipes and load data.

Note:
Performance uaing this type of parallel loading depends on a number of factors such as distribution of incoming data and underlying DBMS capabilities. Under some circumstances, it is possible that specifying

parallel loaders can be detrimental to performance. Always test the parallel loading process before moving to production.

9.10.3.2.1 To configure the bulk loader for parallel processing

1. On the target table **Options** tab, specify the **Number of loaders** to control the number of data files or named pipes. Data Services will write data to these files in batches of 999 rows.
2. On the **Bulk Loader Options** tab, for "Bulk loader" choose **Parallel Transporter**.
3. For **File Option**, choose the type of file (**Data file**, **Generic named pipe**, or **Named pipes access module**) to contain the data to bulk load.
4. If you chose **Data file** or **Generic named pipe** in **File Option**, specify the number of read and load instances in the loading scripts.

   If you set **Number of instances** to 2 (load operators) and **Number of DataConnector instances** to 2 (read operators), Parallel Transporter will assign the first read operator instance to read one data file and the other instance to read another data file in parallel. The read operator instances then pass the data to the load operator instances for parallel loading into Teradata.

   Note:
   If you chose **Data file**, the value you specify for DataConnector instances (read operators) should be less than or equal to the number of data files.

5. If you chose **Named pipes access module** for **File Option**, specify **Number of instances** (load operators) in the loading scripts.

   Teradata uses the value you specify in **Number of loaders** to determine the number of read operator instances, as well as the number of named pipes. The DataConnector instances is not applicable when you use Named Pipes Access Module.

   For example, if you set **Number of loaders** to 2, Parallel Transporter generates two named pipes and assigns one read operator instance to read from one pipe and the other instance to read the other pipe in parallel. If you set **Number of instances** to 2 (load operators), the read operator instances pass the data to the load operator instances for parallel loading into Teradata.

6. If you specified **Named pipes access module** for **File Option**, you can override the default settings for the following Teradata Access Module parameters: Log directory, Log level, Block size, Fallback file name, Fallback directory, Signature checking.

   The Teradata Access Module creates a log file to record the load status and writes information to fallback data files. If the job fails, the Teradata Access Module uses the fallback data files to restart the load. The Access Module log file differs from the build log that you specify in the **Log directory** option in the Teradata datastore.

   Note:
   Data Services sets the bulk loader directory as the default value for both Log Directory and Fallback Directory.

   For more information about these parameters, see the relevant Teradata tools and utilities documentation.

Related Topics

• Reference Guide: Objects, Teradata target table options

9.10.4 Teradata standalone utilities

In addition to the Parallel Transporter interface, SAP BusinessObjects Data Services supports several Teradata utilities that load to and extract from the Teradata database. Each load utility is a separate executable designed to move data into a Teradata database. Choose from the following bulk loader utilities:

Utility	Description
FastLoad	Loads unpopulated tables only. Both the client and server environments support FastLoad. Provides a high-performance load (inserts only) to one empty table each session.
MultiLoad	Loads large quantities of data into populated tables. MultiLoad also supports bulk inserts, updates, upserts, and deletions against populated tables.
TPump	Uses standard SQL/DML to maintain data in tables. It also contains a method that you can use to specify the percentage of system resources necessary for operations on tables. Allows background maintenance for insert, update, upsert, and delete operations to take place at any time you specify. Used with small data volumes.
Load Utility	Invokes one of the above utilities (MultiLoad, FastLoad, or TPump) with the interface prior to Data Services version 11.5.1.

9.10.4.1 FastLoad

This procedure describes how to bulk load a table using the Teradata FastLoad utility.

1. Ensure that your Teradata datastore specifies a value in **TdpId** (Teradata Director Program Identifier). This option identifies the name of the Teradata database to load and is mandatory for bulk loading.
2. In the Bulk Loader Options tab of the target table editor, choose **FastLoad** in the **Bulk loader** drop-down list.
3. For **File option**, choose the type of file (**Data file**, **Generic named pipe**, or **Named pipes access module**) to contain the data to bulk load.
4. You can specify the following FastLoad parameters:

FastLoad parameter	Description
Data encryption	Encrypt data and requests in all sessions used by the job. The default is not to encrypt all sessions.
Print all requests	Prints every request sent to the Teradata database. The default is not to reduce print output.
Buffer size	Number of kilobytes for the output buffer that FastLoad uses for messages to the Teradata database. The default is 63 kilobytes which is also the maximum size.
Character set	Particular mapping between characters and byte strings (such as ASCII or UTF-8).

For more information about these parameters, see the Teradata FastLoad Reference.

5. In **Attributes**, you can usually use the default settings for the following attributes in the FastLoad script that SAP BusinessObjects Data Services generates.

Script attribute	Description
AccountId	Identifier, of up to 30 characters, associated with the user name that will logon to the Teradata database.
CheckpointRate	The number of rows sent to the Teradata database between checkpoint operations. The default is not to checkpoint.
ErrorLimit	Maximum number of rejected records that Teradata can write to the error table 1 while inserting into a FastLoad table.
ErrorTable1	FastLoad uses this table to store records that were rejected for errors other than unique primary index or duplicate row violation.
ErrorTable2	FastLoad uses this table to store records that violated the unique primary index constraint.
MaxSessions	Maximum number of FastLoad sessions for the load job.
MinSessions	Minimum number of FastLoad sessions required for the load job to continue.
TenacityHours	Number of hours that the FastLoad utility continues trying to logon when the maximum number of load jobs are already running on the Teradata database.
TenacitySleep	Number of minutes that the FastLoad utility waits before it retries a logon operation. The default is six minutes.

Note:
By default, Data Services uses the bulk loader directory to store the script, data, error, log, and command (bat) files.

For more information about these parameters, see the Teradata FastLoad Reference.

6. If you specified **Data file** for **File Option**, you can increase the **Number of loaders** on the **Options** tab, which increases the number of data files. The software can use parallel processing to write data to multiple data files in batches of 999 rows.

   If you specified **Generic named pipe** or **Named pipes access module**, Data Services supports only one loader and disables the **Number of loaders** option.

9.10.4.2 MultiLoad

This procedure describes how to bulk load a table using the Teradata MultiLoad utility.

1. Ensure that your Teradata datastore specifies a value in **TdpId** (Teradata Director Program Identifier). This option identifies the name of the Teradata database to load and is mandatory for bulk loading.
2. In the Bulk Loader Options tab of your target table editor, choose **MultiLoad** in the **Bulk loader** drop-down list.
3. In **File Option**, choose the type of file (**Data File**, **Generic named pipe**, or **Named pipes access module**) to contain the data to bulk load. The default is **Data File**.
4. You can specify the following MultiLoad parameters:

MultiLoad parameter	Short description
Reduced print output	The default is not to reduce print output.
Data Encryption	The default is not to encrypt all sessions.
Character set	Particular mapping between characters and byte strings (such as ASCII or UTF-8).

   For more information about these parameters, see the Teradata MultiLoad Reference.

5. In **Attributes**, you can usually use the default settings for the following attributes in the MultiLoad script that SAP BusinessObjects Data Services generates.

Script attribute	Short description
LogTable	Table in which Teradata stores the load job status. Specify the restart log table that will maintain the checkpoint information for your MultiLoad job.
AccountId	Identifier, of up to 30 characters, associated with the user name that will logon to the Teradata database.

Script attribute	Short description
WorkTable	Teradata uses this table to stage input data.
ErrorTable1	Teradata uses this table to store errors that it detects during the acquisition phase of the MultiLoad import task.
ErrorTable2	Teradata uses this table to store errors that it detects during the application phase of the MultiLoad import task.
ErrorLimit	Maximum number of rejected records that Teradata can write to the error table 1 during the acquisition phase of the MultiLoad import task. If used with ErrorPercentage, **ErrorLimit** specifies the number of records that must be sent to the Teradata database before **ErrorPercentage** takes effect.
ErrorPercentage	Approximate percentage (expressed as an integer) of total records sent so far (**ErrorLimit**) to the Teradata database that the acquisition phase might reject.
CheckpointRate	Interval between checkpoint operations during the acquisition phase. Express this value as either: • The number of rows read from your client system or sent to the Teradata database • An amount of time in minutes
MaxSessions	Maximum number of MultiLoad sessions for the load job.
MinSessions	Minimum number of MultiLoad sessions required for the load job to continue.
TenacityHours	Number of hours that the MultiLoad utility continues trying to logon when the maximum number of load jobs are already running on the Teradata database.
TenacitySleep	Number of minutes that the MultiLoad utility waits before it retries a logon operation. The default is six minutes.
TableWait	Number of hours that MultiLoad continues trying to start when one of the target tables is being loaded by some other job.
AmpCheck	Specifies how MultiLoad should respond when an Access Module Processor (AMP) is down.
IgnoreDuplicate	Select IgnoreDuplicate to not place duplicate rows in error table 2. The default is to load the duplicate rows.

Note:
By default, Data Services uses the bulk loader directory to store the script, data, error, log, and command (bat) files.

For more information about these parameters, see the Teradata MultiLoad Reference.

6. If you specified **Data file** in **File Option**, you can increase the **Number of loaders** in the **Options** tab which increase the number of data files. Data Services can use parallel processing to write data to multiple data files in batches of 999 rows.

If you specified **Generic named pipe** or **Named pipes access module**, Data Services supports only one loader and disables the **Number of loaders** option.

Related Topics

• Reference Guide: Objects, Target tables (Teradata target table options)

9.10.4.3 TPump

This procedure describes how to bulk load a table using the Teradata TPump utility.

1. Ensure that your Teradata datastore specifies a value in **TdpId** (Teradata Director Program Identifier). This option identifies the name of the Teradata database to load and is mandatory for bulk loading.
2. On the **Bulk Loader Options** tab of the target table editor, choose **TPump** in the **Bulk loader** drop-down list.
3. For **File Option**, choose the type of file (**Data file**, **Generic named pipe**, or **Named pipes access module**) to contain the data to bulk load.
4. You can specify the following TPump parameters:

FastLoad parameter	Short description
Reduced print output	Reduce the print output of TPump to the minimal information required to determine the success of the job. The default is not to reduce print output.
Retain Macros	Keep macros that were created during the job run. You can use these macros as predefined macros for subsequent runs of the same job.
Data Encryption	Encrypt data and requests in all sessions used by the job. The default is not to encrypt all sessions.
Number of buffers	Number of request buffers that TPump uses for SQL statements to maintain the Teradata database.
Character set	Particular mapping between characters and byte strings (such as ASCII or UTF-8).
Configuration file	Configuration file for the TPump job.
Periodicity value	Controls the rate at which TPump transfers SQL statements to the Teradata database. Value can be between 1 and 600, which specifies the number of periods per minute. The default value is 4 15-second periods per minute.
Print all requests	Turns on verbose mode which provides additional statistical data in addition to the regular statistics.

For more information about these parameters, see the Teradata Parallel Data Pump Reference.

5. In **Attributes**, you specify Data Services parameters that correspond to Teradata parameters in TPump commands. You can usually use the default settings for the following parameters in the TPump script that the software generates.

TPump command	Data Services parameter in Attributes pane	Description
NAME	AccountId	Identifier, of up to 30 characters, associated with the user name that will logon to the Teradata database.
BEGIN LOAD	Append	Use the error table specified in ErrorTable. If the table does not exist, TPump creates it. If the structure of the existing error table is not compatible with the error table TPump creates, the job will run into an error when TPump tries to insert or update the error table.
BEGIN LOAD	CheckpointRate	Number of minutes between checkpoint operations. Value must be an unsigned integer from 0 through 60, inclusive. The default is to checkpoint every 15 minutes.
BEGIN LOAD	ErrorLimit	Maximum number of rejected records that TPump can write to the error table while maintaining a table. The default is no limit. If you specify ErrorPercentage , **ErrorLimit** specifies the number of records that must be sent to the Teradata database before **ErrorPercentage** takes effect. For example, if ErrorLimit is 100 and ErrorPercentage is 5, 100 records must be sent to the Teradata database before the approximate 5% rejection limit is applied. If only 5 records were rejected when the 100th record is sent, the limit is not exceeded. However, if six records were rejected when the 100th record is sent, TPump stops processing because the limit is exceeded.
BEGIN LOAD	ErrorPercentage	Integer value that represents the approximate percent of the total number of records sent to the Teradata Database that might be rejected during the TPump task. You cannot specify this parameter without ErrorLimit.
BEGIN LOAD	ErrorTable	Name of the table in which TPump stores information about errors and the rejected records.
EXECUTE	ExecuteMacro	Name of macro to execute. Using predefined macros saves time because TPump does not need to create and drop new macros each time you run a TPump job script.
DML LABEL	Ignore duplicate inserts	Select **Ignore duplicate inserts** to not place duplicate rows in the error table.

TPump command	Data Services parameter in Attributes pane	Description
NAME	JobName	Character string that identifies the name of a job. The maximum length is 16 characters.
BEGIN LOAD	Latency	Number of seconds that the oldest record resides in the buffer before TPump flushes it to the Teradata database. Value cannot be less than one second. If the **SerializeOn** is not specified, only the current buffer can possibly be stale. If you specify **SerializeOn**, the number of stale buffers can range from zero to the number of sessions.
Other TPump commands	LogTable	Name of the table to use to write checkpoint information that is required for the safe and automatic restart of a TPump job. The default name has the following format: `owner.table_LT`
BEGIN LOAD	MacroDatabase	Name of database to contain any macros TPump uses or builds. The default is to place macros in the same database that contains the TPump target table.
BEGIN LOAD	MaxSessions	Maximum number of sessions for TPump to use to update the Teradata database. SAP BusinessObjects Data Services uses a default of 3.
BEGIN LOAD	MinSessions	Minimum number of sessions for TPump to use to update the Teradata database.
BEGIN LOAD	NoDrop	Do not drop the error table, even if it is empty, at the end of a job. You can use **NoDrop** with **Append** to persist the error table, or you can use it alone.
BEGIN LOAD	NoMonitor	Prevents TPump from checking for statement rate changes from, or update status information for, the TPump Monitor.
IMPORT INFILE	NoStop	Prevents TPump from terminating because of an error associated with a variable-length record.
BEGIN LOAD	Pack	Number of SQL statements to pack into a multiple-statement request. The default is 20 statements per request. The maximum value is 600.
BEGIN LOAD	PackMaximum	Select PackMaximum to have TPump dynamically determine the number of records to pack within one request. The maximum value is 600.

TPump command	Data Services parameter in Attributes pane	Description
BEGIN LOAD	Rate	Initial maximum rate at which TPump sends SQL statements to the Teradata database. Value must be a positive integer. If unspecified, Rate is unlimited.
BEGIN LOAD	Robust	Specifies whether or not to use robust restart logic. Value can be **YES** or **NO**. • **NO** specifies simple restart logic, which cause TPump to begin where the last checkpoint occurred in the job. TPump redoes any processing that occurred after the checkpoint. • **YES** specifies robust restart logic, which you would use for DML statements that change the results when you repeat the operation. Examples of such statements include the following: INSERTs into tables which allow duplicate rows `UPDATE foo SET A=A+1...`
BEGIN LOAD	SerializeOn	Specify a comma separated list of columns to use as the key for rows and guarantee that operations on these rows occur serially.
BEGIN LOAD	TenacityHours	Number of hours that the utility tries to log on sessions required to perform the TPump job. The default is four hours.
BEGIN LOAD	TenacitySleep	Number of minutes that TPump waits before it retries a logon operation. The default is six minutes.

Note:
By default, SAP BusinessObjects Data Services uses the bulk loader directory to store the script, data, error, log, and command (bat) files.

For more information about these parameters, see the Teradata Parallel Data Pump Reference.

6. If you specified **Data file** in **File Option**, you can increase the **Number of loaders** in the Options tab which increase the number of data files. The software can use parallel processing to write data to multiple data files in batches of 999 rows.

   If you specified Generic named pipe or Named pipe access module, Data Services supports only one loader and disables the Number of loaders option.

9.10.4.4 Load Utility

To bulk load a Teradata table using the Load Utility:

1. On the **Bulk Loader Options** tab of your target table editor, choose **Load** in the **Bulk loader** drop-down list.
2. For **File Option**, choose the type of file (**Data File**, **Generic named pipe**, or **Named pipes access module**) to contain the data to bulk load.
3. Enter a command to be invoked by Data Services in the "Command line" text box. For example, `fastload<C:\tera_script\float.ctl`.
4. If you chose **Data file** for **File Option**, enter (or browse to) the directory path where you want the software to place your data file.
5. If you chose **Generic named pipe** or **Named pipes access module** for **File Option**, enter the pipe name.

9.10.5 Using the UPSERT bulk-loading operation

The purpose of the Teradata UPSERT operation is to update a row, but if no row matches the update, the row is inserted.

In SAP BusinessObjects Data Services, you can only use the Teradata UPSERT operation with the following **Bulk loader** methods:

- MultiLoad
- TPump
- Parallel Transporter

  The UPSERT operation is available only with the following Operators:

  - Stream
  - Update

In Data Services, you enable UPSERT on the target table editor's **Bulk Loader Options** tab. In the **Data Services options** section, for **Bulk Operation**, select **Upsert** (the default is **Insert**).

The additional **Attributes** available when you select **Upsert** include:

- **Ignore missing updates**: Select whether or not to write the missing update rows into the error table. The default is **yes**.
- **Ignore duplicate updates**: Select whether or not to write an updated duplicate row into the error table. The default is **no**.

After selecting **Upsert**, note that you can also enable the **Use input keys** option on the target editor's **Options** tab.

Related Topics

• Reference Guide: Objects, Target tables

9.11 Bulk loading using DataDirect's Wire Protocol SQL Server ODBC driver

Use the DataDirect's Wire Protocol SQL Server ODBC driver bulk load feature to quickly insert and update a large number of records into a database. You don't need to use a separate database load utility because the bulk load feature is built into the driver. DataDirect drivers are included in the Data Services installation.

For more detailed information about the Wire Protocol SQL Server ODBC driver, see the DataDirect documentation.

Some general considerations when using the bulk load feature:

- Enable the bulk load option only when you want to optimize load performance. Leaving the bulk load option enabled at all times could lead to undesired results or even corrupt data.
- Do not select the **Enable Bulk Load** option if any of the following Data Services loader options are enabled:
  - Include in Transaction
  - Use Overflow File
  - Auto Correct Load
  - Load Triggers
- You should create a different DSN and datastore when you are:
  - using the same SQL Server database server in different datastores.
  - using loaders that have different bulk load options.

9.11.1 Enabling the DataDirect bulk load feature in Windows

To enable the DataDirect bulk load feature for MS SQL Server on Windows, do the following:

1. Open the ODBC Data Source Administrator and go to the **Use DSN** tab. Click **Add**.
2. In the "Create New Data Source" window, select the DataDirect SQL Server Wire Protocol driver and click **Finish.**

   The "ODBC SQL Server Wire Protocol Driver Setup" window opens.
3. Enter driver setup information, such as the name of the data source, the host number, and so on.
4. On the Bulk tab, select the **Enable Bulk Loading** option.
5. Set the Bulk Options.

Option	Description
Keep Identity	Keeps source identity values.
Check Constraints	Checks constraints while data is being inserted into the database.
Keep Nulls	Keeps null values in the destination table.
Table Lock	Locks the table while the bulk copy operation is taking place. This option is checked by default.
Fire Triggers	Executes a trigger each time a row is being inserted into the database.
Bulk Binary Threshold (KB)	Maximum amount of data that you want exported to the bulk data file.
Batch Size	Number of rows you want the driver to send to the database at one time.
Bulk Character Threshold (KB)	Maximum amount of character data that you want exported to the bulk data file.

9.11.2 Enabling the DataDirect bulk load feature in UNIX

To enable the DataDirect bulk load feature for MS SQL Server on UNIX, do the following:

1. Using a text editor, open the `odbc.ini` file associated with the DSN for which you are bulk loading.
2. Set the EnableBulkLoad option to 1.
3. Enter a BulkLoadOptions value. The value you enter depends on what options you enable.

Option	Description
Keep Identity	A value of 1 keeps source identity values.
Check Constraints	A value of 16 checks constraints while data is being inserted into the database.
Keep Nulls	A value of 64 keeps null values in the destination table.
Table Lock	A value of 2 locks the table while the bulk copy operation is taking place.
Fire Triggers	A value of 32 executes triggers when a row is being inserted into the database.

   Add the option values together and use the total as the BulkLoadOption value. For example, 16 (Check Constraints) + 32 (Fire Triggers) + 1 (Keep Identity) = 49.

   ```
   BulkLoadOptions=49.
   ```

4. Set the BulkLoadBatchSize option. This option sets the number of rows you want the driver to send to the database at one time. The default value is 1024.

Example: **odbc.ini file**

```
[ddsql]
Driver=/build/ds41/dataservices/DataDirect/odbc/lib/DAsqls25.so
Description=DataDirect 6.1 SQL Server Wire Protocol
AlternateServers=
AlwaysReportTriggerResults=0
AnsiNPW=1
ApplicationName=
ApplicationUsingThreads=1
AuthenticationMethod=1
BulkBinaryThreshold=32
BulkCharacterThreshold=-1
BulkLoadBatchSize=1024
BulkLoadOptions=2
ConnectionReset=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=ods
EnableBulkLoad=1
EnableQuotedIdentifiers=0
EncryptionMethod=0
FailoverGranularity=0
FailoverMode=0
FailoverPreconnect=0
FetchTSWTZasTimestamp=0
FetchTWFSasTime=1
GSSClient=native
HostName=vantgvmwin470
HostNameInCertificate=
InitializationString=
Language=
LoadBalanceTimeout=0
LoadBalancing=0
LoginTimeout=15
LogonID=
MaxPoolSize=100
MinPoolSize=0
PacketSize=-1
Password=
Pooling=0
PortNumber=1433
QueryTimeout=0
ReportCodePageConversionErrors=0
SnapshotSerializable=0
TrustStore=
TrustStorePassword=
ValidateServerCertificate=1
WorkStationID=
XML Describe Type=-10
```

Other Tuning Techniques

The previous chapters describe the following tuning techniques:

- Maximizing push-down operations
- Using Caches
- Using Parallel Execution
- Distributing Data Flow Execution
- Using Bulk Loading

This section describes other tuning techniques that you can use to adjust performance:

- Source-based performance options
  - Join ordering
  - Minimizing extracted data
  - Using array fetch size
- Target-based performance options
  - Loading method
  - Rows per commit
- Job design performance options
  - Loading only changed data
  - Minimizing data type conversion
  - Minimizing locale conversion
  - Precision in operations

These techniques require that you monitor the change in performance carefully as you make small adjustments.

10.1 Source-based performance options

10.1.1 Join ordering

10.1.1.1 Join rank settings

You can use join rank to control the order in which sources are joined. Join rank indicates the rank of a source relative to other tables and files joined in a data flow. When considering join rank, the Data Services Optimizer joins sources with higher join ranks before joining sources with lower join ranks.

Join rank must be a non-negative integer. The default value is 0.

Although it is possible to set join rank in the Query editor FROM tab or in a source editor, the best practice is to specify the join rank directly in the Query editor.

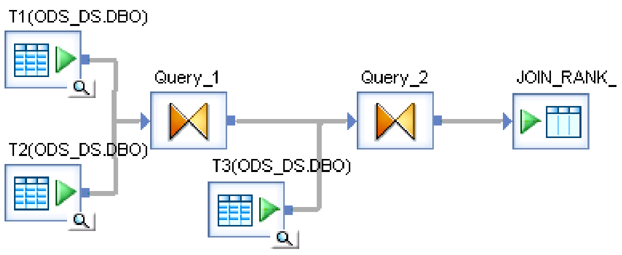
Note:
Join ranks set in sources are not displayed in the Query editor. If a join rank value is specified in the source, to find the value you must open the source editor.

The Data Services Optimizer gives preference to settings entered in the Query editor over settings entered in a source editor. If any one input schema has a join rank specified in the Query editor, then the Data Services Optimizer considers only Query editor join rank settings and ignores all source editor join rank settings.

Additionally, in a data flow containing multiple adjacent Query transforms, upstream join rank settings may be considered.

The join rank determination for multiple adjacent Query transforms can be complex because the Data ServicesOptimizer may combine these Query transforms into a single Query transform.

Consider a case where there is a join T1 inner join T2 in a query, Query_1; and the result of that join is used in another join in a downstream query, Query_2, as Query_1 inner join T3. The Optimizer would combine these two inner joins into a new Query, Query_2'.



Scenario 1

If the join rank values are set as follows:

Query editor	Table	Join rank
Query_1	T1	30
	T2	40
Query_2	Query_1 result set	10
	T3	20

The combined query, Query_2', would have the following join rank values:

Query editor	Table	Join rank
Query_2'	T1	30
	T2	40
	T3	41

The join rank value for T3 is adjusted to 41 because in the original Query_2 T3 has a higher join rank value than the result of T1 join T2 (Query_1 result set).

Scenario 2

In Query_2, if no join rank value is specified, then the default join rank of 0 is applied to both the Query_1 result set and T3. The join rank values are set as follows:

Query editor	Table	Join rank
Query_1	T1	30
	T2	40
Query_2	Query_1 result set	not set (default=0)
	T3	not set (default=0)

The combined query, Query_2', would have the following join rank values:

Query editor	Table	Join rank
Query_2'	T1	30
	T2	40
	T3	40

The join rank value for T3 is adjusted to 40 because in the original Query_2 T3 has the same join rank value as the result of T1 join T2 (Query_1 result set).

Scenario 3

Assume join ranks are not set in the source tables. In Query_1, if no join rank value is specified, then the default join rank of 0 is applied to both T1 and T2. Values are set in the Query_2 Query editor as follows:

Query editor	Table	Join rank
Query_1	T1	not set (default=0)
	T2	not set (default=0)
Query_2	Query_1 result set	10
	T3	20

The combined query, Query_2', would have the following join rank values:

Query editor	Table	Join rank
Query_2'	T1	10
	T2	10
	T3	20

Scenario 4

If join rank values are not specified in the Query_1 and Query_2 Query editors, then the combined query, Query_2', would have no join rank values specified (default=0).

10.1.1.2 Join rank tips

For an inner join between two tables, in the Query editor assign a higher join rank value to the larger table and, if possible, cache the smaller table.

For a join between a table and file:

- If the file is small and can be cached, then assign it a lower join rank value and cache it.
- If you cannot cache the file, then assign it a higher join rank value so that it becomes the "outer table" in the join.

For a join between two files, assign a higher rank value to the larger file and, if possible, cache the smaller file.

10.1.1.3 About join ordering

The Data Services Optimizer determines the order it joins two or more tables based on the type of join and, where applicable, join rank. Although the join order has no effect on the actual result produced, controlling join order can often have a profound effect on the performance of producing the join result.

Join ordering is relevant only in cases where the Data Services engine performs the join. In cases where the query is pushed down to the database, the database determines how a join is done.

Join order in left outer joins

The result of a left outer join depends on the order of the sources. When considering left outer joins, the Data Services Optimizer does not change the join order from that specified in the Query editor. Since join order is not changed, join rank is not considered in the current query. However in a downstream query that uses the results of the left outer join as one of the sources, the join rank may be considered. Caching is implemented based on the cache settings specified in the Query editor.

Join order in inner joins

The result of an inner join is not dependent on the order of the sources. The Data Services Optimizer considers join rank and uses the source with the highest join rank as the left source. The Data Services Optimizer may choose to join tables in a different order than the order defined in the Query editor.

Viewing optimized join order

You can print a trace message to the Monitor log file which allows you to see the order in which the Data Services Optimizer performs the joins. This information may help you identify ways to improve performance.

To add the trace, select **Optimized Data Flow** from the list of traces in the **Trace** tab of the "Execution Properties" dialog.

View the results of the trace in the Trace log. The message begins `Join Order is:`.

Join order in real-time jobs

Using a left outer join is often preferable with real-time jobs because often you want the whole message passed on whether or not conditions a join looks for in the inner source exist.

In fact, if you do not use a left outer join to order joins for a message, the software will still process the message as if it has the highest join rank. The message source editor lacks a Join rank option because the software automatically gives it the highest join rank.

Related Topics

- Maximizing Push-Down Operations

10.1.1.4 Inner join example

When performing inner joins, the results of the join are identical no matter what order the joins are performed.

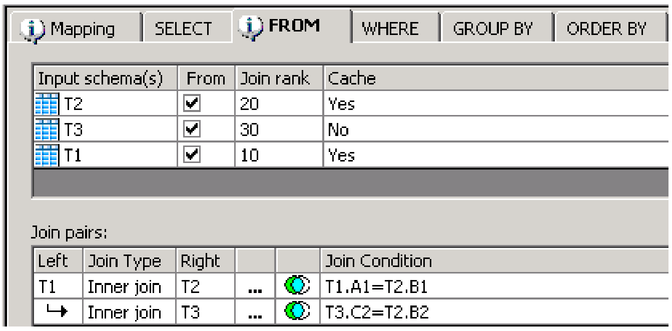
Because join rank is considered, for inner joins, the Data Services Optimizer may choose to join tables in a different order than the order defined in the Query editor.

When assigning join rank, also consider the effect of cache. If you assign a high join rank to a source, it will likely become the left (outer) source for the join. Generally, right (inner) tables are cached. When choosing to cache a source, ensure that you have sufficient memory or pageable cache.

Consider the following information:

Table	Column (data type)	Column (data type)	Number of Rows
T1	A1 (int)	A2 (int)	1,000,000
T2	B1 (int)	B2 (int)	1000
T3	C1 (int)	C2 (int)	1000

Suppose you wanted to create inner joins between T1, T2, and T3. Taking the number of rows in each table into consideration, in the Query editor you might specify join rank, cache, and join pairs as shown in the following screenshot:



Input schema(s)	From	Join rank	Cache
T2	✓	20	Yes
T3	✓	30	No
T1	✓	10	Yes

Join pairs:

Left	Join Type	Right			Join Condition
T1	Inner join	T2	...		T1.A1=T2.B1
↳	Inner join	T3	...		T3.C2=T2.B2

Although the join order is expressed as `T1 INNER JOIN T2 ON T1.A1=T2.B1 INNER JOIN T3 ON T3.C2=T2.B2`, the Data Services Optimizer determines the join order for inner joins. For the first join, the Optimizer takes source with the largest join rank, T3, as the left source and the table with the next highest join rank, T2, as the right source. The result of that inner join would then be joined with the remaining table, T1. The optimized SQL would appear as follows:

```
SELECT T1.A1, T3.C2
FROM (T3 INNER JOIN T2 ON (T3.C2=T2.B2))
INNER JOIN T1 ON (T1.A1=T2.B1)
```

10.1.1.5 Join order with mixed joins example

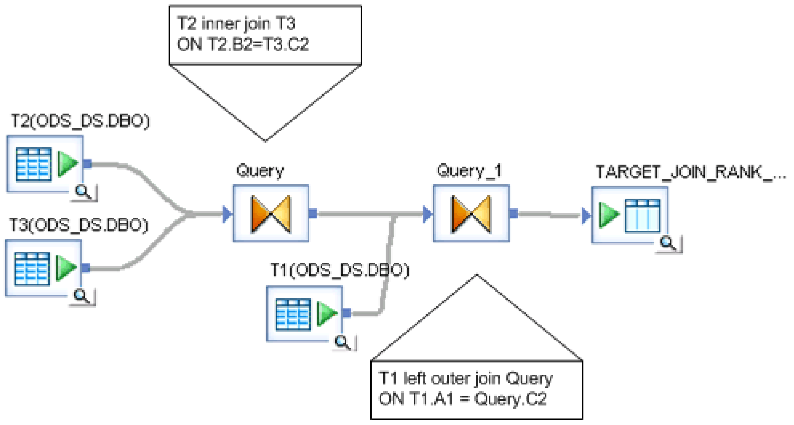
Consider a case where you want to join T1 with the result of an inner join between T2 and T3.

```
FROM T1 LEFT OUTER JOIN (T2 INNER
    JOIN T3 ON T2.B2=T3.C2)
 ON T1.A1=T3.C2
```

.

Although you can specify mixed joins within the Query editor, you may specify the left table only for the first join. Subsequent joins take the result of the previous join as the left source. The data flow for this case must contain two queries. The first query will inner join T2 and T3; the second query is a left outer join with T1 as the left source and the result of the first query as the right source.

The data flow would look as follows:



In the Query_1 transform, C2 originates in T3.

Assume the source information in the following table:

Table	Column (data type)	Column (data type)	Number of Rows	Join Rank	Cache
T1	A1 (int)	A2 (int)	1,000,000	10	Yes
T2	B1 (int)	B2 (int)	1000	20	No
T3	C1 (int)	C2 (int)	1000	30	No

In the first query, T3 has the higher join rank so the Data Services Optimizer would perform the inner join as `T3 INNER JOIN T2 ON T3.C2 = T2.B2` even if the outer table specified in the Query editor is T2. In this case the cache setting for T2 is No, so the table would not be cached.

In the second query, T1 is left outer joined with the results of the first query. Join rank is not considered because this is a left outer join. In a scenario where join rank is relevant, the maximum join rank from the previous join is inherited. Additionally the result of the join T2 and T3 would be cached if either T2 or T3 has a cache setting of Yes.

10.1.2 Minimizing extracted data

The best way to minimize the amount of data extracted from the source systems is to retrieve only the data that has changed since the last time you performed the extraction. This technique is called changed-data capture.

Related Topics

• Designer Guide: Capturing Changed Data

10.1.3 Using array fetch size

SAP BusinessObjects Data Services provides an easy way to request and obtain multiple data rows from source databases. The array fetch size feature allows you to retrieve data using fewer requests, thereby significantly reducing traffic on your network. Tuning array fetch size can also reduce the amount of CPU use on the Job Server computer.

The array fetch feature lowers the number of database requests by "fetching" multiple rows (an array) of data with each request. Enter the number of rows to fetch per request in the **Array fetch size** option on any source table editor or SQL transform editor. The default setting is 1000, meaning that with each database request, The software will automatically fetch 1000 rows of data from your source database. The maximum array fetch size that you can specify is 5000 bytes.

It is recommended that you set the array fetch size based on network speed.

Note:
Higher array fetch settings will consume more processing memory proportionally to the length of the data in each row and the number of rows in each fetch.

Regardless of the array fetch setting, sources reading columns with an Oracle LONG data type cannot take advantage of this feature. If a selected data column is of type LONG, the array fetch size internally defaults to `1` row per request.

10.1.3.1 To set the Array fetch size parameter

1. Use either a source table editor or an SQL transform editor.

   To use a source table editor:

   a. Double-click a source table in the Designer's workspace.
   b. In the **Performance** section of the **Source** tab, enter a number in the **Array fetch size** text box.

   To use an SQL transform editor:

   a. Double-click an SQL transform in the Designer's workspace.
   b. In the SQL transform editor, enter a number in the **Array fetch size** text box.

   Array Fetch Size indicates the number of rows returned in a single fetch call to a source table. The default value is 1000. This value reduces the number of round-trips to the database and can improve performance for table reads.

   The Array Fetch Size option does not support long column data types. If the **SELECT** list contains a long column, the software sets the **Array Fetch Size** to 1 and reads one row of data at a time from the database.

2. Click **OK**.

10.1.3.2 Tip

The optimal number for **Array fetch size** depends on the size of your table rows (the number and type of columns involved) as well as the network round-trip time involved in the database requests and responses. If your computing environment is very powerful, (meaning that the computers running the Job Server, related databases, and connections are extremely fast), then try higher values for **Array fetch size** and test the performance of your jobs to find the best setting.

10.2 Target-based performance options

10.2.1 Loading method

You can choose to use regular loading or bulk loading. For a regular load, the **Parameterized SQL** option is automatically selected when generating, parsing, and compiling the statement. By using parameterized SQL, the software can minimize these efforts by using one handle for a set of values instead of one handle per value.

Many databases do not support bulk loading with the following options; see the specific options for your target database in the *Reference Guide*.

- Auto-correct load
- Enable Partitioning
- Number of Loaders
- Full push down to a database

  The software automatically selects this optimizer process when the following conditions are met:

  - The source and target in a data flow are on the same database
  - The database supports the operations in the data flow

  If the optimizer pushes down source or target operations, then it ignores the performance options set for sources (Array fetch size, Caching, and Join rank) because it is not solely processing the data flow.

- Overflow file
- Transactional loading

To improve performance for a regular load (parameterized SQL), you can select the following options from the target table editor. Note that if you use one, you cannot use the others for the same target.

- Enable Partitioning

  Parallel loading option. The number of parallel loads is determined by the number of partitions in the target table.

- Number of Loaders

  Parallel loading option. The number of parallel loads is determined by the number you enter for this option.

Related Topics

- Push-down operations
- Table partitioning
- Bulk Loading and Reading

10.2.2 Rows per commit

Rows per commit for regular loading defaults to 1000 rows. Setting the **Rows per commit** value significantly affects job performance. Adjust the rows per commit value in the target table editor's **Options** tab, noting the following rules:

- Do not use negative number signs and other non-numeric characters.
- If you enter nothing or `0`, the text box will automatically display `1000`.
- If you enter a number larger than `5000`, the text box automatically displays `5000`.

It is recommended that you set rows per commit between `500` and `2000` for best performance. You might also want to calculate a value. To do this, use the following formula:

`max_IO_size/row size` (in bytes)

For most platforms, `max_IO_size` is 64K. For Solaris, `max_IO_size` is 1024K.

Note that even with a value greater than one set for **Rows per commit**, SAP BusinessObjects Data Services will submit data one row at a time if the following conditions exist:

- You are loading into a database (this scenario does not apply to Oracle databases), and have a column with a LONG datatype attribute.
- You are using an overflow file where the transaction failed. However, once all the rows are loaded successfully, the commit size reverts to the number you entered. In this case, depending on how often a load error happens, performance might be come worse than setting **Rows per commit** to `1`.

Related Topics

- Caching sources

10.3 Job design performance options

10.3.1 Loading only changed data

Identifying and loading only changed data is called changed-data capture (CDC), which includes only incremental data that has changed since the last refresh cycle. Performance improves because with less data to extract, transform, and load, the job typically takes less time.

Related Topics

• Designer Guide: Capturing Changed Data

10.3.2 Minimizing data type conversion

SAP BusinessObjects Data Services offers very robust and easy-to-use data type conversions via column mappings of different data types. It is recommended that you:

- Avoid unnecessary data conversions.
- Verify that SAP BusinessObjects Data Services is performing the implicit conversions (selected when you drag and drop columns from input to output schemas in the query transform) as expected. This can be done by looking at the warnings generated during job validation.

10.3.3 Minimizing locale conversion

If your jobs do not require the use of different or multi-byte locales, you can increase performance by ensuring that locales are single-byte and not mismatched.

10.3.4 Precision in operations

SAP BusinessObjects Data Services supports the following precision ranges: 0-28, 29-38, 39-67, 68-96. Note that as you decrease precision, performance increases for arithmetic operations and comparison operations. In addition, when processing an arithmetic or boolean operation that includes decimals in different precision ranges, the software converts all to the highest precision range value because it cannot process more than one decimal data type precision range for a single operation. For example, if the software must perform an arithmetic operation for decimals with precision 28 and 38, it converts both to precision 38 before completing the operation.

Index

A

aggregations, pushing to database 34
al_engine process 19
al_jobserver process 19
analyzing trace log files 24
array fetch size 136
auto correct load
  enabling push down 41

B

bulk loading
  DB2 89
  DB2, using import utility 95
  Informix 97
  Oracle 102
  Oracle, conventional-path load 102
  Oracle, direct-path load 102
  Sybase 107
  Teradata 108
  Teradata, Load Utilities 117
  Teradata, Parallel Transporter 114
  using 125
  vs. regular loading 138

C

caching
  joins 48
  Table_Comparison transform 51
  tables 47
  using a lookup function 50
caching data 30, 47
caching lookups vs. setting source table as outer join 50
changed-data capture, performance, using for 136
conventional-path loading in Oracle 102

D

data
  capturing changes 139
Data Services, optimizing data transformations 33
data transformations
  optimize by caching 22, 30, 47
  optimizing 33

data transformations *(continued)*
  pushing to source database 37
data type conversion 140
database links
  and datastores 42
  defined 42
  example data flow 43
  importing 42
  software support 42
  tuning performance 45
datastores
  adding database link support 42
  and database links 42
  persistent cache 52
DB2, bulk loading 89
degree of parallelism, enabled for functions 66
degree of parallelism, for data flows 62

E

extracting data
  caching 47
  minimizing 136

F

filtering, pushing to database 34
functions, pushing to database 34

I

import utility for DB2 bulk loading 95
Informix, bulk loading 97
IPC communications
  between data flows 86
  between sub data flows 87

J

join order
  real-time jobs 133
join ordering 133
join rank 130
joins, caching 48
joins, pushing to database 34

L

linked datastores
  creating 42
  example data flow 43
  tuning performance 45
Loading 138
logs
  performance monitor 54
lookup function, caching 50

M

Microsoft SQL Server
  bulk loading 98
minimize data extracted 136
monitor log files, performance 25

O

operational dashboards 26
optimizing data transformations 33
optimizing join order 133
Oracle
  bulk loading 102
  bulk loading, conventional-path load 102
  bulk loading, direct-path loading 102
  parallel loading 104
ordering, pushing to database 34

P

parallel execution 57, 73
  degree of parallelism 62
  Degree Of Parallelism 62, 78
  degree of parallelism, enabled for functions 66
  file multi-threading 70
  for different data flows and work flows 57
  partitioned tables 58
  partitioned tables, creating 59
  partitioned tables, enabling 61
  using table partitioning and DOP 68
  within a single data flow 58, 73
parallel loading in Oracle 104

peer-to-peer communications
  between data flows 86
  between sub data flows 87
performance
  changed-data capture and 139
  data transformations and 37
  improving, DS environment 13, 17
  improving, with bulk loading 125
  improving, with parallel execution 57, 73
  measuring, with DI options 20
  measuring, with DS options 27
  tuning, with DS job design options 30, 129
  tuning, with DS source options 30, 129
  tuning, with DS target options 30, 129
performance monitor log 54
Performance Monitor, reading execution statistics 26
persistent cache datastore 52
projection, pushing to database 34
pushing down operations to a database
  join example 37
  joins, pushing to database 37
pushing operations to database
  example 35
  logic 33
  overview 33

R

rows
  duplicate, avoiding loading 41
rows per commit 139
rows, retrieving multiple 21, 136

S

server groups
  grid computing 84
sizing tables 47
sources
  retrieving multiple rows 21, 136
SQL
  pushing to database 33
Sybase, bulk loading 107

T

Table_Comparison transform, performance, improving 51
tables
  caching 47
  retrieving multiple rows 21, 136
  sizing 47
Teradata
  bulk loading 108
  bulk loading using Load Utilities 117
  bulk loading using Parallel Transporter 114
  load utilities 117, 119, 121, 125
throughput, improving 30
trace log files, analyzing 24
tuning techniques
  array fetch size 136
  caching data 30, 47
  minimize data extracted 136
  minimize data type conversion 140
  rows per commit 139