Information Builders

WebFOCUS

Hyperstage for PostgreSQL Reference Guide

WebFOCUS Reporting Server Release 8.2 DataMigrator Server Release 7 Version 7.07

December 15, 2017

Contents

Preface

This document describes how to set up and use WebFOCUS Hyperstage directly, beyond the scope of a typical WebFOCUS installation.

How This Manual Is Organized

This manual includes the following chapters:

	Chapter/Appendix	Contents
1	About WebFOCUS Hyperstage	Introduces WebFOCUS Hyperstage, a column-oriented, high performance analytic engine designed for analytic applications.
2	Installing and Configuring the Hyperstage Database	Describes the Hyperstage technical requirements, and explains the installation and configuration steps.
3	Using the Hyperstage Database Beyond WebFOCUS	Describes how to work with Hyperstage, check its version, and use Hyperstage to improve ETL performance.
4	Managing Hyperstage Tables	Describes the Hyperstage tables.
5	Data Manipulation Statements	Describes the statements supported by Hyperstage.
6	Character Set Support	Describes how character sets are supported by the Hyperstage analytic engine.
7	Importing and Exporting Data in Hyperstage	Describes how to import and export data using the Hyperstage analytic engine.
8	Running Queries in Hyperstage	Describes how to run queries in Hyperstage.
9	Hyperstage Backup and Recovery	Describes how to backup and restore the Hyperstage databases.
A	Functions and Operators	Describes the functions and operators supported by Hyperstage.
B	Hyperstage Data Tools	Describes the data tools used by Hyperstage.

Documentation Conventions

The following table describes the documentation conventions that are used in this manual.

Convention	Description

`THIS TYPEFACE`
or
`this typeface`
	Denotes syntax that you must enter exactly as shown.

`this typeface`	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
`underscore`	Indicates a default setting.
this typeface	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option that you can click or select.
Key + Key	Indicates keys that you must press simultaneously.
`{ }`	Indicates two or three choices. Type one of them, not the braces.
`[ ]`	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the parameter in the brackets, not the brackets.
`|`	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
`...`	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis (...).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our Technical Content Library at *http://documentation.informationbuilders.com*. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have questions about this product?

Join the Focal Point community. Focal Point is our online developer center and more than a message board. It is an interactive network of more than 3,000 developers from almost every profession and industry, collaborating on solutions and sharing tips and techniques. Access Focal Point at *http://forums.informationbuilders.com/eve/forums*.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our website, *http://www.informationbuilders.com*. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of *www.informationbuilders.com* also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

Call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your questions. Information Builders consultants can also give you general guidance regarding product capabilities. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Information You Should Have

To help our consultants answer your questions effectively, be prepared to provide the following information when you call:

- ❑ Your six-digit site code (*xxxx.xx*).
- ❑ Your WebFOCUS configuration:
  - ❑ The front-end software you are using, including vendor and release.
  - ❑ The communications protocol (for example, TCP/IP or HLLAPI), including vendor and release.

- The software release.
- Your server version and release. You can find this information using the Version option in the Web Console.

- The stored procedure (preferably with line numbers) or SQL statements being used in server access.
- The Master File and Access File.
- The exact nature of the problem:
  - Are the results or the format incorrect? Are the text or calculations missing or misplaced?
  - Provide the error message and return code, if applicable.
  - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, your security system, communications protocol, or front-end software changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two data sources, have you tried executing a query containing just the code to access the data source?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

In an effort to produce effective documentation, the Technical Content Management staff welcomes your opinions regarding this document. You can contact us through our website *http://documentation.informationbuilders.com/connections.asp*.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our website (*http://education.informationbuilders.com*) or call (800) 969-INFO to speak to an Education Representative.

Chapter 1

About WebFOCUS Hyperstage

Thank you for choosing to install WebFOCUS Hyperstage. Hyperstage is a column-oriented, high performance analytic engine designed for analytic applications and data marts that need fast query response across large data volumes. Hyperstage was designed specifically for large volume data analytics applications with up to 50 Terabytes of data.

In this chapter:

Hyperstage Overview

Hyperstage uses a unique and patent-pending approach to compressing, storing, and processing data that allows it to be installed and run on commodity hardware with little or no DBA intervention. Hyperstage requires little tuning to support ad hoc or complex business analytic queries.

Hyperstage is a database engine utilizing the PostgreSQL database environment. As such, Hyperstage is fully compatible with all PostgreSQL-compliant Business Intelligence tools and utilizes the PostgreSQL administrative interface to reduce the learning curve for system administrators.

Hyperstage provides a versatile, highly-compressed database system optimized for analytic-type queries. The ratio of possible compression and the speed of data import and retrieval are optimized at the expense of some transactional features of the engine performance, like the frequent data updating.

Hyperstage executes complex or ad hoc queries across vast amounts of data with a low cost of ownership.

Hyperstage and PostgreSQL

Hyperstage combines the Hyperstage storage engine with PostgreSQL server implementation. Hyperstage consists of several layers. The upper layers are provided by the PostgreSQL server implementation, and the lower layers are provided by Hyperstage.

Hyperstage includes both its own optimizer and executor along with the storage engine. The PostgreSQL query engine can be used with Hyperstage. However, since the PostgreSQL storage engine interface is row oriented, it cannot take full advantage of the column orientation or the Knowledge Grid and hence query execution through this path is reduced. Queries will be directed to the Hyperstage optimizer whenever possible.

Hyperstage ships with the full PostgreSQL binaries required. PostgreSQL is used to store catalog information (as with other storage engines). You can use the PostgreSQL instance for other purposes, but joining PostgreSQL and Hyperstage tables may result in reduced performance as the PostgreSQL query engine will be used.

PostgreSQL provides:

- Mature connectors, tools and resources.
- Interconnectivity and certification with BI tools.
- Management services and utilities.

Hyperstage provides:

- Load function that compresses data.
- Column-oriented storage engine.
- Knowledge Grid metadata layer that contains information about the compressed data.
- Optimizer/executor that uses the Knowledge Grid.

Chapter 2

Installing and Configuring the Hyperstage Database

The following section describes the installation and configuration steps for Hyperstage.

In this chapter:

Technical Requirements

Before installing Hyperstage, review the following technical requirements.

Hyperstage Technical Requirements	
Platforms	Windows Server 2008 and 2012 Red Hat Enterprise Linux 6.x and 7.x CentOS 6.x and 7.x Debian 6 Novell SUSE Linux Enterprise 11
Processor Architecture	Intel 64-bit AMD 64-bit
For Personal Evaluation and Application Development	
CPU Speed	1.8GHz minimum, 2.0GHz or faster dual core or quad core recommended
Memory	4GB minimum, 8GB recommended
For Multi-User Evaluation or Production Deployment	

Hyperstage Technical Requirements	
CPU Speed	2.0GHz minimum, 8 cores minimum
Memory	16GB minimum, 32GB recommended

Installing Hyperstage

Hyperstage is packaged as part of the Hyperstage version of the Reporting Server installation. This version installs the WebFOCUS Reporting Server, Hyperstage, and automatically configures the WebFOCUS Reporting Server for use with Hyperstage.

Note: The type of server you install determines the default names for the program folder and product directory.

- If you install the WebFOCUS Reporting Server Release 8.2, then the default names will indicate “82”. For example:

  c:\ibi\srv82

- If you install DataMigrator Server Release 7 Version 7.07, then the default names will indicate “77”. For example:

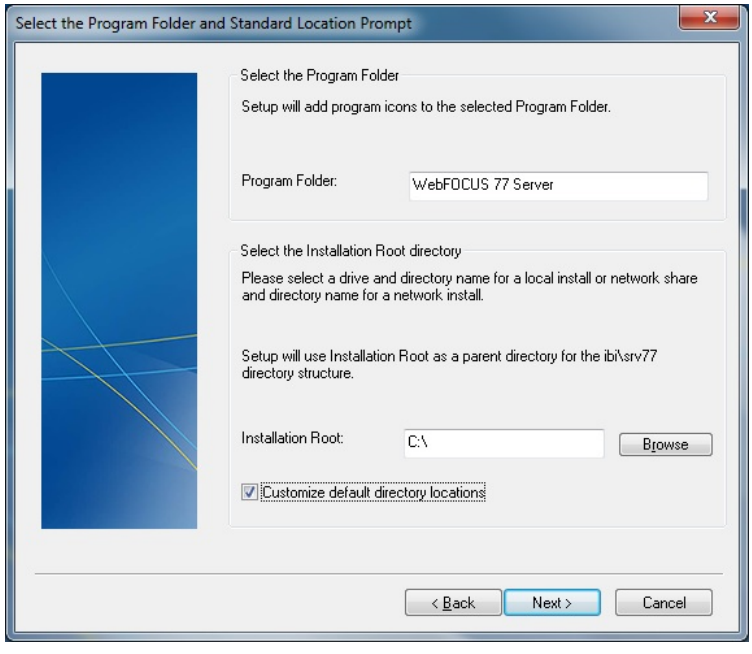
  c:\ibi\srv77

Procedure: How to Install Hyperstage

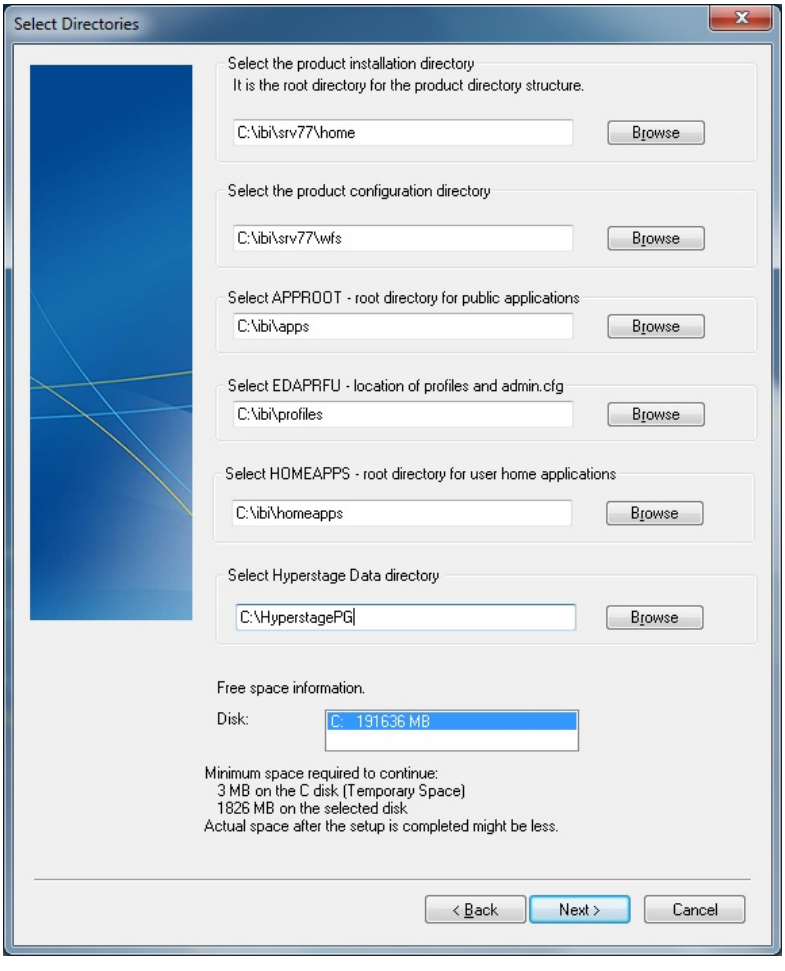
Important: Only one instance of Hyperstage can run on a single machine.

1. Download the installation package for Hyperstage for the desired platform.
2. Follow the Reporting Server installation instructions in the *Server Installation* manual.
3. By default, Hyperstage data directories (ib_data and pg_data) will be installed under the ibi \HyperstagePG directory.

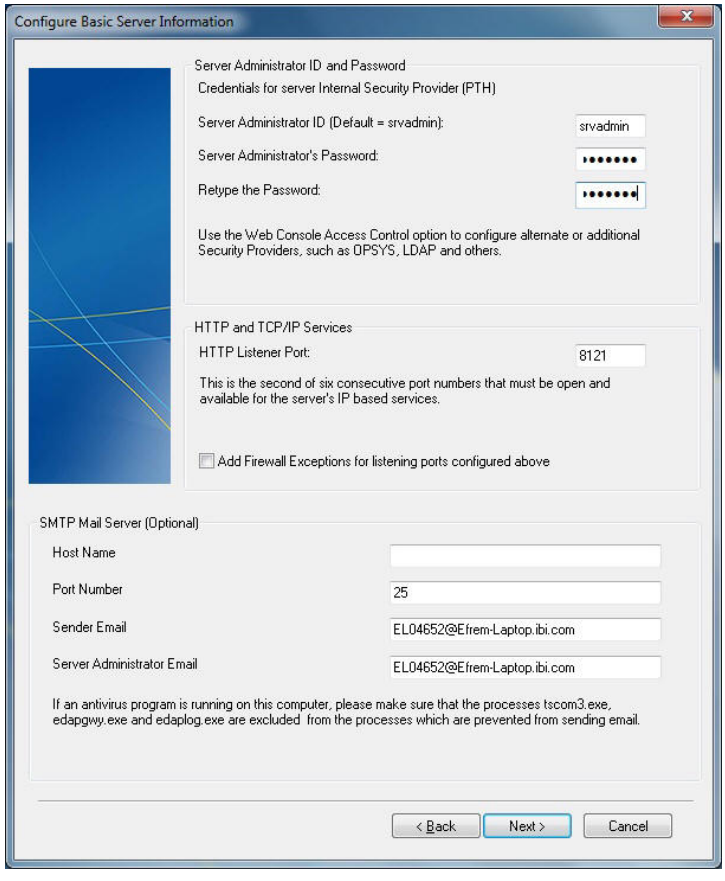
In order to customize the location of the Hyperstage data directories, select the *Customize default directory locations* check box in the Select the Program Folder and Standard Location Prompt dialog box, as shown in the following image.



4. Enter the desired Hyperstage directory location, as shown in the following image.



5. By default, the HTTP Listener Port on the Configure Basic Server Information dialog box is 8121, as shown in the following image.



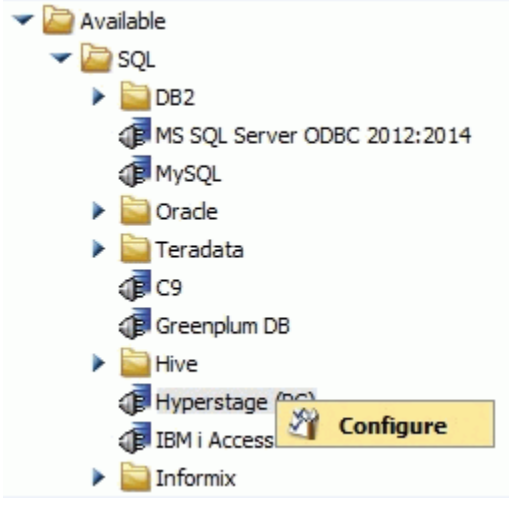
The port for Hyperstage will automatically configure to three port numbers higher than the HTTP Listener Port (for example, 8124).

Configuring the Hyperstage (PG) Adapter

When installing the Hyperstage version of the WebFOCUS Reporting Server, the Hyperstage (PG) adapter will automatically be configured. If the WebFOCUS Reporting Server and Hyperstage version of the Reporting Server exist on different boxes, then the Hyperstage (PG) adapter needs to be manually configured on the WebFOCUS Reporting Server pointing to the Hyperstage port.

Procedure: How to Configure the Hyperstage (PG) Adapter

1. Launch the Web Console and click the *Adapters* tab.
2. Expand *Available* and then expand the *SQL* folder.
3. Right-click *Hyperstage (PG)* and click *Configure*, as shown in the following image.



The Add Hyperstage (PG) Configuration window opens.

Note: It is a good practice to copy the entire *hs* directory structure below the *home* directory of the Hyperstage version of the Reporting Server to a location where the WebFOCUS Reporting Server exists.

It is very common that the WebFOCUS Reporting Server and the Hyperstage version of the Reporting Server would exist on different machines. For example, you can copy *\ibi\home \hs* from Machine #1 to *\ibi\HyperstagePG* on Machine #2.

This will create the entire *hs* directory structure from the Hyperstage version of the Reporting Server under the \ibi\HyperstagePG directory where the WebFOCUS Reporting Server exists.

It is a good practice to keep the version of the Hyperstage Reporting Server and the WebFOCUS Reporting Server in sync. An upgrade of the Hyperstage Reporting Server might contain adapter changes that the WebFOCUS Reporting Server would require. If the WebFOCUS Reporting Server is upgraded for reasons not related to Hyperstage, the Hyperstage Reporting Server should also be upgraded to keep the adapter in sync with the Hyperstage version.

4. Complete the following fields:

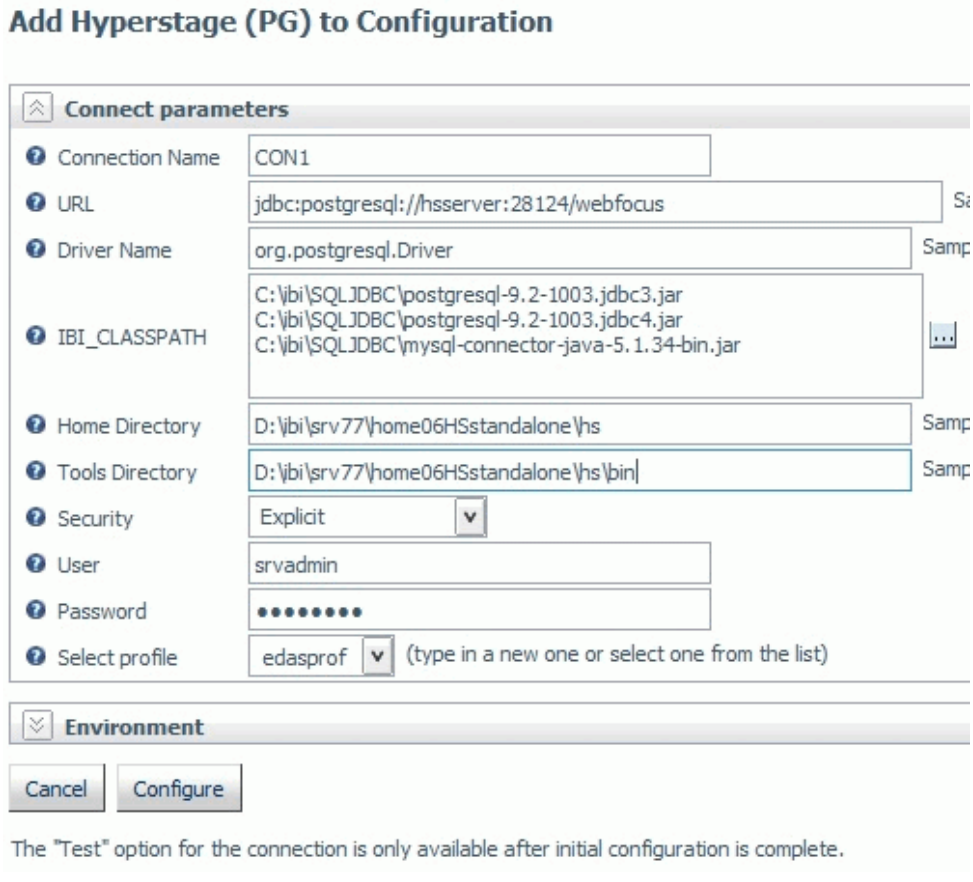
   - In the Connection Name box, type a name for the connection.
   - In the URL box, type the URL to the Hyperstage port and database, for example, jdbc:postgresql://hsserver:28124/webfocus.
   - In the Driver Name box, type *org.postgresql.Driver*.
   - In the IBI_CLASSPATH box, add the location of the JDBC Jar file.

     Note: The PostgreSQL jar files exist in the home\hs\java directory of the Hyperstage version of the Reporting Server, for example, \ibi\srv77\home06HSstandalone\hs \java. These could be copied to a location on the machine where the WebFOCUS Reporting Server resides, for example, C:\ibi\SQLJDBC\postgresql-9.2-1003.jdbc3.jar and C:\ibi\SQLJDBC\postgresql-9.2-1003.jdbc4.jar.
   - In the Home Directory box, enter the location of the home directory for Hyperstage (PG).

     Note: The location is the home\hs directory of the Hyperstage version of the Reporting Server, for example, D:\ibi\srv77\home06HSstandalone\hs.
   - In the Tools Directory box, type the location of the tools directory for Hyperstage (PG).

     Note: The location is the home\hs\bin directory of the Hyperstage version of the Reporting Server, for example, D:\ibi\srv77\home06HSstandalone\hs\bin.
   - Type the User and Password for the PostgreSQL database. By default, the credentials are *srvadmin/srvadmin*.

The following image shows the window with all of the fields completed.



5. Click *Configure*.

Configuring Hyperstage

The Hyperstage configuration file is called infobright.cnf and is located in the ib_data subdirectory within the Hyperstage Data installation directory for example, C:\ibi\HyperstagePG \ib_data.The configuration file is a text file containing the Hyperstage configuration parameters.

Each parameter is shown on a separate line.

If a parameter is not present in the configuration file or if the configuration file does not exist, the default values are used. Blank lines and comments (lines starting with #) are ignored.

Be sure to customize the following parameters to optimize performance. These tuning parameters are case sensitive and must be typed as shown in the following table.

Note: The values are commented out (preceded by #) in the infobright.cnf file, which causes them to default to the application minimum allowed values of 600 and 320 for ServerMainHeapSize and LoaderMainHeapSize, respectively.

Hyperstage Configuration Parameters

Parameter Name	Description
LicenseFile	Specifies the path or name of the newly required License file.
LogLevel	Controls how much information is written to logs. This is similar to the obsolete ControlMessages parameter.
LogRotateSize	Specifies how large the log file can be before it is rotated and archived.
LogRotateFiles	Specifies how many log archive files are kept.
KNFolder	Specifies the folder where Knowledge Grid is stored.
CacheFolder	Specifies the folder where temporary objects are stored.
ServerMainHeapSize	Specifies the size (in MB) of the main memory heap.
ThrottleLimit	Controls how many SELECT queries can run concurrently.

Configuration Tips and Examples

Important: You must properly configure your memory settings to ensure optimal performance.

The following table shows sample, recommended memory configurations for different systems.

System Memory	Server Main Heap Size	Loader Main Heap Size
64GB	48000	800

System Memory	Server Main Heap Size	Loader Main Heap Size
48GB	32000	800
32GB	24000	800
16GB	10000	800
8GB	4000	800
4GB	1300	400
2GB	600	320

In most cases, the loader does not benefit from larger memory settings. However, increasing the LoaderMainHeapSize can help when:

- A table to be loaded has very long text values.

  or

- The table has many columns (for example, 1000 columns).

You can use more memory at import if you are planning to execute several concurrent load tasks to different data tables. However, disk access may become a bottleneck.

ServerMainHeapSize should be as large as possible, but safely smaller than the amount of physical memory on the machine. If performance decreases because of memory swapping by the operating system, try to set lower heap sizes. We also recommend decreasing the heap size if many users are running queries in parallel.

Note: Hyperstage may use additional memory for heavy loads or queries. Also, other applications on your server will use memory for their processes. It is important that the total of ServerMainHeapSize is less than the total available physical memory. If the system needs to swap memory, performance will be severely impacted.

Chapter 3

Using the Hyperstage Database Beyond WebFOCUS

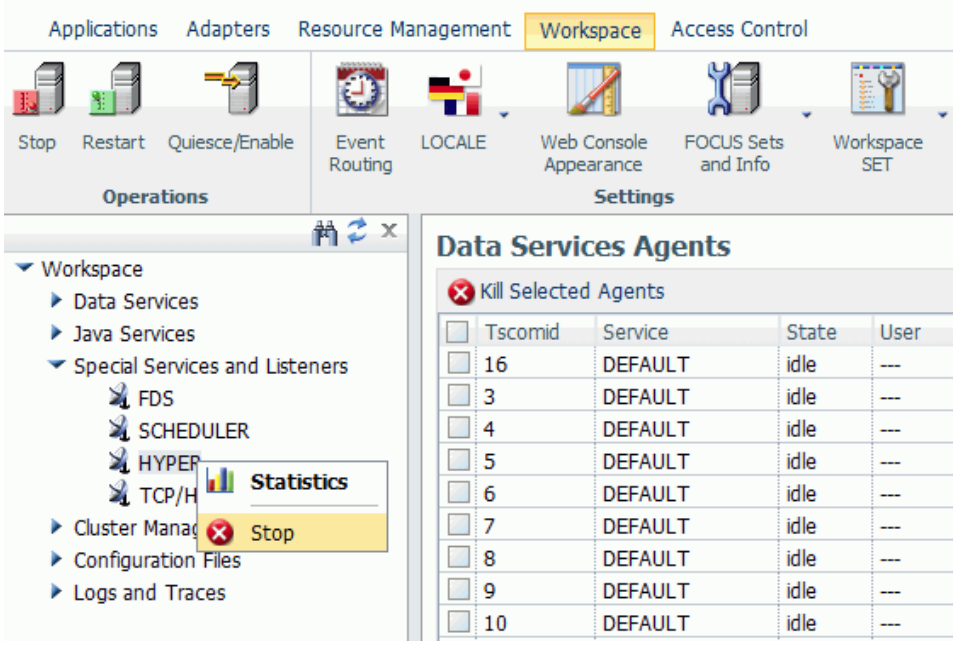
The following section describes how to work with the Hyperstage server.

In this chapter:

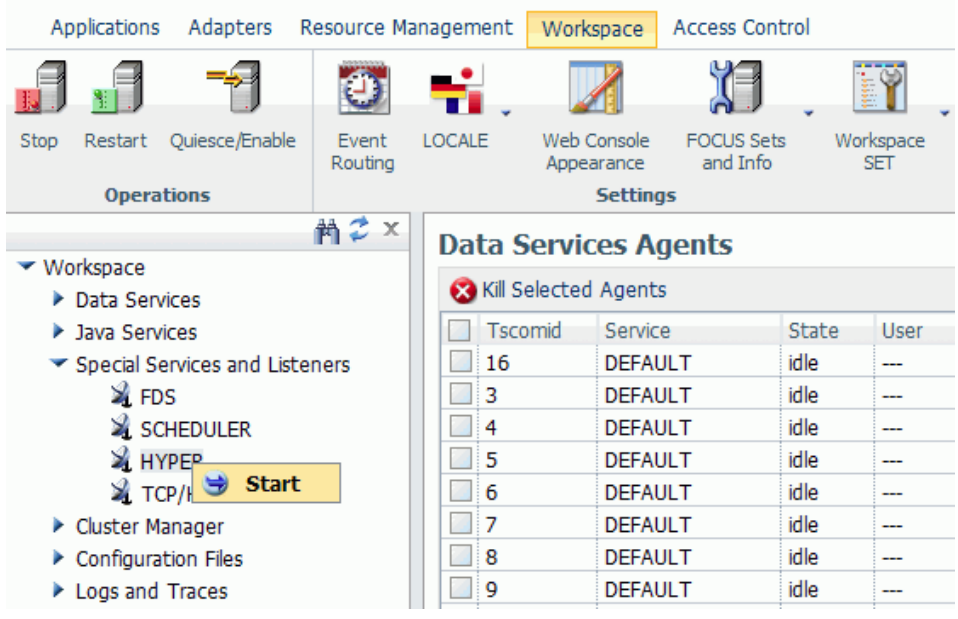
Starting and Stopping the Hyperstage Server Under Windows

The Hyperstage Server starts and stops automatically when starting and stopping the Hyperstage version of the Reporting Server.

To manually stop the Hyperstage Server, from the Workspace tab, expand the *Select Special Services and Listeners* section of the Reporting Server Web Console, right-click the *HYPER* service and click *Stop*, as shown in the following image.



To manually start the Hyperstage server, from the Workspace tab, expand the *Select Special Services and Listeners* section of the Reporting Server Web Console, right-click the *HYPER* service and select *Start*, as shown in the following image.



Working With the Hyperstage Server

You can use the tools provided with PostgreSQL, such as the psql client program, with the Hyperstage server. For more information, see the *PostgreSQL 9.2 Documentation*.

Note: The PostgreSQL 9.2 Documentation is referenced throughout this manual. You can access the content from the following link: *http://www.postgresql.org/docs/9.2/static/index.html*

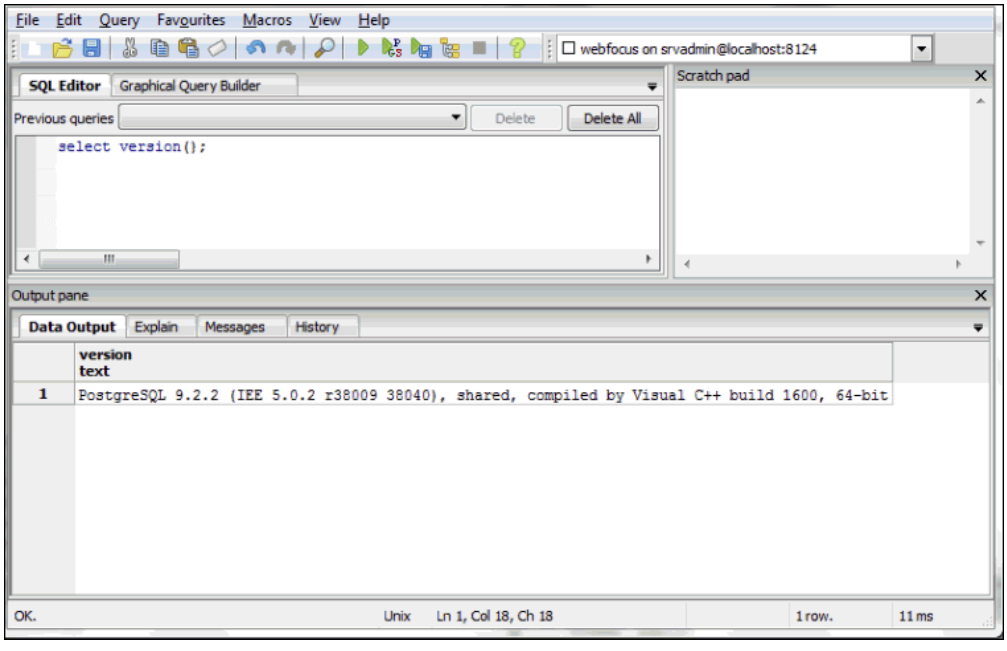
You can also use GUI tools, such as the pgAdmin Workbench, to query Hyperstage databases in a more graphical manner.

Checking the Hyperstage Version

You can use the following SQL to check the version of the Hyperstage system.

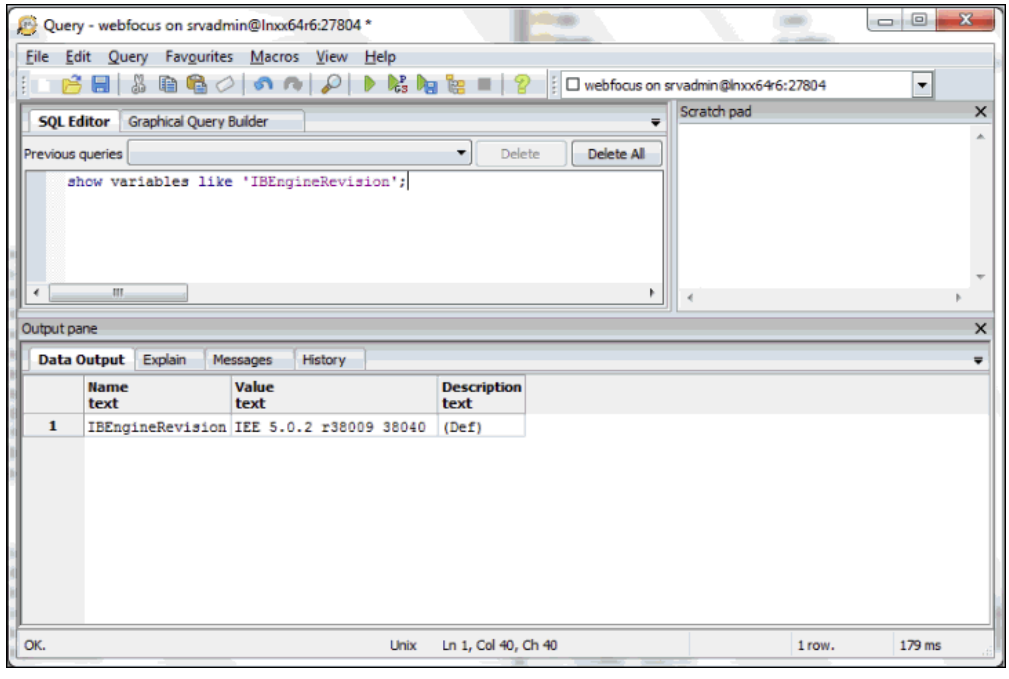
Windows:

```
select version();
```



Linux:

```
show variables like 'IBEEngineRevision';
```



Quick Copy For Hyperstage Using Extended Bulk Load Utility

Note: Hyperstage only supports DataMigrator, Quick Copy, and Custom Copy as ETL tools.

The Quick Copy tool allows for the copying of all data from a Source table into Hyperstage. The *Bulk Load* option should be selected in order for data to be loaded quickly. If the *Bulk Load* option is cleared, the data will take much longer to load.

The Custom Copy tool allows for the copying of selected columns, presorting data within selected columns, and filtering of columns from a Source table into Hyperstage.

The DataMigrator tool is a comprehensive ETL tool. ETL flows can be created to copy data from various sources, transform the data, and load the data into targets.

To access the Quick Copy tool, right-click the name of the synonym corresponding to the table or data you wish to copy into Hyperstage, and select *Quick Copy*.

The following configuration setting options are available:

Load Option

New/Replace. Recreate the target table before loading the data.

Append to Existing. Data is loaded to an existing table.

Adapter

The list of adapters currently configured on the Reporting Server. The Hyperstage adapter must be configured with at least one connection in order to appear in the drop-down menu.

Connection

The Hyperstage connection used for the load operation.

Synonym Application

The target application on the Reporting Server where the target synonym will be stored.

Synonym

The name of the target synonym defining the Target Table Name.

Table Name

The name of the Hyperstage table where the data will be loaded.

Bulk Load

When selected, data will be loaded using the Hyperstage Bulk Load functionality. Bulk Load is the recommended approach for loading data into Hyperstage.

When cleared, data will be loaded using Insert/Update. Insert/Update is not recommended and will perform extremely slow.

About Log Files

Hyperstage uses the PostgreSQL server logs and also creates several new logs. For more information about PostgreSQL logs, see the *PostgreSQL 9.2 Documentation*.

Hyperstage Log Files	

Error log	Errors starting, stopping, and running the Hyperstage server. To generate this log, add the following lines to *my.cnf*:
	`log-error=<filename>`
	`log-output=FILE`

Hyperstage Log Files	
General query log	Connection and statement information received from clients.
Hyperstage log	Server start and stop information. Also contains missing configuration settings.

It is possible to turn on the display of diagnostic information. By default, this information is redirected to the Hyperstage console, unless an error log is specified (see table above). To turn on diagnostic messages you have to modify your infobright.cnf configuration file (see *Configuring Hyperstage* on page 22) and set parameter ControlMessages to 1 (log actions), 2 (to add a time stamp to each message), or 3 (to add memory and CPU resource information).

Note: In general, more detail in the log may have an impact on performance, so it is recommended that users find and use the setting that strikes the best balance in terms of performance versus log details.

About Errors

Hyperstage reports the same errors as the standard PostgreSQL server. For more information, see *PostgreSQL Error Codes* in the *PostgreSQL 9.2 Documentation*. There are a few additional errors specific to Hyperstage import and export commands.

About SQL Command Syntax

The syntax for Hyperstage SQL commands is exactly the same as the syntax for PostgreSQL commands. For more information, see the *PostgreSQL 9.2 Documentation*.

There are special considerations when using the following commands with Hyperstage. All other SQL commands can be used with Hyperstage as they are with the standard PostgreSQL.

Using PostgreSQL Commands With Hyperstage	
CREATE TABLE, DROP TABLE	*Creating and Dropping Tables* on page 35.
SHOW TABLE STATUS, SHOW FULL COLUMNS	*Viewing Compression Ratio Statistics* on page 41.
INSERT, UPDATE, DELETE	*Data Manipulation Statements* on page 43.

Using PostgreSQL Commands With Hyperstage	
LOAD DATA INFILE	*Importing and Exporting Data in Hyperstage* on page 51.
SELECT	*Running Queries in Hyperstage* on page 67.
VIEW	*Creating VIEWs in Hyperstage* on page 69.

About SQL ISO Standards

As mentioned in the previous section, Hyperstage uses the same syntax as the standard PostgreSQL commands. For information about the compliance of the PostgreSQL language with ISO SQL standards, see the *PostgreSQL 9.2 Documentation*.

Hyperstage is approaching full ISO SQL compliance. However, certain sections of the ISO SQL standard are open to interpretation and each DBMS, including Hyperstage, may implement these sections slightly differently. Consequently, Hyperstage query results may differ from those of other databases. For example, the SQL standard does not define a default collation for string comparisons, which affects the ordering of query results.

Chapter 4

Managing Hyperstage Tables

The following section describes how to work with the Hyperstage tables and lists the data types supported.

In this chapter:

About the Hyperstage Database Files

Hyperstage tables are located in the ib_data subdirectory in your Hyperstage installation directory. Within the ib_data subdirectory, Hyperstage databases are stored in separate subdirectories.

Important: Do not manually copy a data table from one database to another by copying the database files. Internal table numbering errors and Knowledge Grid inconsistencies may occur. To copy a table, use import and export commands (see *Importing and Exporting Data in Hyperstage* on page 51) or backup the entire database directory (see *Hyperstage Backup and Recovery* on page 73).

The following path and image shows the content of the ib_data directory, containing the Hyperstage databases webfocus and utf8test, as well as the BH_RSI_Repository directory, which holds the Knowledge Notes:

```
D:\dbms\Hyperstage\ib_data>dir
08/15/2014 05:11 PM <DIR> BH_RSI_Repository
08/18/2014 12:56 PM <DIR> utf8test
06/10/2014 03:16 PM <DIR> webfocus
```

About Supported Data Types

The following data types are supported in Hyperstage. Note that numeric data types ranges are 1 less than the PostgreSQL minimums and maximums.

Numeric Types		
BOOLEAN	Values are either 0 or 1.	
SMALLINT	-32767	32767
INT (INTEGER)	-2147483647	2147483647
BIGINT	-9223372036854775807	-9223372036854775807
REAL	-3.402823466E+38	3.402823466E+38
DOUBLE PRECISION	-1.7976931348623157E +308	1.7976931348623157E +308
Numeric(M, D) where: 0 < M <= 18 and 0 <= D <= M	-(1E+M - 1) / (1E+D)	(1E+M - 1) / (1E+D)

Date and Time Types			
DATE	100-01-01	9999-12-31	YYYY-mm-dd
Time (without timezone)	00:00:00	24:00:00	HH:MM:SS

Date and Time Types			
TIMESTAMP (without timezone)	100-01-01 00:00:00	9999-12-31 23:59:59	YYYY-mm-dd HH:MM:SS
TIME0053TAMP (with timezone)	1970-01-01 00:00:00 in UTC	2038-01-01 00:59:59 in UTC	
Interval	-178000000 years	178000000 years	

String Type	
BYTEA (binary string)	0 < N <= 65536
CHAR(N)	Fixed-length. Maximum length depends on character set (encoding). 0 < N * B <= 65536 where B is the maximum number of bytes for a single character.
VARCHAR(N)	Maximum length depends on character set (encoding). 0 < N * B <= 65536, where B is the maximum number of bytes for a single character. For example, for UTF-8 it is 4 bytes, so the maximum number of characters that can be stored in a (VAR)CHAR column is 65536 / 4 = 16384

Creating and Dropping Tables

Use the standard PostgreSQL commands to create and drop tables in Hyperstage, the same as you would with a PostgreSQL table. For detailed syntax information, see the *PostgreSQL 9.2 Documentation*.

Important: Do not manually copy a data table from one database to another by copying the database files. Internal table numbering errors and Knowledge Grid inconsistencies may occur. To copy a table from one database to another, export from the source database and then import into the target database (see *Importing and Exporting Data in Hyperstage* on page 51) or back up the entire database directory (see *Hyperstage Backup and Recovery* on page 73). You can rename the entire database by renaming the folder. However, you should not copy a database folder from one active instance to another, or within the same active instance.

To create a table, enter the following command:

```
psql> create table <table_name> (<column(s)>) with (ENGINE=INFOBRIGHT);
```

Note:

- 'with (ENGINE=INFOBRIGHT)' syntax is necessary when creating tables manually, to specify that the table will be stored as part of the Hyperstage-specific Infobright engine. Without this syntax, the table will be created and stored as a regular PostgreSQL table.
- When creating a table, as a matter of practice, you should always use the ENGINE= option to ensure that the correct database engine is used. Hyperstage is shipped with DEFAULT ENGINE=INFOBRIGHT, but this can be changed. The name of the engine can be specified explicitly at the end of the create table statement.

To drop a table, enter the following command:

```
psql> drop table table_name;
```

For information on supported and unsupported options when creating columns, see *About Column Options* on page 37.

Modifying Table Structures

Hyperstage supports common ALTER TABLE commands to add columns to existing tables and modify table structures, the same as you would with a PostgreSQL table. For detailed syntax information, see the *PostgreSQL 9.2 Documentation*.

- To add a column to an existing table, enter the following command:

  ```
  psql> ALTER TABLE table_name ADD [COLUMN] col_name col_definition;
  ```

- To add multiple columns at once, enter the following command:

  ```
  psql> ALTER TABLE table_name ADD [COLUMN] col1_name col1_definition,
  ADD [COLUMN] col2_name col2_definition;
  ```

- To remove the most recently added column from an existing table, enter the following command:

```
psql> ALTER TABLE table_name DROP [COLUMN] col_name;
```

Note: Hyperstage only supports DROP of the last added column for the purposes of *undo* capability. Any column DROP will be supported in a future release.

- To rename an existing table, enter the following command:

```
psql> ALTER TABLE tbl_name RENAME [TO] new_tbl_name;
```

About Column Options

Hyperstage supports NULL and NOT NULL specifications for columns.

- NULL allows NULL values for the column.
- NOT NULL replaces the imported NULL values with default values such as 0 (zero) for numeric columns and an empty string ('') for string columns.

LOOKUP Columns

Hyperstage provides an additional modifier for string data type columns, called a LOOKUP column. The LOOKUP column uses an integer substitution for values. You can declare a LOOKUP column on a CHAR or VARCHAR column to increase its compression and performance in queries. However, to use a LOOKUP column, the CHAR or VARCHAR column should meet the following criteria:

- There is no fixed upper limit for unique values in the column (cardinality), but the cardinality of the column should be low. The total size of a dictionary, being the total length of all distinct values, will be loaded into RAM (for example, 1 million distinct values that are each 100-characters wide will permanently occupy 100 MB of RAM).
- The column must contain a large number of duplicate values: the ratio of total number of records to distinct values should be greater than 10.

Typically, a LOOKUP column is useful for fields like state, gender, category, and similar fields where the number of instances is very high, but the number of unique values is very low. To determine the ratio of records to distinct values, determine the number of distinct values using SELECT COUNT (DISTINCT <COLUMN>) FROM… Then, compare this to the number of records using a SELECT COUNT(<COLUMN>) FROM…

Note: Using a LOOKUP on a column where there are more than 10,000 distinct values will result in greatly reduced load speeds.

To declare a column as a LOOKUP column, use the following example syntax when creating a table:

```
CREATE TABLE (a VARCHAR(200), b VARCHAR(200) LOOKUP=TRUE, c INTEGER) with
(engine=infobright);
```

In this example, a LOOKUP attribute is associated with columns **a** and **b**, but column **c** is a standard integer column.

Note:

- You can only declare a column as a LOOKUP modifier at the time the table is created. Modifying the column using ALTER TABLE to add or remove the LOOKUP modifier is not supported.
- In certain releases of Hyperstage, LOOKUP columns were called DIMENSION columns. Columns previously defined as DIMENSION columns will now be associated with LOOKUP columns.

  For backwards compatibility, LOOKUP columns can now be defined by either using the new syntax lookup=true, or the previous syntax dimension=true.
- Issuing a \d <table name> command will display whether the lookup modifier has been used for any column in the table.

Optimizing Columns for INSERTs

Hyperstage provides an additional modifier for columns to help optimize for INSERT operations, called a for_insert column. The for_insert modifier ensures that the most recent data pack is left uncompressed allowing for faster INSERTs in the case of a large number of single INSERTs with AUTOCOMMIT enabled, or small frequent LOADs (< 10000 rows each) with AUTOCOMMIT enabled.

If you are expecting a large number of individual INSERTs or small frequent LOADs with AUTOCOMMIT enabled, you should consider setting the for_insert modifier on character columns and large numeric columns (for example, 64-bit random identifiers or partNo). Small numeric columns (for example, color number or region ID) can be decompressed and recompressed with ease, and are unlikely to gain performance benefit from the for_insert modifier. For columns marked as lookup, the for_insert modifier may give very little benefit only. For smaller machines, you may wish to leave the for_insert modifier off in order to maximize compression for disk space.

Note: Currently, you can only set the for_insert modifier at the time of table creation. Modifying the column using ALTER TABLE to add or remove the for_insert modifier will be supported in a future release.

To declare a column as a for_insert column, add the comment 'for_insert' on the column. Enter the following command:

```
psql> create table …
(…
<<column name>> <<column type>> … comment 'for_insert'  …
…)
engine=infobright;
```

Issuing a SHOW CREATE TABLE command will display if the 'for_insert' modifier has been used for each column.

Unsupported Column Options

The following column options are ignored by Hyperstage:

- Default values.
- References to other tables.

Unsupported Indices Options

Hyperstage uses Knowledge Grid technology instead of standard indices and does not support explicit indices. The following elements of CREATE TABLE syntax related to indices are not allowed:

- Keys
- Indices
- Unique columns
- Auto-increment columns

Converting Oracle DDL to Hyperstage

If you have an existing Oracle schema definition, you should perform the following steps to make it work on Hyperstage:

- Convert MEDIUMTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert LONGTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert DOUBLE(A,B) to DECIMAL(A,B).
- INTEGER types may be converted to BIGINT.
- Convert VARCHAR2/CHAR2 to VARCHAR/CHAR.

Note: These steps are for converting DDL manually without having to use WebFOCUS Quickcopy or iWay software to migrate data.

Converting SQL Server to Hyperstage

If you have an existing SQL Server schema definition, you should perform the following steps to make it work on Hyperstage:

- Convert MEDIUMTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert LONGTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert DOUBLE(A,B) to DECIMAL(A,B).
- Convert MONEY to DECIMAL(18,4).
- Convert SMALLMONEY to DECIMAL(6,4).
- INTEGER types may be converted to BIGINT.
- NCHAR/NVARCHAR should be converted to CHAR/VARCHAR.
- Convert NUMBER to INTEGER.
- Convert NUMBER(A,B) to DECIMAL(A,B).

Note: These steps are for converting DDL manually without having to use WebFOCUS Quickcopy or iWay software to migrate data.

Converting PostgreSQL to Hyperstage

If you have an existing PostgreSQL schema definition, you should perform the following steps to ensure compliance with Hyperstage:

- Convert MEDIUMTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert LONGTEXT to VARCHAR (*N*), where *N* is only as large as necessary.
- Convert DOUBLE(A,B) to DECIMAL(A,B).

Note: These steps are for converting DDL manually without having to use WebFOCUS Quickcopy or iWay software to migrate data.

Viewing Compression Ratio Statistics

Hyperstage provides specific statistics on table and column compression. The compression ratio is calculated in relation to the *natural size* of uncompressed data in the table or column. The ratio equal to *n* means that the compressed data, including statistics and technical description of a column, is *n* times smaller than its theoretical natural size.

The following natural sizes (in bytes) are defined for various data types. Note the following:

- For all data types, if the column is not declared as NOT NULL, add 1-bit per value for NULL indicators.
- These data sizes take into account the typical format of data display (for example, yyyy-mm-dd for DATE or decimal point for DEC). The size also counts the bytes that store the actual text length (VARCHAR).

The natural size of the data type is approximately equal to the binary import/export format.

Data Types and Natural Sizes	
CHAR(*n*), BINARY(*n*)	*n**(number of rows)
BIGINT, INT, MEDIUMINT, SMALLINT, TINYINT, BOOL	(8 or 4 or 3 or 2 or 1 or 1)*(number of rows)
YEAR	4*(number of rows)
DATE	10*(number of rows)
TIME	8*(number of rows)
TIMESTAMP/DATETIME	19*(number of rows)
DEC(x,y)	(x+1)*(number of rows)
FLOAT	4*(number of rows)
REAL,DOUBLE	8*(number of rows)
VARCHAR(*n*), VARBINARY(*n*)	Total number of bytes used. For example, the total length of all strings, excluding terminating characters + 2*(number of rows).

Comparison of Calculated Compression Ratio to Physical Size

The compression ratio calculated above will differ from the compression ratio calculated from physical sizes of files on disk. The compression ratio based on physical size will be slightly smaller, due to extra files that are generated containing statistics on the imported data, such as Knowledge Nodes. Knowledge Nodes are used to optimize query execution and are discussed further in *About the Knowledge Grid* on page 67.

Chapter 5

Data Manipulation Statements

The following section describes the data manipulation statements that are supported by Hyperstage.

In this chapter:

Design of DML in Hyperstage

Hyperstage has been designed specifically for data warehousing applications, which are primarily load and read applications. Although Hyperstage supports INSERT, UPDATE, and DELETE, these constructs are designed for specific use cases.

The PostgreSQL Loader replicates a large bulk INSERT function and is typically used for loading small volumes of data where the format can vary, and where the higher level of error handling resilience is beneficial.

UPDATE is utilized for updating slowly changing dimensions. The DELETE function is ideal for the removal or archiving of older data from tables, and for the correction of invalid loads (the *oops* factor).

Hyperstage is not designed for OLTP type applications and its transaction model is limited. Using Hyperstage for an OLTP solution will result in poor performance, and incremental effort will be required to enforce referential integrity.

INSERT

Hyperstage supports the INSERT statement. For more information, see the *PostgreSQL 9.2 Documentation*.

```
INSERT [LOW_PRIORITY|DELAYED|HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES|VALUE} ({expr|DEFAULT},...),(...),...
```

Important: To use INSERT in bulk or batch load, you must set AUTOCOMMIT=0 and explicitly use COMMIT to complete the transaction. If AUTOCOMMIT=1, then each insert will result in the decompression and recompression of data packs, causing very slow performance. Explicit commits ensure that compression is only done once.

Inserting a Query Result in a PostgreSQL Table

- You can use the INSERT command to insert the result of a Hyperstage query into a PostgreSQL table. Enter the following command:

```
Autocommit=0;
insert into <psql_table> (<columns>) select <columns> from
<hyperstage_table> …;
commit;
```

- The following example shows the use of the INSERT command as described above:

```
psql> drop table if exists temp;
Query OK, 0 rows affected (0.00 sec)
psql> create table temp (sums int);
Query OK, 0 rows affected (0.00 sec)
psql> insert into temp (sums) select sum(i1) from tint;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0
psql> select * from temp;
+-----------+
| sums      |
+-----------+
| 87        |
+-----------+
1 row in set (0.00 sec)
```

The CREATE TABLE statement can be used in combination with a select statement to generate a series of INSERTs from one table into another. The format is as follows:

```
CREATE TABLE <table_name> with (ENGINE=INFOBRIGHT) AS (SELECT …);
```

This will result in the creation and population of a new table based on the SELECT criteria. There are a few things to look out for:

- Although the ENGINE=<*engine_name*> is optional, if not specified it will default to the default ENGINE for the database.

  Important: It is strongly recommended that you always include the ENGINE=<*engine_name*> in the CREATE TABLE statement.

- If the SELECT contains functions (for example, CONCAT), the output field attributes will be based on the defined output of the function. In the case of concat, the field attribute will be aggregate of the field sizes defined in the concat.

- The target column names, where functions are used, will be the function name unless an alias is provided:
  - Concat(a, b) will result in a target column name of Concat(a, b).
  - Concat(a, b) as c will result in a target column name of c.

UPDATE

Hyperstage supports the UPDATE statement. For more information, see the *PostgreSQL 9.2 Documentation*.

```
UPDATE tbl_name    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|
DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

AUTOCOMMIT should also be turned off during large UPDATE operations and the transaction explicitly committed as described in the INSERT statement section.

UPDATE can be used to maintain slowly changing dimensions, but if there are massive changes to the dimension, you might consider recreating the dimension with an ETL tool and simply dropping and reloading the dimension in the warehouse as this will improve performance.

DELETE

Hyperstage supports the DELETE statement. For more information, see the *PostgreSQL 9.2 Documentation*.

```
DELETE FROM tbl_name    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

AUTOCOMMIT is also important for good DELETE performance. DELETE is designed to remove the older data from a table and free up disk space. It should be used sparingly to randomly delete data over a large fact table as this may cause a performance impact over time.

Occasionally, data is incorrectly loaded to a fact table. DELETE can be used effectively in this case to remove the fresh incorrect data and replace it with the corrected data.

Chapter 6

Character Set Support

The following section describes the character sets supported by Hyperstage.

In this chapter:

Supported Character Sets

Hyperstage storage supports all ANSI and UTF-8 character sets. This means that Hyperstage can store and retrieve data encoded in 8-bit and multi-byte character sets.

Important: Queries that evaluate against UTF-8 character data columns will execute with less performance than an equivalent query against ASCII character data, due to ASCII support of Character Maps in the Knowledge Grid (see *Running Queries in Hyperstage* on page 67). UTF-8 specific Knowledge Grid extensions will be available in an upcoming release.

Collations and Comparisons

Hyperstage supports all custom UTF-8 collations supported by PostgreSQL:

❑ utf8_bin	❑ utf8_polish_ci
❑ utf8_czech_ci	❑ utf8_roman_ci
❑ utf8_danish_ci	❑ utf8_romanian_ci
❑ utf8_esperanto_ci	❑ utf8_slovak_ci
❑ utf8_estonian_ci	❑ utf8_slovenian_ci
❑ utf8_general_ci (default)	❑ utf8_spanish2_ci
❑ utf8_hungarian_ci	❑ utf8_spanish_ci
❑ utf8_icelandic_ci	❑ utf8_swedish_ci
❑ utf8_latvian_ci	❑ utf8_turkish_ci
❑ utf8_lithuanian_ci	❑ utf8_unicode_ci*
❑ utf8_persian_ci	

*utf8_unicode_ci properly handles both French and German collation, so specific collation types for these languages are not necessary.

For more information, see the *PostgreSQL 9.2 Documentation*.

The SQL standard does not define a default collation. Therefore, many DBMS engines have different default collations and produce different results. As a result, there are several differences between Hyperstage and other DBMS engines.

- ❑ For Hyperstage, character data types are case-sensitive. For example, the condition 'toronto'='Toronto' is not true in Hyperstage. Similarly, the condition, LIKE 'Abc%' is not true for 'abcde'.
- ❑ The Hyperstage sorting order is A…Z a…z (for example 'Zeta' < 'alfa'), which is the same sorting order as used by Oracle. The Hyperstage sorting order is different than the default PostgreSQL sorting order, which mixes lowercase and uppercase. The SQL Server order, which is aAbB…zZ; and the DB2 order, which is AaBb…Zz.

- The Hyperstage sorting order affects ORDER BY results, GROUP BY results (which is the order of groups and their definitions (for example, 'aaa' and 'AAA' define different groups) and DISTINCT results. WHERE conditions may also be affected if you are expecting a different sorting order than the one used by Hyperstage.
- To simulate Hyperstage collation in the PostgreSQL engine, set latin1_bin collation while creating a table (for more information, see the *PostgreSQL 9.2 Documentation*). Enter the following command:

  ```
  psql> create table … collate ascii_bin;
  ```

Padding

Hyperstage treats padding differently than other DBMS engines. Hyperstage assumes literal comparisons of text fields, including all whitespace characters. Therefore, a string containing two spaces is different than a string containing one space or an empty (0 length) string, which is also different than the NULL value.

The Hyperstage padding definition is compatible with the SQL standard. However, most DBMS systems have defined less restricted, customizable rules regarding text comparison. For example, 'abc ' = 'abc' may be true in some databases, but is not true in Hyperstage.

Note: In CHAR columns, trailing spaces are trimmed on LOAD, INSERT, and UPDATE, whereas in VARCHAR columns values are loaded with all spaces.

Chapter 7

Importing and Exporting Data in Hyperstage

The following section describes how to import data in Hyperstage.

In this chapter:

About Importing and Exporting Data

Hyperstage provides three ways to import data:

- INSERT statement
- Distributed Load Processor (DLP)
- COPY FROM statement

INSERT is described in *Data Manipulation Statements* on page 43 and is the slowest load approach. The DLP is the fastest load method but supports less load syntax.

Distributed Load Processor (DLP)

- Fastest loader
- Less error handling diagnostics (only the source file row number pertaining to the error is returned)

- Strict input file formats (supports delimited text and binary formats)
- Variable Data Pack size
- Load-time clustering

INSERT

- Supported by virtually all ETL tools
- Can be very slow depending upon the approach and commit rate

If you are using an ETL tool, using the DLP or COPY TO method with the binary format is most efficient, although this approach may require more data preparation. For large fact tables, using the DLP or COPY TO method with either binary or text input is recommended.

COPY FROM

- Supports capability similar to the DLP (does not support cluster on load or varying packrow size in this release)
- Primarily used for local instance only
- File-based loading
- Can work within transactions

Hyperstage COPY FROM Syntax

COPY FROM allows for very fast loading of file data in a single step. This is equivalent to LOAD DATA INFILE, for those familiar with MySQL. The COPY FROM syntax works in a manner similar to standard PostgreSQL (COPY FROM). However, there are differences in the options supported. The examples and table below outline these differences.

Usage Examples

```
copy tab1 from '/tmp/data' with (format txt_variable, lines_terminated_by
e'\n', delimiter ';')
copy tab1 from '/tmp/data' with (format infobright)
copy tab1 from '/tmp/data' with (format ib_binary)

COPY table_name
    FROM { 'filename' | STDIN }
    [ [ WITH ] ( option [, ...] ) ]
```

where:

option

Can be one of the following:

- format: {'txt_variable' | 'infobright' | 'ib_binary'}
- fields-terminated-by
- fields-enclosed-by
- escaped-by
- data-chartset 'encoding_name'
- lines_terminated_by
- reject-file-path
- abort-on-count
- abort-on-threshold
- input type {'client' | 'server'}
- timeout

PARAMETERS			
format		txt_variable infobright ib_binary	txt_variable
delimiter	fields-terminated-by	only a single one-byte character	\t
quote	fields-enclosed-by	only a single one-byte character	(empty)
escape	escaped-by	only a single one-byte character	(empty)

PARAMETERS			
encoding	data-chartset	only supported encodings by Hyperstage	(database encoding) *4-byte UTF-8 characters (CHAR, VARCHAR types) are replaced with question marks (?)
lines_terminated_by	lines-terminated-by		(empty)
reject_file_path	reject-file-path		(not specified)
abort_on_count	abort-on-count		(disabled)
abort_on_threshold	abort-on-threshold	in range (0,1)	(disabled)
pipe_mode	input-type	client server	(not used)
timeout	timeout		600s

Data Format (Mandatory)

You must set the data format parameter. Possible values are:

- **@txt_variable,** which is readable text
- **ib_binary,** which is a native binary representation, as found in *Hyperstage Binary Format* on page 60
- **infobright,** which is created by the DLP

Hyperstage Loader Reject File

By default, the COPY FROM command aborts on the first record that cannot be correctly parsed. However, in some cases, you may want the load process to continue and then later review rows that cannot be loaded. You can use the Reject File functionality to accomplish this.

Reject File is disabled, by default. To enable it, specify *reject_file_path*. This is the path to a file that will contain the rejected rows after load. You can set the number of records that can be rejected prior to the load being aborted and rolled back. To accomplish this, set the *abort_on_count* or *abort_on_threshold* parameter. If only *reject_file_path* is set, the processor will fail (terminate) on the first error and the row that was incorrect will be output to the reject file.

Usage example:

```
copy from '<path to file with data>' with (format txt_variable, …,
reject_file_path '<path to reject file>', abort_on_count 3)
```

The above command would fail and the load would be terminated if there were more than three incorrect rows in the input file. All rejected rows will be added to <path to reject file>.

HYPERSTAGE LOADER REJECT FILE OPTIONS	
reject_file_path	Path to the file where rejected rows are stored. Rejected rows are placed into the reject file in the order they are rejected. The original format is preserved to allow the operator to correct and rerun the load for only the rejected rows. **Note:** If reject_file_path is set, abort_on_count *or* abort_on_threshold must be set, as well.
abort_on_count	Abort and rollback the load if the number of rejected rows exceeds this value. If this value is not set, the load will be rolled back to the first bad record if the load fails. A value of -1 means never abort. A value of 0 means abort on first rejected row. There is no upper limit on this value. **Note:** abort_on_count and abort_on_threshold are mutually exclusive.
abort_on_threshold	Abort and rollback the load if the relative number of rejected rows to total processed rows exceeds this value (threshold test starts after one packrow row has been processed). Value must be in the range (0,1). This is an open interval. For example: set @ abort_on_threshold=0.01 / 0.5 / 0.99 means that 1% / 50% / 99% of all processed lines corrupted will terminate the Hyperstage Loader and save the problematic rows in the reject file. **Note:** abort_on_count and abort_on_threshold are mutually exclusive.

Importing Files With Invalid Values

Hyperstage may abort a load when invalid values are found. Certain invalid values, however, can be loaded in Hyperstage. The following rules are used with invalid data:

- If a numeric value is invalid, the value is replaced by 0.
- If a TIME, DATE, or TIMESTAMP is invalid, the value is replaced with a minimum value for the given data type.
- If a NULL value is imported into a column defined as NOT NULL (except for TIMESTAMP columns), it is replaced by 0 (for numerical, date, and time columns) or by an empty string (for string columns).

OPTIONS FOR DIFFERENT FORMATS				
oids	As in PostgreSQL	Not supported	Not supported	Not supported
null	As in PostgreSQL	Not supported	Not supported	Not supported
header	As in PostgreSQL	Not supported	Not supported	Not supported
force_quote	As in PostgreSQL	Not supported	Not supported	Not supported
force_not_null	As in PostgreSQL	Not supported	Not supported	Not supported
delimiter	As in PostgreSQL	Supported	Not supported	Not supported
quote	As in PostgreSQL	Supported	Not supported	Not supported
escape	As in PostgreSQL	Supported	Not supported	Not supported
encoding	As in PostgreSQL	Supported	Not supported	Not supported

OPTIONS FOR DIFFERENT FORMATS				
lines_terminated_by	Not supported	Supported	Not supported	Not supported
reject_file_path	Not supported	Supported	Not supported	Not supported
abort_on_count	Not supported	Supported	Not supported	Not supported
abort_on_threshold	Not supported	Supported	Not supported	Not supported
pipe_mode	Not supported	Supported	Supported	Supported
timeout	Not supported	Supported	Supported	Supported

Hyperstage COPY TO Syntax

COPY TO can be used for export Hyperstage table data to a file. It allows for fast exporting of data from a select statement. This is equivalent to SELECT INTO OUTFILE in MySQL.

The COPY TO syntax works in a manner similar to PostgreSQL (COPY TO), but supports a different set of options. The examples and table below outline these differences.

Usage Examples

```
copy (select ...) to '/tmp/data' with (format csv, delimiter ';')
copy (select ...) to '/tmp/data' with (format txt_variable,
lines_terminated_by e'\n', delimiter ';')
copy (select ...) to '/tmp/data' with (format ib_binary)
```

```
COPY { table_name [ ( column_name [, ...] ) ] | ( query ) }
    TO { 'filename' | PROGRAM 'command' | STDOUT }
    [ [ WITH ] ( option [, ...] ) ]
```

where:

option

Can be one of the following:

- format: {'txt_variable' | 'infobright' | 'ib_binary'}
- fields-terminated-by
- fields-enclosed-by
- escaped-by
- data-chartset 'encoding_name'
- lines_terminated_by
- reject-file-path
- abort-on-count
- abort-on-threshold
- input type {'client' | 'server'}
- timeout

For Hyperstage (ENGINE=INFOBRIGHT) tables, the following formats are supported:

- Hyperstage formats: **txt_variable, ib_binary**
- PostgreSQL formats: **text, csv, binary**

OPTIONS FOR DIFFERENT FORMATS			
oids	As in PostgreSQL	Not supported	Not supported
null	As in PostgreSQL	Supported	Not supported
header	As in PostgreSQL	Not supported	Not supported
force_quote	As in PostgreSQL	Not supported	Not supported
force_not_null	As in PostgreSQL	Not supported	Not supported
delimiter	As in PostgreSQL	Supported	Not supported

OPTIONS FOR DIFFERENT FORMATS			
quote	As in PostgreSQL	Supported	Not supported
escape	As in PostgreSQL	Supported	Not supported
encoding	As in PostgreSQL	Supported	Not supported
lines_terminated_by	Not supported	Supported	Not supported
reject_file_path	Not supported	Not supported	Not supported
abort_on_count	Not supported	Not supported	Not supported
abort_on_threshold	Not supported	Not supported	Not supported
pipe_mode	Not supported	Supported	Supported
timeout	Not supported	Not supported	Not supported

Single-character Delimiter

Hyperstage for PostgreSQL support single-character delimiters only.

About Transactions

About Transaction Behavior

While a write operation is being performed on a table, the following occurs:

- Queries to the table are executed against the state of the database before the write operation began. Once the current LOAD or INSERT/UPDATE/DELETE is complete and the operation is committed, then subsequent queries execute against the new state of the database.
- Until the current write operation is committed, all subsequent write commands to the table are queued. They will wait for the write lock to be released before proceeding in the order they were received.

While a read query is being executed on a table, the following occurs:

- All subsequent queries run concurrently with the current query.
- A subsequent LOAD or INSERT/UPDATE/DELETE will run concurrently with the current queries. Further write operations with queue (as described above).

In general, Hyperstage uses table-level locking where only one write operation (INSERT, UPDATE, DELETE, or LOAD) can execute at one time.

Failure Handling

If the Hyperstage server (ibengine) is terminated during an export operation to a disk file, the following occurs:

- A non-empty file is saved on disk. However, the last row in the saved file is inconsistent.
- The database files are not harmed by the failed export operation. To export data, repeat the export operation.

If Hyperstage tries to import data from a file created during a failed export session, the following occurs:

- No data is inserted because the input file consists of corrupted table rows. No new records are added to the database files, so no harm is done.

About Export Differences in Hyperstage

There are several important differences between exporting data from Hyperstage and exporting data from other DBMS engines.

Escape Characters

The Hyperstage and PostgreSQL Loaders support escape character definition and usage.

Exporting NULL Values

Hyperstage recognizes the following representations of NULL values when loading data from a text file:

```
NULL, \N, <field delimiter><field delimiter>
```

However, by default, Hyperstage only exports NULL values in the following representation:

```
\N
```

This can be modified using the null modifier in the COPY TO command.

Hyperstage Binary Format

With Hyperstage binary format load, individual rows are not separated by any special characters. There are also no value delimiters or qualifiers.

The structure of binary data files is as follows:

Data is stored contiguously: <row_size><nulls><data_col_1>...<data_col_n> and then the next data rows, without any line separator.

<row_size>	2-byte short integer indicating total number of bytes in this row (including all header bytes),
<nulls>	Binary map of null values, every byte reflecting to eight consecutive columns. Bit 0 means a normal value, bit 1 means null value. The length of <nulls> section is floor((number_of_columns+7)/8). For example, minimal number of bytes to cover the number of columns (one bit per column).
<data_col_1>	Data itself, depending on column type. ❏ Floating point values are stored here as 8-byte values. ❏ Most numerical values (for example, integers, dates) are stored as 4-byte integers. ❏ Fixed size texts (for example, CHAR(*n*)) are stored on the fixed number of n bytes. ❏ Other text types (for example, VARCHAR(*n*)) have their length stored on the first two bytes, followed by the text.

For example, we have two floating point columns. In this case, the binary file will look like the following:

```
11, 0, 0, a1, a2, a3, a4, a5, a6, a7, a8, b1, b2, b3, b4, b5, b6, b7, b8
```

where:

`(11, 0)`

Is the 2-byte (HEX) representation of the record length after the first 0. The second 0 is null map (no nulls in this case).

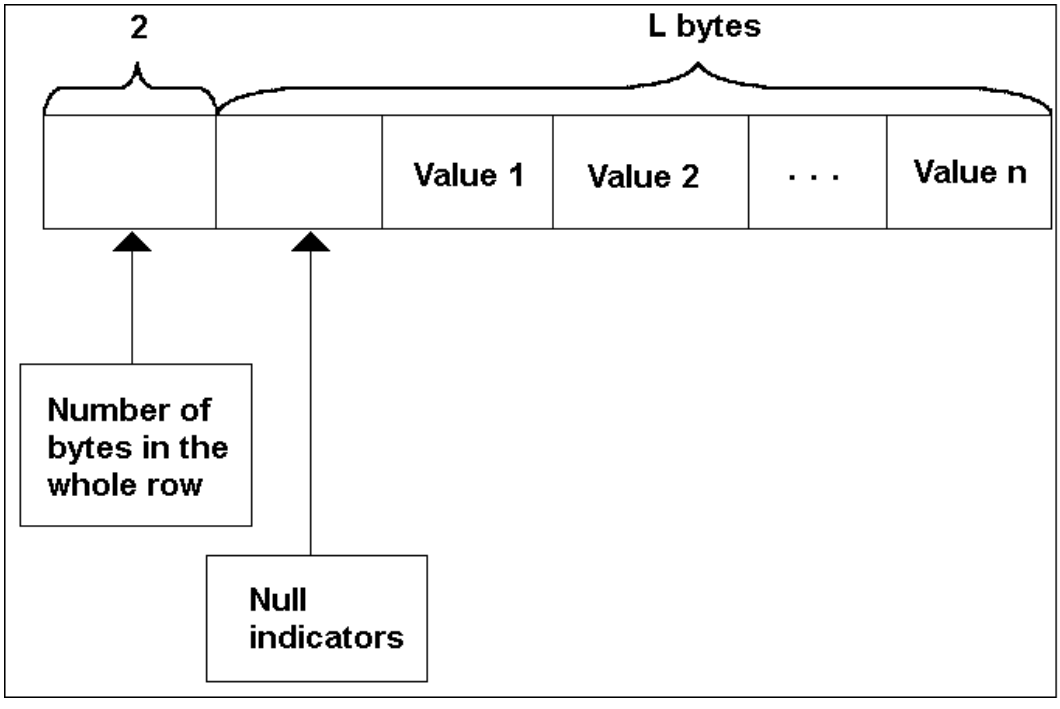
`(a1a2a3a4a5a6a7a8)`

Is an 8-byte representation of the first double.

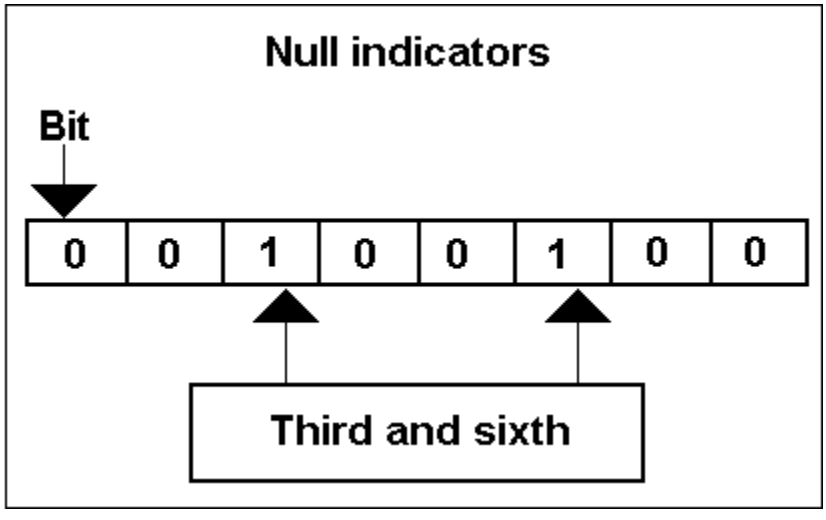
`(b1b2b3b4b5b6b7b8)`

Is an 8-byte representation of the second double. If the file contained 1000 rows, it will have a length of 19000 bytes.

The following schema illustrates the format of one row in the BINARY format.



Every row starts with L (2-byte integer) that specifies number of following bytes of data. Null indicators are an array of bits (one bit per each column). 1 on m-th bit means that the m-th value in the row is NULL.



The number of columns in a record determines the numbers of bytes in NULL indicators. For example, for a record that contains from one to eight column indicator bits are stored on one byte. If a record contains from nine to 16 columns, two bytes are used, and so on.

NULL indicators array is followed by *N* values, where *N* is a number of columns in a row.

Formats and Lengths in Bytes for Particular Data Types

SMALLINT		2
INTEGER		4
BIGINT		8
REAL	IEEE 4-byte Float	4
DOUBLE PRECISION	IEEE 8-byte Double	8
NUMERIC (N,M)	(Actual value) * 10^M	
TIME	[sign] [h] hh:mm:ss	8-10

Formats and Lengths in Bytes for Particular Data Types		
DATE	4-byte integer yyyymmdd where: yyyy Is the year (1900).	4
TIMESTAMP	yyyy-mm-dd hh:mm:ss	19
CHAR (N)	N characters	N
VARCHAR (N)	2-byte integer of value L followed by L characters	2+L
BYTEA (N)	2-byte integer of value L followed by L bytes	2+L

Note that CHAR is constant sized, whereas VARCHAR occupies only the size needed for the actual value. Integer and floating-point data are stored as a natural binary representation of these values (little endian).

Exporting and Importing Query Results

After exporting the results of a query to an output file, you may not be able to import the file back into the same definition of the accessed table. This is because the query may contain aggregates that will produce values beyond the boundaries of the original data types. In order to load the output file, you may need to create a new table with the appropriate data types for the values to be imported.

The following table shows the required data type conversions when using the binary format.

Data Type Conversions (Binary Format)		
SUM	Smallint Int Bigint	BigInt

Data Type Conversions (Binary Format)		
SUM	Float Double	Double
SUM	Numeric(N,M)	Decimal(18, M)
AVG	Smallint Int BigInt Double Numeric(N,M)	Double
COUNT	Smallint Int BigInt Double Numeric(N,M)	BigInt

Chapter 8

Running Queries in Hyperstage

The following section describes how to run queries and create VIEWs in Hyperstage.

In this chapter:

About the Knowledge Grid

The Knowledge Grid is a set of Hyperstage metadata used by the Hyperstage storage engine (named Infobright) to optimize query execution. The Knowledge Grid consists of Knowledge Nodes, which are optimization data for particular tables and columns. Knowledge Nodes are stored on disk in a special directory, specified in the infobright.cnf configuration file. Knowledge Nodes can be lost without losing data integrity.

About Knowledge Node

There are four kinds of Knowledge Nodes:

- **Histogram.** Used by Hyperstage to enhance the speed of most queries consisting of numerical conditions (including date/time, decimal, and so on).

  Histograms are created automatically during data load.

- **Character Map.** Used by Hyperstage to enhance the speed of most queries consisting of text conditions.

  Character Maps are created automatically during data load.

- **Pack/Pack.** Used to enhance joining of tables. Created or updated automatically while executing user queries.

- **Data Pack Nodes (DPN).** Statistical metadata that describes the content of the Data Pack. Used to both assist in data access and in rough operations.

DPNs are created automatically during data load.

Running Queries

To run queries on Hyperstage tables, use standard PostgreSQL syntax like in the simple request below:

```
psql> select [field name] from [table name];
```

The Hyperstage Optimizer is the primary engine used to resolve queries. While significant additions have been made to the library of supported SQL, there are cases where the query will still be executed by the PostgreSQL query engine, instead of the Hyperstage engine. In this event, query response time tends to suffer due to the fact that the PostgreSQL engine is row-oriented and therefore cannot make use of the Knowledge Grid information. In some cases, it can be too slow to be usable. For best performance, ensure your queries (and VIEWs) contain only syntax supported by the Hyperstage Optimizer.

Viewing Queries Redirected to the PostgreSQL Engine

When a query is redirected from the Hyperstage Optimizer to the PostgreSQL query engine, a warning is reported. For example:

```
400 rows in set, 1 warning (0.00 sec)
```

This will occur when functions not optimized in Hyperstage are used. If you get poor query performance, you should execute the command below to identify if a query has been directed to the PostgreSQL query engine.

After running a query, enter the following command to view any warnings:

```
psql> show warnings;
```

The following message indicates that the query was directed to PostgreSQL for processing:

```
1105 | Query syntax not implemented in Infobright, executed by PostgreSQL
engine.
```

Important: When queries are executed on Hyperstage tables by the standard PostgreSQL engine, performance can be significantly slower than when queries are executed by Hyperstage.

Preventing Queries From Redirecting to the PostgreSQL Engine

You can prevent queries from being redirected to the PostgreSQL query engine by editing the infobright.cnf file within the data directory:

```
AllowMySQLQueryPath=0
```

If the PostgreSQL query path is disabled, then the following message will be returned if the query would have been directed to PostgreSQL for processing:

```
The query includes syntax that is not supported by the Hyperstage
Optimizer. Hyperstage suggests either restructure the query with supported
syntax, or enable the PostgreSQL Query Path in the infobright.cnf file to
execute the query with reduced performance.
```

Terminating a Query

If you want to terminate a query executed from a client session before the query is complete, do the following:

1. Use the show [full] process list command to determine the process ID of the query.
2. Use the kill <id> command to terminate the query.

or

If you are using a command-line PostgreSQL client, you can also press Ctrl+C to terminate the query.

Creating VIEWs in Hyperstage

Hyperstage supports the creation of VIEWs. Note that the VIEW must contain Hyperstage optimized syntax, or the VIEW will be run in the PostgreSQL query engine.

Create VIEW Syntax

The syntax to create a VIEW is as follows:

```
CREATE
    [OR REPLACE]
    VIEW view_name [(column_list)]
    AS select_statement
```

A VIEW must contain unique column names. If you select two columns with the same name from separate tables, at least one must be aliased or the column list option must be used.

If the select statement of VIEW contains functionality that is not supported in the Hyperstage optimizer, then the VIEW will perform sub-optimally since it will always flip over to the PostgreSQL query engine.

SELECT Syntax Supported in Hyperstage

The following SELECT syntax is supported in Hyperstage.

SELECT Syntax

For more information, see the *PostgreSQL 9.2 Documentation*.

```
SELECT [ALL|DISTINCT|DISTINCTROW]
SELECT_expr, …
[FROM table_references[WHERE where_condition]
[GROUP BY {col_name|expr|position}]
[HAVING where_condition]
[ORDER BY {col_name|expr|position} [ASC|DESC ], …]
[LIMIT {[offset,] row_count|row_count OFFSET offset} ]
[INTO OUTFILE 'file_name' export_options- AS alias_name- ORDER BY NULL]
```

JOIN Syntax

For more information, see the *PostgreSQL 9.2 Documentation*.

Hyperstage supports the following JOIN syntax for the table_references part of SELECT statements (as described in *SELECT Syntax* on page 70).

```
table_references: table_reference [, table_references]
table_reference: table_factor | join_table

table_factor:
tbl_name [[AS] alias]

join_table:
table_reference [INNER|CROSS] JOIN table_factor [join_condition]
| table_reference STRAIGHT_JOIN table_factor
| table_reference STRAIGHT_JOIN table_factor ON condition
| table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition

join_condition:
ON conditional_expr | USING (column_list)
```

Union Syntax

For more information, see the *PostgreSQL 9.2 Documentation*.

```
SELECT ….
UNION [ALL|DISTINCT] SELECT …
[UNION [ALL|DISTINCT] SELECT … ]
```

Subqueries

For more information, see the *PostgreSQL 9.2 Documentation*.

```
SELECT * FROM t1 WHERE column1 = (SELECT max(column1) FROM t2);
```

The following functions are also supported:

- Subquery as scalar operand
- Subquery with ANY, IN, SOME, and ALL
- EXISTS and NOT EXISTS
- Correlated subqueries
- Subqueries in the FROM clause
- VIEWs in the FROM clause

Query Performance

Due to the Hyperstage column-oriented data organization and other Hyperstage-specific features, query optimization in Hyperstage is slightly different than in traditional DBMS approaches.

- Hyperstage works well with data tables containing many columns, where only necessary columns are accessed by query (as opposed to SELECT *). The traditional approach suggests keeping records as small as possible (for example, using schema normalization and table decomposition). However, in Hyperstage, only necessary columns are used in calculations. Therefore, queries with many limiting conditions on many columns of the same table are especially well optimized in Hyperstage.
- In traditional DBMS systems, better performance can be achieved by creating indices. In Hyperstage, Knowledge Nodes are used instead of indices (Knowledge Nodes are created automatically). To further enhance performance, you can try to influence the data loading procedure by keeping similar data (for example, for similar time frames) close together. The order in which data is loaded may influence both compression ratio and query speed.
- Avoid using OR in queries and, if possible, use IN instead. In some cases, ORs can be translated to UNION ALL or IN. For example:

  ```
  ...WHERE a=1 OR a=2...
  ```

  could be replaced by

  ```
  ...WHERE a IN (1,2)...
  ```

- Try to replace correlated subqueries with joins and independent subqueries.

- Executing queries in steps may also help with missing function support. For instance, execute the bulk of the query in Hyperstage and export the data to a PostgreSQL table. Then, execute the function query on the result set.

Temp tables may be used to manage intermediate steps without needing to do database cleanup.

To optimize your query performance, avoid the following, which will result in the query being handled by the PostgreSQL query engine:

- Using functions or type cast operators.
- Creating queries containing mixed Hyperstage and PostgreSQL tables.
- Performing comparisons or arithmetical operations on two different data types (such as numbers and text).
- Creating JOINs with the JOIN condition defined as NOT BETWEEN.

Chapter 9

Hyperstage Backup and Recovery

The following section provides instructions on how to backup and restore the Hyperstage databases.

In this chapter:

Backup Procedure

Use the following procedures to back up Hyperstage.

- To back up the Hyperstage databases, copy the ib_data and pg_data directories.
- You can take advantage of incremental backups, since only some of the database files are updated when new data is imported. Be sure to do a full backup occasionally.

Important: Some files in the KNFolder are updated when queries (using JOIN) are run, so be sure to back up the KNFolder on a regular basis.

Restore Procedure

To restore the Hyperstage databases from a backup copy, do the following:

1. Replace the ib_data and pg_data directories with the backup copy.
2. Replace the KNFolder with the backup copy (if the KNFolder is not inside the data directory).

Important: Do not manually modify database files or move them from one database to another. This may lead to data corruption and unpredictable results.

Appendix A

Functions and Operators

The following section lists the functions and operators supported by Hyperstage.

In this appendix:

Hyperstage Optimizer Supported Functions and Operators

Comparison Functions and Operators

COALESCE	YES

Control Flow Functions

CASE	YES
COALESCE	TBD
NULLIF	YES

String Functions

BIT_LENGTH	YES
CONCAT	YES
LEFT	YES
LENGTH	YES
LOCATE	YES
LOWER	YES

LPAD	YES
LTRIM	YES
OCTET_LENGTH	YES
POSITION	YES
RIGHT	YES
RPAD	YES
RTRIM	YES
SUBSTR	YES
TRIM	YES
TRUNC	TBD
UPPER	YES

Numeric Functions

Modulo (%)	YES

ABS	YES
ACOS	YES
ASIN	YES
ATAN2, ATAN	YES
ATAN	YES
CEIL	YES
COS	YES
COT	YES

DEGREES	YES
EXP	YES
FLOOR	YES
LN	YES
LOG	YES
MOD	YES
PI	YES
POWER	YES
RADIANS	YES
RANDOM	TBD
SIGN	YES
SIN	YES
SQRT	YES
TAN	YES

Date and Time Functions

CURRENT_DATE	YES
CURRENT_TIME	YES
DATE	YES
DAY	YES
DAYOFYEAR	YES
FROM_UNIXTIME	YES
HOUR	YES

MINUTE	YES
MONTH	YES
NOW	YES
QUARTER	YES
SECOND	YES
TIME	YES
YEAR	No

Text Search and Other Functions

CAST	YES
MD5	TBD

Group By Aggregate Functions

AVG	YES
BIT_OR	No
BIT_AND	No
COUNT(DISTINCT)	TBD
COUNT	YES
MIN	YES
MAX	YES
STD, STDDEV	YES
STDDEV_POP	YES
STDDEV_SAMP	YES

SUM	YES
VAR_POP	YES
VAR_SAMP	YES
VARIANCE	YES

Appendix B

Hyperstage Data Tools

The following section describes the data tools used by Hyperstage.

In this appendix:

Hyperstage Consistency Manager

Hyperstage provides a tool to validate Hyperstage-specific metadata structures. The Hyperstage Consistency Manager is an external stand-alone application that can be run against a Hyperstage instance to verify and repair most Hyperstage data structures, including the Knowledge Grid and Data Packs.

If you are seeing unexpected behavior with Hyperstage, such as server crashes, it can help to run the Hyperstage Consistency Manager for information for support and to perform repairs.

Note: Currently, the Hyperstage database must be offline in order to run the Hyperstage Consistency Manager.

Hyperstage Consistency Manager Tests

The Hyperstage Consistency Manager runs tests, as described in the following table.

Test	Description
Delete mask consistency check	Checks that the delete mask headers contain the proper sum for the delete mask body. If any inconsistency is found between the header and body, the Hyperstage Consistency Manager returns the list of blocks of delete mask where inconsistencies were found.

Test	Description
Number of objects in columns equality	Compares the stored number of objects in each column file related to the table. If any inconsistency is found in the number of objects, the Hyperstage Consistency Manager returns the first two columns with different object numbers.
Comparison of maximal value in DIMENSION dictionary versus DPN	Executes only for DIMENSION columns. Compares the maximal key value stored in the DIMENSION column dictionary and in DPNs. If the values differ, the Hyperstage Consistency Manager writes them to the log.
Comparison of number of objects in first-column DPN versus delete mask	Compares the metadata stored in the headers of the delete mask and DPN file related to the number of objects. If any inconsistencies are found, the Hyperstage Consistency Manager returns both numbers. The Hyperstage Consistency Manager compares only the first column because there is an independent test comparing this value between columns. If the test does not find the proper delete mask file or the proper DPN file, the Hyperstage Consistency Manager reports corruption.
Knowledge Grid consistency for column	Checks if the histograms report the proper value of the fixed parameter. A basic test of the Knowledge Node, ensuring the file has a proper format and the type of Knowledge Node corresponds to the column.
Knowledge Grid format for column	Each Knowledge Node is stored in a separate file. This test validates that the header data of each file is in the proper format.
Test for overlapping Data Packs in data files	Checks if there are Data Packs in files that overlap each other. If this situation occurs, the Hyperstage Consistency Manager returns a list of pairs of Data Packs numbers that are overlapping.
Tests of table metadata consistency	Verifies if the table metadata is valid. Includes verification of files used to store items, such as table name, number of columns and their names, types, and constrains like NOT NULL. These are the files created on CREATE TABLE and modified only on ALTER TABLE.

Test	Description
Test of DPNs for non-binary collation	Verifies Data Packs specifically for non-binary collation types (for example: Latin1_swedish.ci). If errors exist, they can be repaired using the Hyperstage Consistency Manager - -repair option.

Syntax:

How to Run the Hyperstage Consistency Manager

To view the run options, run Hyperstage Consistency Manager with the - -help flag:

```
Icm-pure --help
```

To run Hyperstage Consistency Manager, use the following command:

```
Icm-pure --datadir=/data_directory_path [parameters]
```

For example:

```
c:\ibi\srv77\home06Hyperstage\hs\bin>icm-pure.exe
  --datadir=C:\HyperstagePG\ib_data --log-file=C:\temp\icm-pure.log
```

Note: Hyperstage Consistency Manager should be run by the 'postgres' user. It should not be executed by 'root' or any rebuilt knowledge nodes will be owned by root (and cannot be edited), which will result in issues when loading any subsequent data into the 'corrected' tables.

The following table describes the Hyperstage Consistency Manager parameters.

Parameter	Description
- -help	Displays help message and exit.
-V [- - version]	Displays version information and exit.
- -basedir	Absolute path to Hyperstage installation directory.
arg	
- -datadir arg	Absolute path to directory. Mandatory.
- -database arg	Name of database chosen for data integrity testing. Optional. If specified, no other databases will be tested.

Parameter	Description
- -table arg	Name of table chosen for data integrity testing. Optional. If specified, no other tables will be tested.
- -log-file arg	Prints output to log file. Optional. If not specified, the logs will be printed to the console.
-F [- -full-check]	Runs full set of tests (may be time-intensive). Running Hyperstage Consistency Manager without the full-check option will result in a quicker test. However, the "Knowledge Grid consistency for column" test will not be run.
- -repair	Repairs found problems.
- -rebuild-kns	Rebuilds the Knowledge Grid. For more information, see *About Rebuilding or Repairing Knowledge Nodes* on page 84.
- -stop-on-error	Stops tests on first error and report.
- -cleanup	In case of an error in the Hyperstage Consistency Manager repair procedure, this option enables Hyperstage Consistency Manager to manually revert the datadir to its previous state. Running Hyperstage Consistency Manager with the - -cleanup option removes the old DPN files (containing incorrect DPNs) from the datadir and also makes the changes performed by Hyperstage Consistency Manager impossible to undo. If the - -cleanup option is not used, the old DPN files remain in the datadir.

About Rebuilding or Repairing Knowledge Nodes

Executing a rebuild of the Knowledge Nodes (using the - -rebuild-kns option) will run the following tests:

- Test of table metadata consistency
- Test of Knowledge Grid format for column
- Test of Knowledge Grid consistency for column

The - -rebuild-kns option will fix any issues found for the first two tests ("Test of table metadata consistency" and "Test of Knowledge Grid format for column").

You can also use the - -repair option along with the - -full-check option to achieve the same results as - -rebuild-kns. Using either of these methods will rebuild any Knowledge Nodes that have been deleted.

About Cleanup Procedures

The Hyperstage Consistency Manager creates backup files when repairing problems related to "Test of DPNs for non-binary collation" (backup files are not created for any other tests). These backup files can be used to revert back to the original data if the Hyperstage Consistency Manager encounters an error during the repair procedure. To revert to the original data, copy or rename the TAXXXXXDPN.icm_bck files to the TAXXXXXDPN.ctb files (found in the ib_data directory).

Hyperstage MySQL to PostgreSQL Migrator (“External Migrator”)

The Hyperstage External Migrator allows for migration of Hyperstage MySQL data to Hyperstage for PostgreSQL. The current version of the utility works under the following basic assumptions and conditions.

Assumptions and Conditions

- Migrates data from version 4 (latest Hyperstage MySQL) to data version 5 (latest PostgreSQL version).
- Destination data directories must be created for PostgreSQL.
- Migration of text types is supported under the following conditions (all conditions must be satisfied):
  - If UTF-8 is a charset in all text columns and no other charset is used.
  - If binary collations used.
  - If max text length from a column does not exceed 16K.
- Both PostgreSQL and the MySQL instances must be offline.
- Columns of time types must not contain 0 (zeros).
- Specific data types will require more space.
  - After the conversion to PostgreSQL, VARCHAR(n) types will require more than 64KB for a single value. Hyperstage for MySQL using UTF-8 may have to up 3 bytes whereas Hyperstage for PostgreSQL uses up to 4 bytes. The maximum value for n is 16K characters.

MySQL to PostgreSQL Data Type Mappings

The following table lists the data type mappings for MySQL to PostgreSQL.

MYSQL TO POSTGRESQL DATA TYPE MAPPINGS	
BOOL	SMALLINT
TINYINT	SMALLINT
MEDIUMINT	INT
INT	INT
BIGINT	BIGINT
FLOAT	REAL
DOUBLE	DOUBLE PRECISION
DECIMAL(M,N)	DECIMAL(M,N)
YEAR	(To be decided)
TIME	INTERVAL HOUR TO SECOND
DATE	DATE
DATETIME	TIMESTAMP WITHOUT TIME ZONE)
TIMESTAMP	TIMESTAMP WITH TIME ZONE)
CHAR(N)	CHAR(N)
VARCHAR(N)	VARCHAR(N)
TINYTEXT	VARCHAR(255)
TEXT	VARCHAR(N)
BINARY(N)	BYTEA(N)
VARBINARY(N)	BYTEA(N)

Limitations and Notes

- Table migration is done by copying the data. In-place migration is not supported.
- Tables with decomposition rules are currently not supported.
- The External Migrator will change all ‘0000-00-00’ DATE values to ‘100-01-01’.
- The External Migrator will change all ‘0000-00-00 00:00:00’ DATETIME and TIMESTAMP values to ‘100-01-01 00:00:00’ and ‘1970-01-01 00:00:00’.
- The External Migrator will apply a common character set to all columns being migrated. This is necessary because Hyperstage for PostgreSQL requires that all columns within a given database have the same character set.
- The External Migrator will recalculate Data Pack Nodes and Knowledge Nodes.
- Tables previously using LOOKUP columns will be migrated to DIMENSION columns.
- There is no support for Default values within PostgreSQL. Therefore, this modifier will not be migrated.

Working With the ibtop Tool

The ibtop tool provides monitoring of Hyperstage database operations and system resource usage. Use ibtop to monitor CPU usage, physical memory usage, disk I/O, cache directory size, query concurrency, and additional insightful metrics.

Note: Starting with release 5.0.1, ibtop utilizes a native C-API. Therefore, previously required modules, such as perl, are no longer needed to run ibtop.

ibtop is available for all supported OS platforms.

Command Options

A description of all command options available when running ibtop can be found by running the ibtop –help command. For example:

```
# ibtop --help
```

ibtop collects the statistics information of the running Hyperstage instance, which include the following:

- /proc/[pid]/stat for CPU/Memory usage under Linux or Windows equivalent.
- show status, such as 'IB%' (MySQL) or show Hyperstage statistics (postgres).

- show engine Hyperstage status (MySQL) or show ibengine status (postgres).
- (IMM only) additional multi-machine specific metrics.

You will not see the output on the screen. All collection goes to file specified by –output-file/-o or –output-dir with a name convention.

Command options include the following:

`-h [ --help]`
Shows the command usage message.

`-H [ --host ] arg`
The host on which the IB instance is running. The default is 127.0.0.1.

`-P [ --port ] arg`
The port number of the IB instance. The default is 5029.

`-L [ --login ] arg`
The sign-in username. Make sure it has the permission to show specific IB status and statistics. If left empty, ibtop will use postgres for the Hyperstage-PG instance, and root for the Hyperstage-MySQL instance.

`-p [ --password ] arg`
The sign-in password.

`-D [ --database ] arg`
The database to connect. If left empty, Hyperstage-PG will use template1, and Hyperstage-MySQL will use information_schema.

`-S [ --server-type ] arg`
The IB instance type. Valid options are:

- mysql
- postgres

The default value is postgres.

`-o [ --outputfile ] arg`
The output file name.

`-i [ --interval ] arg`
The interval, in seconds, between each collection. It can be as frequent as one (1) second. The default value is 60 seconds.

`-f [ --flush-on-intervals ] arg`
ibtop will keep collection *x* intervals, and flush into the output file. The default is 60 intervals for each flush.

`-R [ --output-directory ] arg`
The directory to hold the output files (JSON or CSV format). The default is /tmp (C:\tmp for Windows OS). If output-file is specified, output-file takes precedence. Otherwise, ibtop will use the following name convention:

/tmp/*hostname*.YYYYMMDD.HHMMSS.*SERVERTYPE_HOST_PORT*.hyperstage.[gpe | csv].

The .gpe file extension is in JSON format, if –enable-json-output is specified.

`-q [ --skip-header ]`
Skips header column names when the report is in CSV format. (This does not apply to JSON format.)

`-b [ --abort-on-error ]`
Aborts collection on error (for example, a lost connection to the underlying IB instance). If left empty, ibtop keeps trying indefinitely to reconnect, and all values in collection will be 0.

`-G [ --enable-json-output ]`
Enables JSON format output, while turning off CSV output. For details on JSON output format, see *Format of JSON Output* on page 96.

`-g [ --debug ]`
Debug mode. Shows more verbose messages to help the ibtop developer.

`-c [ --config-file ] arg`
The configuration file path.

`-v [ --version ]`
Shows the current ibtop version.

Running ibtop

The ibtop tool can be run either remotely or locally on the same server that the Hyperstage database engine is running. To run ibtop and collect database instance metrics, enter a command, such as the following:

```
$ ibtop -H 192.168.20.105 -P 5030 -L root -S postgres -i 1 -f 10 -o /tmp/
colo105.ibtop.csv
```

Based on the above command, after 10 seconds (1-second (interval) * 10 (flush-on-interval)), information will be written to the file /tmp/colo105.ibtop.csv.

Note: Once output is collected, it can be opened directly in any spreadsheet software (CSV), json reader (JSON), or loaded into a database like Hyperstage for analysis. By default, for CSV, the first line contains the column header. You can omit the column header by specifying –skip-header=yes in the command line.

Reference: Using a Configuration File to Protect a Database Password

As an alternative to entering ibtop options at the command line, it is also possible to specify them using a configuration file. This may be especially useful in order to protect passwords from command line sniffing.

For example, in order to run ibtop using a configuration file (called ibtop.cnf), enter the following command:

```
$ ibtop --config-file=ibtop.cnf
```

A sample configuration file is shown below. To use it, it is necessary to uncomment, by removing the leading # character from each line, and specifying the appropriate parameter value.

```
###==== begin of ibtop.cnf =====
#host=127.0.0.1
#port=5029
#login=
#password=
#database=
#server-type=postgres
#output-file=
#interval=60
#flush-on-intervals=60
#output-directory=/tmp
#skip-header=no
#abort-on-error=no
#enable-json-output=no
###==== end of ibtop.cnf =====
```

Collecting Database Process CPU/Memory Utilization From the Operating System

ibtop is able to optionally collect database process CPU/Memory utilization information from the operating system. To enable this functionally, the following must be true:

- ibtop must be run locally.
- For Linux operating systems, ibtop must be run using the same OS user at the database instance (for example, postgres) or as a super user (for example, root).
- For Windows operating systems, ibtop must be run as a super user (for example, administrator).

Note: Under Linux, the information collection is achieved by executing the cat /proc/[IB instance pid]/stat command. Under Windows, the information collection is achieved by issuing the GetProcessTimes() and GetProcAddress() function calls.

Collecting Hyperstage Statistics

Inside the Hyperstage engine, a global data cache is accessible by all threads. The global data cache consists of three heaps:

- Main Heap
- Large Temporary Heap
- System Heap

All IB data structures are inherited from a base class called TrackableObject, with the acronym TO. All data structures will allocate on one of three heaps. The type of TrackableObject could be one the following listed examples:

- **TO_PACK:** The actual compressed/decompressed data pack. By default, each pack contains 65536 elements for a column.
- **TO_SPLICE:** Metadata information including min/max/null/sum of each pack. Because a single DPN structure is small, the engine will store them in splices.
- **TO_RSINDEX:** Mirror of files from BH_RSI_Repository. It can be CMAP (Character Map) for a string column or HIST (Histogram) for a numeric column.
- **TO_FTREE:** Mirror for lookup dictionary file (for example, $datadir/table.bht/TA#####.dic)
- **TO_SORTER:** Temporary structure used by a query when sorting is needed.
- **TO_CACHEDBUFFER+TO_INDEXTABLE:** Temporary structure used by a query.
- **TO_FILTER:** Delete mask.
- **TO_TEMPORARY:** Miscellaneous temporary structure used by a query, (for example, aggregation work buffer, join buffer, and so on).
- **others:** everything else.

ibtop collects the above heap and *TO* metrics for both size (in bytes) and block count.

Summary of Information Collected by ibtop

The following tables describe all the information that is collected by ibtop.

ibtop Information - General

Variable Name	Note
TimeStamp	ibtop always uses UTC timestamp (for example, 2016-04-28 13:21:46 +0000)
UniqueId	Identifier if you collect multiple ibtop. It takes command line server_type + remote_ip + port

ibtop Information - DB Instance CPU/Memory usage from the OS

Variable Name	Note
PID	/proc/[pid]/stat column #1 pid
NumThreads	/proc/[pid]/stat column #20 num_threads
UserCPU	/proc/[pid]/stat column #14+#16 utime (include child process). The collected value is per-second usage.
SystemCPU	/proc/[pid]/stat column #15+#17 stime (include child process). The collected value is per-second usage.
VmSize	/proc/[pid]/stat column #23 vmsize
VmRSS	/proc/[pid]/stat column #24 rss

ibtop Information - from the *show Hyperstage statistics* command

Variable Name	Note
IB_gdc_false_wakeup	Metric to measure efficiency of internal global data cache.

Variable Name	Note
IB_gdc_hits	Number of retrievals, which are obtained from memory directly, without going through the disk reading and decompression process.
IB_gdc_load_errors	Metric to measure efficiency of internal global data cache.
IB_gdc_misses	Number of retrievals, which are not from the data cache, but go through the disk reading and decompression process.
IB_gdc_pack_loads	Number of DataPack loaded and cached.
IB_gdc_prefetched	Metric to measure efficiency of internal global data cache.
IB_gdc_read_wait_in_progress	Metric to measure efficiency of internal global data cache.
IB_gdc_readwait	Metric to measure efficiency of internal global data cache.
IB_gdc_redecompress	obsoleted
IB_gdc_released	Objects release from global data cache.
IB_mm_alloc_blocs	Blocks allocated, per second.
IB_mm_alloc_objs	Number of objects doing allocating, per second.
IB_mm_alloc_pack_size	Bytes allocated by Datapack objects, per second.
IB_mm_alloc_packs	Blocks allocated by Datapack objects, per second.
IB_mm_alloc_size	Allocated memory bytes, per second.

Variable Name	Note
IB_mm_alloc_temp	Number of temporary blocks allocated, per second.
IB_mm_alloc_temp_size	Temporary allocation bytes, per second.
IB_mm_free_blocks	Number of blocks deallocated, per second.
IB_mm_free_pack_size	Bytes-per-second of blocks deallocated by Datapack objects.
IB_mm_free_packs	Blocks-per-second deallocated by Datapack objects.
IB_mm_free_size	Bytes-per-second deallocated.
IB_mm_free_temp	Number of temporary blocks deallocated, per second.
IB_mm_free_temp_size	Bytes of temporary deallocation, per second.
IB_mm_freeable	Total allocated memory that is currently in the releasable state.
IB_mm_release1	Dependent on specific object release algorithm.
IB_mm_release2	Dependent on specific object release algorithm.
IB_mm_release3	Dependent on specific object release algorithm.
IB_mm_release4	Dependent on specific object release algorithm.
IB_mm_reloaded	Number of times a datapack was loaded after eviction, but before falling off the history list.

Variable Name	Note
IB_mm_scale	Integer factor representing the magnitude of maximum buffer sizes that can be allocated in a query.
IB_mm_unfreeable	Total allocated memory that is currently in the un-releasable state.
IB_readbytes	Read from disk, in bytes-per-second.
IB_readcount	Read operation count, in count-per-second.
IB_writebytes	Write to disk, in bytes-per-second.
IB_writecount	Writer operation count, in count-per-second.

ibtop Information - from *show ibengine status* command

Variable Name	Note
System Heap Total(size)	Size (in bytes) for heaps, trackable objects.
Main Heap Total(size)	Size (in bytes) for heaps, trackable objects.
Large Temporary Heap(size)	Size (in bytes) for heaps, trackable objects.
TO_PACK objects(size)	Size (in bytes) for heaps, trackable objects.
TO_SORTER objects(size)	Size (in bytes) for heaps, trackable objects.
TO_CACHEDBUFFER+TO_INDEXTABLE objects(size)	Size (in bytes) for heaps, trackable objects.
TO_FILTER objects(size)	Size (in bytes) for heaps, trackable objects.
TO_RSINDEX objects(size)	Size (in bytes) for heaps, trackable objects.
TO_SPLICE objects(size)	Size (in bytes) for heaps, trackable objects.
TO_TEMPORARY objects(size)	Size (in bytes) for heaps, trackable objects.
TO_FTREE objects(size)	Size (in bytes) for heaps, trackable objects.

Variable Name	Note
other objects(size)	Size (in bytes) for heaps, trackable objects.
System Heap Total(block)	Block count for heaps, trackable objects.
Main Heap Total(block)	Block count for heaps, trackable objects.
Large Temporary Heap(block)	Block count for heaps, trackable objects.
TO_PACK objects(block)	Block count for heaps, trackable objects.
TO_SORTER objects(block)	Block count for heaps, trackable objects.
TO_CACHEDBUFFER+TO_INDEXTABLE objects(block)	Block count for heaps, trackable objects.
TO_FILTER objects(block)	Block count for heaps, trackable objects.
TO_RSINDEX objects(block)	Block count for heaps, trackable objects.
TO_SPLICE objects(block)	Block count for heaps, trackable objects.
TO_TEMPORARY objects(block)	Block count for heaps, trackable objects.
TO_FTREE objects(block)	Block count for heaps, trackable objects.
other objects(block)	Block count for heaps, trackable objects.
cache_folder_size	This metric is the summary size (in bytes) from all files under CacheFolder, which is defined in infobright.cnf. IB instance uses it to store temporary intermediate results if there is not enough memory.

Format of JSON Output

When ibtop option enable-json-output=yes, the output file is in JSON format. Each JSON output file has two level-1 sections: meta and data.

- The meta section contains 1 variable named *internval*. This value is equal to the input variable –interval.

- The data section contains a series of key-value collections. The first key is the timestamp. The value of this key is a nested structure of statistics. The content of this structure is similar to the following:

```
ibtop ->
      instance_unique_id ->
                  trend
                  tag
                  config
```

To generate instance_unique_id, ibtop concatenates server_type, IP, and port. This combination provides a unique identifier in the event of monitoring or comparing two distinct ibtop outputs.

In the nested structure for a given instance_unique_id, ibtop collects the following:

- **trend:** The collected metrics. This is the same set of data as CSV output.
- **tag**: The fixed element. This value will always be IBTOP@HYPERSTAGE.
- **config:** The Hyperstage instance configuration parameters. For example, ServerMainHeapSize.

An example of JSON output is shown below:

```
{
     "meta": {
         "interval": 1
     },
     "data": {
         "2016-05-02 17:27:56 +0000": {
             "ibtop": {
                 "postgres_127_0_0_1_5029": {
                     "trend": {
                        "gdc_false_wakeup": 0,
                        "gdc_hits": 0,
                        "gdc_load_errors": 0,
                        "gdc_misses": 0,
                        "gdc_pack_loads": 0,
                        "gdc_prefetched": 0,
                        "gdc_read_wait_in_progress": 0,
                        "gdc_readwait": 0,
                        "gdc_redecompress": 0,
                        "gdc_released": 0,
                        "mm_alloc_blocs": 0,
                        "mm_alloc_objs": 0,
                        "mm_alloc_pack_size": 0,
                        "mm_alloc_packs": 0,
                        "mm_alloc_size": 0,
                        "mm_alloc_temp": 0,
                        "mm_alloc_temp_size": 0,
                        "mm_free_blocks": 0,
                        "mm_free_pack_size": 0,
                        "mm_free_packs": 0,
                        "mm_free_size": 0,
                        "mm_free_temp": 0,
                        "mm_free_temp_size": 0,
```

```
        "mm_freeable": 0,
        "mm_release1": 0,
        "mm_release2": 0,
        "mm_release3": 0,
        "mm_release4": 0,
        "mm_reloaded": 0,
        "mm_scale": 5,
        "mm_unfreeable": 4,
        "readbytes": 0,
        "readcount": 0,
        "writebytes": 0,
        "writecount": 0,
        "largetemporaryheap_block": 0,
        "largetemporaryheap_size": 0,
        "mainheap_block": 12,
        "mainheap_size": 4,
        "numthreads": 0,
        "pid": 0,
        "systemheap_block": 0,
        "systemheap_size": 0,
        "systemcpu": 0,
        "cachedbuffer_indextable_block": 0,
        "cachedbuffer_indextable_size": 0,

        "filter_block": 6,
        "filter_size": 0,
        "ftree_block": 0,
        "ftree_size": 0,
        "pack_block": 4,
        "pack_size": 0,
        "rsindex_block": 0,
        "rsindex_size": 0,
        "sorter_block": 0,
        "sorter_size": 0,
        "splice_block": 1,
        "splice_size": 0,
        "temporary_block": 0,
        "temporary_size": 0,
        "usercpu": 0,
        "vmrss": 0,
        "vmsize": 0,
        "cache_folder_size": 0,
        "other_block": 1,
        "other_size": 4
    },
    "tag": {
        "add": [
            "IBTOP@HYPERSTAGE"
        ]
    },
```

```
                        "config": {
                            "CfgName": "postgres_127_0_0_1_5029",
                            "CfgCollectionInterval": "1",
                            "CacheFolder": "cache",
                            "ConnectTimeout": "5",
                            "FET": "0",
                            "FETInterval": "0",
                            "HandshakeTimeout": "15",
                            "IBEngineRevision":
                            "Hyperstage_4.8.3_r35390_36166",
                            "KNFolder": "BH_RSI_Repository",
                            "KNLevel": "1",
                            "LicenseFile": "hyperstage.lic",
                            "LoaderSaveThreads": "16",
                            "LogLevel": "W",
                            "LogRotateFiles": "9",
                            "LogRotateSize": "250",
                            "MemoryHardLimit": "0",
                            "MemoryLargeTempPercentage": "20",
                            "MemoryScaleReduction": "0",
                            "ParallelAggrThreads": "1024",
                            "ParallelJoinThreads": "1024",
                            "ParallelScanDPsAtOnce": "1",
                            "ParallelScanDPsPerThread": "10",
                            "ParallelScanThreads": "1024",
                            "ParallelSortThreads": "1024",

                            "PeerCommitTimeout": "120",
                            "PrefetchQueueLength": "18",
                            "PrefetchThreads": "6",
                            "ServerMainHeapSize": "8834",
                            "ServerMainHeapThreshold": "5",
                            "SpliceSize": "128",
                            "SyncBuffers": "0",
                            "ThrottleLimit": "0",
                            "ThrottleScheduler": "0",
                            "ses_LogLevel": ""
                        }
                    }
                }
            },
            "2016-05-02 17:27:57 +0000": {...},
            "2016-05-02 17:27:58 +0000": {...},
            "2016-05-02 17:27:59 +0000": {...}
        }
}
```


Create Hyperstage Table Syntax for CSV Output

When the ibtop output file is in CSV format, which is the default, one option for further analysis is to load the data into a Hyperstage table. You can do this by accommodating the *create table* syntax. An example of this is shown below:

```
create table ibtop_collection_hyperstage (
"timestamp" varchar(32),
uniqueid varchar(64),
pid int,
numthreads int,
usercpu int,
systemcpu int,
vmsize int,
vmrss int,
ib_gdc_false_wakeup int,
ib_gdc_hits int,
ib_gdc_load_errors int,
ib_gdc_misses int,
ib_gdc_pack_loads int,
ib_gdc_prefetched int,
ib_gdc_read_wait_in_progress int,
ib_gdc_readwait int,
ib_gdc_redecompress int,
ib_gdc_released int,
ib_mm_alloc_blocs int,
ib_mm_alloc_objs int,
ib_mm_alloc_pack_size int,
ib_mm_alloc_packs int,
ib_mm_alloc_size int,
ib_mm_alloc_temp int,
ib_mm_alloc_temp_size int,
ib_mm_free_blocks int,
ib_mm_free_pack_size int,
ib_mm_free_packs int,
ib_mm_free_size int,
ib_mm_free_temp int,
ib_mm_free_temp_size int,
```

```
ib_mm_freeable int,
ib_mm_release1 int,
ib_mm_release2 int,
ib_mm_release3 int,
ib_mm_release4 int,
ib_mm_reloaded int,
ib_mm_scale int,
ib_mm_unfreeable int,
ib_readbytes int,
ib_readcount int,
ib_writebytes int,
ib_writecount int,
system_heap_total_size int,
main_heap_total_size int,
large_temporary_heap_size int,
to_pack_objects_size int,
to_sorter_objects_size int,
to_cachedbuffer_to_indextable_objects_size int,
to_filter_objects_size int,
to_rsindex_objects_size int,
to_splice_objects_size int,
to_temporary_objects_size int,
to_ftree_objects_size int,
other_objects_size int,
system_heap_total_block int,
main_heap_total_block int,


large_temporary_heap_block int,
to_pack_objects_block int,
to_sorter_objects_block int,
to_cachedbuffer_to_indextable_objects_block int,
to_filter_objects_block int,
to_rsindex_objects_block int,
to_splice_objects_block int,
to_temporary_objects_block int,
to_ftree_objects_block int,
other_objects_block int,
cache_folder_size int ) with (engine=hyperstage);
```

Using the External Migrator

To use the External Migrator, run the following command:

```
./ibextmigrator options
```

The available External Migrator options are listed in the following table.

Option	Description
-h [- -help]	Prints help messages.

Option	Description
-f [- -force]	Continues migration, even if an error occurs.
-v [- -verbose]	Shows more details.
-b [- -pg-bin] arg	PostgreSQL installation path.
-u [- -pg-user] arg	PostgreSQL user used to create the migration database.
-s [- -src-datadir] arg	Source MySQL datadir.
-i [- -dst-ibdatadir] arg	Destination Hyperstage Server datadir (ib_data).
-p [- -dst-pgdatadir] arg	Destination PostgreSQL datadir (pg_data).
-d [- -dst-db] arg	Destination PostgreSQL database name.
-t [- -tables] arg	List of tables to migrate in the form "db1.t1 db2.t3 db2.*" where * implies migration of every table in the database. If not specified, the External Migrator will attempt to migrate the entire datadir.
-c [- -dst-schema] arg (=public)	Name of destination schema to which tables specified with the -t option should be migrated. Defaults to public.
- -connection-db arg (=template1)	Database that External Migrator will use to connect to PostgreSQL.
- -force-charset-conversion [=arg(=utf8)]	Specifying this option will turn off the option to check if all data selected for migration has a common character set, and will trigger the conversion to the specified character set, if necessary. The default character set is UTF8. You can also use this option to trigger conversion of all data to specified charset. **Note:** Using this setting will significantly increase the time that it takes to complete the migration.
- -version	Print program version number and exit.

The following code is an example of the migration command for all the tables within a MySQL database named salesdatabase, to a PostgreSQL database named salesdatabase.

```
c:\ibi\srv77\home06Hyperstage\hs\bin>ibextmigrator.exe
-b "C:\ibi\srv77\home06Hyperstage\hs\bin" -u srvadmin
-s "C:\HyperstageMySQL\Data"
-p "C:\HyperstagePG\pg_data"
-i "C:\HyperstagePG\ib_data"
-d salesdatabase
-t "salesdatabase.*"
```

The following code is an example of the migration command for all of the tables within a MySQL database named salesdatabase, to a PostgreSQL database named salesdatabase. In this example, the option to check if all data selected for migration has a common character set is also included, and will force the character set conversion to UTF8.

```
c:\ibi\srv77\home06Hyperstage\hs\bin>ibextmigrator.exe
-b "C:\ibi\srv77\home06Hyperstage\hs\bin" -u srvadmin
-s "C:\HyperstageMySQL\Data"
-p "C:\HyperstagePG\pg_data"
-i "C:\HyperstagePG\ib_data"
-d salesdatabase
-t "salesdatabase.*"
--force-charset-conversion=utf8
```

Feedback

Customer success is our top priority. Connect with us today!

Information Builders Technical Content Management team is comprised of many talented individuals who work together to design and deliver quality technical documentation products. Your feedback supports our ongoing efforts!

You can also preview new innovations to get an early look at new content products and services. Your participation helps us create great experiences for every customer.

To send us feedback or make a connection, contact Sarah Buccellato, Technical Editor, Technical Content Management at *Sarah_Buccellato@ibi.com.*

To request permission to repurpose copyrighted material, please contact Frances Gambino, Vice President, Technical Content Management at *Frances_Gambino@ibi.com.*

Information Builders

WebFOCUS

Hyperstage for PostgreSQL Reference Guide

WebFOCUS Reporting Server Release 8.2 DataMigrator Server Release 7 Version 7.07

DN4501642.1016

Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898