

Oracle® Big Data SQL

User's Guide



Release 4.1
F22883-02
October 2020

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Big Data SQL User's Guide, Release 4.1

F22883-02

Copyright © 2012, 2020, Oracle and/or its affiliates.

Primary Author: Frederick Kush, Lauran Serhal, Drue Swadener

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	ix
Documentation Accessibility	ix
Related Documents	ix
Conventions	ix
Backus-Naur Form Syntax	x
Changes in This Release	x

1 Introducing Oracle Big Data SQL

1.1 What Is Oracle Big Data SQL?	1-1
1.1.1 About Oracle External Tables	1-2
1.1.2 About the Access Drivers for Oracle Big Data SQL	1-2
1.1.3 About Smart Scan for Big Data Sources	1-2
1.1.4 About Storage Indexes	1-3
1.1.5 About Predicate Push Down	1-5
1.1.6 About Pushdown of Character Large Object (CLOB) Processing	1-6
1.1.7 About Aggregation Offload	1-8
1.1.8 About Oracle Big Data SQL Statistics	1-9
1.2 Installation	1-12

2 Using Oracle Big Data SQL for Data Access

2.1 Creating External Tables	2-1
2.1.1 About the SQL CREATE TABLE Statement	2-1
2.1.1.1 Basic Syntax	2-1
2.1.1.2 About the External Table Clause	2-2
2.1.2 Creating an Oracle External Table for Hive Data	2-3
2.1.2.1 Obtaining Information About a Hive Table	2-4
2.1.2.2 Using the CREATE_EXTDDL_FOR_HIVE Function	2-4
2.1.2.3 Using Oracle SQL Developer to Connect to Hive	2-6
2.1.2.4 Developing a CREATE TABLE Statement for ORACLE_HIVE	2-8
2.1.2.5 Hive to Oracle Data Type Conversions	2-10

2.1.3	Creating an Oracle External Table for Oracle NoSQL Database	2-11
2.1.3.1	Creating a Hive External Table for Oracle NoSQL Database	2-11
2.1.3.2	Creating the Oracle Database Table for Oracle NoSQL Data	2-13
2.1.3.3	About Oracle NoSQL to Oracle Database Type Mappings	2-13
2.1.3.4	Example of Accessing Data in Oracle NoSQL Database	2-14
2.1.4	Creating an Oracle External Table for Apache HBase	2-17
2.1.4.1	Creating a Hive External Table for HBase	2-17
2.1.4.2	Creating the Oracle Database Table for HBase	2-18
2.1.5	Creating an Oracle External Table for HDFS Files	2-18
2.1.5.1	Using the Default Access Parameters with ORACLE_HDFS	2-18
2.1.5.2	ORACLE_HDFS LOCATION Clause	2-19
2.1.5.3	Overriding the Default ORACLE_HDFS Settings	2-20
2.1.6	Creating an Oracle External Table for Kafka Topics	2-22
2.1.6.1	Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics	2-23
2.1.6.2	Creating an Oracle Big Data SQL Table for Kafka Topics	2-26
2.1.7	Creating an Oracle External Table for Object Store Access	2-27
2.1.7.1	Create Table Example for Object Store Access	2-30
2.1.7.2	Accessing a Local File through an Oracle Directory Object	2-31
2.1.7.3	Conversions to Oracle Data Types	2-32
2.1.7.4	ORACLE_BIGDATA Support for Compressed Files	2-35
2.2	Querying External Tables	2-36
2.2.1	Granting User Access	2-36
2.2.2	About Error Handling	2-36
2.2.3	About the Log Files	2-36
2.2.4	About File Readers	2-36
2.2.4.1	Using Oracle's Optimized Parquet Reader for Hadoop Sources	2-37
2.3	About Oracle Big Data SQL on the Database Server (Oracle Exadata Machine or Other)	2-37
2.3.1	About the bigdata_config Directory	2-38
2.3.2	Common Configuration Properties	2-38
2.3.2.1	bigdata.properties	2-38
2.3.2.2	bigdata-log4j.properties	2-40
2.3.3	About the Cluster Directory	2-41
2.3.4	About Permissions	2-41
2.4	Oracle SQL Access to Kafka	2-41
2.4.1	About Oracle SQL Access to Kafka	2-42
2.4.2	Get Started with Oracle SQL Access to Kafka	2-42
2.4.3	Register a Kafka Cluster	2-43
2.4.4	Create Views to Access CSV Data in a Kafka Topic	2-44
2.4.5	Create Views to Access JSON Data in a Kafka Topic	2-45
2.4.6	Query Kafka Data as Continuous Stream	2-46

2.4.7	Exploring Kafka Data from a Specific Offset	2-48
2.4.8	Exploring Kafka Data from a Specific Timestamp	2-49
2.4.9	Load Kafka Data into Tables Stored in Oracle Database	2-49
2.4.10	Load Kafka Data into Temporary Tables	2-50
2.4.11	Customize Oracle SQL Access to Kafka Views	2-50
2.4.12	Reconfigure Existing Kafka Views	2-51

3 Information Lifecycle Management: Hybrid Access to Data in Oracle Database and Hadoop

3.1	About Storing to Hadoop and Hybrid Partitioned Tables	3-1
3.2	Use Copy to Hadoop	3-2
3.2.1	What Is Copy to Hadoop?	3-2
3.2.2	Getting Started Using Copy to Hadoop	3-3
3.2.2.1	Table Access Requirements for Copy to Hadoop	3-3
3.2.3	Using Oracle Shell for Hadoop Loaders With Copy to Hadoop	3-4
3.2.3.1	Introducing Oracle Shell for Hadoop Loaders	3-4
3.2.4	Copy to Hadoop by Example	3-4
3.2.4.1	First Look: Loading an Oracle Table Into Hive and Storing the Data in Hadoop	3-4
3.2.4.2	Working With the Examples in the Copy to Hadoop Product Kit	3-7
3.2.5	Querying the Data in Hive	3-9
3.2.6	Column Mappings and Data Type Conversions in Copy to Hadoop	3-9
3.2.6.1	About Column Mappings	3-9
3.2.6.2	About Data Type Conversions	3-9
3.2.7	Working With Spark	3-10
3.2.8	Using Oracle SQL Developer with Copy to Hadoop	3-11
3.3	Enable Access to Hybrid Partitioned Tables	3-11
3.4	Store Oracle Tablespaces in HDFS	3-12
3.4.1	Advantages and Limitations of Tablespaces in HDFS	3-12
3.4.2	About Tablespaces in HDFS and Data Encryption	3-13
3.4.3	Moving Tablespaces to HDFS	3-14
3.4.3.1	Using bds-copy-tbs-to-hdfs	3-15
3.4.3.2	Manually Moving Tablespaces to HDFS	3-18
3.4.4	Smart Scan for TableSpaces in HDFS	3-21

4 Oracle Big Data SQL Security

4.1	Accessing Oracle Big Data SQL	4-1
4.2	Multi-User Authorization	4-1
4.2.1	The Multi-User Authorization Model	4-2
4.3	Sentry Authorization in Oracle Big Data SQL	4-2

4.3.1	Sentry and Multi-User Authorization	4-3
4.3.2	Groups, Users, and Role-Based Access Control in Sentry	4-3
4.3.3	How Oracle Big Data SQL Uses Sentry	4-4
4.3.4	Oracle Big Data SQL Privilege-Related Exceptions for Sentry	4-4
4.4	Hadoop Authorization: File Level Access and Apache Sentry	4-5
4.5	Compliance with Oracle Database Security Policies	4-5

5 Working With Query Server

5.1	About Oracle Big Data SQL Query Server	5-1
5.2	Important Terms and Concepts	5-2
5.3	Query Server Features	5-2
5.4	Specifying the Hive Databases to Synchronize With Query Server	5-3
5.4.1	Specifying the Hive Databases in the bds-config.json Configuration File	5-4
5.4.2	Updating the Hive Databases With the sync_hive_db_list Configuration Parameter	5-5
5.5	Synchronizing Query Server With Hive	5-5
5.5.1	Restarting Query Server Manually by Using Cloudera Manager	5-6
5.5.2	Synchronizing Query Server Manually by Using Cloudera Manager	5-6
5.5.3	Synchronizing Query Server Manually by Using the PL/SQL API	5-7
5.5.4	Enabling Query Server Full Synchronization	5-7
5.6	Query Server Restarts and Metadata Persistence	5-8
5.7	Query Server Security	5-8

6 Oracle Big Data SQL Reference

6.1	CREATE TABLE ACCESS PARAMETERS Clause	6-1
6.1.1	Syntax Rules for Specifying Properties	6-1
6.1.2	ORACLE_HDFS Access Parameters	6-2
6.1.2.1	Default Parameter Settings for ORACLE_HDFS	6-2
6.1.2.2	Optional Parameter Settings for ORACLE_HDFS	6-3
6.1.3	ORACLE_HIVE Access Parameters	6-3
6.1.3.1	Default Parameter Settings for ORACLE_HIVE	6-4
6.1.3.2	Optional Parameter Values for ORACLE_HIVE	6-4
6.1.4	Full List of Access Parameters for ORACLE_HDFS and ORACLE_HIVE	6-4
6.1.4.1	com.oracle.bigdata.buffersize	6-5
6.1.4.2	com.oracle.bigdata.datamode	6-5
6.1.4.3	com.oracle.bigdata.colmap	6-6
6.1.4.4	com.oracle.bigdata.erroropt	6-7
6.1.4.5	com.oracle.bigdata.fields	6-8
6.1.4.6	com.oracle.bigdata.fileformat	6-10
6.1.4.7	com.oracle.bigdata.log.exec	6-11

6.1.4.8	com.oracle.bigdata.log.qc	6-12
6.1.4.9	com.oracle.bigdata.overflow	6-12
6.1.4.10	com.oracle.bigdata.rowformat	6-13
6.1.4.11	com.oracle.bigdata.tablename	6-15
6.1.5	ORACLE_BIGDATA Access Parameters	6-16
6.2	Static Data Dictionary Views for Hive	6-24
6.2.1	ALL_HIVE_DATABASES	6-24
6.2.2	ALL_HIVE_TABLES	6-25
6.2.3	ALL_HIVE_COLUMNS	6-26
6.2.4	DBA_HIVE_DATABASES	6-27
6.2.5	DBA_HIVE_TABLES	6-27
6.2.6	DBA_HIVE_COLUMNS	6-27
6.2.7	USER_HIVE_DATABASES	6-27
6.2.8	USER_HIVE_TABLES	6-28
6.2.9	USER_HIVE_COLUMNS	6-28
6.3	DBMS_BDSQL PL/SQL Package	6-28
6.3.1	ADD_USER_MAP	6-29
6.3.2	REMOVE_USER_MAP	6-30
6.3.3	Multi-User Authorization Security Table	6-31
6.4	DBMS_BDSQS_ADMIN PL/SQL Package	6-33
6.5	DBMS_HADOOP PL/SQL Package	6-33
6.5.1	CREATE_EXTDDL_FOR_HIVE	6-33
6.5.1.1	Example	6-34

Part I Appendices

A Manual Steps for Using Copy to Hadoop for Staged Copies

A.1	Generating the Data Pump Files	A-1
A.1.1	About Data Pump Format Files	A-1
A.1.2	Identifying the Target Directory	A-2
A.1.3	About the CREATE TABLE Syntax	A-2
A.2	Copying the Files to HDFS	A-3
A.3	Creating a Hive Table	A-3
A.3.1	About Hive External Tables	A-3
A.4	Example Using the Sample Schemas	A-4
A.4.1	About the Sample Data	A-4
A.4.2	Creating the EXPDIR Database Directory	A-4
A.4.3	Creating Data Pump Format Files for Customer Data	A-4
A.4.3.1	CREATE TABLE Example With a Simple SELECT Statement	A-5

A.4.3.2	CREATE TABLE Example With a More Complex SQL SELECT Statement	A-5
A.4.4	Verifying the Contents of the Data Files	A-5
A.4.5	Copying the Files into Hadoop	A-6
A.4.6	Creating a Hive External Table	A-6

B Using Copy to Hadoop With Direct Copy

B.1	Manual Steps for Using Copy to Hadoop for Direct Copies	B-1
B.2	Copy to Hadoop Property Reference	B-4

C Using mtactl to Manage the MTA extproc

D Diagnostic Tips and Details

D.1	Running Diagnostics with bdschecksw	D-1
D.2	How to do a Quick Test	D-4
D.3	Oracle Big Data SQL Database Objects	D-5
D.4	Other Database-Side Artifacts	D-7
D.5	Hadoop Datanode Artifacts	D-13
D.6	Step-by-Step Process for Querying an External Table	D-14
D.7	Step-by-Step for a Hive Data Dictionary Query	D-17
D.8	Key Administration Tasks for Oracle Big Data SQL	D-18
D.9	Additional Java Diagnostics	D-20
D.10	Checking for Correct Oracle Big Data SQL Patches	D-21
D.11	Debugging SQL.NET Issues	D-21

E Oracle Big Data SQL Software Accessibility Recommendations

E.1	Tips on Using Screen Readers and Braille Displays	E-1
E.2	Tips on Using Screen Magnifiers	E-2

Index

Preface

The *Oracle Big Data SQL User's Guide* describes how to use and manage the Oracle Big Data SQL product.

Audience

This guide is intended for administrators and users of Oracle Big Data SQL, including:

- Application developers
- Data analysts
- Data scientists
- Database administrators
- System administrators

The guide assumes that the reader has basic knowledge of Oracle Database single-node and multinode systems, the Hadoop framework, the Linux operating system, and networking concepts.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

See the *Oracle Big Data SQL Installation Guide* for instructions on installing the product.

See the *Oracle Big Data Appliance Owner's Guide* for information about using the Oracle Big Data SQL with Oracle Big Data Appliance.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
# prompt	The pound (#) prompt indicates a command that is run as the Linux root user.

Backus-Naur Form Syntax

The syntax in this reference is presented in a simple variation of Backus-Naur Form (BNF) that uses the following symbols and conventions:

Symbol or Convention	Description
[]	Brackets enclose optional items.
{ }	Braces enclose a choice of items, only one of which is required.
	A vertical bar separates alternatives within brackets or braces.
...	Ellipses indicate that the preceding syntactic element can be repeated.
delimiters	Delimiters other than brackets, braces, and vertical bars must be entered as shown.
boldface	Words appearing in boldface are keywords. They must be typed as shown. (Keywords are case-sensitive in some, but not all, operating systems.) Words that are not in boldface are placeholders for which you must substitute a name or value.

Changes in This Release

The following are changes in this and previous releases.

- [Changes in Oracle Big Data SQL 4.1](#)
- [Changes in Oracle Big Data SQL 4.0](#)
- [Changes in Oracle Big Data SQL 3.2](#)

Changes in Oracle Big Data SQL 4.1

The following are new features and updates in Oracle Big Data SQL 4.1.

Oracle SQL Access to Kafka

Oracle SQL Access to Kafka enables Oracle external tables to query data from Kafka topics. This feature allows Kafka data to be queried and joined with data in Oracle Database tables without requiring Hive metadata. See [Oracle SQL Access to Kafka](#).

Object Store Reader Enhancements

The following ORACLE_BIGDATA driver updates are available:

- Support for Apache ORC file format. [Creating an Oracle External Table for Object Store Access](#).
- Support for complex data types with ORC, Parquet and Avro. This capability was supported in Hadoop, and is now supported with ORACLE_BIGDATA. See [Creating an Oracle External Table for Object Store Access](#).
- Enhancements to Text Driver
New text-based access parameters available. See [ORACLE_BIGDATA Access Parameters](#).

Support for Microsoft's Azure Blob Object Storage

Microsoft's Azure Blob Storage is supported in this release. The ORACLE_BIGDATA driver has been extended to support the creation of external tables that map to data stored in Azure Blob storage. See [Creating an Oracle External Table for Object Store Access](#).

Supports Oracle Database 19c

The Oracle Big Data SQL 4.1 release supports Oracle Database 19c, along with earlier Oracle Database 12c and 18c releases. For more information refer to [Oracle Help Center for Oracle Database 19c](#)

 **Note:**

Sentry is not supported for Big Data SQL installations on Oracle Database 12.1 systems.

- Hybrid Partitioned Tables (HPT)
See [Information Lifecycle Management: Hybrid Access to Data in Oracle Database and Hadoop](#).
- In-Memory External Tables
See In-Memory External Tables in *Oracle Database Database In-Memory Guide 19c*.

Installer Improvements

- Alternate Kerberos Principals
On the database side of the Big Data SQL installation, Release 4.1 supports Kerberos authentication via the principal of a user other than the database owner. The --alternate-principal parameter is described in Command Line Parameter Reference for bds-database-install.sh in *Oracle Big Data SQL Installation Guide*.
- Alternate Repositories for Cloudera Enterprise 6 Release (CDH 6)
On CDH 6 systems only, you can now specify an arbitrary repository for the installer to acquire the Hadoop, Hive, and HBase clients needed to complete the Oracle Big Data SQL installation on the database side. The --alternate-repo

parameter is described in Command Line Parameter Reference for `bds-database-install.sh` in *Oracle Big Data SQL Installation Guide*.

- **Support for Non-Default Hive and Hadoop Owners and Groups**
Release 4.1 includes new Jaguar configuration parameters to identify the correct user and/or group for HDFS and Hive daemons when these have been changed from the defaults. See `hadoop_ids` in Jaguar Configuration Parameter and Command Reference in *Oracle Big Data SQL Installation Guide*.

Changes in Oracle Big Data SQL 4.0

The following are new features and updates in Oracle Big Data SQL Release 4.0.

Support for Oracle Database 18c as well as Backward Compatibility for Oracle Database 12.2 and 12.1

To take advantage of the new capabilities in Oracle Big Data SQL 4.0, you need use Oracle Database 18c or later. However, use of Oracle Database 12.1 and 12.2 is fully supported (even though you can't leverage the new 4.0 capabilities with these database versions). This backward compatibility enables you to install and administer release 4.0 in a mixed environment that includes both Oracle Database 18c and 12c.

Big Data SQL Query Server

Big Data SQL Query Server is a lightweight, zero-maintenance Oracle Database. It gives you an easy way to query data in Hadoop without the need for a full-fledged Oracle Database service. The services consist of the Oracle SQL query engine only. It provides no persistent storage except for certain categories of metadata that are useful to retain across sessions.

- **Installs Automatically and Requires no Maintenance**

Big Data SQL Query Server is included as part of the standard Oracle Big Data SQL installation. The only thing you need to provide is the address of an edge node where you would like the service installed. The installation itself is fully automated and requires no post-installation configuration.

- **Provides Single and Multi-User Modes**

The service provides two modes – single-user and multi-user. Single-user mode utilizes a single user for accessing the Query Server. All users connect to the Query Server as the BDSQL user with the password specified during the installation. In multi-user mode Hadoop cluster users log into the Query Server using their Kerberos principal.

- **Works with Kerberos, Automatically Imports Kerberos Principals**

A Kerberos-secured cluster can support both single user and multi-user mode.

During installation on a secured cluster, the installer automatically queries the KDC to identify the Kerberos principals and then sets up externally identified users based on the principals. After installation, the administrator can manually add or remove principals.

- **Resets to Initial State After Each Query Server Restart**

Each time Big Data SQL Query Server is restarted, the database instance is reset to the original state. This also happens if a fatal error occurs. This reset enables you to start again from a “clean slate.” A restart preserves external tables (both

ORACLE_HIVE and HDFS types), associated statistics, and user-defined views. A restart deletes regular tables containing user data

- **Can be Managed Through Hortonworks Ambari or Cloudera Manager**

Big Data SQL Query Service is automatically set up as a service in Ambari or Cloudera Manager. You can use these administrative tools to monitor and stop/start the process, view warning, error, and informational messages, and perform some Big Data SQL Query Service operations such as statistics gathering and Hive metadata import.

Query Server is provided under a limited use license described in Oracle Big Data SQL Licensing in *Oracle Big Data SQL Installation Guide*.

New ORACLE_BIGDATA Driver for Accessing Object Stores

In addition to ORACLE_HIVE and ORACLE_HDFS, release 4.0 also includes the new ORACLE_BIGDATA driver. This driver enables you to create external tables over data within object stores in the cloud. Currently Oracle Object Store and Amazon S3 are supported. ORACLE_BIGDATA enables you to create external tables over Parquet, Avro, and text files in these environments. For development and testing, you can also use it to access local data files through Oracle Database directory objects. The driver is written in C and does not execute any Java code.

In release 4.0, ORACLE_BIGDATA supports the return of scalar fields from Parquet files. More complex data types as well as multi-part Parquet files are not supported at this time. Because the reader does not support complex data types in the Parquet file, the column list generated omits any complex columns from the external table definition. Most types stored in Parquet files are not directly supported as types for columns in Oracle tables.

Oracle Big Data SQL's Smart Scan, including the new aggregation offload capability, work with object stores by offloading data from the object store to processing cells on the Hadoop cluster where Oracle Big Data SQL is installed.

Authentication against object stores is accomplished through a credential object that you create using the DBMS_CREDENTIAL package. You include the name of the credential object as well as a location URI as parameters of the external table create statement.



See Also:

[Creating an Oracle External Table for Object Store Access](#) which provides create statement examples as well as conversion tables for Parquet and Avro data types to Oracle data types.

Aggregation Offload

Oracle Big Data SQL can now utilize Oracle In-Memory technology to offload aggregations to the Oracle Big Data SQL cells. Oracle Big Data SQL leverages the processing power of the Hadoop cluster to distribute aggregations across the cluster nodes. The performance gains can be significantly greater than for aggregations that do not offload, especially when there are a moderate number of summary groupings.

Oracle Big Data SQL cells support single table and multi-table aggregations (for example, dimension tables joining to a fact table). For multi-table aggregations, the

Oracle Database uses the key vector transform optimization in which the key vectors are pushed to the cells for the aggregation process. This transformation type is useful for star join sql queries that use typical aggregation operators (for example, SUM, MIN, MAX, and COUNT) which are common in business queries.

**See Also:**

["About Aggregation Offload"](#)

Sentry Authorization in Oracle Big Data SQL

In addition to supporting authorization for HDFS file access, Oracle Big Data SQL supports Sentry policies, which authorize access to Hive metadata. Sentry enables fine-grained control over user access, down to the column level.

**See Also:**

Sentry Authorization in Big Data SQL in the *Oracle Big Data SQL Installation Guide*.

Installer Improvements

- The Jaguar installer provides easy installation of the optional Query Server database. Several new parameters are added to the Jaguar configuration file for the installation of this component.
- Oracle Big Data SQL now includes its own JDK. You no longer need to download it from the Oracle Technology Network. Other versions of the JDK may be present, but do not change the JDK path that Oracle Big Data SQL uses.
- The installer now validates principals entered in the Kerberos section of the configuration file against the corresponding keytab file and flags an error if these do not match.
- Cluster edge nodes are automatically excluded from the requirements pre-check.
- In the installation pre-check, hardware factors (cores and memory) are validated only on nodes where Oracle Big Data SQL processing cells will be installed.
- On the database side, the install now validates the subnet (for InfiniBand connections), the LD_LIBRARY_PATH, and the hostnames of Hadoop systems on the other side of the connection.
- In an uninstall on the database side, the operation now removes all Oracle Big Data SQL artifacts from the database server and reverts all changes to `cellinit.*ora` and database parameters.
- The Jaguar `updatenodes` operation is deprecated in this release. Use `reconfigure` instead to change cluster settings, create database-side install bundles, and expand or shrink the configuration.
- Two new scripts to help predetermine readiness for installation.

Prior to installing the Hadoop side of Oracle Big Data SQL, you can run `bds_node_check.sh` on each DataNode of the cluster to check if the node meets the installation prerequisites.

Prior to installing on the Oracle Database system, you can run `bds-validate-grid-patches.sh` to ensure that Oracle Grid includes the patches required by the Oracle Big Data SQL release.

- The script `bds_cluster_node_helper.sh`, which you can run on each Hadoop node, provides status on the Oracle Big Data SQL installation on the node and also collects log data and other information useful for maintenance. There are three options for the scope of the log data collection.

Changes in Oracle Big Data SQL 3.2

Oracle Big Data SQL Release 3.2 includes major improvements in performance, secure network connectivity, authentication, and user administration, as well as installation and configuration.

JSON CLOB Predicate Pushdown

Much improved filtering and parsing of JSON CLOB data in Hadoop enables Oracle Big Data SQL to push more processing for these large objects down to the Hadoop cluster. JSON Data can now be filtered on the Oracle Big Data SQL cells in Hadoop for CLOB columns up to 1 MB, depending on character set of the input document. The eligible JSON filter expressions for storage layer evaluation include simplified syntax, `JSON_VALUE`, and `JSON_QUERY`. In addition, Oracle Big Data SQL can project up to 32 KB of CLOB data from select list expression evaluation in Hadoop to Oracle Database. Processing falls back to Oracle Database only when column sizes exceed these two values.

Customers can disable or re-enable this functionality to suit their own needs.

In Release 3.2, this enhancement currently applies only to JSON expressions returning CLOB data. The same support will be provided for other CLOB types (such as `substr` and `instr`) as well as for BLOB data in a future release.

 **Note:**

The new JSON CLOB predicate pushdown functionality requires Oracle Database version 12.1.0.2.180417 or greater, as well as the following patches:

- The April 2018 Proactive DBBP (Database Bundle Patch). This is patch 27486326.
- The one-off patch 27767148.

Install the one-off patch on all database compute nodes.

The one-off patch 26170659, which is required on top of earlier DBBPs, is not required on top of the April DBBP.

This functionality is not available through the January 2018 and August 2017 Proactive DBBPs

See the [Oracle Big Data SQL Master Compatibility Matrix](#) (Doc ID 2119369.1 in [My Oracle Support](#)) for the most up-to-date information on software version and patch requirements.

Support for Querying Kafka Topics

Release 3.2 provides Hive and Oracle Big Data SQL the ability to query Kafka topics via a new Hive storage handler. You can use this storage handler to create external Hive tables backed by data residing in Kafka. Oracle Big Data SQL or Hive can then query the Kafka data through the external tables. The Kafka key, value, offset, topic name, and partition id are mapped to Hive columns. You can explicitly designate the offset for each topic/partition pair, otherwise the offset will start from the earliest offset in the topic and end with the latest offset in the topic for each partition.

Improved Processing of Parquet Files

Oracle has introduced its own Parquet reader for processing data in Parquet format. This new reader provides significant performance and resource utilization improvements over the existing Hive Parquet driver. These include:

- More intelligent column processing retrieval. The reader uses “lazy materialization” to process only columns with rows that satisfy the filter, thereby improving I/O.
- Leveraging of dictionaries during filter predicate processing to improve CPU usage.
- Streamlined data conversion, which also contributes to more efficient CPU usage.

The Big Data SQL installation enables the Oracle's Parquet reader by default. You have the option to disable it and revert to the generic Parquet reader.

Multi-User Authorization

In previous releases of Oracle Big Data SQL, all queries against Hadoop and Hive data are executed as the `oracle` user and there is no option to change users. Although `oracle` is still the underlying user in all cases, Oracle Big Data SQL 3.2 now uses Hadoop Secure Impersonation to direct the `oracle` account to execute tasks on behalf of other designated users. This enables HDFS data access based on the user that is currently executing the query, rather than the singular `oracle` user.

Administrators set up the rules for identifying the query user. They can provide rules for identifying the currently connected user and mapping the connected user to the user that is impersonated. Because there are numerous ways in which users can connect to Oracle Database, this user may be a database user, a user sourced from LDAP, from Kerberos, or a user from other sources. Authorization rules on the files apply for that user and HDFS auditing identifies the actual user running the query.

See Also:

Administration for Multi-User Authorization is done through the DBMS_BDSQL PL/SQL Package.

Authentication Between Oracle Database and Oracle Big Data SQL Cells

This authentication is between Oracle Database and the Big Data SQL cells on the Hadoop cluster, facilitating secure communication. The Database Authentication enhancement provides a safeguard against impersonation attacks, in which a rogue service attempts to connect to the Oracle Big Data offload server process running on a cluster node.

Kerberos Ticket Renewal Automation

On a Kerberos-secured network you can configure the installation to set up automated Kerberos ticket renewal for the `oracle` account used by Oracle Big Data SQL. This is done for both the Hadoop cluster and Oracle Database sides of the installation. You must provide the principal name and the path to the keytab file in the `bds-config.json` configuration file. A template is provided in the configuration file:

```
"kerberos" : {
"principal" : "oracle/mycluster@MY.DOMAIN.COM",
"keytab" : "/home/oracle/security/oracle.keytab"
}
```

If you provide the Kerberos parameters in the configuration file, then Oracle Big Data SQL installation sets up cron jobs on both the Hadoop cluster and Oracle Database servers. These jobs renew the Kerberos tickets for the principal once per day.

The principal and keytab file must already exist.

Automatic Upgrade

The current release can now be installed over an earlier release with no need to remove the older software on either the Hadoop or Oracle Database side. The previous installation is upgraded to the current release level.

Common Installation Bundle for all Platforms

In previous releases, customers needed to unpack the Oracle Big Data SQL installation bundle and choose the correct package for their Hadoop system (CDH or HDP). Now the bundle contains a single installation package that works for all supported Hadoop systems.

Simpler and Faster Installation with the new “Jaguar” Installer

The Jaguar installer replaces `setup-bds.sh`, the installer in previous releases. Jaguar includes these changes:

- Automatic Check for Installation Prerequisites on Hadoop Nodes**
 Jaguar checks for installation readiness on each Hadoop DataNode and reports any missing prerequisites.
- No Need to Manually Generate the Database-Side Installation Bundle**
 The database-side installation bundle that previously was manually generated by the customer can now be generated automatically. You still need to copy the bundle to the Oracle Database nodes and install it.
- Faster Overall Installation Time on the Hadoop Side**
 Installation time will vary, but on the Hadoop Side the installation may take approximately eight minutes if all resources are local, possibly 20 minutes if Hadoop clients must be downloaded from the Internet, depending on download speed.
- Prerequisite Apache Services on CDH can now be Installed as Either Packages or Parcels**
 Previously on CDH systems, the Oracle Big Data SQL installation required that the HDFS, YARN, and HIVE components had been installed as parcels. These components can now be installed on CDH as either packages or parcels. There is no change for HDP, where they must be installed as stacks.

Note:

On CDH systems, if the Hadoop services required by Oracle Big Data SQL are installed as packages, be sure that they are installed from within Cloudera Manager. Otherwise, Cloudera Manager will not be able to manage these services. This is not an issue with parcels.

- In the CLI, the Jaguar utility Replaces `./setup-bds`**
 The Jaguar utility is now the primary tool for Hadoop-side installation, de-installation, and configuration changes, as in these examples:


```
# ./jaguar install bds-config.json
# ./jaguar reconfigure bds-config.json
# ./jaguar uninstall bds-config.json
```
- The Default Configuration File Name is `bds-config.json`, but Alternate File Names are Also Accepted**
 You can now drop the explicit `bds-config.json` argument and allow the installer default to `bds-config.json`, as in the first example below. You can also specify an alternate configuration file of any name, though it must adhere to the same internal format as `bds-config.json` and should be given the `.json` file type.


```
# ./jaguar install
# ./jaguar install cluster2-config.json
```

You can create configurations files with settings that are tailored to the requirements of each cluster. For example, you may want to apply different security parameters to Oracle Big Data SQL installations on test and production clusters.

- **Configuration Parameters Have Changed Significantly**

Users of previous releases will see that the Jaguar configuration file includes a number of new parameters. Most of them are “optional” in the sense that they are not uniformly required, although your particular installation may require some of them. See the Related Links section below for links to the table of installer parameters as well as an example of a configuration file that uses all available parameters.

- **New `updatenodes` Command for Easier Maintenance**

Oracle Big Data SQL must be installed on each Hadoop cluster node that is provisioned with the DataNode role. It has no function on nodes where DataNode is not present. The new Jaguar utility includes the `updatenodes` command which scans the cluster for instances of the DataNode within the cluster. If the DataNode role has been removed or relocated, or if nodes provisioned with the DataNode have been added or removed, then the script installs or uninstalls Oracle Big Data SQL components from nodes as needed.

- **An Extra Installation Step is Required to Enable Some Security Features**

If you choose to enable Database Authentication between Oracle Database and Oracle Big Data SQL cells in the Hadoop cluster, or, Hadoop Secure Impersonation, then an additional “Database Acknowledge” step is required. In this process, the installation on the database server generates a ZIP file of configuration information that you must copy back to the Hadoop cluster management server for processing.

- **On the Database Side, Connections to Clusters are no Longer Classified as Primary and Secondary.**

An Oracle Database system can have Oracle Big Data SQL connections to multiple Hadoop clusters. In previous releases, the first these connections was considered the primary (and had to be uninstalled last) and the others were secondary. In the current release, management of multiple installation is simpler and `--uninstall-as-primary` and `--uninstall-as-secondary` parameters of the database-side installer are obsolete. However there is now a default cluster. The *Important Terms and Concepts* section of this guide explains the significance of the default cluster.

Support for Oracle Tablespaces in HDFS Extended to Include All Non-System Permanent Tablespaces

Previous releases supported the move of permanent online tablespaces only to HDFS. This functionality now supports online, read-only, as well as offline permanent tablespaces.

Important Change in Behavior of the “`mtactl start`” Command

Oracle Big Data SQL 3.1 introduced the option to install Oracle Big Data SQL on servers where Oracle Grid Infrastructure is not present. In these environments, you can use the `start` subcommand of the `mtactl` utility (`mtactl start`) to start the MTA (Multi-Threaded Agent) `extproc`.

Note that in the current release, the `mtactl start` command works differently from the original Release 3.1 implementation.

- Current behavior: `mtactl start` starts an MTA extproc using the init parameter values that are stored in the repository. It uses the default values only if the repository does not exist.
- Previous behavior (Oracle Big Data SQL 3.1): `mtactl start` always uses the default init parameters regardless of whether or not init parameter values are stored in the repository.

1

Introducing Oracle Big Data SQL

Welcome to Oracle Big Data SQL.

- [What Is Oracle Big Data SQL?](#)
- [Installation](#)

1.1 What Is Oracle Big Data SQL?

Oracle Big Data SQL supports queries against non-relational data stored in multiple big data sources, including Apache Hive, HDFS, Object Stores, Oracle NoSQL Database, Apache Kafka, Apache HBase, and other NoSQL databases. It enables unified query for distributed data and therefore the ability to view and analyze data from disparate data stores seamlessly, as if it were all stored in an Oracle database.

Oracle Big Data SQL enables you to execute highly complex SQL `SELECT` statements against data in the Hadoop ecosystem, either manually or through your existing applications. For example, if you are a user of Oracle Advanced Analytics, Oracle Big Data SQL enables you to extend your Oracle Database data mining models to big data in Hadoop.

Components of an Oracle Big Data SQL Installation

The Oracle Big Data SQL architecture consists of an installation on an Oracle Database system (single node or RAC) that works in conjunction with a parallel installation on a Hadoop (or NoSQL) cluster. The two systems may be networked via either Ethernet or InfiniBand. Hadoop and Hive clients on the compute nodes of the Oracle Database system enable communication between the database and the Oracle Big Data SQL process (known as Oracle Big Data SQL “cell”) that runs on each of the DataNodes of the Hadoop cluster. Through this mechanism, Oracle Database can query data on the Hadoop cluster. In addition, an Oracle Big Data SQL Query Server can be deployed on an edge node on a cluster and can also connect to the Oracle Big Data SQL cells. For details see [Working With Query Server](#).

Since data in the Hadoop HDFS file system and Object Storage is stored in an undetermined format, SQL queries require some constructs to parse and interpret data for it to be processed in rows and columns. Oracle Big Data SQL leverages available Hadoop constructs to accomplish this for HDFS, notably InputFormat and SerDe Java classes, optionally through Hive metadata definitions. For object storage, Big Data SQL uses highly optimized C-drivers to access data in text, Parquet, ORC and Avro file formats. The Oracle Big Data SQL processing cells on the DataNodes are managed by YARN and are integrated into the Hadoop infrastructure. Three key features provided by the cells are *Smart Scan*, *Storage Indexes*, and *Aggregation Offload*. See [About Smart Scan for Big Data Sources](#), [About Storage Indexes](#), and [About Aggregation Offload](#).

1.1.1 About Oracle External Tables

Oracle Big Data SQL provides external tables with extreme performance gains. An **external table** is an Oracle Database object that identifies and describes the location of data outside of a database. You can query an external table using the same SQL `SELECT` syntax that you use for any other database tables.

Big Data SQL uses fan out parallelism to offer highly scalable processing with external tables. For example, a serial query against a Big Data SQL external table takes advantage of the Hadoop cluster parallel processing capacity.

External tables use **access drivers** to parse the data outside the database. There are multiple external data drivers to process data from different sources. The first, **ORACLE_HIVE**, leverages metadata defined in Apache Hive. The second, **ORACLE_HDFS**, accesses data directly in HDFS - with metadata specified as part of the external table definition. And the third, **ORACLE_BIGDATA**, accesses data in Object Storage; similar to HDFS, the metadata is part of the table definition.

1.1.2 About the Access Drivers for Oracle Big Data SQL

By querying external tables, you can access data stored in external sources as if that data was stored in tables in an Oracle database. Oracle Database accesses the data via the metadata provided by the external table.

Oracle Database supports these access drivers for Oracle Big Data SQL:

- **ORACLE_HIVE**: Enables you to create Oracle external tables over Apache Hive data sources. Use this access driver when you already have Hive tables defined for your HDFS data sources. **ORACLE_HIVE** can also access data stored in other locations, such as HBase, that have Hive tables defined for them and Kafka.
- **ORACLE_HDFS**: Enables you to create Oracle external tables directly over files stored in HDFS. This access driver uses Hive syntax to describe a data source, assigning default column names of `COL_1`, `COL_2`, and so forth. You do not need to create a Hive table manually as a separate step.

Instead of acquiring the metadata from a Hive metadata store the way that **ORACLE_HIVE** does, the **ORACLE_HDFS** access driver acquires all of the necessary information from the access parameters. The **ORACLE_HDFS** access parameters are required to specify the metadata, and are stored as part of the external table definition in Oracle Database.

- **ORACLE_BIGDATA**: Enables external table creation over files in object stores. Use this access driver for querying data captured in object stores. **ORACLE_BIGDATA** supports text (i.e. delimited, JSON, and XML), ORC, Parquet, and Avro file types. Similar to the **ORACLE_HDFS** driver, you use access parameters to describe the structure of the file.

1.1.3 About Smart Scan for Big Data Sources

Oracle external tables do not have traditional indexes. Queries against these external tables typically require a full table scan. The Oracle Big Data SQL processing agent on the DataNodes of the Hadoop cluster extends Smart Scan capabilities (such as filter-predicate off-loads) to Oracle external tables. Smart Scan has been used for some time on the Oracle Exadata Database Machine to do column and predicate filtering in

the Storage Layer before query results are sent back to the Database Layer. In Oracle Big Data SQL, Smart Scan is a final filtering pass done locally on the Hadoop server to ensure that only requested elements are sent to Oracle Database. Oracle storage servers running on the Hadoop DataNodes are capable of doing Smart Scans against various data formats in HDFS, such as CSV text, Avro, and Parquet.

This implementation of Smart Scan leverages the massively parallel processing power of the Hadoop cluster to filter data at its source. It can preemptively discard a huge portion of irrelevant data—up to 99 percent of the total. This has several benefits:

- Greatly reduces data movement and network traffic between the cluster and the database.
- Returns much smaller result sets to the Oracle Database server.
- Aggregates data when possible by leveraging scalability and cluster processing.

Query results are returned significantly faster. This is the direct result reduced traffic on the network and reduced load on Oracle Database.

See Also:

See [Store Oracle Tablespaces in HDFS](#) for instructions on how to set up data files for smart scanning.

See [Oracle Database Concepts](#) for a general introduction to external tables and pointers to more detailed information in the Oracle Database documentation library

1.1.4 About Storage Indexes

For data stored in HDFS, Oracle Big Data SQL maintains Storage Indexes automatically, which is transparent to Oracle Database. Storage Indexes contain the summary of data distribution on a hard disk for the data that is stored in HDFS. Storage Indexes reduce the I/O operations cost and the CPU cost of converting data from flat files to Oracle Database blocks. You can think of a storage index as a "negative index". It tells Smart Scan that data does not fall within a block of data, which enables Smart Scan to skip reading that block. This can lead to significant I/O avoidance.

Storage Indexes can be used only for external tables that are based on HDFS and are created using either the ORACLE_HDFS driver or the ORACLE_HIVE driver. Storage Indexes cannot be used for external tables based on object stores or external tables that use StorageHandlers, such as Apache HBase and Oracle NoSQL.

A Storage Index is a collection of in-memory region indexes, and each region index stores summaries for up to 32 columns. There is one region index for each split. The content stored in one region index is independent of the other region indexes. This makes them highly scalable, and avoids latch contention.

Storage Indexes maintain the minimum and maximum values of the columns of a region for each region index. The minimum and maximum values are used to eliminate unnecessary I/O, also known as I/O filtering. The cell XT granule I/O bytes saved by the Storage Indexes statistic, available in the `V$SYSSTAT` view, shows the number of bytes of I/O saved using Storage Indexes.

Queries using the following comparisons are improved by the Storage Indexes:

- Equality (=)
- Inequality (<, !=, or >)
- Less than or equal (<=)
- Greater than or equal (>=)
- IS NULL
- IS NOT NULL

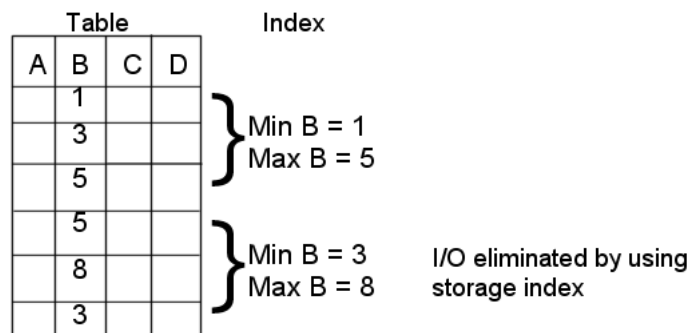
Storage Indexes are built automatically after Oracle Big Data SQL service receives a query with a comparison predicate that is greater than the maximum or less than the minimum value for the column in a region.

 **Note:**

- The effectiveness of Storage Indexes can be improved by ordering the rows in a table based on the columns that frequently appear in the WHERE query clause.
- Storage Indexes work with any non-linguistic data type, and works with linguistic data types similar to non-linguistic index.

Example 1-1 Elimination of Disk I/O with Storage Indexes

The following figure shows a table and region indexes. The values in **column B** in the table range from 1 to 8. One region index stores the minimum 1, and the maximum of 5. The other region index stores the minimum of 3, and the maximum of 8.



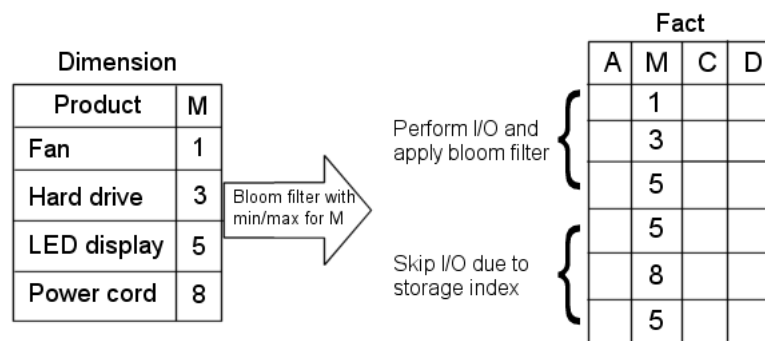
For a query such as the one below, only the first set of rows match. Disk I/O is eliminated because the minimum and maximum of the second set of rows do not match the WHERE clause of the query.

```
SELECT *
FROM TABLE
WHERE B < 2;
```


Example 1-2 Using Storage Indexes and Bloom Filters

Using Storage Indexes allows table joins to skip unnecessary I/O operations. For example, the following query would perform an I/O operation and apply a Bloom filter to only the first block of the fact table. Bloom filters are the key to improved join performance. In the example, a predicate is on the dimension table - not the fact table. The Bloom Filter is created based on "dim.name=Hard drive" and this filter is then applied to the fact table. Therefore, even though the filter is on the dimension table, you are able to filter the data at its source (i.e. Hadoop) based on the results of the dimension query. This also enables optimizations like Storage Indexes to engage.

```
SELECT count(*)
FROM fact, dimension dim
WHERE fact.m=dim.m and dim.product="Hard drive";
```



The I/O for the second block of the fact table is completely eliminated by Storage Indexes as its minimum/maximum range (5,8) is not present in the Bloom filter.

1.1.5 About Predicate Push Down

Many Big Data systems support some level of predicate off-loading, either through the filetype itself (e.g. Apache Parquet), or through Hive's partitioning and StorageHandler APIs. Oracle Big Data SQL takes advantage of these off-load capabilities by pushing predicates from the Oracle Database into supporting systems. For example, predicate push down enables the following automatic behaviors:

- Queries against partitioned Hive tables are pruned, based on filter predicates on partition columns.
- Queries against Apache Parquet and Apache ORC files reduce I/O by testing predicates against the internal index-like structures contained within these file formats.

Note:

Predicate pushdown in queries against Parquet files is inefficient unless the files are generated through Hive using the workaround described in the next section.

- Queries against Oracle NoSQL Database or Apache HBase use predicates to drive subscons of data in the remote data store.

Required Datatypes to Enable Predicate Push Down

Predicate push down requires that certain mappings between Hive Datatypes and Oracle Datatypes be present. These mappings are described in the following table.

Hive Datatype	Mapped To Oracle Datatype
CHAR(m)	CHAR(n), VARCHAR2(n) where n is >= m
VARCHAR(m)	CHAR(n), VARCHAR2(n) where n is >= m.
string	CHAR(n), VARCHAR2(n)
DATE	DATE
TIMESTAMP	TIMESTAMP(9) Hive TIMESTAMP has nanoseconds, 9 digit fractional seconds.
TINYINT	NUMBER(3) preferably, but NUMBER or NUMBER(n) for any value of n is valid.
SMALLINT	NUMBER(5) preferably, but NUMBER or NUMBER(n) for any value of n is valid.
INT	NUMBER(10) preferably, but NUMBER or NUMBER(n) for any value of n is valid.
BIGINT	NUMBER(19) preferably, but NUMBER or NUMBER(n) for any value of n is OK
DECIMAL(m)	NUMBER(n) where m = n preferably, but NUMBER or NUMBER(n) for any value of n is valid.
FLOAT	BINARY_FLOAT
DOUBLE	BINARY_DOUBLE
BINARY	RAW(n)
BOOLEAN	CHAR(n), VARCHAR2(n) where n is >= 5, values 'TRUE', 'FALSE'
BOOLEAN	NUMBER(1) preferably, but NUMBER or NUMBER(n) for any value of n is valid. Values 0 (false), 1 (true).

1.1.6 About Pushdown of Character Large Object (CLOB) Processing

Queries against Hadoop data may involve processing large objects with potentially millions of records. It is inefficient to return these objects to Oracle Database for filtering and parsing. Oracle Big Data SQL can provide significant performance gains by pushing CLOB processing down to its own processing cells on the Hadoop cluster. Filtering in Hadoop reduces the number of rows returned to Oracle Database. Parsing reduces the amount of data returned from a column within each filtered row.

Customers can disable or re-enable CLOB processing pushdown to suit their own needs.

In the current release, this functionality currently applies only to JSON expressions returning CLOB data. The eligible JSON filter expressions for storage layer evaluation include simplified syntax, JSON_VALUE, and JSON_QUERY.

The same support will be provided for other CLOB types (such as substr and instr) as well as for BLOB data in a future release.

Oracle Big Data SQL can push processing down to Hadoop for CLOBs within these size constraints:

- Filtering for CLOB columns up to 1 MB in size.

The actual amount of data that can be consumed for evaluation in the storage server may vary, depending upon the character set used.

- Parsing for columns up to 32 KB.

This limit refers to the select list projection from storage for the CLOB datatype.

Processing falls back to the Oracle Database only when column sizes exceed these two values.

Example 1-3 JSON Document Processing

For queries into large JSON documents, pushdown of CLOB processing to Oracle Big Data SQL processing cells in Hadoop can be highly effective. Consider the following example, where purchase orders information is stored in JSON. Assume that this record could be up to 25K in size and several millions of such records must be processed.

```
{ "ponumber":9764,"reference":"LSMITH-20141017","requestor":"Lindsey Smith","email": "Lindsey@myco.com", "company":"myco" ...}
```

You can create the external table to access this data as follows. Notice there is a single CLOB column.

```
CREATE TABLE POS_DATA
  ( pos_info CLOB )
  ORGANIZATION EXTERNAL
  ( TYPE ORACLE_HDFS
    DEFAULT DIRECTORY DEFAULT_DIR
    LOCATION ( '/data/pos/*' )
  )
  REJECT LIMIT UNLIMITED;
```

You can then query the data with this simple syntax:

```
SELECT p.pos_info.email, p.pos_info.requestor
FROM POS_DATA p
WHERE p.pos_info.company='myco'
```

The query example above engages two data elimination optimizations:

- The data is filtered by the Oracle Big Data SQL cells in the Hadoop cluster. Only records pertaining to the company “myco” are parsed (and after parsing only selected data from these records is returned to the database).
- The Oracle Big Data SQL cells in the cluster parse the filtered set of records and from each record only the values for the two attributes requested (p.pos_info.email and p.pos_info.requestor) are returned to the database.

The table below shows some other examples where CLOB processing pushdown is supported. Remember that projections (references on the select side of the CLOB

column) are limited to 32 KB of CLOB data, while predicate pushdown is limited to 1 MB of CLOB data.

Query	Comment
<pre>SELECT count(*) FROM pos_data p WHERE pos_info is json;</pre>	In this case, the predicate ensures that only columns which comply with JSON format are returned.
<pre>SELECT pos_info FROM pos_data p WHERE pos_info is json;</pre>	The same predicate as in the previous case, but now the CLOB value is projected.
<pre>SELECT json_value(pos_info, '\$.reference') FROM pos_data p WHERE json_value(pos_info, '\$.ponumber') > 9000</pre>	Here, the predicate is issued on a field of the JSON document, and we also execute a JSON value to retrieve field "reference" on top of the projected CLOB JSON value.
<pre>SELECT p.pos_info.reference FROM pos_data p WHERE p.pos_info.ponumber > 9000;</pre>	This is functionally the same query as the previous example, but expressed in simplified syntax.
<pre>SELECT p.pos_info.email FROM po_data p WHERE json_exists(pos_info, '\$.requestor') and json_query(pos_info, '\$.requestor') is not null;</pre>	This example shows how <code>json_exists</code> and <code>json_query</code> can also be used as predicates.

1.1.7 About Aggregation Offload

Oracle Big Data SQL uses Oracle In-Memory technology to push aggregation processing down to the Oracle Big Data SQL cells. This enables Oracle Big Data SQL to leverage the processing power of the Hadoop cluster for distributing aggregations across the cluster nodes. The performance gains can be significantly faster compared to aggregations that do not offload especially when there are a moderate number of summary groupings. For single table queries, the aggregation operation should consistently offload.

Oracle Big Data SQL cells support single table and multi-table aggregations (for example, dimension tables joining to a fact table). For multi-table aggregations, the Oracle Database uses the key vector transform optimization in which the key vectors are pushed to the cells for the aggregation process. This transformation type is useful for star join SQL queries that use typical aggregation operators (for example, `SUM`, `MIN`, `MAX`, and `COUNT`) which are common in business queries.

A vector transformation query is a more efficient query that uses bloom filter for joins. When you use a vector transformed query with Oracle Big Data SQL Cells, the performance of joins in the query is enhanced by the ability to offload filtration for rows used for aggregation. You see a "KEY VECTOR USE" operation in the query plan during this optimization.

In Oracle Big Data SQL cells, vector transformed queries benefit from more efficient processing due to the application of group-by columns (key vectors) to the Oracle Big Data SQL Storage Index.

You may not see the benefit of aggregation offload in certain instances:

- **Missing predicate**

If the `SYS_OP_VECTOR_GROUP_BY` predicate is missing in the explain plan, aggregation offload is affected. The predicate can be missing due to the following reasons:

- Presence of a disallowed intervening row source between the table scan and group-by row sources.
- The table scan does not produce rowsets.
- Presence of an expression or data type in the query that can not be offloaded.
- Vector group-by is manually disabled.
- The table of table scan or configuration does not expect gains from aggregation offload.

- **Missing smart scan**

The cell interconnect bytes returned by XT smart scan and cell XT granules requested for predicate offload statistics must be available.

- **Missing key vectors**

The limit on the data transmitted to the cells is 1 MB. If this threshold is exceeded, then queries can benefit from intelligent key vector filtering but not necessarily offloaded aggregation. This condition is known as Key Vector Lite mode. Due to their large size, some of the key vectors are not fully offloaded. They get offloaded in lite mode along with the key vectors that do not support aggregation offload. Key vectors are not completely serialized in lite mode. The vector group-by offload is disabled when key vectors are offloaded in lite mode.



See Also:

[Oracle Database In-Memory Guide](#) for information about how aggregation works in Oracle Database

1.1.8 About Oracle Big Data SQL Statistics

Oracle Big Data SQL provides a number of statistics that can contribute data for performance analyses.

Five Key Cell XT and Storage Index Statistics

If a query is off-loadable, the following XT-related statistics that can help you to determine what kind of I/O savings you can expect from the offload and from Smart Scan.

- **cell XT granules requested for predicate offload**

Note that number of granules requested depends on a number of factors, including the HDFS block size, Hadoop data source splittability, and the effectiveness of Hive partition elimination.

- **cell XT granule bytes requested for predicate offload**

The number of bytes requested for the scan. This is the size of the data on Hadoop to be investigated after Hive partition elimination and before Storage Index evaluation.

- **cell interconnect bytes returned by XT smart scan**
The number of bytes of I/O returned by an XT smart scan to Oracle Database.
- **cell XT granule predicate offload retries**
The number of times that a Big Data SQL process running on a DataNode could not complete the requested action. Oracle Big Data SQL automatically retries failed requests on other DataNodes that have a replica of the data. The retries value should be zero.
- **cell XT granule IO bytes saved by storage index**
The number of bytes filtered out by storage indexes at the storage cell level. This is data that was not scanned, based information provided by the storage indexes.

You can check these statistics before and after running queries as follows. This example shows the values at null, before running any queries.

```
SQL> SELECT sn.name,ms.value
FROM V$MYSTAT ms, V$STATNAME sn
WHERE ms.STATISTIC#=sn.STATISTIC# AND sn.name LIKE '%XT%';
```

NAME	VALUE
cell XT granules requested for predicate offload	0
cell XT granule bytes requested for predicate offload	0
cell interconnect bytes returned by XT smart scan	0
cell XT granule predicate offload retries	0
cell XT granule IO bytes saved by storage index	0

You can check some or all of these statistics after execution of a query to test the effectiveness of the query, as in:

```
SQL> SELECT n.name, round(s.value/1024/1024)
FROM v$mystat s, v$statname n
WHERE s.statistic# IN (462,463)
AND s.statistic# = n.statistic#;

cell XT granule bytes requested for predicate offload 32768
cell interconnect bytes returned by XT smart scan 32
```

Five Aggregation Offload Statistics

The following statistics can help you analyze the performance of aggregation offload.

- **vector group by operations sent to cell**
The number of times aggregations can be offloaded to the cell.
- **vector group by operations not sent to cell due to cardinality**
The number of scans that were not offloaded because of large wireframe.
- **vector group by rows processed on cell**
The number of rows that were aggregated on the cell.
- **vector group by rows returned by cell**
The number of aggregated rows that were returned by the cell.

- **vector group by rowsets processed on cell**

The number of rowsets that were aggregated on the cell.

You can review these statistics by running the queries as follows:

```
SQL> SELECT count(*) FROM bdsq1_parq.web_sales;
```

```

COUNT(*)
-----
287301291

```

```
SQL> SELECT substr(n.name, 0,60) name, u.value
FROM v$statname n, v$mystat u
WHERE ((n.name LIKE 'key vector%') OR
       (n.name LIKE 'vector group by%') OR
       (n.name LIKE 'vector encoded%') OR
       (n.name LIKE '%XT%') OR
       (n.name LIKE 'IM %' AND n.name NOT LIKE '%spare%'))
AND u.sid=userenv('SID')
AND n.STATISTIC# = u.STATISTIC#
AND u.value > 0;
```

NAME	VALUE
cell XT granules requested for predicate offload	808
cell XT granule bytes requested for predicate offload	2.5833E+10
cell interconnect bytes returned by XT smart scan	6903552
vector group by operations sent to cell	1
vector group by rows processed on cell	287301291
vector group by rows returned by cell	808

Nine Key Vector Statistics

The following statistics can help you analyze the effectiveness of key vectors that were sent to the cell.

- **key vectors sent to cell**

The number of key vectors that were offloaded to the cell.

- **key vector filtered on cell**

The number of rows that were filtered out by a key vector on the cell.

- **key vector probed on cell**

The number of rows that were tested by a key vector on the cell.

- **key vector rows processed by value**

The number of join keys that were processed by using their value.

- **key vector rows processed by code**

The number of join keys that were processed by using the dictionary code.

- **key vector rows filtered**

The number of join keys that were skipped due to skip bits.

- **key vector serializations in lite mode for cell**
The number of times a key vector was not encoded due to format or size.
- **key vectors sent to cell in lite mode due to quota**
The number of key vectors that were offloaded to the cell for non-exact filtering due to the 1 MB metadata quota.
- **key vector efilters created**
A key vector was not sent to a cell, but an efilter (similar to a bloom filter) was sent.

You can review these statistics by running the queries as follows:

```
SELECT substr(n.name, 0,60) name, u.value
FROM v$statname n, v$mystat u
WHERE ((n.name LIKE 'key vector%') OR
       (n.name LIKE 'vector group by%') OR
       (n.name LIKE 'vector encoded%') OR
       (n.name LIKE '%XT%'))
AND u.sid=userenv('SID')
AND n.STATISTIC# = u.STATISTIC#
```

NAME	VALUE
cell XT granules requested for predicate offload	250
cell XT granule bytes requested for predicate offload	61,112,831,993
cell interconnect bytes returned by XT smart scan	193,282,128
key vector rows processed by value	14,156,958
key vector rows filtered	9,620,606
key vector filtered on cell	273,144,333
key vector probed on cell	287,301,291
key vectors sent to cell	1
key vectors sent to cell in lite mode due to quota	1
key vector serializations in lite mode for cell	1
key vector efilters created	1



Tip:

The [Oracle Big Data SQL Quickstart](#) blog, published in the [Data Warehouse Insider](#), provides a series of code and functionality walkthroughs that show you how to use these statistics to analyze the performance of Oracle Big Data SQL. See [Part 2](#), [Part 7](#), and [Part 10](#).

1.2 Installation

Oracle Big Data SQL requires installation of components on the Hadoop system where the data resides and also on the Oracle Database server which queries the data.

See the following resources for installation information:

- Introduction

This guide describes installation and configuration procedures for supported Hadoop system/Oracle Database server combinations.

- [Oracle Big Data SQL Master Compatibility Matrix](#)

This is Document 2119369.1 in [My Oracle Support](#). Check the matrix for up-to-date information on Big Data SQL compatibility with the following:

- Oracle Engineered Systems.
- Other systems.
- Linux OS distributions and versions.
- Hadoop distributions.
- Oracle Database releases, including required patches.

2

Using Oracle Big Data SQL for Data Access

Oracle Big Data SQL enables you to query diverse data sources using the full power of Oracle SQL SELECT statements.

This chapter describes how to create Oracle Big Data SQL enabled external tables over data from Hive, Hadoop, Apache Kafka, Oracle NoSQL Database, and object stores.

2.1 Creating External Tables

Oracle Big Data SQL enables you to query external data through Oracle Big Data SQL enabled external tables from the Oracle Database using the full power of Oracle SQL SELECT statements. It also enables you to write queries that join Oracle tables and that external data, leverage robust Oracle Database security features, and take advantage of advanced SQL capabilities like analytic functions, JSON handling, and others.

The section contains the following topics:

- [About the SQL CREATE TABLE Statement](#)
- [Creating an Oracle External Table for Hive Data](#)
- [Creating an Oracle External Table for Oracle NoSQL Database](#)
- [Creating an Oracle External Table for Apache HBase](#)
- [Creating an Oracle External Table for HDFS Files](#)
- [Creating an Oracle External Table for Kafka Topics](#)
- [Creating an Oracle External Table for Object Store Access](#)

2.1.1 About the SQL CREATE TABLE Statement

The SQL CREATE TABLE statement has a clause specifically for creating external tables. The information that you provide in this clause enables the access driver to read data from an external source and prepare the data for the external table.

2.1.1.1 Basic Syntax

The following is the basic syntax of the CREATE TABLE statement for external tables:

```
CREATE TABLE table_name (column_name datatype,  
                           column_name datatype[,...])  
  ORGANIZATION EXTERNAL (external_table_clause);
```

You specify the column names and data types the same as for any other table. ORGANIZATION EXTERNAL identifies the table as an external table.

The *external_table_clause* identifies the access driver and provides the information that it needs to load the data. See "[About the External Table Clause](#)".

2.1.1.2 About the External Table Clause

`CREATE TABLE ORGANIZATION EXTERNAL` statement takes the external table clause as its argument. The external table clause has the following subclauses:

TYPE

The `TYPE` clause identifies the access driver. The type of access driver determines how the other parts of the external table definition are interpreted.

Specify one of the following values for Oracle Big Data SQL:

- `ORACLE_HDFS`: Accesses files in an HDFS directory.
- `ORACLE_HIVE`: Accesses a Hive table.
- `ORACLE_BIGDATA`: Accesses files in an object store.



Note:

The `ORACLE_DATAPUMP` and `ORACLE_LOADER` access drivers are not associated with Oracle Big Data SQL.

DEFAULT DIRECTORY

The `DEFAULT DIRECTORY` clause identifies an Oracle Database directory object. The directory object identifies an operating system directory with files that the external table reads and writes.

`ORACLE_HDFS`, `ORACLE_BIGDATA`, and `ORACLE_HIVE` use the default directory solely to write log files on the Oracle Database system.

LOCATION

The `LOCATION` clause identifies the data source.

REJECT LIMIT

The `REJECT LIMIT` clause limits the number of conversion errors permitted during a query of the external table before Oracle Database stops the query and returns an error.

Any processing error that causes a row to be rejected counts against the limit. The reject limit applies individually to each parallel query (PQ) process. It is not the total of all rejected rows for all PQ processes.

ACCESS PARAMETERS

The `ACCESS PARAMETERS` clause provides information that the access driver needs to load the data correctly into the external table.

2.1.2 Creating an Oracle External Table for Hive Data

You can leverage Hive metadata when creating your Oracle Big Data SQL external tables.

External table creation is a technique to access data not only data in HDFS, but also data in other storage types, including Oracle NoSQL Database, Apache Kafka, HBase, and object stores.

To enable Oracle Big Data SQL to query Hive data, you must first define an Oracle external table for your Hive data. There are a number of tools available to help you create the Oracle external table definition. These tools leverage the underlying hive metadata, making it easier to create the tables.

The external table provides a level of abstraction. The underlying partitions or file type may change, but the Oracle external table definition can remain the same. It automatically picks up these changes when you query the table.

As part of an external table definition, you specify the table columns and their data types as well as a pointer to the source table in Hive. The rest of the metadata is derived at query execution time, including the data location, the file type and partitioning information.

- **DBMS_HADOOP**

DBMS_HADOOP is a PL/SQL package that contains the `CREATE_EXTDDL_FOR_HIVE` procedure. This procedure generates the DDL to create an Oracle external table for a given Hive table. You can optionally edit the text of the generated DDL before execution in order to customize the external table properties. This procedure enables you to easily automate the definition of many tables at one time.

- **The Big Data SQL wizard in Oracle SQL Developer**

The most recent versions of the free Oracle SQL Developer tool include a Big Data SQL wizard that guides you easily through the process of creating Oracle external table definitions.

If you have a configured Hive connection in Oracle SQL Developer, then in the **Connections** navigator, drill down from the connection entry point to a Hive table and do the following:

1. Right-click on the table icon and select **Use in Oracle Big Data SQL...**
2. When prompted, select an Oracle Database connection for the import of the Hive table.
3. Select an Oracle Big Data SQL-enabled target database.
4. In the Create Table dialog, check over the current configuration for columns, external table properties, and storage. Modify as needed. You can also preview the text of the DDL that will be generated.
5. Click **OK** when you are satisfied with the table definition. The wizard will create the external table at the designated location.

- **The Oracle SQL Developer Data Modeler**

This is free graphical design tool that you can use to connect to a Hive metastore and generate an external table. You can select and import one or multiple Hive tables, modify table properties as needed, and then generate the DDL that you can copy into an SQL Worksheet and then run in order to create an Oracle external table. Although the Data Modeler is a more complex tool to use than the other options, its advantage is that you can use it to work on multiple Hive tables

See [Oracle SQL Developer & Data Modeler Support for Oracle Big Data SQL](#) in the Oracle Blog space for a demonstration of how to use the Data Modeler.

See Also:

For instructions on how to install Oracle SQL Developer and connect to Hive in order to create external tables, see [Using Oracle SQL Developer to Connect to Hive](#).

2.1.2.1 Obtaining Information About a Hive Table

The `DBMS_HADOOP` PL/SQL package contains a function named `CREATE_EXTDDL_FOR_HIVE`. It returns the data dictionary language (DDL) to create an external table for accessing a Hive table. This function requires you to provide basic information about the Hive table:

- Name of the Hadoop cluster
- Name of the Hive database
- Name of the Hive table
- Whether the Hive table is partitioned

You can obtain this information by querying the `ALL_HIVE_TABLES` data dictionary view. It displays information about all Hive tables that you can access from Oracle Database.

This example shows that the current user has access to an unpartitioned Hive table named `RATINGS_HIVE_TABLE` in the default database. A user named `JDOE` is the owner.

```
SQL> SELECT cluster_id, database_name, owner, table_name, partitioned FROM  
all_hive_tables;
```

CLUSTER_ID	DATABASE_NAME	OWNER	TABLE_NAME	PARTITIONED
hadoop1	default	jdoe	ratings_hive_table	UN-PARTITIONED

See Also:

["Static Data Dictionary Views for Hive"](#)

2.1.2.2 Using the `CREATE_EXTDDL_FOR_HIVE` Function

With the information from the data dictionary, you can use the `CREATE_EXTDDL_FOR_HIVE` function of `DBMS_HADOOP`. This example specifies a database table name of `RATINGS_DB_TABLE` in the current schema. The function returns the text

of the `CREATE TABLE` command in a local variable named `DDLout`, but does not execute it.

```
DECLARE
  DDLout VARCHAR2(4000);
BEGIN
  dbms_hadoop.create_extddl_for_hive(
    CLUSTER_ID=>'hadoop1',
    DB_NAME=>'default',
    HIVE_TABLE_NAME=>'ratings_hive_table',
    HIVE_PARTITION=>FALSE,
    TABLE_NAME=>'ratings_db_table',
    PERFORM_DDL=>FALSE,
    TEXT_OF_DDL=>DDLout
  );
  dbms_output.put_line(DDLout);
END;
/
```

When this procedure runs, the `PUT_LINE` function displays the `CREATE TABLE` command:

```
CREATE TABLE ratings_db_table (
  c0 VARCHAR2(4000),
  c1 VARCHAR2(4000),
  c2 VARCHAR2(4000),
  c3 VARCHAR2(4000),
  c4 VARCHAR2(4000),
  c5 VARCHAR2(4000),
  c6 VARCHAR2(4000),
  c7 VARCHAR2(4000))
ORGANIZATION EXTERNAL
  (TYPE ORACLE_HIVE DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS
    (
      com.oracle.bigdata.cluster=hadoop1
      com.oracle.bigdata.tablename=default.ratings_hive_table
    )
  ) PARALLEL 2 REJECT LIMIT UNLIMITED
```

You can capture this information in a SQL script, and use the access parameters to change the Oracle table name, the column names, and the data types as desired before executing it. You might also use access parameters to specify a date format mask.

The `ALL_HIVE_COLUMNS` view shows how the default column names and data types are derived. This example shows that the Hive column names are `C0` to `C7`, and that the Hive `STRING` data type maps to `VARCHAR2(4000)`:

```
SQL> SELECT table_name, column_name, hive_column_type, oracle_column_type FROM
all_hive_columns;
```

TABLE_NAME	COLUMN_NAME	HIVE_COLUMN_TYPE	ORACLE_COLUMN_TYPE
ratings_hive_table	c0	string	VARCHAR2(4000)
ratings_hive_table	c1	string	VARCHAR2(4000)
ratings_hive_table	c2	string	VARCHAR2(4000)
ratings_hive_table	c3	string	VARCHAR2(4000)
ratings_hive_table	c4	string	VARCHAR2(4000)
ratings_hive_table	c5	string	VARCHAR2(4000)
ratings_hive_table	c6	string	VARCHAR2(4000)

```
ratings_hive_table    c7          string      VARCHAR2(4000)
8 rows selected.
```

**See Also:**

["DBMS_HADOOP PL/SQL Package"](#)

2.1.2.3 Using Oracle SQL Developer to Connect to Hive

Oracle SQL Developer provides methods to connect to a Hive metastore and create Oracle external tables over Hive.

Follow these steps to set up Oracle SQL Developer to work with Oracle Big Data SQL.

1. Install Oracle SQL Developer
2. Download the Hive JDBC Drivers
3. Add the new Hive JDBC Drivers to Oracle SQL Developer
4. Create a database connection to Hive.

Installing Oracle SQL Developer

Install Oracle SQL Developer 4.2 or greater. Starting with this version, support is included for *Copy To Hadoop*, a useful Oracle Big Data SQL tool for off-loading Oracle Database tables to HDFS.

The installation is simple. Just download the package and extract it.

1. Go to the [Oracle SQL Developer download site](#) on the Oracle Technology Network (OTN).
2. Accept the license agreement and download the version that is appropriate for your platform.

For most users, **Windows 64-bit with JDK 8 included** is the correct choice.

3. Extract the downloaded ZIP file to your local drive.

You can extract to any folder name.

See *Installing and Getting Started with SQL Developer* in the *Oracle SQL Developer User's Guide* for further installation and configuration details.

Downloading and Installing the Hive JDBC Drivers for Cloudera Enterprise

To connect Oracle SQL Developer to Hive in the Hadoop environment, you need to download and install the Hive JDBC drivers for Cloudera Enterprise. These drivers are not included in the Oracle SQL Developer download package.

 **Note for HDP Users:**

At this time, SQL Developer 4.2 requires the Cloudera JDBC drivers for Hive. However, these drivers appear to work against Hortonworks clusters as well. HDP users should test to determine if these drivers meet their needs.

1. Download the latest Cloudera JDBC drivers for Hive from the [Cloudera](#) website to any local directory.

You can search for “cloudera hive jdbc drivers download” on the Cloudera website to locate the available driver packages.

You are prompted to select the driver version, OS, and OS version (32/64 bit). At this time, the latest drive version is 2.5.18. You can choose the newest version available.

2. Unzip the archive:

```
unzip hive_jdbc_<version>.zip
```

3. View the extracted content. Notice that under the top-level folder there are multiple ZIP files. Each is for a different JDBC version. For this setup, only JBDC 4.0 is usable. Select the **JDBC4_** ZIP file (JDBC4_<version>.zip).

 **Important:**

Choose *only* the **JDBC4_** ZIP file, which contains the drivers for JDBC 4.0. This is the only compatible version. The drivers in other packages, such as JDBC41_*, are not compatible with SQL Developer 4.2 and will return errors upon connection attempts.

4. Unzip the JDBC4 archive to a target directory that is accessible to Oracle SQL Developer, for example, `./home/oracle/jdbc` :

```
# unzip Cloudera_HiveJDBC4_<version>.zip -d /home/oracle/jdbc/
```

The extracted content should be similar to this:

```
Cloudera_HiveJDBC4_2.5.18.1050\Cloudera-JDBC-Driver-for-Apache-Hive-Install-Guide.pdf
Cloudera_HiveJDBC4_2.5.18.1050\Cloudera-JDBC-Driver-for-Apache-Hive-Release-Notes.pdf
Cloudera_HiveJDBC4_2.5.18.1050\commons-codec-1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\commons-logging-1.1.1.jar
Cloudera_HiveJDBC4_2.5.18.1050\HiveJDBC4.jar
Cloudera_HiveJDBC4_2.5.18.1050\hive_metastore.jar
Cloudera_HiveJDBC4_2.5.18.1050\hive_service.jar
Cloudera_HiveJDBC4_2.5.18.1050\httpclient-4.1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\httpcore-4.1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\libfb303-0.9.0.jar
Cloudera_HiveJDBC4_2.5.18.1050\libthrift-0.9.0.jar
Cloudera_HiveJDBC4_2.5.18.1050\log4j-1.2.14.jar
```



```
Cloudera_HiveJDBC4_2.5.18.1050\out.txt
Cloudera_HiveJDBC4_2.5.18.1050\ql.jar
Cloudera_HiveJDBC4_2.5.18.1050\slf4j-api-1.5.11.jar
Cloudera_HiveJDBC4_2.5.18.1050\slf4j-log4j12-1.5.11.jar
Cloudera_HiveJDBC4_2.5.18.1050\TCLIServiceClient.jar
Cloudera_HiveJDBC4_2.5.18.1050\zookeeper-3.4.6.jar
```

Add the new Hive JDBC Drivers to Oracle SQL Developer

Next, start up SQL Developer and copy all of the extracted driver files into “Third Party JDBC Drivers” in the **Preferences** window.

1. Start SQL Developer.
2. In the SQL Developer menu bar, select **Tools>Preferences**.
3. In the file explorer of the **Preferences** window, expand **Database** and then click **Third Party JDBC Drivers**.
4. Click **Add Entry**.
5. Navigate to the folder where you sent the files extracted from `Cloudera_HiveJDBC4_<version>.zip`. Copy all of the JAR files from the ZIP extraction into this window and then click **OK**.
6. Restart Oracle SQL Developer.

Create a Database Connection to Hive

After the drivers are installed, you can create a connection to Hiveserver2.

If you are creating a Kerberos-secured connection, you will need a user ID, the Kerberos connection parameters, and the number of the port where Hiveserver2 is running on the Hadoop system (typically, port 10000). A keytab must exist for the user.

If you're not using Kerberos, you will need a user ID (the `oracle` user or a user with equivalent privileges), the account password, and the Hiveserver2 port number.

See *Create/Edit/Select Database Connection* in the *Oracle SQL Developer User's Guide* for a explanation of the fields in the Oracle and Hive tabs in the **New/Select Database Connection** dialog.

2.1.2.4 Developing a CREATE TABLE Statement for ORACLE_HIVE

Whichever method you use to create an Oracle external table over Hive (DBMS_HADOOP, Oracle SQL Developer Data Modeler, Oracle Big Data Wizard in Oracle SQL Developer, or manual coding), you may need to set some access parameters to modify the default behavior of `ORACLE_HIVE`.

Note:

Do not include the `LOCATION` clause with `ORACLE_HIVE`. It raises an error. The data is stored in Hive, and the access parameters and the metadata store provide the necessary information.

2.1.2.4.1 Using the Default ORACLE_HIVE Settings

The following statement creates an external table named `ORDER` to access Hive data:

```
CREATE TABLE order (cust_num  VARCHAR2(10),
                    order_num VARCHAR2(20),
                    description VARCHAR2(100),
                    order_total NUMBER (8,2))
    ORGANIZATION EXTERNAL (TYPE oracle_hive);
```

Because no access parameters are set in the statement, the `ORACLE_HIVE` access driver uses the default settings to do the following:

- Connects to the default Hadoop cluster.
- Uses a Hive table named `order`. An error results if the Hive table does not have fields named `CUST_NUM`, `ORDER_NUM`, `DESCRIPTION`, and `ORDER_TOTAL`.
- Sets the value of a field to `NULL` if there is a conversion error, such as a `CUST_NUM` value longer than 10 bytes.

2.1.2.4.2 Overriding the Default ORACLE_HIVE Settings

You can set properties in the `ACCESS PARAMETERS` clause of the external table clause, which override the default behavior of the access driver. The following clause includes the `com.oracle.bigdata.overflow` access parameter. When this clause is used in the previous example, it truncates the data for the `DESCRIPTION` column that is longer than 100 characters, instead of throwing an error:

```
(TYPE oracle_hive
 ACCESS PARAMETERS (
   com.oracle.bigdata.overflow={"action":"truncate", "col":"DESCRIPTION" } ))
```

The next example sets most of the available parameters for `ORACLE_HIVE`:

```
CREATE TABLE order (cust_num VARCHAR2(10),
                    order_num VARCHAR2(20),
                    order_date DATE,
                    item_cnt NUMBER,
                    description VARCHAR2(100),
                    order_total (NUMBER(8,2)) ORGANIZATION EXTERNAL
    (TYPE oracle_hive
     ACCESS PARAMETERS (
       com.oracle.bigdata.tablename: order_db.order_summary
       com.oracle.bigdata.colmap:    {"col":"ITEM_CNT", \
                                     "field":"order_line_item_count"}
       com.oracle.bigdata.overflow:  {"action":"TRUNCATE", \
                                     "col":"DESCRIPTION"}
       com.oracle.bigdata.erroropt:  [{"action":"replace", \
                                     "value":"INVALID_NUM" , \
                                     "col":["CUST_NUM", "ORDER_NUM"]} ,\
                                     {"action":"reject", \
                                     "col":"ORDER_TOTAL"}
     ))
```

The parameters make the following changes in the way that the `ORACLE_HIVE` access driver locates the data and handles error conditions:

- `com.oracle.bigdata.tablename`: Handles differences in table names. `ORACLE_HIVE` looks for a Hive table named `ORDER_SUMMARY` in the `ORDER.DB` database.
- `com.oracle.bigdata.colmap`: Handles differences in column names. The Hive `ORDER_LINE_ITEM_COUNT` field maps to the Oracle `ITEM_CNT` column.
- `com.oracle.bigdata.overflow`: Truncates string data. Values longer than 100 characters for the `DESCRIPTION` column are truncated.
- `com.oracle.bigdata.erroropt`: Replaces bad data. Errors in the data for `CUST_NUM` or `ORDER_NUM` set the value to `INVALID_NUM`.

 **See Also:**

The section [CREATE TABLE ACCESS PARAMETERS Clause](#) provides the complete list of access parameters for `ORACLE_HIVE`, `ORACLE_HDFS`, and `ORACLE_BIGDATA`.

2.1.2.5 Hive to Oracle Data Type Conversions

When the access driver loads data into an external table, it verifies that the Hive data can be converted to the data type of the target column. If they are incompatible, then the access driver returns an error. Otherwise, it makes the appropriate data conversion.

Hive typically provides a table abstraction layer over data stored elsewhere, such as in HDFS files. Hive uses a serializer/deserializer (SerDe) to convert the data as needed from its stored format into a Hive data type. The access driver then converts the data from its Hive data type to an Oracle data type. For example, if a Hive table over a text file has a `BIGINT` column, then the SerDe converts the data from text to `BIGINT`. The access driver then converts the data from `BIGINT` (a Hive data type) to `NUMBER` (an Oracle data type).

Performance is better when one data type conversion is performed instead of two. The data types for the fields in the HDFS files should therefore indicate the data that is actually stored on disk. For example, JSON is a clear text format, therefore all data in a JSON file is text. If the Hive type for a field is `DATE`, then the SerDe converts the data from string (in the data file) to a Hive date. Then the access driver converts the data from a Hive date to an Oracle date. However, if the Hive type for the field is string, then the SerDe does not perform a conversion, and the access driver converts the data from string to an Oracle date. Queries against the external table are faster in the second example, because the access driver performs the only data conversion.

The table below identifies the data type conversions that `ORACLE_HIVE` can make when loading data into an external table.

Table 2-1 Supported Hive to Oracle Data Type Conversions

Hive Data Type	VARCHAR2, CHAR, NCHAR2, NCHAR, CLOB	NUMBER, FLOAT, BINARY_INTEGER, NUMBER, BINARY_FLOAT	BLOB	RAW	DATE, TIMESTAMP, TIMESTAMP WITH TZ, TIMESTAMP WITH LOCAL TZ	INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
INT SMALLINT TINYINT BIGINT	yes	yes	yes	yes	no	no
DOUBLE FLOAT	yes	yes	yes	yes	no	no
DECIMAL	yes	yes	no	no	no	no
BOOLEAN	yes ¹	yes ²	yes ²	yes	no	no
BINARY	yes	no	yes	yes	no	no
STRING	yes	yes	no	no	yes	yes
TIMESTAMP	yes	no	no	no	yes	no
STRUCT ARRAY UNIONTYPE E MAP	yes	no	no	no	no	no

¹ FALSE maps to the string FALSE, and TRUE maps to the string TRUE.

² FALSE maps to 0, and TRUE maps to 1.

2.1.3 Creating an Oracle External Table for Oracle NoSQL Database

You can use the `ORACLE_HIVE` access driver to access data stored in Oracle NoSQL Database. However, you must first create a Hive external table that accesses the KVStore. Then you can create an external table in Oracle Database over it, similar to the process described in "[Creating an Oracle External Table for Hive Data](#)".

This section contains the following topics:

- [Creating a Hive External Table for Oracle NoSQL Database](#)
- [Creating the Oracle Database Table for Oracle NoSQL Data](#)
- [About Oracle NoSQL to Oracle Database Type Mappings](#)
- [Example of Accessing Data in Oracle NoSQL Database](#)

2.1.3.1 Creating a Hive External Table for Oracle NoSQL Database

To provide access to the data in Oracle NoSQL Database, you create a Hive external table over the Oracle NoSQL table. Oracle Big Data SQL provides a `StorageHandler`

named `oracle.kv.hadoop.hive.table.TableStorageHandler` that enables Hive to read the Oracle NoSQL Database table format.

The following is the basic syntax of a Hive `CREATE TABLE` statement for a Hive external table over an Oracle NoSQL table:

```
CREATE EXTERNAL TABLE tablename colname coltype[, colname coltype,...]
STORED BY 'oracle.kv.hadoop.hive.table.TableStorageHandler'
TBLPROPERTIES (
    "oracle.kv.kvstore" = "database",
    "oracle.kv.hosts" = "nosql_node1:port[, nosql_node2:port...]",
    "oracle.kv.hadoop.hosts" = "hadoop_node1[,hadoop_node2...]",
    "oracle.kv.tableName" = "table_name");
```

Hive `CREATE TABLE` Parameters

tablename

The name of the Hive external table being created.

This table name will be used in SQL queries issued in Oracle Database, so choose a name that is appropriate for users. The name of the external table that you create in Oracle Database must be identical to the name of this Hive table.

Table, column, and field names are case insensitive in Oracle NoSQL Database, Apache Hive, and Oracle Database.

colname coltype

The names and data types of the columns in the Hive external table. See [Table 2-2](#) for the data type mappings between Oracle NoSQL Database and Hive.

Hive `CREATE TABLE TBLPROPERTIES` Clause

oracle.kv.kvstore

The name of the KVStore. Only upper- and lowercase letters and digits are valid in the name.

oracle.kv.hosts

A comma-delimited list of host names and port numbers in the Oracle NoSQL Database cluster. Each string has the format `hostname:port`. Enter multiple names to provide redundancy in the event that a host fails.

oracle.kv.hadoop.hosts

A comma-delimited list of all host names in the Hadoop cluster with Oracle Big Data SQL enabled.

oracle.kv.tableName

The name of the table in Oracle NoSQL Database that stores the data for this Hive external table.

See Also:

Apache Hive Language Manual DDL at

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create/Drop/TruncateTable>

2.1.3.2 Creating the Oracle Database Table for Oracle NoSQL Data

Use the following syntax to create an external table in Oracle Database that can access the Oracle NoSQL data through a Hive external table:

```
CREATE TABLE tablename(colname colType[, colname colType...])
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_HIVE DEFAULT DIRECTORY directory
  ACCESS PARAMETERS
    (access parameters)
  )
  REJECT LIMIT UNLIMITED;
```

In this syntax, you identify the column names and data types. For more about this syntax, see "[About the SQL CREATE TABLE Statement](#)".

2.1.3.3 About Oracle NoSQL to Oracle Database Type Mappings

When Oracle Big Data SQL retrieves data from Oracle NoSQL Database, the data is converted twice to another data type:

- To a Hive data type when the data is read into the columns of the Hive external table.
- To an Oracle data type when the data is read into the columns of an Oracle Database external table.

In order to execute a Big Data SQL query against data stored in an Oracle NoSQL Database table, a Hive *external table* must first be created with a schema mapped from the schema of the desired Oracle NoSQL Database table. [Table 2-2](#) identifies the supported data types Oracle NoSQL Database table API and their mappings to Hive.

Table 2-2 Mapping Hive Data Types to the NoSQL Database Table API Data Model

Oracle NoSQL Database Table API	Hive
FieldDef.Type.STRING	STRING
FieldDef.Type.BOOLEAN	BOOLEAN
FieldDef.Type.BINARY	BINARY
FieldDef.Type.FIXED_BINARY	BINARY
FieldDef.Type.INTEGER	INT
FieldDef.Type.LONG	BIGINT
FieldDef.Type.FLOAT	FLOAT
FieldDef.Type.DOUBLE	DOUBLE
FieldDef.Type.ENUM	STRING
FieldDef.Type.ARRAY	ARRAY
FieldDef.Type.MAP	MAP<STRING, data_type>
FieldDef.Type.RECORD	STRUCT<col_name : data_type, ...>

 **Note:**

To complete this mapping a corresponding Oracle Database *external table* must be created with a schema mapped from the schema of the Hive table.

Also note that the following Hive data types are not applicable to the mapping of Oracle NoSQL data types to Oracle Database data types: VARCHAR, CHAR, TINYINT, SMALLINT, DECIMAL, TIMESTAMP, DATE, UNION TYPE.

 **See Also:**

[Hive to Oracle Data Type Conversions](#) provides details on Hive to Oracle Database data type mappings. Predicate Pushdown in Oracle Big Data SQL requires that certain mappings between Hive Datatypes and Oracle Datatypes be present. See [About Predicate Push Down](#).

2.1.3.4 Example of Accessing Data in Oracle NoSQL Database

This example uses the sample data provided with the Oracle NoSQL Database software:

- [Creating the Oracle NoSQL Database Example Table](#)
- [Creating the Example Hive Table for vehicleTable](#)
- [Creating the Oracle Table for VEHICLES](#)

2.1.3.4.1 Creating the Oracle NoSQL Database Example Table

Verify that the following files reside in the `examples/hadoop/table` directory:

```
create_vehicle_table.kvs  
CountTableRows.java  
LoadVehicleTable.java
```

This example runs on a Hadoop cluster node named `some1node07` and uses a KVStore named `SOME1KV`.

To create and populate the sample table in Oracle NoSQL Database:

1. Open a connection to an Oracle NoSQL Database node on your Hadoop cluster.
2. Create a table named `vehicleTable`. The following example uses the `load` command to run the commands in `create_vehicle_table.kvs`:

```
$ cd NOSQL_HOME  
$ java -jar lib/kvcli.jar -host some1node07 -port 5000 \  
  load -file examples/hadoop/table/create_vehicle_table.kvs
```

3. Compile `LoadVehicleTable.java`:

```
$ javac -cp examples:lib/kvclient.jar examples/hadoop/table/  
LoadVehicleTable.java
```

4. Execute the `LoadVehicleTable` class to populate the table:

```
$ java -cp examples:lib/kvclient.jar hadoop.table.LoadVehicleTable -host
some1node07 -port 5000 -store SOME1KV
{"type":"auto","make":"Chrysler","model":"PTCruiser","class":"4WheelDrive","c
olo
r":"white","price":20743.240234375,"count":30}
{"type":"suv","make":"Ford","model":"Escape","class":"FrontWheelDrive","color
":
.
.
.
10 new records added
```

The `vehicleTable` table contains the following fields:

Table 2-3 Fields in the `vehicleTable` Example

Field Name	Data Type
<code>type</code>	STRING
<code>make</code>	STRING
<code>model</code>	STRING
<code>class</code>	STRING
<code>color</code>	STRING
<code>price</code>	DOUBLE
<code>count</code>	INTEGER

2.1.3.4.2 Creating the Example Hive Table for `vehicleTable`

The following example creates a Hive table named `VEHICLES` that accesses `vehicleTable` in the `SOME1KV` KVStore. In this example, the system is configured with a Hadoop cluster in the first six servers (`some1node01` to `some1node06`) and an Oracle NoSQL Database cluster in the next three servers (`some1node07` to `some1node09`).

```
CREATE EXTERNAL TABLE IF NOT EXISTS vehicles
(type STRING,
 make STRING,
 model STRING,
 class STRING,
 color STRING,
 price DOUBLE,
 count INT)
COMMENT 'Accesses data in vehicleTable in the SOME1KV KVStore'
STORED BY 'oracle.kv.hadoop.hive.table.TableStorageHandler'
TBLPROPERTIES
 ("oracle.kv.kvstore" = "SOME1KV",
 "oracle.kv.hosts" =
 "some1node07.example.com:5000,some1node08.example.com:5000",
 "oracle.kv.hadoop.hosts" =
 "some1node01.example.com,some1node02.example.com,some1node03.example.com,some1nod
e04.example.com,some1node05.example.com,some1node06.example.com",
 "oracle.kv.tableName" = "vehicleTable");
```

The `DESCRIBE` command lists the columns in the `VEHICLES` table:


```
hive> DESCRIBE vehicles;
OK
type                string                from deserializer
make                 string                from deserializer
model                string                from deserializer
class                string                from deserializer
color                string                from deserializer
price                double                from deserializer
count                int                    from deserializer
```

A query against the Hive `VEHICLES` table returns data from the Oracle NoSQL `vehicleTable` table:

```
hive> SELECT make, model, class
      FROM vehicletable
      WHERE type='truck' AND color='red'
      ORDER BY make, model;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
.
.
.
Chrysler      Ram1500        RearWheelDrive
Chrysler      Ram2500        FrontWheelDrive
Ford           F150           FrontWheelDrive
Ford           F250           RearWheelDrive
Ford           F250           AllWheelDrive
Ford           F350           RearWheelDrive
GM             Sierra         AllWheelDrive
GM             Silverado1500  RearWheelDrive
GM             Silverado1500  AllWheelDrive
```

2.1.3.4.3 Creating the Oracle Table for VEHICLES

After you create the Hive table, the metadata is available in the Oracle Database static data dictionary views. The following SQL `SELECT` statement returns information about the Hive table created in the previous topic:

```
SQL> SELECT table_name, column_name, hive_column_type
      FROM all_hive_columns
      WHERE table_name='vehicles';
TABLE_NAME      COLUMN_NAME      HIVE_COLUMN_TYPE
-----
vehicles         type             string
vehicles         make             string
vehicles         model            string
vehicles         class            string
vehicles         color            string
vehicles         price            double
vehicles         count            int
```

The next SQL `CREATE TABLE` statement generates an external table named `VEHICLES` over the Hive `VEHICLES` table, using the `ORACLE_HIVE` access driver. The name of the table in Oracle Database must be identical to the name of the table in Hive. However, both Oracle NoSQL Database and Oracle Database are case insensitive.

```
CREATE TABLE vehicles
  (type VARCHAR2(10), make VARCHAR2(12), model VARCHAR2(20),
   class VARCHAR2(40), color VARCHAR2(20), price NUMBER(8,2),
```

```

count NUMBER)
ORGANIZATION EXTERNAL
  (TYPE ORACLE_HIVE DEFAULT DIRECTORY DEFAULT_DIR
   ACCESS PARAMETERS
     (com.oracle.bigdata.debug=true com.oracle.bigdata.log.opt=normal))
REJECT LIMIT UNLIMITED;

```

This SQL `SELECT` statement retrieves all rows for red trucks from `vehicleTable` in Oracle NoSQL Database:

```

SQL> SELECT make, model, class
      FROM vehicles
      WHERE type='truck' AND color='red'
      ORDER BY make, model;

```

MAKE	MODEL	CLASS
Chrysler	Ram1500	RearWheelDrive
Chrysler	Ram2500	FrontWheelDrive
Ford	F150	FrontWheelDrive
Ford	F250	AllWheelDrive
Ford	F250	RearWheelDrive
Ford	F350	RearWheelDrive
GM	Sierra	AllWheelDrive
GM	Silverado1500	RearWheelDrive
GM	Silverado1500	4WheelDrive
GM	Silverado1500	AllWheelDrive

2.1.4 Creating an Oracle External Table for Apache HBase

You can also use the `ORACLE_HIVE` access driver to access data stored in Apache HBase. However, you must first create a Hive external table that accesses the HBase table. Then you can create an external table in Oracle Database over it. The basic steps are the same as those described in ["Creating an Oracle External Table for Oracle NoSQL Database"](#).

2.1.4.1 Creating a Hive External Table for HBase

To provide access to the data in an HBase table, you create a Hive external table over it. Apache provides a storage handler and a SerDe that enable Hive to read the HBase table format.

The following is the basic syntax of a Hive `CREATE TABLE` statement for an external table over an HBase table:

```

CREATE EXTERNAL TABLE tablename colname coltype[, colname coltype,...]
ROW FORMAT
  SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe'
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES (
  'serialization.format'='1',
  'hbase.columns.mapping'=':key,value:key,value:

```

 **See Also:**

- *Apache Hive Language Manual DDL* at <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create/Drop/TruncateTable>
- *Hive HBase Integration* at <https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration#HBaseIntegration-StorageHandlers>
- Class `HBaseSerDe` in the Apache Hive Javadocs at <https://hive.apache.org/javadoc.html>

2.1.4.2 Creating the Oracle Database Table for HBase

Use the following syntax to create an external table in Oracle Database that can access the HBase data through a Hive external table:

```
CREATE TABLE tablename(colname colType[, colname colType...])
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_HIVE DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS
    (access parameters)
  )
  REJECT LIMIT UNLIMITED;
```

In this syntax, you identify the column names and data types. To specify the access parameters, see "[About the SQL CREATE TABLE Statement](#)".

2.1.5 Creating an Oracle External Table for HDFS Files

The `ORACLE_HDFS` access driver enables you to access many types of data that are stored in HDFS, but which do not have Hive metadata. You can define the record format of text data, or you can specify a SerDe for a particular data format.

You must create the external table for HDFS files manually, and provide all the information the access driver needs to locate the data, and parse the records and fields. The following are some examples of `CREATE TABLE ORGANIZATION EXTERNAL` statements.

2.1.5.1 Using the Default Access Parameters with `ORACLE_HDFS`

The following statement creates a table named `ORDER` to access the data in all files stored in the `/usr/cust/summary` directory in HDFS:

```
CREATE TABLE ORDER (cust_num VARCHAR2(10),
                    order_num VARCHAR2(20),
                    order_total NUMBER (8,2))
  ORGANIZATION EXTERNAL
  ( TYPE oracle_hdfs
  DEFAULT DIRECTORY DEFAULT_DIR
  )
  LOCATION ('hdfs:/usr/cust/summary/*');
```

Because no access parameters are set in the statement, the `ORACLE_HDFS` access driver uses the default settings to do the following:

- Connects to the default Hadoop cluster.
- Reads the files as delimited text, and the fields as type `STRING`.
- Assumes that the number of fields in the HDFS files match the number of columns (three in this example).
- Assumes the fields are in the same order as the columns, so that `CUST_NUM` data is in the first field, `ORDER_NUM` data is in the second field, and `ORDER_TOTAL` data is in the third field.
- Rejects any records in which the value causes a data conversion error: If the value for `CUST_NUM` exceeds 10 characters, the value for `ORDER_NUM` exceeds 20 characters, or the value of `ORDER_TOTAL` cannot be converted to `NUMBER`.

See Also:

The section [CREATE TABLE ACCESS PARAMETERS Clause](#) provides the complete list of access parameters for `ORACLE_HIVE`, `ORACLE_HDFS`, and `ORACLE_BIGDATA`.

2.1.5.2 ORACLE_HDFS LOCATION Clause

The `LOCATION` clause for `ORACLE_HDFS` contains a comma-separated list of file locations. The files must reside in the HDFS file system on the default cluster.

A location can be any of the following:

- A fully qualified HDFS directory name, such as `/user/hive/warehouse/hive_seed/hive_types`. `ORACLE_HDFS` uses all files in the directory.
- A fully qualified HDFS file name, such as `/user/hive/warehouse/hive_seed/hive_types/hive_types.csv`
- A URL for an HDFS file or a set of files, such as `hdfs:/user/hive/warehouse/hive_seed/hive_types/*`. It is invalid to use the directory name alone.

The file names can contain any pattern-matching character described in [Table 2-4](#).

Table 2-4 Pattern-Matching Characters

Character	Description
<code>?</code>	Matches any one character
<code>*</code>	Matches zero or more characters
<code>[abc]</code>	Matches one character in the set <code>{a, b, c}</code>
<code>[a-b]</code>	Matches one character in the range <code>{a...b}</code> . The character must be less than or equal to <code>b</code> .
<code>[^a]</code>	Matches one character that is not in the character set or range <code>{a}</code> . The carat (<code>^</code>) must immediately follow the left bracket, with no spaces.
<code>\c</code>	Removes any special meaning of <code>c</code> . The backslash is the escape character.

Table 2-4 (Cont.) Pattern-Matching Characters

Character	Description
{ab\,cd}	Matches a string from the set { <i>ab, cd</i> }. The escape character (\) removes the meaning of the comma as a path separator.
{ab\,c{de\,fh}	Matches a string from the set { <i>ab, cde, cfh</i> }. The escape character (\) removes the meaning of the comma as a path separator.

2.1.5.3 Overriding the Default ORACLE_HDFS Settings

You can use many of the same access parameters with `ORACLE_HDFS` as `ORACLE_HIVE`.

2.1.5.3.1 Accessing a Delimited Text File

The following example is equivalent to the one shown in "[Overriding the Default ORACLE_HIVE Settings](#)". The external table accesses a delimited text file stored in HDFS.

```
CREATE TABLE taxis
(
  dispatching_base_num varchar2(100),
  pickup_date varchar2(100),
  location_id varchar2(100)
)
ORGANIZATION EXTERNAL
  (TYPE ORACLE_HDFS
   DEFAULT DIRECTORY DEFAULT_DIR
   ACCESS PARAMETERS
   (
     com.oracle.bigdata.fileformat=TEXTFILE
     com.oracle.bigdata.rowformat=DELIMITED FIELDS TERMINATED BY ','
   )
  LOCATION ('/data/taxi-trips/'))
REJECT LIMIT UNLIMITED;
```

Note that there is no `colmap` field, since this source contains no metadata that describes columns. The only access parameters needed in this case are `fileformat` and `rowformat`.

Tip:

Instead of a colon (:) as the delimiter between fields and values, it is better to use the equal sign (=). This is because certain tools prompt you for a parameter value when they encounter the colon delimiter.

2.1.5.3.2 Accessing Avro Container Files

The next example uses a SerDe to access Avro container files.

```
CREATE TABLE order (cust_num VARCHAR2(10),
                    order_num VARCHAR2(20),
                    order_date DATE,
```

```

        item_cnt NUMBER,
        description VARCHAR2(100),
        order_total NUMBER(8,2)
    ORGANIZATION EXTERNAL
    (
        TYPE oracle_hdfs
        DEFAULT DIRECTORY DEFAULT_DIR
        ACCESS PARAMETERS (
            com.oracle.bigdata.rowformat: \
            SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
            com.oracle.bigdata.fileformat: \
            INPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'\
            OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
            com.oracle.bigdata.colmap: { "col":"item_cnt", \
            "field":"order_line_item_count"}
            com.oracle.bigdata.overflow: {"action":"TRUNCATE", \
            "col":"DESCRIPTION"}
        )
        LOCATION ('hdfs:/usr/cust/summary/*');

```

The access parameters provide the following information to the ORACLE_HDFS access driver:

- `com.oracle.bigdata.rowformat`: Identifies the SerDe that the access driver needs to use to parse the records and fields.
- `com.oracle.bigdata.fileformat`: Identifies the Java classes that can extract records and output them in the desired format.
- `com.oracle.bigdata.colmap`: Handles differences in column names. ORACLE_HDFS matches `ORDER_LINE_ITEM_COUNT` in the HDFS files with the `ITEM_CNT` column in the external table.
- `com.oracle.bigdata.overflow`: Truncates string data. Values longer than 100 characters for the `DESCRIPTION` column are truncated.

2.1.5.3.3 Accessing JSON Data

Oracle Big Data SQL user functionality built into Oracle SQL in order to parse data in JSON format.

Oracle SQL can parse JSON data accessible in columns (which may be external data or data stored inside the database).

For example, here is a JSON file called `station_information.json`, stored in HDFS.

```

{
  "station_id": "72", "name": "W 52 St & 11 Ave", "short_name": "6926.01",
  "lat": 40.76727216, "lon": -73.99392888, "region_id": 71, "rental_methods":
  ["CREDITCARD", "KEY"],
  "capacity": 39, "rental_url": "http://app.citibikenyc.com/S6Lr/IBV092JufD?
  station_id=72",
  "eightd_has_key_dispenser": false} {"station_id": "79", "name": "Franklin St
  & W Broadway",
  "short_name": "5430.08", "lat": 40.71911552, "lon": -74.00666661, "region_id":
  71,
  "rental_methods": ["CREDITCARD", "KEY"],

```

```
"capacity":33,"rental_url":"http://app.citibikenyc.com/S6Lr/IBV092JufD?
station_id=79",
"eightd_has_key_dispenser":false}{ "station_id":"82",
"name":"St James Pl & Pearl
St", "short_name":"5167.06", "lat":40.71117416,
"lon":-74.00016545, "region_id":71, "rental_methods":["CREDITCARD", "KEY"],
"capacity":27, "rental_url":"http://app.citibikenyc.com/S6Lr/IBV092JufD?
station_id=82",
"eightd_has_key_dispenser":false
}
```

To query this data, do the following.

1. First create an HDFS external table over this data. Add a single text as column as VARCHAR2. (For large JSON objects, you can use CLOB instead.)

```
CREATE TABLE
  bikes.stations_ext (
    doc varchar2(4000)
  )
  ORGANIZATION EXTERNAL
  ( TYPE ORACLE_HDFS
    DEFAULT DIRECTORY DEFAULT_DIR
    LOCATION ('/data/bike-stations')
  )
  REJECT LIMIT UNLIMITED;
```

2. Then query the external table. Note the use of Oracle SQL's `JSON_VALUE` function.

```
select
  s.doc.station_id, s.doc.name, s.doc.lon as longitude,
  s.doc.lat as latitude, s.doc.capacity,
  s.doc.eightd_has_key_dispenser, s.doc.rental_methods,
  json_value(doc, '$.rental_methods[0]'),
  json_value(doc, '$.rental_methods[1]') from stations_exts
```



See Also:

The [Oracle SQL Language Reference](#) provides syntax and more examples for `JSON_VALUE`.

2.1.6 Creating an Oracle External Table for Kafka Topics

Oracle Big Data SQL has two options for working with Kafka data: using the Hive storage handler for Kafka, and using Oracle SQL access to Kafka.

- **Using the Hive storage handler for Kafka**
The `ORACLE_HIVE` access driver can access Kafka topics using the Hive storage handler that is part of the Oracle Big Data SQL install. You first create a Hive external table that accesses the Kafka topics, and then create an Oracle Big Data

SQL table over it. See [Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics](#).

- **Using Oracle SQL Access to Kafka**

Oracle SQL Access to Kafka enables Oracle SQL to access Kafka topics, without using Hive or Hadoop. The `ORA_KAFKA` PL/SQL package contains functions and procedures to enable this. You register a Kafka cluster in your database schema using `ORA_KAFKA.REGISTER_CLUSTER`, and then create views using the `ORA_KAFKA.CREATE_VIEWS` procedure. This procedure creates external tables under the covers, you do not have to explicitly create external tables. See [Oracle SQL Access to Kafka](#).

Comparison between the above two options:

Hive Storage Handler for Kafka	Oracle SQL Access to Kafka
Use Big Data SQL syntax	Use <code>ORA_KAFKA</code> functions and procedures.
Requires Hive and Hadoop	Does not require Hive and Hadoop, requires a Java process to run on the database system.
Uses predicate pushdown when answering queries: Best option for queries that require predicate pushdown. For example, queries that are searching through large volumes of data in Kafka. Predicate pushdown is supported for conjunctive queries (a AND b AND c) defined on the "partitioned", "topic", "offset" and "timestamp" columns.	Uses Kafka offset and Kafka timestamp management to read messages by offset or timestamp: Best option for queries looking for the most recent Kafka messages (for example, messages from 5 seconds ago until current time), or a sequence of messages starting from a specific offset or timestamp (for example a 1000 messages starting from timestamp t or offset o).
Supports messages in Avro (schema-less or schema-full) formats	Supports messages in CSV and JSON formats
Enables Hive query of Kafka topics	Enables multiple database views to query Kafka, each application can have its own set of views and offset management so that each application can read a different part of the Kafka stream.

2.1.6.1 Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics

The Hive storage handler for Kafka enables Hive and Oracle Big Data SQL to query Kafka topics.

To provide access to Kafka data, you create a Hive external table over the Kafka topics. The Oracle Big Data SQL storage handler that enables Hive to read the Kafka data format is `oracle.hadoop.kafka.hive.KafkaStorageHandler`.

You can use this storage handler to create external Hive tables backed by data residing in Kafka. Big Data SQL can then query the Kafka data through the external Hive tables.

The Hive DDL is demonstrated by the following example, where topic1 and topic2 are two topics in Kafka broker whose keys are serialized by Kafka's String serializer and whose values are serialized by Kafka's Long serializer.

```
CREATE EXTERNAL TABLE test_table
row format serde 'oracle.hadoop.kafka.hive.KafkaSerDe'
stored by 'oracle.hadoop.kafka.hive.KafkaStorageHandler'
tblproperties('oracle.kafka.table.key.type='string',
              'oracle.kafka.table.value.type='long',
              'oracle.kafka.bootstrap.servers='nshgc0602:9092',
              'oracle.kafka.table.topics='topic1,topic2');
```

The example below shows the resulting Hive table. The Kafka key, value, offset, topic name, and partitionid are mapped to Hive columns. You can explicitly designate the offset for each topic/partition pair through a WHERE clause in you Hive query.

```
hive> describe test_table;
OK
topic          string          from deserializer
partitionid    int            from deserializer
key            string          from deserializer
value          bigint         from deserializer
offset         bigint         from deserializer
timestamptype  smallInt       from deserializer
timestamp      timestamp      from deserializer
Time taken: 0.084 seconds, Fetched: 7 row(s)
```

The content of the table is a snapshot of the Kafka topics when the Hive query is executed. When new data is inserted into the Kafka topics, you can use the offset column or the timestamp column to track the changes to the topic. The offsets are per topic/partition. For example, the following query will return new messages after the specified offsets in the where clause for each topic/partition:

```
hive> SELECT * \
FROM test_table \
WHERE (topic="topic1" and partitoinid=0 and offset > 199) \
OR (topic="topic1" and partitionid=1 and offset > 198) \
OR (topic="topic2" and partitionid=0 and offset > 177) \
OR (topic="topic2" and partitionid=1 and offset > 176);
```

You need to keep track of the offsets for all topics and partitions. For example, you can use an Oracle table to store these offsets. A more convenient way to keep track of new data is using the timestamp column. You can query data after a specific time point using the following query:

```
hive> SELECT * FROM test_table WHERE timestamp > '2017-07-12 11:30:00';
```

See the Property Reference section below for descriptions of all table properties

Property Reference

Table 2-5 Table Properties of Hive Storage Handler for Kafka

Property Name	Requirement	Description
oracle.kafka.table.topics	Required	A comma-separated list of Kafka topics. Each Kafka topic name must consist of only letters (uppercase and lowercase), numbers, <code>.</code> (dot), <code>_</code> (underscore), and <code>-</code> (minus). The maximum length for each topic name is 249. These topics must have the same serialization mechanisms. The resulting Hive table consists of records from all the topics listed here. A Hive column “topic” will be added and it will be set to the topic name for each record.
oracle.kafka.bootstrap.servers	Required	This property will be translated to the “bootstrap.servers” property for the underlying Kafka consumer. The consumer makes use of all servers, irrespective of which servers are specified here for bootstrapping. This list only impacts the initial hosts used to discover the full set of servers. This list should be in the form <code>host1:port1,host2:port2,...</code> . Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers. For availability reasons, you may want to list more than one server.
oracle.kafka.table.key.type	Optional	The key type for your record. If unset, then the key part of the Kafka record will be ignored in the Hive row. Only values of “string”, “integer”, “long”, “double”, “avro”, “avro_confluent” are supported. “string”, “integer”, “double” and “long” correspond to the built-in primitive serialization types supported by Kafka. If this property is one of these primitive types, then the Kafka key for each record will be mapped to one single Hive Column. If this property is set to “avro” or “avro_confluent”, then <code>oracle.kafka.table.key.schema</code> is required. The Kafka key for each record will be deserialized into an Avro Object. If the Avro schema is of record type then each first level field of the record will be mapped to a single Hive column. If the Avro schema is not of Record Type, then it will be mapped to a single Hive Column named “key”. The difference between “avro” and “avro_confluent” is that the wire format for the serialization is slightly different. For “avro”, the entire bytes array of the key consists of the bytes of avro serialization. For “avro_confluent”, the bytes array consists of a magic byte, a version number, then the bytes of avro serialization of the key.
oracle.kafka.table.value.type	Optional	The value type of your record. If unset, then the value part of Kafka record will be ignored in the Hive row. Use of this property is similar to use of <code>oracle.kafka.table.key.type</code> . The difference between them is: when the Avro Schema for Kafka value is not of record type. The whole Avro object will be mapped to a single Hive Column named “value” instead of “key”.
oracle.kafka.table.key.writer.schema	Optional	An optional writer schema for the Kafka key’s Avro serialization. It’s required when the reader schema for the key is different from the schema in which the keys are written to Kafka brokers. It must be the exact schema in which Kafka keys are serialized.

Table 2-5 (Cont.) Table Properties of Hive Storage Handler for Kafka

Property Name	Requirement	Description
oracle.kafka.table.key.schema	Required when "oracle.kafka.table.key.type" is "avro" or "avro_confluent"	The JSON string for the Kafka key's Avro reader schema. It doesn't need to be exactly the same as the Kafka key's writer Avro schema. As long as the reader schema is compatible with the Kafka key or the converted object from the converter, it is valid. This enables you to rename Hive columns and choose what fields to keep from the Kafka key in the Hive row. If the schema in this property is different from the schema in which the Kafka keys are serialized, then oracle.kafka.table.key.writer.schema is required.
oracle.kafka.table.value.writer.schema	Optional	An optional writer schema for the Kafka value's Avro serialization. Its use is similar to oracle.kafka.table.key.writer.schema.
oracle.kafka.table.value.schema	Required when "oracle.kafka.table.value.type" is "avro" or "avro_confluent"	The JSON string for the Kafka value's Avro reader schema. Its use is similar to oracle.kafka.table.key.schema.
oracle.kafka.table.extra.columns	Optional, default to "true"	A boolean flag to control whether to include extra Kafka columns: partitionid, offset, timestamptype.
oracle.kafka.chop.partition	Optional, default to false	A Boolean flag to control whether to chop Kafka partitions into smaller chunks. This is useful when the number of Kafka partitions is small and the size of each Kafka partition is large.
oracle.kafka.partition.chunk.size	Optional	When oracle.kafka.chop.partition is true, this property controls the number of Kafka records in each partition chunk. It should be set a value estimated by (Ideal size of a split)/(Average size of a Kafka record). For example, if the ideal size of a split is 256 MB and the average size of a Kafka record is 256 Bytes, then this property should be set to 1000000.

2.1.6.2 Creating an Oracle Big Data SQL Table for Kafka Topics

Big Data SQL can use the ORACLE_HIVE access driver to query the Kafka source that is described using Hive metadata.

After you create a Hive table over Kafka data by using the Hive storage handler for Kafka, there are no special procedures for generating a Big Data SQL table from the resulting Hive table. The ORACLE_HIVE settings for Kafka sources are identical to other Hive sources. Below is an example that shows how to create the Oracle external table. Once created, you can query it as you would other Oracle tables.

```
CREATE TABLE test_table(
topic varchar2(50),
```

```

partitionid integer,
key varchar2(50),
value integer,
offset integer,
timestamptype integer,
timestamp      timestamp
)
ORGANIZATION EXTERNAL
(TYPE ORACLE_HIVE DEFAULT DIRECTORY DEFAULT_DIR
ACCESS PARAMETERS
(
com.oracle.bigdata.cluster=hadoop1
com.oracle.bigdata.tablename=default.test_table
)
) PARALLEL 2 REJECT LIMIT UNLIMITED

```

Some Common Questions

- *Is Oracle Big Data SQL access to Kafka Brokers parallelized? For example, if I have six nodes running Oracle Big Data SQL, will all six nodes participate in a single query to Kafka so that we have to create a consumer group across all nodes to read the Kafka topic? Or, will only one node be used for a single SELECT statement to query from Kafka?*

Like any Big Data SQL query, a query to Kafka will engage all of the nodes where Oracle Big Data SQL is installed.

- *In a Kafka query, how can we accommodate new incoming data? Can I have a query that waits (for a specified timeout) for new data to come into Kafka?*

To pick up new data, you can run the Oracle Big Data SQL query periodically and filter by offset and timestamp to only retrieve the new rows (rows since the last read).

See Also:

The following section of the Big Data SQL Quick Start blog provides more information on accessing Kafka through Oracle Big Data SQL – [Big Data SQL Quick Start. Big Data SQL over Kafka – Part 23](#)

2.1.7 Creating an Oracle External Table for Object Store Access

The `ORACLE_BIGDATA` access driver enables you to create an external table over data stored in object stores. Oracle Big Data SQL currently supports access to Oracle Object Store, Amazon S3, and Azure Blob Storage.

`ORACLE_BIGDATA` is primarily intended to support queries against object stores. It uses Smart Scan and Oracle Big Data SQL cells for scale out and performance against these stores. You can also use this driver to query local data, which is useful for testing and smaller data sets.

The `ORACLE_BIGDATA` driver is similar to the `ORACLE_HDFS` driver in that you create tables over the raw files. It does not use a metadata store like Hive. You specify the metadata as part of the table definition.

However, unlike `ORACLE_HDFS`, `ORACLE_BIGDATA` does not use Java drivers and the standard Hadoop mechanisms for data access (SerDes, InputFormats, etc.). `ORACLE_BIGDATA` uses optimized C-drivers for all data access. It supports the text, Avro, ORC and Parquet file types. The text file type support is robust. For example, you can specify parameters for delimited text. You can also utilize Oracle's extensive JSON processing capabilities.

Steps for Accessing Data in Object Store

There are two steps required in order to access data in an object store:

- Create a credential object
A credential object stores object store credentials in an encrypted format. The identity specified by the credential must have access to the underlying data in the object store.
- Create an Oracle Big Data SQL external table whose type is `ORACLE_BIGDATA`.
The create table statement must reference the credential object, which provides authentication against the object store. It also requires a `LOCATION` clause, which provides the URI to the files within the object store.

Creating the Credential Object

Use the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure to create your credential object. This object contains the username and password information needed to access the object store. This credential password must match the Auth Token created for the username in your cloud service.

```
execute dbms_credential.create_credential(  
    credential_name => '<my_credential>',  
    username       => '<username>',  
    password       => '<password>' )  
);
```

For the Native Oracle Cloud Infrastructure style credentials the object also contains the Key to access the object store. The Key must match the API Key for the specified user.

```
execute dbms_credential.create_credential(  
    credential_name => '<my_credential>',  
    username       => '<user_ocid>',  
    password       => '',  
    key            => '{  
        "tenancy_ocid": "<tenancy_ocid>",  
        "private_key": "<private_key>",  
        "fingerprint": "<fingerprint>"  
    }')
```

Then specify the credential object name in the `com.oracle.bigdata.credential.name` parameter.

 **See Also:**

The Oracle Database PL/SQL Packages and Types Reference describes how to use `DBMS_CREDENTIAL.CREATE_CREDENTIAL`

Defining the LOCATION Clause

The LOCATION is a URI pointing to data in the object store. Currently supported object stores are Oracle Object Store and Amazon S3. There are different ways of specifying the URI, depending on its source. Here are some examples.

- For Native Oracle Cloud Infrastructure Object Storage, the URI format is:

```
location ('https://objectstorage.<region>.oraclecloud.com/n/  
<namespace>/b/<bucket>/o/<objectname>')
```

- For Oracle Cloud Infrastructure Object Storage, the URI format for files is:

```
location ('https://swiftobjectstorage.<region>.oraclecloud.com/v1/  
<namespace>/<bucket>/<filename>')
```

- For an Amazon S3 location, see <https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html#access-bucket-intro>.

The hosted-style URI format is as follows:

```
location ('https://<bucket>.<host>/<objectname>')
```

The path-style URI format is as follows:

```
location ('https://<host>/<bucket>/<objectname>')
```

A possible path-style example is:

```
location ('https://s3-us-west-2.amazonaws.com/adwc/<filename>')
```

- For an Azure Blob Storage location, the URI format is:

```
location ('https://<host>:<port>/<container>/<blob>')
```

A possible example is:

```
location ('https://myaccount.blob.core.windows.net/mycontainer/  
myblob')
```

The credential object is required for object store access only. If the credential parameter is omitted, then the object must be in a public bucket. The user id associated with this credential must have access to read the data from object storage.

If you are testing access for data in object storage using local storage, you need to specify an Oracle directory object in the location - similar to what you do for `ORACLE_LOADER` data sources.

Setting Access Parameters

Like ORACLE_HDFS, ORACLE_BIGDATA requires information about how to access and parse the data. You provide this through access parameters. The following access parameter is required to specify the file format type:

```
com.oracle.bigdata.fileformat={textfile|avro|parquet|orc}
```

For delimited text files, the rowformat parameter is also required.

See Also:

[ORACLE_BIGDATA Access Parameters](#) in this guide provides tables that list common access parameters as well as those specific to each file type.

Note:

You can use ORACLE_BIGDATA to access local files for testing purposes or simple querying. In this case, the LOCATION field value is the same as what you would use for ORACLE_LOADER. You can use an Oracle directory object followed by the name of the file in the LOCATION field. For local files, a credential object is not required. However, you must have privileges over on the directory object in order to access the file.

2.1.7.1 Create Table Example for Object Store Access

This section describes how to build a CREATE TABLE statement for object store access.

As shown in the example below, the required components are as follows:

- The schema for the external table.
- The credential object.
- The fileformat parameter and any access parameters that are particular to the file format. For example, delimited Text files require the rowformat parameter. Parquet and Avro require only the fileformat parameter. Note that the default file format is Parquet.
- The correct LOCATION clause syntax for the particular object store.
- Use of a DEFAULT DIRECTORY clause with a LOCATION that is local rather than in an object store.

See Also:

[ORACLE_BIGDATA Access Parameters](#) for the full set of available parameters for each file type.

Example: Accessing a File in an Object Store

 **Note:**

Remember that for object store access you must first use `DBMS_CREDENTIAL.CREATE_CREDENTIAL` in the `DBMS_CREDENTIAL` PL/SQL package to create the credential object.

```
exec dbms_credential.create_credential(
credential_name => '<my_credential_object_name>',
username       => '<username>',
password       => '<password>'
);
```

Then within the `ACCESS PARAMETER` clause of the statement, assign the name of the object to `com.oracle.bigdata.credential.name` as shown in the statement. A Parquet file in an object store is the target in this example.

```
CREATE TABLE tkexbaseballtab
(date1      date,
date2      timestamp,
name       varchar2(30),
nationality varchar2(20),
age        number,
team       varchar2(20),
active     char(1),
average    float,
payroll    char(1),
exid       VARCHAR2(20))
ORGANIZATION EXTERNAL
(TYPE ORACLE_BIGDATA
ACCESS PARAMETERS
(
  com.oracle.bigdata.debug=TRUE
  com.oracle.bigdata.credential.name=MY_CRED
  com.oracle.bigdata.fileformat=parquet
)
location ('https://<domain>.com/BIGDATA_PARQUET/<filename>.parquet'))
) REJECT LIMIT UNLIMITED;
```

2.1.7.2 Accessing a Local File through an Oracle Directory Object

You can also use the `ORACLE_BIGDATA` driver to create an external table over a local file.

The statement below creates an external table over a text file in a local directory. For a JSON file, set the file format to `textfile`. No other access parameters are needed. You do not need to supply a credential when accessing local files. Do the same for a delimited text file, but if the fields terminator and/or line terminator used in the file are other than the default values, define those as well. In this example, we set the

field terminator to the vertical bar ('|'). This statement does not include a DEFAULT DIRECTORY clause. Instead in the LOCATION clause we include the name of the directory object where the file is located – DEF_DIR1 .

```
CREATE TABLE b19724081
(
  CONTRACT_STATUS_CODE VARCHAR2(1),
  CONTRACT_STATUS_DESCRIPTION VARCHAR2(200),
  POSTING_AGENT VARCHAR2(50),
  DATA_ORIGIN VARCHAR2(50),
  WAREHOUSE_POSTING_TIMESTAMP DATE,
  WAREHOUSE_UPDATE_TIMESTAMP DATE
)
ORGANIZATION EXTERNAL
( TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY "DEF_DIR1"
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.fileformat=textfile
    com.oracle.bigdata.csv.rowformat.fields.terminator = '|'
  )
  location ("DEF_DIR1": "<filename>.csv")
)
REJECT LIMIT UNLIMITED;
```

2.1.7.3 Conversions to Oracle Data Types

Oracle Big Data SQL supports conversion from Parquet, ORC, and Avro file data types to Oracle data types, but for scalar fields only. Non-scalar data types are converted to VARCHAR2(4000).

Complex types, such as arrays, structs, and so on, map to VARCHAR2 data types and are represented by a JSON string in a VARCHAR2 field.

Note:

If data for a column encounters a conversion error, for example, the target column is not large enough to hold the converted value, the value for the column will be set to NULL.

Table 2-6 ORACLE_BIGDATA Mappings From Parquet, ORC and Avro to Oracle Data Types

Type Description	Parquet	ORC	Avro	Supported Conversions to Oracle Data Types
decimal: arbitrary-precision signed decimal number of the form $\text{unscaled} \times 10^{\text{scale}}$	¹ decimal (int32, int64, fixed_len_byte_array, byte_array)	² decimal (int64, int128)	³ decimal (bytes, fixed)	number(p), number(p,s)
UTF8: UTF-8 encoded character string	string/utf8	char/string/ varchar	string	varchar2
byte array/binary	⁴ byte_array	binary	bytes	blob
byte array with fixed length	fixed_len_byte_array	-	fixed	blob
UUID	UUID (fixed_len_byte_array)	-	-	blob
Json	JSON (byte_array)	-	-	varchar2 2
Bson	Bson (byte_array)	-	-	-
date: number of days from the Unix epoch, 1 January 1970	date (int32)	date	date (int)	date
time (millis): number of milliseconds after midnight, 00:00:00.000	time-millis (int32)	-	time-millis (int)	timestamp(3)
time (micros): number of microseconds after midnight, 00:00:00.000000	time-micros (int64)	-	time-micros (long)	timestamp(6)
time (nanos): number of nanoseconds after midnight, 00:00:00.000000	time-nanos (int64)	-	-	timestamp(9)
time (millis) UTC	time-millis-UTC (int32)	-	-	timestamp(3)
time (micros) UTC	time-micros-UTC (int64)	-	-	timestamp(6)
time (nanos) UTC	time-nanos-UTC (int64)	-	-	timestamp(9)
timestamp (millis)	timestamp-millis (int64)	-	-	timestamp(3)

Table 2-6 (Cont.) ORACLE_BIGDATA Mappings From Parquet, ORC and Avro to Oracle Data Types

Type Description	Parquet	ORC	Avro	Supported Conversions to Oracle Data Types
timestamp (micros)	timestamp-micros (int64)	-	-	timestamp(6)
timestamp (nanos)	timestamp-nanos (int64)	timestamp	-	timestamp(9)
timestamp (millis) UTC: number of milliseconds from the unix epoch, 1 January 1970 00:00:00.000 UTC	timestampmillis-UTC (int64)	-	timestampmillis (long)	timestamp(3)
timestamp (micros) UTC: number of microseconds from the unix epoch, 1 January 1970 00:00:00.000000 UTC	timestampmicros-UTC (int64)	-	timestampmicros (long)	timestamp(6)
timestamp (nanos) UTC: number of nanoseconds from the unix epoch, 1 January 1970 00:00:00.000000 UTC	timestampnanos-UTC (int64)	-	-	timestamp(9)
duration: number of months, days and milliseconds	interval (fixed, size: 12)	-	duration(fixed, size:12)	-
8-bit signed integer	int_8 (int32)	-	-	number(3), tinyint
16-bit signed integer	int_16 (int32)	-	-	number(5), smallint
32-bit signed integer	int_32 (int32)	-	int	number(10), int
64-bit signed integer	int_64 (int64)	-	long	number(20), bigint
8-bit unsigned integer	uint_8 (int32)	byte	-	number(3), tinyint
16-bit unsigned integer	uint_16 (int32)	short	-	number(5), smallint
32-bit unsigned integer	uint_32 (int32)	int	-	number(10), int
64-bit unsigned integer	uint_64 (int64)	long	-	number(20), bigint

Table 2-6 (Cont.) ORACLE_BIGDATA Mappings From Parquet, ORC and Avro to Oracle Data Types

Type Description	Parquet	ORC	Avro	Supported Conversions to Oracle Data Types
96-bit signed integer	⁵ int96	-	-	number(29), timestamp(9)
IEEE 32-bit floating point	float	float	float	binary_float
IEEE 64-bit floating point	double	double	double	binary_double
boolean	boolean	boolean	boolean	number(1)
null/no value	null	-	null	varchar2(1)
enum	enum(int)	-	enum(int)	varchar2

1. In Parquet, a decimal can be stored in int_32, int_64, as a fixed length byte array or as a byte array whose size is determined by the precision.
2. In ORC, decimals are stored in 128 bit integers.
3. In Avro, decimals are internally stored as byte arrays (fixed or not). Depending on the Avro writer, some of them store the string representation of the decimal, while others store the unscaled value. To avoid presenting ambiguous data to the user, it is recommended that the access parameter `com.oracle.bigdata.avro.decimaltype` is used to explicitly declare which representation is used in the file. If this parameter is not explicitly specified we assume that the unscaled representation of the data is stored in the decimal columns of the file. See [ORACLE_BIGDATA Access Parameters](#).
4. The binary type for Parquet is only available as a blob when `com.oracle.bigdata.prq.binary_as_string` is set to `FALSE`. See [ORACLE_BIGDATA Access Parameters](#).
5. The int_96 type for Parquet is only available as number when `com.oracle.bigdata.prq.int96_as_timestamp` access parameter is set to `FALSE`. See [ORACLE_BIGDATA Access Parameters](#).

2.1.7.4 ORACLE_BIGDATA Support for Compressed Files

ORACLE_BIGDATA driver support for access to compressed files in object stores is as follows:

- Compressed Text, Parquet, Avro and ORC files
The driver can read from text, Parquet, Avro and ORC files compressed with gzip, bzip2, or zlib. The compression format can be detected automatically or specified. No parameters to handle compressed files are needed in the external table create statement.

The following ORACLE_BIGDATA access parameter is used to define the compression type:

```
com.oracle.bigdata.compressiontype=detect|gzip|zlib|bzip2
```

Here is an example of defining a gzip compression type:

```
com.oracle.bigdata.compressiontype=gzip
```

See [ORACLE_BIGDATA Access Parameters](#) for further details.

2.2 Querying External Tables

Users can query external tables using the SQL `SELECT` statement, the same as they query any other table.



Note:

The `MODIFY EXTERNAL` clause is not allowed for any external table created through the `ORACLE_BIGDATA` driver.

2.2.1 Granting User Access

Users who query the data on a Hadoop cluster must have `READ` access in Oracle Database to the external table and to the database directory object that points to the cluster directory. See "[About the Cluster Directory](#)".

2.2.2 About Error Handling

By default, a query returns no data if an error occurs while the value of a column is calculated. Processing continues after most errors, particularly those thrown while the column values are calculated.

Use the [com.oracle.bigdata.erroropt](#) and [com.oracle.bigdata.overflow](#) parameters to determine how errors are handled.

2.2.3 About the Log Files

You can use these access parameters to customize the log files:

- [com.oracle.bigdata.log.exec](#)
- [com.oracle.bigdata.log.qc](#)

2.2.4 About File Readers

2.2.4.1 Using Oracle's Optimized Parquet Reader for Hadoop Sources

For reading parquet files, you have the option of using the custom Parquet reader for Hadoop sources. This proprietary driver improves performance and makes more efficient use of cluster resources.

Disabling or Re-Enabling the Custom Parquet Reader

The Parquet reader optimization is enabled by default. It can be disabled for an individual table by adding the following access parameter to the external table definition:

```
com.oracle.bigdata.useOracleParquet=false
```

You can add this setting to the cluster properties file to disable the optimization for all Parquet-based external tables. Remove the setting to return to the default.

Compatibility with Previously Created Parquet Format Data

Use of the custom reader requires no changes to data format. However, for best performance, the format must provide min and max values for each column for each Parquet block. These values are used by the standard Hadoop Parquet InputFormat, as well as the custom Parquet reader, to optimize the query. The resulting optimization significantly improves query performance with both Hive and Oracle Big Data SQL.

Note that Parquet files created by Impala do not include min and max values for each column for each Parquet block.

To ensure that min and max values are available, it is recommended that you write Parquet files with Hive or other tools that generate output in the standard Hadoop Parquet InputFormat, such as PrestoDB and Spark.

To check if a file includes these values, you can run the `help about parquet-tools`. On a CDH Hadoop distro, the `parquet-tools` command may also be configured in your path.

Here's an example of how to run parquet tools against a file called `sales.parq`:

```
parquet-tools meta sales.parq
```

The resulting output should contain for each column a min, max and number of nulls value:

```
c_quantity : ... :[min: 0, max: 109572, num_nulls: 55]
c_total_sales : ... :[min: 0, max: 109571, num_nulls: 41]
```

2.3 About Oracle Big Data SQL on the Database Server (Oracle Exadata Machine or Other)

This section explains the changes that the Oracle Big Data SQL installation makes to the Oracle Database system (which may or may not be an Oracle Exadata Machine).

The section contains the following topics:

- [About the bigdata_config Directory](#)
- [Common Configuration Properties](#)
- [About the Cluster Directory](#)
- [About Permissions](#)

2.3.1 About the bigdata_config Directory

The directory `bigdata_config` contains configuration information that is common to all Hadoop clusters. This directory is located on the Oracle Database system under `$ORACLE_HOME/bigdatasql`. The oracle file system user (or whichever user owns the Oracle Database instance) owns `bigdata_config`. The Oracle Database directory `ORACLE_BIGDATA_CONFIG` points to `bigdata_config`.

2.3.2 Common Configuration Properties

The installation store these files in the `bigdata_config` directory under `$ORACLE_HOME/bigdatasql`:

- [bigdata.properties](#)
- [bigdata-log4j.properties](#)

The Oracle DBA can edit these configuration files as necessary.

2.3.2.1 bigdata.properties

The `bigdata.properties` file in the common directory contains property-value pairs that define the Java class paths and native library paths required for accessing data in HDFS.

These properties must be set:


- [bigdata.cluster.default](#)
- [java.classpath.hadoop](#)
- [java.classpath.hive](#)
- [java.classpath.oracle](#)

The following list describes all properties permitted in `bigdata.properties`.

bigdata.properties

Property	Description
bigdata.cluster.default	The name of the default Hadoop cluster. The access driver uses this name when the access parameters do not specify a cluster. Required. Changing the default cluster name might break external tables that were created previously without an explicit cluster name.
bigdata.cluster.list	A comma-separated list of Hadoop cluster names. Optional.

Property	Description
java.classpath.hadoop	The Hadoop class path. Required.
java.classpath.hive	The Hive class path. Required.
java.classpath.oracle	The path to the Oracle JXAD Java JAR file. Required.
java.classpath.user	The path to user JAR files. Optional.
java.libjvm.file	The full file path to the JVM shared library (such as <code>libjvm.so</code>). Required.
java.options	<p>A comma-separated list of options to pass to the JVM. Optional.</p> <p>This example sets the maximum heap size to 2 GB, and verbose logging for Java Native Interface (JNI) calls:</p> <pre>Xmx2048m,-verbose=jni</pre>
java.options2	<p>A space-delimited list of options to pass to the JVM. Optional. The delimiter must be a space character, not a tab or other whitespace character.</p> <p>This example sets the maximum heap size to 2 GB, and verbose logging for Java Native Interface (JNI) calls:</p> <pre>Xmx2048m -verbose=jni</pre>
LD_LIBRARY_PATH	<p>A colon separated (:) list of directory paths to search for the Hadoop native libraries. Recommended.</p> <p>If you set this option, then do not set <i>java.library</i> path in java.options.</p>

 **Note:**

Notice that `java.options` is comma-delimited, while `java.options2` is space delimited. These two properties can coexist in the same `bigdata.properties` file.

Example 2-1 shows a sample `bigdata.properties` file.

Example 2-1 Sample bigdata.properties File

```
# bigdata.properties
#
```



```

# Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
#
#   NAME
#     bigdata.properties - Big Data Properties File
#
#   DESCRIPTION
#     Properties file containing parameters for allowing access to Big Data
#     Fixed value properties can be added here
#

java.libjvm.file=$ORACLE_HOME/jdk/jre/lib/amd64/server/libjvm.so
java.classpath.oracle=$ORACLE_HOME/hadoopcore/jlib/*:$ORACLE_HOME/hadoop/jlib/
hwer-2/*:$ORACLE_HOME/dbjava/lib/*
java.classpath.hadoop=$HADOOP_HOME/*:$HADOOP_HOME/lib/*
java.classpath.hive=$HIVE_HOME/lib/*
LD_LIBRARY_PATH=$ORACLE_HOME/jdk/jre/lib
bigdata.cluster.default=hadoop_cl_1

```

2.3.2.2 bigdata-log4j.properties

The `bigdata-log4j.properties` file in the common directory defines the logging behavior of queries against external tables in the Java code. Any `log4j` properties are allowed in this file.

[Example 2-2](#) shows a sample `bigdata-log4j.properties` file with the relevant `log4j` properties.

Example 2-2 Sample bigdata-log4j.properties File

```

# bigdata-log4j.properties
#
# Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
#
#   NAME
#     bigdata-log4j.properties - Big Data Logging Properties File
#
#   DESCRIPTION
#     Properties file containing logging parameters for Big Data
#     Fixed value properties can be added here

bigsql.rootlogger=INFO,console
log4j.rootlogger=DEBUG, file
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{2}:
%m%n
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{2}: %m%n
log4j.logger.oracle.hadoop.sql=ALL, file

bigsql.log.dir=.
bigsql.log.file=bigsql.log
log4j.appender.file.File=$ORACLE_HOME/bigdatalogs/bigdata-log4j.log

```

 **See Also:**

Apache Logging Services documentation at

<http://logging.apache.org/log4j/1.2/manual.html>

2.3.3 About the Cluster Directory

The cluster directory contains configuration information for a Hadoop cluster. Each cluster that Oracle Database accesses using Oracle Big Data SQL has a cluster directory. This directory is located on the Oracle Database system under `$ORACLE_HOME/bigdatasql/clusters/`. For example, cluster `bda1_cl_1` would have a directory `$ORACLE_HOME/bigdatasql/clusters/bda1_cl_1` and under `$ORACLE_HOME/bigdatasql/clusters/bda1_cl_1/config` would be the following files for client configuration files for accessing the cluster:

- `bigdata.hosts` (not editable by customers)
- `core-site.xml`
- `hdfs-site.xml`
- `hive-site.xml`
- `mapred-site.xml` (optional)
- `log4j` property files (such as `hive-log4j.properties`)

`$ORACLE_HOME/bigdatasql/databases/<database name>/bigdata_config/default_cluster` is a soft link to the directory of the default cluster.

A database directory object points to the cluster directory. Users who want to access the data in a cluster must have read access to the directory object.

2.3.4 About Permissions

On the Oracle database server, ensure that the `oracle` user (or whatever user owns the Oracle Database installation directory) has READ/WRITE access to the database directory that points to the log directory.

On the Hadoop side, when you run Database Acknowledge (`# ./jaguar databaseack [config file]`) this operation creates an account for the database owner and grants required permissions.

2.4 Oracle SQL Access to Kafka

Oracle Big Data SQL allows you to access Oracle external tables connected to Kafka brokers and use Oracle SQL to read and load Kafka topic data.

Topics:

- [About Oracle SQL Access to Kafka](#)
- [Get Started with Oracle SQL Access to Kafka](#)
- [Register a Kafka Cluster](#)

- [Create Views to Access CSV Data in a Kafka Topic](#)
- [Create Views to Access JSON Data in a Kafka Topic](#)
- [Query Kafka Data as Continuous Stream](#)
- [Exploring Kafka Data from a Specific Offset](#)
- [Exploring Kafka Data from a Specific Timestamp](#)
- [Load Kafka Data into Tables Stored in Oracle Database](#)
- [Load Kafka Data into Temporary Tables](#)
- [Customize Oracle SQL Access to Kafka Views](#)
- [Reconfigure Existing Kafka Views](#)

2.4.1 About Oracle SQL Access to Kafka

Oracle SQL access to Kafka (OSaK) is a PL/SQL package that enables Oracle SQL to query Apache Kafka topics via database views and underlying external tables. Apache Kafka versions 0.10.2 and higher are supported.

The ORA_KAFKA PL/SQL package has functions and procedures to register a Kafka cluster in a database schema, create views to query Kafka topics (using external tables under the covers), query data from specified offsets or specified timestamps, and more. Kafka data does not have to be persisted in the Oracle Database. OSaK also allows Kafka data to be loaded into tables stored in the Oracle Database.

Kafka topics in JSON and CSV (delimited text) format are supported, along with offset management to ensure Kafka messages are not queried twice, and Kafka messages are not missed. OSaK is the ideal option for querying the most recent Kafka messages (for example, messages from 5 seconds ago till current time), or a sequence of messages starting from a specific offset or timestamp (for example a 1000 messages starting from timestamp t or offset o).

To get started with OSaK, see [Get Started with Oracle SQL Access to Kafka](#).

For complete reference information for the ORA_KAFKA package, refer to the documentation within the package specification which can be found in the sql/orakafkas.sql file in the unzipped OSaK kit location.

2.4.2 Get Started with Oracle SQL Access to Kafka

In order to successfully use Oracle SQL to read and load data from a Kafka topic, you need to install and configure the Oracle SQL Access to Kafka (OSaK) kit, register the Kafka cluster, and create views over the data within the Kafka topic.

Workflow for Accessing Kafka using Oracle SQL

The following table describes each major task:

Task	Description	Where to find more information
Install and configure access	Install and configure the OSaK kit before performing the following tasks.	See "Install and Configure Oracle SQL Access to Kafka" in <i>Oracle Big Data SQL Installation Guide</i> .

Task	Description	Where to find more information
Register the Kafka cluster	Before reading or loading data, register the Kafka cluster.	See Register a Kafka Cluster .
Create views over data in Kafka topics	In addition to registering the Kafka cluster, create views over the Kafka topic data before reading or loading data from the topic. The views are created differently depending on the type of data in the topic.	For CSV data, see Create Views to Access CSV Data in a Kafka Topic For JSON data, see Create Views to Access JSON Data in a Kafka Topic
Read data from a Kafka topic	You can read data as a continuous stream, or read data from a specific offset or timestamp.	To read a continuous data stream, see Query Kafka Data as Continuous Stream . To read data from a specific offset, see Exploring Kafka Data from a Specific Offset . To read data from a specific timestamp, see Exploring Kafka Data from a Specific Timestamp
Load data from a Kafka topic	Data can be loaded into Oracle Database tables or temporary tables.	For loading data into Oracle Database tables, see Load Kafka Data into Tables Stored in Oracle Database . For loading data into temporary tables, see Load Kafka Data into Temporary Tables .
Manage your Kafka views	You can customize the number of views or reconfigure existing views.	To customize the number of views, see Customize Oracle SQL Access to Kafka Views . To modify views after Kafka topic partitions have been added, see Reconfigure Existing Kafka Views .

2.4.3 Register a Kafka Cluster

Before data can be read from a Kafka topic using Oracle SQL access to Kafka (OSaK), the cluster must be registered and assigned a unique name.

A PL/SQL package, or `ORA_KAFKA`, is installed in a user schema as part of the installation and configuration process. This package contains procedures to register and drop a Kafka cluster, create and drop views for reading data from Kafka topics, manage offsets for each partition of a Kafka topic, and load an Oracle table with data from a Kafka topic.

Also during installation and configuration, the cluster configuration, location, and default directories are created on the operating system. The associated database directory objects are also created: `<cluster>_CONF_DIR`, `<user>_KAFKA_LOC_DIR`,

and `<user>_KAFKA_DEF_DIR`. These directories are required input for the `ORA_KAFKA.REGISTER_CLUSTER` procedure which is described next.

The following example registers the cluster 'MA1':

```
SQL> execute ORA_KAFKA.REGISTER_CLUSTER
      ('MA1',                               -- The name of the cluster
      '<bootstrap servers>',              -- The bootstrap servers for the Kafka
      cluster
      '<user>_KAFKA_DEF_DIR',             -- The Oracle default directory for
      External Tables
      '<user>_KAFKA_LOC_DIR',             -- The Oracle location file directory for
      External Tables
      'MA1_CONF_DIR',                     -- The configuration directory for the
      Kafka cluster
      'Testing DBMS_KAFKA');              -- A text description of the cluster
```

2.4.4 Create Views to Access CSV Data in a Kafka Topic

In order to query data from a Kafka topic, you first need to create the Oracle SQL Access to Kafka (OSaK) views that map to the Kafka topic.

When the Kafka record format is CSV, before creating the views, you must create a reference table whose schema is mapped to records in the Kafka topic.

Note:

If the Kafka record format is `JSON_VARCHAR2`, a reference table should be passed as `NULL`. See [Create Views to Access JSON Data in a Kafka Topic](#).

The following example creates reference table `SENSOR_RECORD_SHAPE` whose schema maps to records in the Kafka topic 'sensor':

```
CREATE TABLE sensor_record_shape(
  msg_number          INTEGER PRIMARY KEY,
  msg_timestamp       TIMESTAMP,
  sensor_type_id      INTEGER,
  sensor_unit_id      INTEGER,
  temperature_setting NUMBER(6,3),
  temperature_reading NUMBER(6,3)
);
```

Now that the reference table has been created, we can create the OSaK views. The following example creates one view per partition in the topic 'sensor' where the record format is CSV. Since the 'sensor' topic has one partition, one view is created.

```
DECLARE
  views_created INTEGER;
  application_id VARCHAR2(128);
BEGIN
  ORA_KAFKA.CREATE_VIEWS
    ('MA1',                               -- The name of the cluster
```

```

(specified in ORA_KAFKA.REGISTER_CLUSTER)
    'QUERYAPP',           -- The name given by the user for
a set of views, corresponds to the concept of a Kafka group
    'sensor',            -- The name of the Kafka topic
    'CSV',               -- The format of the topic record
    'SENSOR_RECORD_SHAPE', -- The name of the database
reference table
    views_created,      -- Output: number of views created
    application_id);   -- Output: the application id of
the set of views
                                -- created

that uniquely identifies the view
objects
    dbms_output.put_line('views created = ' || views_created);
    dbms_output.put_line('application id = ' || application_id);
END;
/

```

The above example causes the following view to be created:

```

SQL> describe KV_MA1_QUERYAPP_SENSOR_0;
Name                               Null?    Type
-----
KAFKA$PARTITION                    NUMBER(38)
KAFKA$OFFSET                        NUMBER(38)
KAFKA$EPOCH_TIMESTAMP              NUMBER(38)
MSG_NUMBER                          NOT NULL NUMBER
MSG_TIMESTAMP                       TIMESTAMP(6)
SENSOR_TYPE_ID                     NUMBER
SENSOR_UNIT_ID                     NUMBER
TEMPERATURE_SETTING                NUMBER(6,3)
TEMPERATURE_READING                NUMBER(6,3)

```

Where `KAFKA$PARTITION` is the Kafka partition id, `KAFKA$OFFSET` is the offset of the Kafka record, `KAFKA$EPOCH_TIMESTAMP` is the timestamp of the Kafka record. The remaining columns represent the fields in the CSV data.

2.4.5 Create Views to Access JSON Data in a Kafka Topic

In order to query data from a Kafka topic containing JSON data, Oracle SQL access to Kafka (OSaK) views must be created specifying `JSON_VARCHAR2` as the format of the topic record.

When the Kafka record format is `JSON_VARCHAR2`, a reference table is not used and should be passed as `NULL`.

The following example creates one view for the single partition in the topic 'sensorj' where the record format is JSON:

```

DECLARE
    views_created INTEGER;
    application_id VARCHAR2(128);
BEGIN

```

```

ORA_KAFKA.CREATE_VIEWS
('MA1', -- The name of the cluster
 'QUERYAPP_JSON', -- The name of the Kafka group
 'sensorj', -- The name of the Kafka topic
 'JSON_VARCHAR2', -- The format of the topic record
 NULL, -- No reference table used for JSON
 views_created, -- Output: number of views created
 application_id); -- Output: the application id of the
set of views
-- created that uniquely identifies
the view
-- objects
dbms_output.put_line('views created = ' || views_created);
dbms_output.put_line('application id = ' || application_id);
END;
/

```

The above example causes the following view to be created:

```

SQL> describe KV_MA1_QUERYAPP_JSON_SENSORJ_0;
Name                               Null?    Type
-----
KAFKA$PARTITION                     NUMBER(38)
KAFKA$OFFSET                         NUMBER(38)
KAFKA$EPOCH_TIMESTAMP                NUMBER(38)
KEY                                   VARCHAR2(32767)
VALUE                                 VARCHAR2(32767)

```

Where `KAFKA$PARTITION` is the Kafka partition id, `KAFKA$OFFSET` is the offset of the Kafka record, `KAFKA$EPOCH_TIMESTAMP` is the timestamp of the Kafka record. The `KEY` and `VALUE` columns contain the JSON data.

2.4.6 Query Kafka Data as Continuous Stream

After creating the Oracle SQL access to Kafka (OSaK) views, the views can be queried using standard SQL. The view name consists of a generated `application_id` (which is the concatenation of cluster name, Kafka group name, topic name) concatenated with `view_id`.

The sensor topic described above has just one partition, and therefore one view. The view name would be “KV_MA1_QUERYAPP_SENSOR_0”. Note, view names can be identified by querying the `ORA_KAFKA_PARTITION` metadata table in the schema in which the `ORA_KAFKA` package was installed.

OSaK views can be accessed continuously, reading from an initial offset or timestamp to the end of the stream. This is the typical usage case for querying the most recent Kafka records for the specified topic.

The example below sets the starting offset to 100 records below the Kafka partition high water mark using the `ORA_KAFKA.INIT_OFFSET` procedure, and reads from there to the end of the stream. The next time through the loop, you read from where you left off last time to the new end of the stream. In the example, the analytics are reduced to a simple example doing a `count(*)` with `LOOP` logic in

PL/SQL. Expected usage is: there is a loop within an application which executes a call to `ORA_KAFKA.NEXT_OFFSET`, queries the OSaK views, performs analytics on retrieved Kafka records, and if satisfied, calls `ORA_KAFKA.UPDATE_OFFSET`, and commits the transaction. The `ORA_KAFKA.NEXT_OFFSET` procedure records the next Kafka offset from which to read, based on the results of `ORA_KAFKA.UPDATE_OFFSET` or `ORA_KAFKA.INIT_OFFSET/ORA_KAFKA.INIT_OFFSET_TS`. `ORA_KAFKA.UPDATE_OFFSET` saved the last Kafka offset read when the view was accessed.

It is also possible to set the point from which to start reading to a particular timestamp. `ORA_KAFKA.INIT_OFFSET_TS` initializes the starting offset related to a timestamp for each Kafka partition belonging to the OSaK view. As with `ORA_KAFKA.INIT_OFFSET`, `ORA_KAFKA.INIT_OFFSET_TS` would normally be called at the outset of a new application instance dedicated to processing the view or when recovering after an application instance shutdown or failure.

 **Note:**

Multiple applications reading the same set of OSaK views can result in duplicate Kafka records being processed or Kafka records being skipped, because each application will attempt to manage the offset. When multiple applications read the same topic, create a set of views for each application by using application-specific Kafka group names. Then each application can use their own offset to determine where to read. One application can call `ORA_KAFKA.INIT_OFFSET` with 100 and use one set of views, another application can call `ORA_KAFKA.INIT_OFFSET` with 550 and use another set of views, and so on.

```
BEGIN
  -- Before entering the loop, we initialize the starting offsets for
  the view relative to the current Kafka high water mark for the Kafka
  partition managed by our view.
  -- Without an INIT_OFFSET call, records are read from either the
  beginning of the stream or from the offset last recorded by a COMMIT
  after an UPDATE_OFFSET call.
  ORA_KAFKA.INIT_OFFSET
    ('KV_MA1_QUERYAPP_SENSOR_0',          -- The view for which to
  initialize offsets
    100,                                  -- The number of records
  before the high water mark that designates the starting offset
    ORA_KAFKA.WATER_MARK_HIGH);          -- The above record count
  parameter is 100 records below the high water mark
  LOOP

    -- Set the offset of the next Kafka record to be processed.
    -- Since we have called INIT_OFFSET,
    -- the starting offset will be 100 records below the high water
  mark.
    ORA_KAFKA.NEXT_OFFSET
      ('KV_MA1_QUERYAPP_SENSOR_0');      -- The view for which to set
  offsets

    -- Now query for rows starting at 100 records below the high water
```



```

mark.
    SELECT count(*) from KV_MA1_QUERYAPP_SENSOR_0;

    -- Now that we've done a query, record the last offset processed.
    ORA_KAFKA.UPDATE_OFFSET
      ('KV_MA1_QUERYAPP_SENSOR_0);      -- The view for which to set
offsets

    COMMIT;
  END LOOP;
END;
/

```

2.4.7 Exploring Kafka Data from a Specific Offset

Oracle SQL access to Kafka (OSaK) allows you to read a specified number of records from a specific offset. This type of access is restricted to applications that create one view per single Kafka topic/partition

The following example creates a view for a new application, name SEEKAPP, to query the 'sensor' topic with records in CSV format. The example uses the `SEEK_OFFSET` procedure to specify offset 100393 and then query 1000 records. If there are fewer than 1000 records available, all available records will be queried.

```

-- First create views for the seek application
DECLARE
  views_created INTEGER;
  application_id VARCHAR2(128);
BEGIN
  ORA_KAFKA.CREATE_VIEWS
    ('MA1',                                -- The name of the cluster
     'SEEKAPP',                             -- The name of the Kafka group
     'sensor',                              -- The name of the Kafka topic
     'CSV',                                 -- The format of the topic record
     'SENSOR_RECORD_SHAPE',                -- The name of the database reference
  table
    views_created,                          -- Output: number of views created
    application_id,                         -- Output: the application id of the
  set of views
    -- created that uniquely identifies the
  view
    -- objects
    0);                                     -- The number of views to create. 0,
  the default,
    -- requests the creation of 1 view per
    -- Kafka partition
  dbms_output.put_line('views created = ' || views_created);
  dbms_output.put_line('application id = ' || application_id);
END;
/

-- Next we seek to offset 100393
SQL> execute ORA_KAFKA.SEEK_OFFSET
      ('KV_MA1_SEEKAPP_SENSOR_0',          -- The name of the OSAK view that maps

```

```

to a
    100393,
    1000);
from the

-- single cluster/topic/partition
-- The offset to which to seek
-- The number of Kafka rows starting
-- offset to be retrieved

-- Now query for at most 1000 rows starting at offset 100393
SQL> SELECT max(temperature) from KV_MA1_SEEKAPP_SENSOR_0;

```

2.4.8 Exploring Kafka Data from a Specific Timestamp

In addition to reading records from a specific offset, Oracle SQL Access to Kafka (OSaK) allows you to position an OSaK view to start at a particular timestamp and read messages in a user defined window of time. As compared to `ORA_KAFKA.SEEK_OFFSET` which is restricted to OSaK views that map to only one Kafka topic/partition, `ORA_KAFKA.SEEK_OFFSET_TS` can be used on any OSaK view. The `ORA_KAFKA.SEEK_OFFSET_TS` procedure is used to specify a window of time for an OSaK view.

2.4.9 Load Kafka Data into Tables Stored in Oracle Database

The Oracle SQL access to Kafka (OSaK) `ORA_KAFKA.LOAD_TABLE` procedure loads data from a Kafka topic into a database table. `ORA_KAFKA.LOAD_TABLE` creates a view which is used internally and maps to all partitions of the Kafka topic.

The view is not deleted at the end of the `ORA_KAFKA.LOAD_TABLE` execution. This means that subsequent calls to `ORA_KAFKA.LOAD_TABLE` with the same cluster, group, and topic arguments as passed previously, will start loading where the previous `ORA_KAFKA.LOAD_TABLE` left off, using the same view.

To continuously load Kafka data into the database, the `ORA_KAFKA.LOAD_TABLE` procedure can be called in a loop.

The following example illustrates a single call to the `ORA_KAFKA.LOAD_TABLE` procedure which loads data from the sensor topic into the Oracle database table `sensortab`.

```

DECLARE
    num_records_loaded INTEGER;
BEGIN
    ORA_KAFKA.LOAD_TABLE
        ('MA1',
         'LOADAPP',
         'sensor',
         'CSV',
         'sensortab',
         -- The name of the cluster
         -- The name of the Kafka group
         -- The name of the topic
         -- The format of the Kafka record
         -- The name of the target table in Oracle.
         -- This table must reflect the shape of
the rows
        num_records_loaded);
    dbms_output.put_line('Kafka records loaded = ' || num_records_loaded);
COMMIT;

```

```
END;  
/
```

2.4.10 Load Kafka Data into Temporary Tables

Oracle SQL Access to Kafka (OSaK) views are Kafka applications which are not transactional within Oracle. Each scan of a view will likely yield new results since the view typically scans Kafka records from an offset to a topic's high water mark (latest record available) which is continually advancing.

This becomes problematic if one wants consistency across several SQL queries of the same data set retrieved from an OSaK view. It also becomes problematic if one is executing complicated joins with OSaK views without careful hinting in a SQL query that ensures the OSaK view is the outermost table in a join.

OSaK procedures (`ORA_KAFKA.LOAD_PRIVATE_TEMP_TABLE` and `ORA_KAFKA.LOAD_GLOBAL_TEMP_TABLE`) solve these problems by creating a temporary table from a `SELECT * FROM <view_name>`, where view name is an OSaK view. This materializes the data retrieved by a single query into a temporary table. The `ORA_KAFKA.LOAD_[PRIVATE|GLOBAL]_TEMP_TABLE` procedure is typically called immediately after calling `ORA_KAFKA.NEXT_OFFSET`, or `ORA_KAFKA.SEEK_OFFSET/ORA_KAFKA.SEEK_OFFSET_TS`. Application logic then queries against the contents of a temporary table rather than directly querying the OSaK view.

Two types of temporary tables are useful: private temporary tables and global temporary tables, which are created by calling the `ORA_KAFKA.LOAD_PRIVATE_TEMP_TABLE` or `ORA_KAFKA.LOAD_GLOBAL_TEMP_TABLE` procedure respectively. A key difference between global temporary tables and private temporary tables is that global temporary tables are more functional and support indexes and triggers, while private temporary tables are lighter weight and don't support indexes or triggers.

2.4.11 Customize Oracle SQL Access to Kafka Views

The Oracle SQL access to Kafka (OSaK) `ORA_KAFKA.CREATE_VIEWS` procedure creates one or more views which map to external tables that retrieve data from partitions of a Kafka topic. Each view retrieves data from one or more partitions in a topic.

The ability to create multiple views over multiple partitions allows an application to scale out and divide the workload across application instances that are running concurrently. Only one application instance should read an Oracle SQL access to Kafka (OSaK) view. Multiple readers may result in duplicate Kafka records being processed or Kafka records being skipped.

By default, the `ORA_KAFKA.CREATE_VIEWS` procedure creates one view per topic partition for a particular cluster/group/topic. For example, if a topic has eight partitions, the default is to create eight views. In some cases it may be useful to create one view over all partitions of a topic. In other cases, it may be useful to create multiple views, each one over multiple Kafka partitions. For example, one view for every 4 partitions. The `ORA_KAFKA.CREATE_VIEWS` procedure has optional parameters that allow the number of views to be specified.

By default, a Kafka topic record format that is specified as 'CSV' is considered to have fields delimited by a comma, and records terminated by new line. The

ORA_KAFKA.CREATE_VIEWS procedure has an optional parameter, `view_properties`, that allows the field and record delimiters to be specified.

```
PROCEDURE CREATE_VIEWS (
    cluster_name          IN  VARCHAR2,
    group_name           IN  VARCHAR2,
    topic_name           IN  VARCHAR2,
    topic_record_format  IN  VARCHAR2,
    ref_table            IN  VARCHAR2,
    views_created        OUT INTEGER,
    application_id       OUT VARCHAR2,
    view_count           IN  INTEGER DEFAULT 0,
    force_view_count     IN  BOOLEAN DEFAULT FALSE,
    view_properties      IN  VARCHAR2 DEFAULT NULL
);
```

The `view_count` parameter allows the application to specify the number of views to create. Legal values are 0 to N, where N is the number of Kafka partitions in the topic. The default value is 0 which instructs `CREATE_VIEWS` to create 1 view per Kafka partition in the topic.

The `force_view_count` parameter can be `TRUE` or `FALSE`, the default value is `FALSE`. If `force_view_count` is set to `TRUE`, `ORA_KAFKA.CREATE_VIEWS` creates 'view_count' views even if that number could create unbalanced views, where different views read from different numbers of Kafka topic partitions. For example, setting a view count of 2 when there are 5 Kafka topic partitions causes `ORA_KAFKA.CREATE_VIEWS` to create one view that maps to 2 Kafka topic partitions and one view that maps to 3 Kafka topic partitions.

The `view_properties` parameter is an optional parameter that allows you to specify custom field and/or record delimiters. It is formatted as a JSON string in the JSON syntax described in the JSON developers guide: [5 SQL/JSON Conditions IS JSON and IS NOT JSON](#) in *JSON Developer's Guide*.

The supported keys are:

- `field_delim`: field delimiter as a json value
- `record_delim`: record delimiter as a json value

Examples:

```
{"field_delim": "\u0001", "record_delim": "\r\n"}
{"record_delim": "\r\n"}
{"field_delim": "\u0001"}
```

2.4.12 Reconfigure Existing Kafka Views

It is possible for additional Kafka partitions to be added to a topic that is being processed by Oracle SQL access to Kafka (OSaK) views. OSaK provides a procedure similar to `ORA_KAFKA.CREATE_VIEWS` to add additional Kafka partitions to an existing set of OSaK views.

The `ORA_KAFKA.ADD_PARTITIONS` procedure preserves the state information about existing Kafka topic partitions and binds new partitions to either existing or new views.

The following example calls the `ORA_KAFKA.ADD_PARTITIONS` procedure to add any new Kafka partitions to the views already created for the 'MA1' cluster:

```
DECLARE
  views_created INTEGER;
  application_id VARCHAR2(128);
BEGIN
  ORA_KAFKA.ADD_PARTITIONS
    ('MA1',          -- The name of the cluster (specified in
ORA_KAFKA.REGISTER_CLUSTER)
    'QUERYAPP',     -- The name given by the user for a set of views,
corresponds to the concept of a Kafka group
    'sensor',       -- The name of the Kafka topic
    views_created); -- Output: number of views created. -1 is
returned if there are no additional partitions
                    -- since the views were created or since the last
call to ORA_KAFKA.ADD_PARTITIONS

  dbms_output.put_line('views created = ' || views_created);
END;
/
```

3

Information Lifecycle Management: Hybrid Access to Data in Oracle Database and Hadoop

Oracle Big Data SQL supports off-loading Oracle Database tables and partitions to the HDFS file system on a Hadoop cluster.

This data can be accessed using tables, external tables or Hybrid Partitioned Tables. This allows you to store the most frequently accessed data in Oracle Database storage for fastest access, while storing read-only archive data in Hadoop.

Topics:

- [About Storing to Hadoop and Hybrid Partitioned Tables](#)
- [Use Copy to Hadoop](#)
- [Enable Access to Hybrid Partitioned Tables](#)
- [Store Oracle Tablespaces in HDFS](#)

3.1 About Storing to Hadoop and Hybrid Partitioned Tables

Oracle Big Data SQL resources support different methods for off-loading Oracle Database tables. These methods include storing to hybrid partitioned tables (HPT), and storing to Oracle Database tablespaces in HDFS.

You can use the Oracle Copy to Hadoop utility to efficiently move data from Oracle Database to Hadoop. That data will be stored in datapump format and can then be queried by both Big Data SQL and native Hadoop tools on the cluster. See [Use Copy to Hadoop](#).

The table below compares the off-loading methods of storing data to hybrid partitioned tables with storing data to Oracle Database tablespaces in HDFS.

Table 3-1 Comparison between off-loading methods

Store to Hybrid Partitioned Tables	Oracle Tablespaces in HDFS
Classical internal partitioned tables are combined with Oracle external partitioned tables to form a hybrid partitioned table. Partitioning breaks up tables into smaller more manageable pieces. This increases performance and allows off-loading data to less expensive storage.	Oracle Database tables or partitions are stored within the tablespace in HDFS in their original Oracle-internal format.

Table 3-1 (Cont.) Comparison between off-loading methods

Store to Hybrid Partitioned Tables	Oracle Tablespaces in HDFS
<p>Access to external partitions support all existing table types and the following access drivers: ORACLE_HDFS, ORACLE_HIVE, ORACLE_BIGDATA.</p> <p>Not only can HPT point to Copy to Hadoop data pump files, it can also point to data that is in more traditional Hadoop file types such as Parquet, ORC, and CSV.</p>	<p>Access is directly through the original Oracle Database tables. External tables are not needed.</p>
<p>Data can be accessed by non-Oracle Database tools. Since the data can be in CSV, Parquet, Avro, and ORC formats any big data technology is able to access seamlessly. With HPT, you store data in the best format for your application.</p> <p>See Enable Access to Hybrid Partitioned Tables.</p>	<p>Data is directly available to Oracle Database only. Data is not accessible to other processes in Hadoop.</p> <p>See Store Oracle Tablespaces in HDFS.</p>

3.2 Use Copy to Hadoop

Learn how to use Copy to Hadoop to copy Oracle Database tables to Hadoop.

- [What Is Copy to Hadoop?](#)
- [Getting Started Using Copy to Hadoop](#)
- [Using Oracle Shell for Hadoop Loaders With Copy to Hadoop](#)
- [Copy to Hadoop by Example](#)
- [Querying the Data in Hive](#)
- [Column Mappings and Data Type Conversions in Copy to Hadoop](#)
- [Working With Spark](#)
- [Using Oracle SQL Developer with Copy to Hadoop](#)

3.2.1 What Is Copy to Hadoop?

Oracle Big Data SQL includes the Oracle Copy to Hadoop utility. This utility makes it simple to identify and copy Oracle data to the Hadoop Distributed File System. It can be accessed through the command-line interface Oracle Shell for Hadoop Loaders.

Data exported to the Hadoop cluster by Copy to Hadoop is stored in Oracle Data Pump format. The Oracle Data Pump files can be queried by Hive or Big Data SQL. The Oracle Data Pump format optimizes queries through Big Data SQL in the following ways:

- The data is stored as Oracle data types – eliminating data type conversions.
- The data is queried directly – without requiring the overhead associated with Java SerDes.

After Data Pump format files are in HDFS, you can use Apache Hive to query the data. Hive can process the data locally without accessing Oracle Database. When

the Oracle table changes, you can refresh the copy in Hadoop. Copy to Hadoop is primarily useful for Oracle tables that are relatively static, and thus do not require frequent refreshes.

Copy to Hadoop is licensed under Oracle Big Data SQL. You must have an Oracle Big Data SQL license in order to use this utility.

3.2.2 Getting Started Using Copy to Hadoop

To install and start using Copy to Hadoop:

1. Follow the Copy to Hadoop and Oracle Shell for Hadoop Loaders installation procedures in the *Oracle Big Data SQL Installation Guide*.

As described in the installation guide, ensure that the prerequisite software is installed on both the Hadoop cluster (on Oracle Big Data Appliance or another Hadoop system) and on the Oracle Database server (Oracle Exadata Database Machine or other).

2. Invoke Oracle Shell for Hadoop Loaders (OHS) to do a direct, one-step copy or a staged, two-step copy of data in Oracle Database to Data Pump format files in HDFS, and create a Hive external table from the files.

OHS will choose `directcopy` by default to do a direct, one-step copy. This is faster than a staged, two-step copy and does not require storage on the database server. However, there are situations where you should do a staged, two-step copy:

- Copying columns from multiple Oracle Database source tables. (The direct, one-step copy copies data from one table only.)
- Copying columns of type `TIMESTAMP_TZ` or `TIMESTAMP_LTZ` to Hive.
Since Hive does not have a data type that supports time zones or time offsets, you must cast these columns to `TIMESTAMP` when manually exporting these columns to Data Pump files
- Copying data from a view. Views are not supported by the `directcopy` option.

The staged two-step copy using the manual steps is demonstrated in "Appendix A: [Manual Steps for Using Copy to Hadoop for Staged Copies](#)".

3. Query this Hive table the same as you would any other Hive table.

Tip:

For Hadoop power users with specialized requirements, the manual option for Direct Copy is recommended. See [Manual Steps for Using Copy to Hadoop for Direct Copies](#) in Appendix B.

3.2.2.1 Table Access Requirements for Copy to Hadoop

To copy a table using Copy to Hadoop, an Oracle Database user must meet one of these requirements.

- The user is the owner of the table, or
- The user is accessing a table in another schema and has the following privileges:

- The `SELECT` privilege on the table.
- The `select_catalog_role` privilege (which provides `SELECT` privileges on data dictionary views).

3.2.3 Using Oracle Shell for Hadoop Loaders With Copy to Hadoop

3.2.3.1 Introducing Oracle Shell for Hadoop Loaders

What is Oracle Shell for Hadoop Loaders?

Oracle Shell for Hadoop Loaders (OHS) is a helper shell that provides an easy-to-use command line interface to Oracle Loader for Apache Hadoop, Oracle SQL Connector for HDFS, and Copy to Hadoop. It has basic shell features such as command line recall, history, inheriting environment variables from the parent process, setting new or existing environment variables, and performing environmental substitution in the command line.

The core functionality of Oracle Shell for Hadoop Loaders includes the following:

- Defining named external resources with which Oracle Shell for Hadoop Loaders interacts to perform loading tasks.
- Setting default values for load operations.
- Running load commands.
- Delegating simple pre and post load tasks to the Operating System, HDFS, Hive and Oracle. These tasks include viewing the data to be loaded, and viewing the data in the target table after loading.

See Also:

The examples directory in the OHS kit contains many examples that define resources and load data using Oracle Shell for Hadoop Loaders. Unzip `<OHS_KIT>/examples.zip` and see `<OHS_KIT>/examples/README.txt` for a description of the examples and instructions on how to run OHS load methods.

3.2.4 Copy to Hadoop by Example

3.2.4.1 First Look: Loading an Oracle Table Into Hive and Storing the Data in Hadoop

This set of examples shows how to use Copy to Hadoop to load data from an Oracle table, store the data in Hadoop, and perform related operations within the

OHSB shell. It assumes that OHSB and Copy to Hadoop are already installed and configured.

What's Demonstrated in The Examples

These examples demonstrate the following tasks:

- Starting an OHSB session and creating the resources you'll need for Copy to Hadoop.
- Using Copy to Hadoop to copy the data from the selected Oracle Database table to a new Hive table in Hadoop (using the resources that you created).
- Using the `load` operation to add more data to the Hive table created in the first example.
- Using the `create` or `replace` operation to drop the Hive table and replace it with a new one that has a different record set.
- Querying the data in the Hive table and in the Oracle Database table.
- Converting the data into other formats

Tip:

You may want to create select or create a small table in Oracle Database and work through these steps.

Starting OHSB, Creating Resources, and Running Copy to Hadoop

1. Start OHSB. (The startup command below assumes that you've added the OHSB path to your PATH variable as recommended.)

```
$ ohsh  
ohsh>
```

2. Create the following resources.

- SQL*Plus resource.

```
ohsh> create sqlplus resource sql0  
connectid="<database_connection_url>"
```

- JDBC resource.

```
ohsh> create jdbc resource jdbc0  
connectid="<database_connection_url>"
```

 **Note:**

For the Hive access shown in this example, only the default `hive0` resource is needed. This resource is already configured to connect to the default Hive database. If additional Hive resources were required, you would create them as follows:

```
ohsh> create hive resource hive_mydatabase
connectionurl="jdbc:hive2:///<Hive_database_name>"
```

3. Include the Oracle Database table name in the `create hive table` command below and run the command below. This command uses the Copy to Hadoop `directcopy` method. Note that `directcopy` is the default mode and you do not actually need to name it explicitly.

```
ohsh> create hive table hive0:<new_Hive_table_name> from oracle
table jdbc0:<Oracle_Database_table_name> from oracle table
jdbc0:<Oracle_Database_table_name> using directcopy
```

The Oracle Table data is now stored in Hadoop as a Hive table.

Adding More Data to the Hive Table

Use the OSHS `load` method to add data to an existing Hive table.

Let's assume that the original Oracle table includes a time field in the format DD-MM-YY and that a number of daily records were added after the Copy to Hadoop operation that created the corresponding Hive table.

Use `load` to add these new records to the existing Hive table:

```
ohsh> load hive table hive0:<Hive_table_name> from oracle table
jdbc0:<Oracle_Database_table_name> where "(time >= '01-FEB-18')"
```

Using OSHS *create or replace*

The OSHS `create or replace` operation does the following:

1. Drops the named Hive table (and the associated Data Pump files) if a table by this name already exists.

 **Note:**

Unlike `create or replace`, a `create` operation fails and returns an error if the Hive table and the related Data Pump files already exist.

2. Creates a new Hive table using the name provided.

Suppose some records were deleted from the original Oracle Database table and you want to realign the Hive table with the new state of the Oracle Database table. Hive

does not support update or delete operations on records, but the `create` or `replace` operation in OSH can achieve the same end result:

```
ohsh> create or replace hive table hive0:<new_hive_table_name> from
oracle table jdbc0:<Oracle_Database_table_name>
```

 **Note:**

Data copied to Hadoop by Copy to Hadoop can be queried through Hive, but the data itself is actually stored as Oracle Data Pump files. Hive only points to the Data Pump files.

Querying the Hive Table

You can invoke a Hive resource in OSH in order to run HiveQL commands. Likewise, you can invoke an SQL*Plus resource to run SQL commands. For example, these two queries compare the original Oracle Database table with the derivative Hive table:

```
ohsh> %sql0 select count(*) from <Oracle_Database_table_name>
ohsh> %hive0 select count(*) from <Hive_table_name>
```

Storing Data in Other Formats, Such as Parquet or ORC

By default, Copy to Hadoop outputs Data Pump files. In a `create` operation, you can use the “`stored as`” syntax to change the destination format to Parquet or ORC:

```
ohsh> %hive0 create table <Hive_table_name_parquet> stored as parquet
as select * from <Hive_table_name>
```

This example creates the Data Pump files, but then immediately copies them to Parquet format. (The original Data Pump files are not deleted.)

3.2.4.2 Working With the Examples in the Copy to Hadoop Product Kit

The OSH product kit provides an `examples` directory at the path where OSH is installed. This section walks you through several examples from the kit.

3.2.4.2.1 Using Copy to Hadoop With the Default Copy Method

The following examples from the Copy to Hadoop product kit show how to use Copy to Hadoop with the default method of loading data. You can find the code in the `examples` directory where the kit is installed (`<OSH_KIT>/examples`).

The following examples assume that OSH and Copy to Hadoop are installed and configured.

Example 3-1 `createreplace_directcopy.ohsh`

This script uses the `create` or `replace` operation to create a Hive external table called `cp2hadoop_fivdti` from the Oracle table `OSHC_CP2HADOOP_FIVDTI`. It then loads

the Hive table with 10000 rows. It uses the default load method `directcopy` to run a map job on Hadoop and split the Oracle table into input splits. The resulting Hive external table includes all of the splits.

```
create or replace hive table hive0:cp2hadoop_fivdti \  
from oracle table jdbc0:ohsh_cp2hadoop_fivdti using directcopy
```

In the example below and in the code samples that follow, `olhp` is a user-defined JDBC resource.

Example 3-2 `load_directcopy.ohsh`

The `load_directcopy.ohsh` script shows how to load the Hive table that was created in `createreplace_directcopy.ohsh` with an additional 30 rows. This script also uses the `directcopy` method.

```
load hive table hive0:cp2hadoop_fivdti from oracle table  
jdbc0:ohsh_cp2hadoop_fivdti \  
using directcopy where "(i7 < 30)";
```

Tip:

You have the option to convert the storage in Hadoop from the default Data Pump format to Parquet or ORC format. For example:

```
%hive0 create table cp2hadoop_fivdti_parquet stored as parquet  
as select * from cp2hadoop_fivdti
```

The original Data Pump files are not deleted.

3.2.4.2.2 Using Copy to Hadoop With the Staged Copy Method

The first example below shows how to use Oracle Shell for Hadoop Loaders (OHS) with Copy to Hadoop to do a staged, two-step copy from Oracle Database to Hadoop. The `stage` method is an alternative to the `directcopy` method.

The second example shows how to load additional rows into the same table. It also uses the `stage` method.

Both examples assume that OHS and Copy to Hadoop have been installed and configured, and that the examples have been configured according to the instructions in `README.txt` in the `examples` directory of the OHS installation. The scripts below and many others are available in the `examples` directory.

Example 3-3 `createreplace_stage.ohsh`

This script uses `create` or `replace` to create a Hive table called `cp2hadoop_fivdti` from the Oracle table `OHS_CP2HADOOP_FIVDTI`. It uses the `stage` command, which automatically does the following:

1. Exports the contents of the source table in Oracle to Data Pump format files on local disk

2. Moves the Data Pump format files to HDFS.
3. Creates the Hive external table that maps to the Data Pump format files in HDFS.

```
create or replace hive table hive0:cp2hadoop_fivdti \  
from oracle table jdbc0:ohsh_cp2hadoop_fivdti using stage
```

In the command above (and also in the next code example), `olhp` is a user-defined JDBC resource.

Example 3-4 load_stage.ohsh

The `load_stage.ohsh` script shows how to load the Hive table created by `createreplace_stage.ohsh` with an additional 30 rows using the `stage` method.

```
load hive table hive0:cp2hadoop_fivdti from oracle table  
jdbc0:ohsh_cp2hadoop_fivdti \  
using stage where "(i7 < 30)";
```

Manual Option

The two-step method demonstrated in the `createreplace_stage.ohsh` and `load_stage.ohsh` example scripts automates some of the tasks required to do staged copies. However, there may be reasons to perform the steps manually, such as:

- You want to load columns from multiple Oracle Database source tables.
- You want to load columns of type `TIMESTAMPZ` or `TIMESTAMPLTZ`.

See [Appendix A: Manual Steps for Using Copy to Hadoop for Staged Copies](#).

3.2.5 Querying the Data in Hive

The following `OHSB` command shows the number of rows in the Hive table after copying from the Oracle table.

```
%hive0 select count(*) from cp2hadoop_fivdti;
```

3.2.6 Column Mappings and Data Type Conversions in Copy to Hadoop

Get help with column mappings and data type conversions in Copy to Hadoop.

3.2.6.1 About Column Mappings

The Hive table columns automatically have the same names as the Oracle columns, which are provided by the metadata stored in the Data Pump files. Any user-specified column definitions in the Hive table are ignored.

3.2.6.2 About Data Type Conversions

Copy to Hadoop automatically converts the data in an Oracle table to an appropriate Hive data type. [Table 3-2](#) shows the default mappings between Oracle and Hive data types.

Table 3-2 Oracle to Hive Data Type Conversions

Oracle Data Type	Hive Data Type
NUMBER	INT when the scale is 0 and the precision is less than 10 BIGINT when the scale is 0 and the precision is less than 19 DECIMAL when the scale is greater than 0 or the precision is greater than 19
CLOB NCLOB	STRING
INTERVALYM INTERVALDS	STRING
BINARY_DOUBLE	DOUBLE
BINARY_FLOAT	FLOAT
BLOB	BINARY
ROWID UROWID	BINARY
RAW	BINARY
CHAR NCHAR	CHAR
VARCHAR2 NVARCHAR2	VARCHAR
DATE	TIMESTAMP
TIMESTAMP	TIMESTAMP
TIMESTAMPtz	Unsupported
TIMESTAMPtz ¹	Unsupported

¹ To copy `TIMESTAMPtz` and `TIMESTAMPtz` data to Hive, follow the instructions in [Appendix A: Manual Steps for Using Copy to Hadoop to do Staged Copies](#). Cast the columns to `TIMESTAMP` when exporting them to the Data Pump files.

3.2.7 Working With Spark

The Oracle Data Pump files exported by Copy to Hadoop can be used in Spark.

The Spark installation must be configured to work with Hive. Launch a Spark shell by specifying the Copy to Hadoop jars.

```
prompt> spark-shell --jars
orahivedp.jar,ojdbc7.jar,oraloader.jar,ora18n.jar,ora-hadoop-common.jar
```

Verify the type of `sqlContext` in spark-shell:

```
scala> sqlContext
```

Your output will look like the following:

```
res0:org.apache.spark.sql.SQLContext =  
org.apache.spark.sql.hive.HiveContext@66ad7167
```

If the default sqlContext is not HiveContext, create it:

```
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

You can now create a Data Frame df that points to a Hive external table over Oracle Data Pump files:

```
scala> val df = sqlContext.table("<hive external table>") <hive  
external table>:  
org.apache.spark.sql.DataFrame = [ <column names> ]
```

Now you can access data via the data frame.

```
scala> df.count  
scala> df.head
```

If a Hive external table had not been created and you only had the Oracle Data Pump files created by Copy to Hadoop, you can create the Hive external table from within Spark.

```
scala> sqlContext.sql("CREATE EXTERNAL TABLE <hive external table> ROW  
FORMAT SERDE  
'oracle.hadoop.hive.datapump.DPSerDe' STORED AS INPUTFORMAT  
'oracle.hadoop.hive.datapump.DPInputFormat' OUTPUTFORMAT  
'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat' LOCATION  
'/user/oracle/oracle_warehouse/<hive database name>')
```

3.2.8 Using Oracle SQL Developer with Copy to Hadoop

Oracle SQL Developer is a free, GUI-based development environment that provides easy to use tools for working Oracle Big Data Connectors, including Copy to Hadoop.

Using Oracle SQL Developer, you can copy data and create a new Hive table, or append data to an existing Hive external table that was created by Copy to Hadoop. In the GUI, you can initiate Copy to Hadoop in Oracle SQL Developer by right-clicking the Tables icon under any Hive schema. You can then append to an existing Hive external table by right-clicking the icon for that Hive table.

See [Installing Oracle SQL Developer](#) in this manual for instructions on where to obtain Oracle SQL Developer and how do the basic installation.

3.3 Enable Access to Hybrid Partitioned Tables

Using hybrid partitioned tables (HPT) enables efficient and economical information lifecycle management. Data copied to HDFS and Oracle Object Storage can be

referenced from HPT. Archive data is saved to HDFS - providing scalable, inexpensive storage.

For example, you may have a table partitioned by month. "Hot" data that is queried frequently and subject to updates would be stored in partitions that are managed by Oracle Database. The older historical data can then be off-loaded to Hadoop.

Query performance benefits from data that is partitioned. When querying data for a particular month, you only need to query a single partition - not the entire table containing data for every month. Queries against recent "hot" data will not even attempt to access the archive data in Hadoop; the archived partitions are pruned at query time.

The data in Hadoop can be stored in various formats - including parquet, CSV, etc. - allowing you to easily use that data by Hadoop tools and applications.

For a complete description of how to use HPT, see Hybrid Partitioned Tables in *Oracle Database VLDB and Partitioning Guide*.

3.4 Store Oracle Tablespaces in HDFS

You can store Oracle read-only tablespaces on HDFS and use Big Data SQL Smart Scan to off-load query processing of data stored in that tablespace to the Hadoop cluster. Big Data SQL Smart Scan performs data local processing - filtering query results on the Hadoop cluster prior to the return of the data to Oracle Database. In most circumstances, this can be a significant performance optimization. In addition to Smart Scan, querying tablespaces in HDFS also leverages native Oracle Database access structures and performance features. This includes features such as indexes, Hybrid Columnar Compression, Partition Pruning, and Oracle Database In-Memory.

Tables, partitions, and data in tablespaces in HDFS retain their original Oracle Database internal format. This is not a data dump. Unlike other means of accessing data in Hadoop (or other noSQL systems), you do not need to create Oracle External table. After copying the corresponding Oracle tablespaces to HDFS, you refer to the original Oracle table to access the data.

Permanent online, read only, and offline tablespaces (including ASM tablespaces) are eligible for the move to HDFS.

Note:

Since tablespaces allocated to HDFS are may not be altered, offline tablespaces must remain as offline. For offline tablespaces, then, what this feature provides is a hard backup into HDFS.

If you want to use Oracle SQL Developer to perform the operations in this section, confirm that you can access the Oracle Database server from your on-premises location. This typically requires a VPN connection.

3.4.1 Advantages and Limitations of Tablespaces in HDFS

The following are some reasons to store Oracle Database tablespaces in HDFS.

- Because the data remains in Oracle Database internal format, I/O requires no resource-intensive datatype conversions.
- All Oracle Database performance optimizations such as indexing, Hybrid Columnar Compression, Partition Pruning, and Oracle Database In-Memory can be applied.
- Oracle user-based security is maintained. Other Oracle Database security features such as Oracle Data Redaction and ASO transparent encryption remain in force if enabled. In HDFS, tablespaces can be stored in zones under HDFS Transparent HDFS encryption.
- Query processing can be off-loaded. Oracle Big Data SQL Smart Scan is applied to Oracle Database tablespaces in HDFS. Typically, Smart Scan can provide a significant performance boost for queries. With Smart Scan, much of the query processing workload is off-loaded to the Oracle Big Data SQL server cells on the Hadoop cluster where the tablespaces reside. Smart Scan then performs predicate filtering in-place on the Hadoop nodes to eliminate irrelevant data so that only data that meets the query conditions is returned to the database tier for processing. Data movement and network traffic are reduced to the degree that smart scan predicate filtering can distill the dataset before returning it to the database.
- For each table in the tablespace, there is only a single object to manage – the Oracle-internal table itself. To be accessible to Oracle Database, data stored in other file formats typically used in HDFS requires an overlay of an external table and a view.
- As is always the case with Oracle internal partitioning, partitioned tables and indexes can have partitions in different tablespaces some of which may be in Exadata, ZFSSA, and other storage devices. This feature adds HDFS as another storage option.

There are some constraints on using Oracle tablespaces in HDFS. As is the case with all data stored in HDFS, Oracle Database tables, partitions, and data stored in HDFS are immutable. Updates are done by deleting and replacing the data. This form of storage is best suited to off-loading tables and partitions for archival purposes. Also, with the exception of OD4H, data in Oracle tablespaces in HDFS is not accessible to other tools in the Hadoop environment, such as Spark, Oracle Big Data Discovery, and Oracle Big Data Spatial and Graph.

3.4.2 About Tablespaces in HDFS and Data Encryption

Oracle Database Tablespaces in HDFS can work with ASO (Oracle Advanced Security) transparent table encryption as well as HDFS Transparent Encryption in HDFS.

Tablespaces With Oracle Database ASO Encryption

In Oracle Database, ASO transparent encryption may be enabled for a tablespace or objects within the tablespace. This encryption is retained if the tablespace is subsequently moved to HDFS. For queries against this data, the `CELL_OFFLOAD_DECRYPTION` setting determines whether Oracle Big Data SQL or Oracle Database decrypts the data.

- If `CELL_OFFLOAD_DECRYPTION = TRUE`, then the encryption keys are sent to the Oracle Big Data server cells in Hadoop and data is decrypted at the cells.

- If `CELL_OFFLOAD_DECRYPTION = FALSE`, encryption keys are not sent to the cells and therefore the cells cannot perform TDE decryption. The data is returned to Oracle Database for decryption.

The default value is `TRUE`.

 **Note:**

In cases where `CELL_OFFLOAD_DECRYPTION` is set to `FALSE`, Smart Scan cannot read the encrypted data and is unable to provide the performance boost that results from the Hadoop-side filtering of the query result set. TDE Column Encryption prevents Smart Scan processing of the encrypted columns only. TDE Tablespace Encryption prevents Smart Scan processing of the entire tablespace.

Tablespaces in HDFS Transparent Encryption Zones

You can move Oracle Database tablespaces into zones under HDFS Transparent Encryption with no impact on query access or on the ability of Smart Scan to filter data.

3.4.3 Moving Tablespaces to HDFS

Oracle Big Data SQL provides two options for moving tablespaces from Oracle Database to the HDFS file system in Hadoop.

- [Using `bds-copy-tbs-to-hdfs`](#)

The script `bds-copy-tbs-to-hdfs.sh` lets you select a preexisting tablespace in Oracle Database. The script automates the move of the selected tablespace to HDFS and performs necessary SQL ALTER operations and datafile permission changes for you. The DataNode where the tablespace is relocated is predetermined by the script. The script uses FUSE-DFS to move the datafiles from Oracle Database to the HDFS file system in the Hadoop cluster.

You can find `bds-copy-tbs-to-hdfs.sh` in the cluster installation directory – `$ORACLE_HOME/BDSJaguar-3.2.0/<string identifier for the cluster>`.

- [Manually Moving Tablespaces to HDFS](#)

As an alternative to `bds-copy-tbs-to-hdfs.sh`, you can manually perform the steps to move the tablespaces to HDFS. You can either move an existing tablespace, or, create a new tablespace and selectively add tables and partitions that you want to off-load. In this case, you can set up either FUSE-DFS or an HDFS NFS gateway service to move the datafiles to HDFS.

The scripted method is more convenient. The manual method is somewhat more flexible. Both are supported.

 **Before You Start:**

As cited in the *Prerequisites* section of the installation guide, both methods require that the following RPMs are pre-installed:

- fuse
- fuse-libs

```
# yum -y install fuse fuse-libs
```

These RPMs are available in the Oracle public yum repository.

3.4.3.1 Using bds-copy-tbs-to-hdfs

On the Oracle Database server, you can use the script `bds-copy-tbs-to-hdfs.sh` to select and move Oracle tablespaces to HDFS. This script is in the `bds-database-install` directory that you extracted from the database installation bundle when you installed Oracle Big Data SQL.

Syntax

`bds-copy-tbs-to-hdfs.sh` syntax is as follows:

```
bds-copy-tbs-to-hdfs.sh
bds-copy-tbs-to-hdfs.sh --install
bds-copy-tbs-to-hdfs.sh --uninstall
bds-copy-tbs-to-hdfs.sh --force-uninstall-script
bds-copy-tbs-to-hdfs.sh --tablespace=<tablespace name> [-pdb=<pluggable
database name>]
bds-copy-tbs-to-hdfs.sh --list=<tablespace name> [--pdb=<pluggable
database name>]
bds-copy-tbs-to-hdfs.sh --show=<tablespace name> [--pdb=<pluggable
database name>]
```

Additional command line parameters are described in the table below.

Table 3-3 bds-copy-tbs-to-hdfs.sh Parameter Options

Parameter List	Description
No parameters	Returns the FUSE-DFS status.
<code>--install</code>	Installs the FUSE-DFS service. No action is taken if the service is already installed.
<code>--uninstall</code>	Uninstalls the FUSE-DFS service and removes the mountpoint.
<code>--grid-home</code>	Specifies the Oracle Grid home directory.
<code>--base-mountpoint</code>	By default, the mountpoint is under <code>/mnt</code> . However, on some systems access to this directory is restricted. This parameter lets you specify an alternate location.

Table 3-3 (Cont.) bds-copy-tbs-to-hdfs.sh Parameter Options

Parameter List	Description
<code>--aux-run-mode</code>	<p>Because Oracle Big Data SQL is installed on the database side as a regular user (not a superuser), tasks that must be done as root and/or the Grid user require the installer to spawn shells to run other scripts under those accounts while <code>bds-copy-tbs-to-hdfs.sh</code> is paused. The <code>--aux-run-mode</code> parameter specifies a mode for running these auxiliary scripts.</p> <p><code>--aux-run-mode=<mode></code></p> <p>Mode options are:</p> <ul style="list-style-type: none"> <code>session</code> — through a spawned session. <code>su</code> — as a substitute user. <code>sudo</code> — through <code>sudo</code>. <code>ssh</code> — through secure shell.
<code>--force-uninstall-script</code>	<p>This option creates a secondary script that runs as root and forces the FUSE-DFS uninstall.</p>

▲ Caution:

Limit use of this option to system recovery, an attempt to end a system hang, or other situations that may require removal of the FUSE-DFS service. Forcing the uninstall could potentially leave the database in an unstable state. The customer assumes responsibility for this choice. Warning message are displayed to remind you of the risk if you use this option.

Table 3-3 (Cont.) bds-copy-tbs-to-hdfs.sh Parameter Options

Parameter List	Description
<code>--tablespace=<tablespace name> [--pdb=<pluggable database name>]</code>	Moves the named tablespace in the named PDB to storage in HDFS on the Hadoop cluster. If there are no PDBs, then the <code>--pdb</code> argument is discarded.
<code>--list=<tablespace name> [--pdb=<pluggable database name>]</code>	Lists tablespaces whose name equals or includes the name provided. The <code>--pdb</code> parameter is an optional scope. <code>--list=*</code> returns all tablespaces. <code>--pdb=*</code> returns matches for the tablespace name within all PDBs.
<code>--show=<tablespace name> [--pdb=<pluggable database name>]</code>	Shows tablespaces whose name equals or includes the name provided and are already moved to HDFS. The <code>--pdb</code> parameter is an optional scope. <code>--show=*</code> returns all tablespaces. <code>--pdb=*</code> returns matches for the tablespace name within all PDBs.

Usage

Use `bds-copy-tbs-to-hdfs.sh` to move a tablespace to HDFS as follows.

1. Log on as the `oracle` Linux user and `cd` to the `bds-database-install` directory where the database bundle was extracted. Find `bds-copy-tbs-to-hdfs.sh` in this directory.

2. Check that FUSE-DFS is installed.

```
$ ./bds-copy-tbs-to-hdfs.sh
```

3. Install the FUSE-DFS service (if it was not found in the previous check). This command will also start the FUSE-DFS the service.

```
$ ./bds-copy-tbs-to-hdfs.sh --install
```

If this script does not find the mount point, it launches a secondary script. Run this script as `root` when prompted. It will set up the HDFS mount. You can run the secondary script in a separate session and then return to this session if you prefer.



For RAC Databases: Install FUSE_DFS on All Nodes:

On a RAC database, the script will prompt you that you must install FUSE-DFS on the other nodes of the database.

4. List the eligible tablespaces in a selected PDB or all PDBs. You can skip this step if you already know the tablespace name and location.

```
$ ./bds-copy-tbs-to-hdfs.sh --list=mytablesapce --pdb=pdb1
```

5. Select a tablespace from the list and then, as `oracle`, run `bds-copy-tbs-to-hdfs.sh` again, but this time pass in the `--tablespace` parameter (and the `--pdb` parameter if specified). The script moves the tablespace to the HDFS file system.

```
$ ./bds-copy-tbs-to-hdfs.sh --tablespace=mytablespace --pdb=pdb1
```

This command automatically makes the tablespace eligible for Smart Scan in HDFS. It does this in SQL by adding the “hdfs:” prefix to the datafile name in the tablespace definition. The rename changes the pointer in the database control file. It does not change the physical file name.

 **Tip:**

If the datafiles are stored in ASM, the extraction will be made using RMAN. At this time, RMAN does not support a direct copy from ASM into HDFS. This will result in an error.

As a workaround, you can use the `--staging-dir` parameter, which enables you to do a two-stage copy – first to a file system directory and then into HDFS. The file system directory specified by `--staging-dir` must have sufficient space for the ASM datafile.

```
$ ./bds-copy-tbs-to-hdfs.sh --tablespace=mytablespace --  
pdb=pdb1 --staging-dir=/home/user
```

For non-ASM datafiles, `--staging-dir` is ignored.

The tablespace should be back online and ready for access when you have completed this procedure.

3.4.3.2 Manually Moving Tablespaces to HDFS

As an alternative to `bds-copy-tbs-to-hdfs.sh`, you can use the following manual steps to move Oracle tablespaces to HDFS.

 **Note:**

In the case of an ASM tablespace, you must first use RMAN or ASMCMD to copy the tablespace to the filesystem.

Oracle Big Data SQL includes FUSE-DFS and these instructions use it to connect to the HDFS file system. You could use an HDFS NFS gateway service instead. The documentation for your Hadoop distribution should provide the instructions for that method.

Perform all of these steps on the Oracle Database server. Run all Linux shell commands as `root`. For SQL commands, log on to the Oracle Database as the `oracle` user.

1. If FUSE-DFS is not installed or is not started, run `bds-copy-tbs-to-hdfs.sh --install`. This script will install FUSE-DFS (if it's not already installed) and then start it.

The script will automatically create the mount point `/mnt/fuse-<clustername>-hdfs`.

 **Note:**

The script `bds-copy-tbs-to-hdfs.sh` is compatible with FUSE-DFS 2.8 only.

2. In SQL, use `CREATE TABLESPACE` to create the tablespace. Store it in a local `.dbf` file. After this file is populated, you will move it to the Hadoop cluster. A single, bigfile tablespace is recommended.

For example:

```
SQL> CREATE TABLESPACE movie_cold_hdfs DATAFILE '/u01/app/oracle/oradata/cdb/orcl/movie_cold_hdfs1.dbf' SIZE 100M reuse AUTOEXTEND ON nologging;
```

3. Use `ALTER TABLE` with the `MOVE` clause to move objects in the tablespace.

For example:

```
SQL> ALTER TABLE movie_fact MOVE PARTITION 2010_JAN TABLESPACE movie_cold_hdfs ONLINE UPDATE INDEXES;
```

You should check the current status of the objects to confirm the change. In this case, check which tablespace the partition belongs to.

```
SQL> SELECT table_name, partition_name, tablespace_name FROM user_tab_partitions WHERE table_name='MOVIE_FACT';
```

4. Make the tablespace read only and take it offline.

```
SQL> ALTER TABLESPACE movie_cold_hdfs READ ONLY;  
SQL> ALTER TABLESPACE movie_cold_hdfs OFFLINE;
```

5. Copy the datafile to HDFS and then change the file permissions to read only.

```
hadoop fs -put /u01/app/oracle/oradata/cdb/orcl/  
movie_cold_hdfs1.dbf /user/oracle/tablespaces/  
hadoop fs -chmod 440 /user/oracle/tablespaces/movie_cold_hdfs1.dbf
```

As a general security practice for Oracle Big Data SQL, apply appropriate HDFS file permissions to prevent unauthorized read/write access.

You may need to source `$ORACLE_HOME/bigdatasql/hadoop_<clustername>.env` before running `hadoop fs` commands.

As an alternative, you could use the LINUX `cp` command to copy the files to FUSE.

- Rename the datafiles, using ALTER TABLESPACE with the RENAME DATAFILE clause.

! Important:

Note the “hdfs:” prefix to the file path in the SQL example below. This is the keyword that tells Smart Scan that it should scan the file. Smart Scan also requires that the file is read only. The cluster name is optional. Also, before running the SQL statement below, the directory \$ORACLE_HOME/dbs/hdfs:<clustername>/user/oracle/tablespaces should include the soft link movie_cold_hdfs1.dbf, pointing to /mnt/fuse-<clustername>-hdfs/user/oracle/tablespaces/movie_cold_hdfs1.dbf.

```
SQL> ALTER TABLESPACE movie_cold_hdfs RENAME
DATAFILE '/u01/app/oracle/oradata/cdb/orcl/movie_cold_hdfs1.dbf' TO
'hdfs:<clustername>/user/oracle/tablespaces/movie_cold_hdfs1.dbf';
```

When you rename the datafile, only the pointer in the database control file changes. This procedure does not physically rename the datafile.

The tablespace must exist on a single cluster. If there are multiple datafiles, these must point to the same cluster.

- Bring the tablespace back online and test it.

```
SQL> ALTER TABLESPACE movie_cold_hdfs ONLINE;
SQL> SELECT avg(rating) FROM movie_fact;
```

Below is the complete code example. In this case we move three partitions from local Oracle Database storage to the tablespace in HDFS.

```
mount hdfs
select * from dba_tablespaces;

CREATE TABLESPACE movie_cold_hdfs DATAFILE '/u01/app/oracle/oradata/cdb/
orcl/movie_cold_hdfs1.dbf' SIZE 100M reuse AUTOEXTEND ON nologging;

ALTER TABLE movie_fact
MOVE PARTITION 2010_JAN TABLESPACE movie_cold_hdfs ONLINE UPDATE
INDEXES;
ALTER TABLE movie_fact
MOVE PARTITION 2010_FEB TABLESPACE movie_cold_hdfs ONLINE UPDATE
INDEXES;
ALTER TABLE movie_fact
MOVE PARTITION 2010_MAR TABLESPACE movie_cold_hdfs ONLINE UPDATE
INDEXES;

-- Check for the changes
SELECT table_name, partition_name, tablespace_name FROM
user_tab_partitions WHERE table_name='MOVIE_FACT';
```

```
ALTER TABLESPACE movie_cold_hdfs READ ONLY;
ALTER TABLESPACE movie_cold_hdfs OFFLINE;

hadoop fs -put /u01/app/oracle/oradata/cdb/orcl/movie_cold_hdfs1.dbf /
user/oracle/tablespaces/
hadoop fs -chmod 444 /user/oracle/tablespaces/ movie_cold_hdfs1.dbf

ALTER TABLESPACE movie_cold_hdfs RENAME DATAFILE '/u01/app/oracle/
oradata/cdb/orcl/movie_cold_hdfs1.dbf' TO 'hdfs:hadoop_cl_1/user/oracle/
tablespaces/movie_cold_hdfs1.dbf';
ALTER TABLESPACE movie_cold_hdfs ONLINE;

-- Test
select avg(rating) from movie_fact;
```

3.4.4 Smart Scan for TableSpaces in HDFS

Smart Scan is an Oracle performance optimization that moves processing to the location where the data resides. In Big Data SQL, Smart Scan searches for datafiles whose path includes the “hdfs:” prefix. This prefix is the key that indicates the datafile is eligible for scanning.

After you have moved your tablespace data to HDFS and the tablespace and have prefixed the datafile path with the “hdfs:” tag, then queries that access the data in these files will leverage Big Data SQL Smart Scan by default. All of the Big Data SQL Smart Scan performance optimizations will apply. This greatly reduces the amount of data that moves from the storage tier to the database tier. These performance optimizations include:

- The massively parallel processing power of the Hadoop cluster is employed to filter data at its source.
- Storage Indexes can be leveraged to reduce the amount of data that is scanned.
- Data mining scoring can be off-loaded.
- Encrypted data scans can be off-loaded.

Disabling or Enabling Smart Scan

The initialization parameter `_CELL_OFFLOAD_HYBRID_PROCESSING` determines whether Smart Scan for HDFS is enabled or disabled. It is enabled by default.

To disable Smart Scan for tablespaces in HDFS do the following.

1. Set the parameter to `FALSE` in `init` or in a parameter file:

```
_CELL_OFFLOAD_HYBRID_PROCESSING=FALSE
```

The underscore prefix is required in this parameter name.

2. Restart the Oracle Database instance.

You can also make this change dynamically using the `ALTER SYSTEM` directive in SQL. This does not require a restart.

```
SQL> alter system set _cell_offload_hybrid_processing=false;
```

One reason to turn off Smart Scan is if you need to move the Oracle tablespace datafiles out of HDFS and back to their original locations.

You can re-enable Smart Scan by resetting `_CELL_OFFLOAD_HYBRID_PROCESSING` to `TRUE`.



Note:

When `_CELL_OFFLOAD_HYBRID_PROCESSING` is set to `FALSE`, Smart Scan is disabled for Oracle tablespaces residing in HDFS.

4

Oracle Big Data SQL Security

Security and protection of data are foremost concerns for customers and for Oracle. Some basic aspects of Oracle Big Data SQL security include granting access, multiuser authorization, and compliance with Oracle Database security policies.

- [Accessing Oracle Big Data SQL](#)
- [Multi-User Authorization](#)
- [Sentry Authorization in Oracle Big Data SQL](#)
- [Hadoop Authorization: File Level Access and Apache Sentry](#)
- [Compliance with Oracle Database Security Policies](#)

To learn more, read *Securing Oracle Big Data SQL* in *Oracle Big Data SQL Installation Guide*.

4.1 Accessing Oracle Big Data SQL

The basics for accessing a cluster within Oracle Big Data SQL are explained here.

At a minimum, you must do the following for each user who needs access to Oracle Big Data SQL:

- Grant the BDSQL_USER role.
- Grant read privileges on the Oracle Big Data SQL configuration directory object. For example, to grant access to user1:

```
SQL> grant BDSQL_USER to user1;  
SQL> grant read on directory ORACLE_BIGDATA_CONFIG to user1;
```

4.2 Multi-User Authorization

By default, queries executed using Oracle Big Data SQL run as the `oracle` user on the Hadoop cluster. All Hadoop audits in this default configuration show that the `oracle` user accessed the files.

Oracle Big Data SQL provides a feature called Multi-User Authorization that enables it to impersonate the connected user when accessing data on the Hadoop cluster. With Multi-User Authorization, the `oracle` identity is no longer used to authorize data access. Instead, the identity of the actual connected user receives authorization. Additionally, Hadoop audits will attribute file access to the connected user, rather than to `oracle`.

Users and applications can connect to Oracle Database in these distinct ways (and more):

- As a database user

- As a Kerberos user
- As an LDAP user
- As an application user

Multi-User Authorization allows the administrator to specify how this connected user should be derived. For example, all users that connect to Oracle Database using their LDAP identity will use their authenticated identity when running queries on the Hadoop cluster. Alternatively, applications that manage their own users may use the Oracle Database client identifier to derive the currently connected user (and use that user's identity to authorize access to data on the Hadoop cluster). Oracle Big Data SQL provides a mapping that contains the rules for identifying the actual user.

 **See Also:**

- [DBMS_BDSQL PL/SQL Package](#), which explains how to use this package to implement Multi-User-Authorization.
- The Apache Foundation documentation at <https://sentry.apache.org>.

4.2.1 The Multi-User Authorization Model

Multi-User Authorization gives you the ability to use Hadoop Secure Impersonation to direct the `oracle` account to execute tasks on behalf of other designated users.

This enables HDFS data access based on the user that is currently executing the query, rather than the singular `oracle` user.

Administrators set up the rules for identifying the query user (the currently connected user) and for mapping this user to the user that is impersonated. Because there are numerous ways in which users can connect to Oracle Database, this user may be a database user, a user sourced from LDAP, from Kerberos, or other sources. Authorization rules on the files apply to the query user and audits will identify the user as the query user.

 **See Also:**

[DBMS_BDSQL PL/SQL Package](#) which describes the Multi-User Authorization security table and the procedures for adding user maps to the table and removing them from the table.

4.3 Sentry Authorization in Oracle Big Data SQL

In addition to supporting authorization for HDFS file access, Oracle Big Data SQL supports Sentry policies, which authorize access to Hive metadata. Sentry enables fine-grained control over user access, down to the column level.

Note: Sentry is not supported for Big Data SQL installations on Oracle Database 12.1 systems.

4.3.1 Sentry and Multi-User Authorization

You can use Oracle Big Data SQL's Multi-User Authorization system to enhance Sentry security.

Oracle Big Data SQL utilizes Sentry policies whenever Sentry is enabled on the Hadoop cluster. Support for Sentry in Oracle Big Data SQL is most effective when used in conjunction with the Multi-User Authorization system. Multi-User Authorization enables Sentry authorization based on the actual identity of the currently connected user.

If Multi-User Authorization is not enabled, then the `oracle proxy` user is used for authorization for all queries.

See Also:

- [The Multi-User Authorization Model](#), the previous section in this chapter.
- [DBMS_BDSQL PL/SQL Package](#) describes `SYS.BDSQL_USER_MAP` and the procedures for managing the table.
- [Jaguar Configuration Parameter and Command Reference in Oracle Big Data SQL Installation Guide](#) shows how to use Jaguar parameters to configure security features in Oracle Big Data SQL. The `impersonation_enabled` parameter enables or disables multi-user authorization (which is based on Apache's Hadoop Secure Impersonation).

4.3.2 Groups, Users, and Role-Based Access Control in Sentry

Oracle Big Data SQL does not directly control access to Hive tables. It respects the access controls defined by the Hive database administrator. For Sentry, these controls are role-based. A given user's access rights are defined by their group memberships and the roles assigned to those groups.

The administrator uses tools such as the HiverServer2 Beeline utility to assign privileges (such as `SELECT` and `INSERT`) to Hive tables and their columns. The administrator also creates Sentry roles, assigns roles to different user groups, and grants privileges to those roles. The Hadoop user operating in the Hadoop environment inherits all of the privileges from each role that is assigned to their group or groups.

For example, to provide access to salary data, an administrator may create a role for this purpose and then grant `SELECT` privileges to the role:

```
CREATE ROLE auditfixedcosts;  
GRANT SELECT ON TABLE salary TO ROLE auditfixedcosts;
```

Grants may also be given to the role `auditfixedcosts` for access to data on other servers or in other databases, tables, or columns.

The administrator assigns the `auditfixedcosts` role to a group:

```
GRANT ROLE fixedcosts TO GROUP finance;
```

Members of the `finance` group then have `SELECT` access to all data in the `salary` table, any other `SELECT` access granted to the `auditfixedcosts` role, as well as access inherited from any other roles granted to the `finance` group.

See Also:

- [Hive SQL Syntax for Use with Sentry](#) on Cloudera's web site provides information on how to configure Sentry permissions.
- See the Apache Foundation documentation at <https://sentry.apache.org> for more details.

4.3.3 How Oracle Big Data SQL Uses Sentry

In Oracle Big Data SQL, Sentry provides a way to grant or withhold the `SELECT` privilege for individual Oracle Database users who attempt to query Oracle external tables over Hive tables.

As a result, Sentry policies do not need to be replicated to the Oracle external tables (i.e. using `GRANT`), which simplifies administration.

Sentry can be used to control the `SELECT` privilege at these levels of scope:

- Server (cluster node)
- Hive database
- Specific columns within a Hive table

Oracle Big Data SQL does not support DML operations. Therefore, only the `SELECT` privilege applies to Oracle Big Data SQL queries. The Sentry privileges `ALL`, `OWNER`, `CREATE`, `INSERT`, and `REFRESH` are not relevant to Oracle Big Data SQL access.

How Oracle Big Data SQL Uses Sentry to Determine Access at Query Execution

When a user executes a query against an Oracle external table over a Hive table, Oracle Big Data SQL checks for role-based Sentry privileges granted to the Hadoop user and applies them against the Oracle external table that is created over the current Hive table. It then compares these with the privileges required to execute the query. If the privileges granted to the user do not fulfill the requirements of the query, then an exception is raised.

4.3.4 Oracle Big Data SQL Privilege-Related Exceptions for Sentry

Oracle Big Data SQL raises one of two exceptions when a user's Sentry privileges are not sufficient to execute a query.

The exceptions and the errors returned by are as follows.

- Table-level privilege exception:

```
"User <the user> does not have SELECT privileges on oracle table
<external table name>
for Server=<server name>->Db=<Hive db name>->Table=<table name>"
```

- Column-level privilege exception:

```
"User <the user> does not have SELECT privileges on oracle table
<external table name>
column <column name> for Server=<server name>->Db=<Hive db name>-
>Table=<table name>->Field=<field name>"
```

4.4 Hadoop Authorization: File Level Access and Apache Sentry

The ability to access source data is based on both the underlying access privileges on the source files and Hive authorization rules defined by Apache Sentry. To populate Oracle Big Data SQL external tables, either the default `oracle` user or the actual connected user (when using Multi-User Authorization) must be authorized to read the source data and/or Hive metadata.

Access to data files in Hadoop is very similar to the POSIX permissions model (the model used by Linux). Each file and directory has an associated owner and group. The file and directory permission bits are used to determine who has access to that information.

Apache Sentry is a role based authorization engine used for Hive metadata. Sentry roles are defined for different data access needs (e.g. finance role, marketing role, etc.). Access to objects (a server, Hive database, table and column) is granted to specific roles. Users can then view those data objects if their group has been given appropriate rights.

Oracle Big Data SQL supports Sentry in addition to supporting file-level authorization. It processes the Sentry policy rules when a user attempt to query Oracle Big Data SQL external tables, down to the column level. This means that authorization rules do not need to be replicated in Oracle Database. A user may have rights to select from an Oracle external table. However, Hadoop authorization only allows the user to see the data if that user has the appropriate Sentry roles and data access privileges.

4.5 Compliance with Oracle Database Security Policies

Oracle Big Data SQL external tables follow the exact same Oracle security policies as other Oracle tables. Users and roles are granted or revoked privileges on the tables.

Advanced security policies, such as redaction and row level security may also be applied. These rules are layered on top of the privileges specified in Hadoop. This means that even when the underlying data source does not have support for advanced security features, such as redaction, those policies can still be enforced when you use Oracle Big Data SQL.

5

Working With Query Server

Learn how to manage Query Server with Oracle Big Data SQL.

5.1 About Oracle Big Data SQL Query Server

Oracle Big Data SQL Query Server is an Oracle Database instance that you can optionally install as a component of Oracle Big Data SQL on an edge node in your Hadoop cluster. You use Query Server to primarily query data stored in the cluster (in HDFS and Hive formats) or object stores using Oracle external tables. This enables you to take advantage of the full SQL capabilities provided by the Oracle Database.

You can define external tables using the `ORACLE_HDFS` and `ORACLE_HIVE` or `ORACLE_BIGDATA` access drivers or have the Query Server automatically define external tables based on the metadata in the Hive metastore. In the latter case, Hive databases map to Oracle Database schemas – and the corresponding Hive tables are defined as Oracle external tables in those schemas. All data authorization is based on authorization rules in Hadoop such as Apache Sentry or HDFS Access Controls Lists (ACLs).

Once installed, Query Server provides an Oracle Database deployment that is automatically configured to query data in your Hadoop cluster using SQL. Restarting the Query Server restores the database to a “clean” state, eliminating management overhead. A restart preserves external tables (`ORACLE_HIVE`, `ORACLE_HDFS`, and `ORACLE_BIGDATA` types), associated statistics, user defined views, and credentials. A restart deletes regular tables containing user data.

If your solution requires High Availability (HA), advanced Oracle security policies, or combining data in Oracle Database with data in Hadoop, then you should leverage a full-blown Oracle Database with Big Data SQL. Oracle supports using both Query Server and a Big Data SQL enabled Oracle Database for a single Hadoop deployment.

To install Query Server, you must specify an existing edge node in your Hadoop cluster in the `bds-config.json` configuration file. You use the same configuration file to specify a list of Hive databases. Query Server automatically creates Oracle external tables corresponding to the tables in the Hive metastore database(s) so that they are ready for querying after a successful installation. The set of external tables in the Query Server can be automatically kept up-to-date with the corresponding Hive metastore tables by running either the **Restart this Big Data SQL Query Server** or the **Synchronize Hive Databases** commands in Cloudera Manager or Apache Ambari cluster management software. You can also use the `dbms_bdsqs.sync_hive_databases` PL/SQL API package procedure.

 **See Also:**

See Introduction in the *Oracle Big Data SQL Installation Guide*, which describes how to install and configure the software on the two sides of an Oracle Big Data SQL configuration.

See [Store Oracle Tablespaces in HDFS](#) for instructions on how to set up data files for smart scanning.

5.2 Important Terms and Concepts

Introduction to edge nodes, edge database, cell nodes, and Hadoop cluster integration.

These terms are key to understanding Query Server.

About Edge Nodes

An edge node in a Hadoop cluster is the interface between the Hadoop cluster and the outside network. Typically, edge nodes are used to run client applications and Hadoop cluster administration tools such as Cloudera Manager and Apache Ambari. Edge nodes can act as a data gateway, by providing HDFS access through NFS or HttpFS, or by running REST servers.

About Cell Nodes

The BDS cells run on the DataNodes, and allows for parts of query processing to be pushed down to the Hadoop cluster DataNodes where the data resides. This ensures both load distribution and reduction in the volume of data that needs to be sent to the database for processing. This can result in significant performance improvements on Big Data workloads.

Hadoop Cluster Integration

Oracle Big Data SQL includes the following three service roles that can you can manage in either Cloudera Manager or Apache Ambari:

- **Big Data SQL Query Server:** Enables you to run SQL queries against the Hadoop cluster. Applications connect to this server using JDBC or SQL*Net.
- **Big Data SQL Agent:** Manages the Big Data SQL installation and is also used by the **Copy to Hadoop** feature.
- **Big Data SQL Server:** Also known as Big Data SQL Cells, allows for parts of query processing to get pushed-down to the Hadoop cluster DataNodes where the data resides.

5.3 Query Server Features

Big Data SQL Query Server provides automatic installation and configuration, integration with Hadoop cluster managers, and automatic integration of cluster metadata:

- **Automatic installation and configuration:** Oracle Big Data SQL installer automatically installs and configures Query Server, if you specify an existing target edge node in the Hadoop cluster in the `bds-config.json` configuration file. To specify the edge node where to install Query Server, you add the `edgedb` parameter and the node and enabled attributes to the `bds-config.json` configuration file to as shown in the following example where `<edgenode_host_name>` is the name of your edge node:

```
"edgedb": {  
  "node" : "dbnode.domain.com",  
  "enabled" : "true",  
  "sync_hive_db_list" : "my_hive_db_1,my_hive_db2"  
}
```

 **Note:**

If the `bds-config.json` configuration file does not include the `edgedb` subsection, then Query Server is not installed.

 **See Also:**

The `bds-config.json` Configuration Example in the installation guide shows a fully-populated `bds-config.json` file. The example includes all available configuration parameters.

- **Integration with Hadoop cluster managers:** You can monitor and manage Query Server as a service using Cloudera Manager or Apache Ambari Hadoop cluster management tools.
- **Synchronization with Hive:** When you start the Oracle Big Data service, Query Server automatically refreshes its metadata from the Hive metastore. After the initial refresh, users can synchronize the Query Server with the latest metadata in the Hive metastore.

5.4 Specifying the Hive Databases to Synchronize With Query Server

Before you can synchronize Query Server with the desired Hive databases in the metastore, you have to specify the list of Hive databases.

Use either of these methods:

- During installation, specify the `sync_hive_db_list` parameter in the `bds-config.json` configuration file.
- After installation, you can update the `sync_hive_db_list` configuration parameter in Cloudera Manager or Apache Ambari.

After installing Query Server, it automatically creates schemas and external tables based on the Hive metastore databases list that you specified. Every subsequent Query Server restart will perform a delta synchronization.

5.4.1 Specifying the Hive Databases in the bds-config.json Configuration File

You can provide the initial list of Hive databases to synchronize with Query Server as part of the installation process using the `bds-config.json` configuration file.

In the configuration file, include the `sync_hive_db_list` configuration parameter followed by a list of the Hive databases. The following example specifies two Hive databases for the `sync_hive_db_list` configuration parameter: `htdb0` and `htdb1`. Only these two databases will be synchronized with Query Server, even if the Hive metastore contains other databases.

```
"edgedb": {  
  "node": "<edgenode_host_name>",  
  "enabled": "true",  
  "sync_hive_db_list": "htdb0,htdb1"  
  . . .  
}
```

To synchronize all Hive databases in the metastore with Query Server, use the `"*"` wildcard character as follows:

```
"edgedb": {  
  "node": "EdgeNode_Host_Name",  
  "enabled": "true"  
  "sync_hive_db_list": "*"  
  . . .  
}
```

If the `bds-config.json` configuration file does not contain the `sync_hive_db_list` configuration parameter, then no synchronization will take place between the Hive databases and Query Server. In that case, you must specify the Hive databases using the `sync_hive_db_list` configuration parameter in Cloudera Manager or Apache Ambari.

Note:

Query Server is not intended to store internal data in Oracle tables. Whenever the Query Server is re-started, it is "reset" to its initial and clean state. This eliminates typical database maintenance such as storage management, database configuration, and so on. The goal of Query Server is to provide a SQL front-end for data in Hadoop, Object Store, Kafka, and NoSQL databases and not a general-purpose RDBMS.

5.4.2 Updating the Hive Databases With the `sync_hive_db_list` Configuration Parameter

You can update the list of the Hive databases to synchronize with Query Server by using Cloudera Manager.

You can update the list of the Hive databases to synchronize with Query Server by using the `sync_hive_db_list` configuration parameter in Cloudera Manager as follows:

1. Login to Cloudera Manager by using your login credentials.
2. In Cloudera Manager, use the **Search** field to search for the **Synchronized Hive Databases** configuration parameter. Enter **/Synchronized Hive Databases** (or enter part of the name until it is displayed in the list) in the **Search** field, and then press Enter.
3. Click the **Big Data SQL: Synchronized Hive Databases** parameter.
4. In the **Synchronized Hive Databases** text box, enter the names of the Hive databases separated by commas, such as `htdb0,htdb1`, and then click **Save Changes**. Only these two Hive databases will be synchronized with Query Server.

To synchronize all Hive databases in the metastore with Query Server, enter the "*" wildcard character in the **Synchronized Hive Databases** text box, and then click **Save Changes**.

5.5 Synchronizing Query Server With Hive

You can synchronize the Query Server with the Hive databases that you specified by using Cloudera Manager, Apache Ambari, or the `dbms_bdsqs.sync_hive_databases` PL/SQL API.

You can synchronize Query Server with the Hive databases in the metastore using one of the following methods:

- Execute the **Restart this Big Data SQL Query Server** command in Cloudera Manager or Apache Ambari.
- Execute the **Synchronize Hive Databases** command in Cloudera Manager or Apache Ambari.
- Invoke the `dbms_bdsqs.sync_hive_databases` PL/SQL API locally on the edge node.

You must specify the Hive databases to use in the synchronization either by using the `bds-config.json` configuration file or by using the `sync_hive_db_list` configuration parameter in Cloudera Manager.

Note that the `dbms_bdsqs.sync_hive_databases` PL/SQL API will only refresh the Hive table definitions for the Hive databases that have already been synchronized through the other two methods.

5.5.1 Restarting Query Server Manually by Using Cloudera Manager

You can synchronize Query Server with the Hive databases that you specified by restarting Query Server in Cloudera Manager or Apache Ambari.

You can use Cloudera Manager or Apache Ambari to manage Query Server such as starting, stopping, and restarting it. When you restart or start Query Server, it synchronizes the metadata with the Hive databases that you specified. Any changes in the Hive databases in the metastore such as dropping or adding tables will be reflected in Query Server. For example, you can restart Query Server in Cloudera Manager as follows:

You must specify the Hive databases to use in the synchronization either by using the `bds-config.json` configuration file or by using the `sync_hive_db_list` configuration parameter in Cloudera Manager.

1. Login to Cloudera Manager using your login credentials.
2. In the list of available services, click the **Big Data SQL** link to display the **Big Data SQL** details page.
3. From the **Status Summary** section, click the **Big Data SQL Query Server** link to display the **Big Data SQL Query Server** details page.
4. From the **Actions** drop-down list, select **Restart this Big Data SQL Query Server**.

A dialog box is displayed. Click **Restart this Big Data SQL Query Server**. Another dialog box is displayed to monitor the status of the synchronization job.

5.5.2 Synchronizing Query Server Manually by Using Cloudera Manager

You can use Cloudera Manager or Apache Ambari to manually synchronize Query Server with the Hive databases that you specified.

After the synchronization, any changes in the Hive databases in the metastore such as dropped or added tables will be reflected in Query Server. For example, you can synchronize Query Server in Cloudera Manager as follows:

1. Login to Cloudera Manager by using your login credentials.
2. In the list of available services, click the **Big Data SQL** link to display the **Big Data SQL** details page.
3. From the **Status Summary** section, click the **Big Data SQL Query Server** link to display the **Big Data SQL Query Server** details page.
4. From the **Actions** drop-down list, select **Synchronize Hive Databases**.

A dialog box is displayed. Click **Synchronize Hive Databases**. Another dialog box is displayed to monitor the status of the synchronization job.

You must specify the Hive databases to use in the synchronization either by using the `bds-config.json` configuration file or by using the `sync_hive_db_list` configuration parameter in Cloudera Manager.

5.5.3 Synchronizing Query Server Manually by Using the PL/SQL API

You can synchronize Query Server with the Hive databases that you specified by using the PL/SQL API.

To do so, invoke the `dbms_bdsqs.sync_hive_databases` PL/SQL API locally on the edge node where the Query Server is installed.

The procedure contains no parameters. It synchronizes all of the Hive databases that are already in Query Server. The API will refresh the Query Server with only the Hive databases listed in the `sync_hive_db_list` configuration parameter. Each successive synchronization (also known as a refresh) will process changes since the last Query Server metadata refresh.

A synchronization captures any tables that were added or dropped in the Hive metastore since the last refresh. This also includes any tables whose schemas might have changed.

5.5.4 Enabling Query Server Full Synchronization

You can specify whether Query Server performs a delta (default) or a full synchronization.

During the Query Server installation process, the Oracle schemas and the appropriate external tables are created based on the Hive databases list that you can specify either in the `bds-config.json` configuration file or the `sync_hive_db_list` configuration parameter. In that case, Query Server performs a full synchronization. By default, Query Server performs a delta synchronization during subsequent restarts or synchronizations.

You can control whether Query Server performs a full or a delta synchronization by using the **Enable full synchronization** configuration parameter in Cloudera Manager or Apache Ambari. This configuration parameter is de-selected by default. To enable Query Server to perform a full synchronization, select this checkbox in Cloudera Manager or Apache Ambari. For example, you can use Cloudera Manager to enable Query Server to perform a full synchronization during a restart or a manual synchronization as follows:

1. Login to Cloudera Manager by using your login credentials.
2. In Cloudera Manager, use the **Search** field to search for the **Enable full synchronization** configuration parameter. Enter **/ Enable full synchronization** (or enter part of the name until it is displayed in the list) in the **Search** field, and then press Enter.
3. Click **Big Data SQL: Enable full synchronization**. The checkbox is de-selected by default. This indicates that Query Server will perform a delta synchronization.
4. To enable full synchronization, select the checkbox, and then click **Save Changes**.

A full synchronization drops all of the existing schemas and external tables from Query Server, and then re-creates new schemas and new external tables based on the Hive databases list that you specified in the `sync_hive_db_list` configuration parameter.

By default, Query Server performs a delta synchronization between the Hive databases in the metastore that you specify and Query Server. Any changes in the Hive databases such as dropping or adding tables will be reflected in Query

Server. However, When you start Query Server for the very first time, it will create Oracle Schemas based on the Hive databases that you specify either in the `bds-config.json` configuration file or in `sync_hive_db_list` configuration parameter in Cloudera Manager or Apache Ambari.

The first time the Query Server synchronizes with Hive the process will be slower than usual. This is because it is importing all of the tables for the specified databases (configured in Cloudera Manager or Apache Ambari) in the Hadoop cluster. Subsequent refreshes should be much faster as it would only refresh the changes that were made to the Hive Metadata such as additions of new tables. During a delta import, the Query Server will also gather new statistics for tables that have been added/modified.

5.6 Query Server Restarts and Metadata Persistence

You can refresh the Query Server metadata using Cloudera Manager, Apache Ambari, or a PL/SQL API.

The following key metadata can be saved so that they can be restored after a Query Server restart:

- **Table statistics**

Gathering statistics can be an expensive operation. Table Statistics are gathered automatically after each metadata synchronization. Subsequent statistics gathering may be captured using the following PL/SQL package procedure and parameters:

```
DBMS_STATS.GATHER_TABLE_STATS (ownname => <schema>,
                               tabname => <table-name>, estimate_percent =>
                               dbms_stats.auto_sample_size);
```

 **Note:**

Make sure you use the `estimate_percent=>dbms_stats.auto_sample_size` parameter.

- Hive external tables that use the `ORACLE_HIVE` access driver.
- HDFS external tables that use the `ORACLE_HDFS` access driver.
- User-defined views.

5.7 Query Server Security

You can connect to Query Server using the single-user or multi-user modes. Query Server users connect to a pluggable database called `BDSQLUSR`. There are two ways to connect to the database, depending on whether the cluster is secure (by means of Kerberos) or non-secure.

Connecting to the Query Server Database

To query data in the Hadoop cluster, users can connect to the Query Server if it is installed. During the Big Data SQL installation, the Big Data SQL installer creates and installs the `BDSQLUSR` Query Server database on the edge node that you specified. In

addition, the installer also installs everything else that you need on that edge node to enable you to query data in the Hadoop cluster.

Query Server users can connect to the Query Server database using the single-user or multi-user modes. By default, Query Server is configured as a single-user mode for a non-secured Hadoop cluster.

Connecting to Query Server in a Single-User Mode

Query Server supports a single-user model for non-secured Hadoop clusters. With this model, all users connect to the `BDSQLUSR` Query Server database as user `bdsq1` with a password that the administrator chooses during the Query Server installation.

Queries run on the cluster with the `oracle` user permissions. This means that the `oracle` user must be authorized to access the underlying Hadoop data – either by using Sentry privileges and/or HDFS authorizations. For example, you can connect to the Query Server database using `SQL*Plus` as follows:

```
sqlplus BDSQL/<bdsq1_password>@BDSQLUSR
```

Note:

Substitute `<bdsq1_password>` in the above command with the actual `BDSQL` password that the administrator specified during the Oracle Big Data SQL installation.

Changing the BDSQL User Password

When installing Oracle Big Data SQL Query Server on a non-secure cluster, you can change the password of the `bdsq1` user with `ALTER USER` as follows:

```
# su - oracle
sqlplus / as sysdba
sql> alter session set container=bdsq1usr;
sql> alter user bdsq1 identified by "<new_password>";
```

Note:

Substitute `<new_password>` with the new password. The new password must conform to the required Oracle secure password guidelines. See [Choosing a Secure Password](#) for information about choosing your new password.

Note that on a Kerberos-secured cluster, the user `bdsq1` is disabled.

Connecting to Query Server in a Single-User Mode (non-secured Hadoop Clusters)

Query Server supports a single-user model for non-secured Hadoop clusters. With this model, all users connect to the `BDSQLUSR` Query Server database as user `BDSQL` with a password that the administrator chooses during the Query Server installation.

Queries run on the cluster as one of two users depending on how Big Data SQL authorization has been configured. When Big Data SQL is setup with multi-user authorization, queries will run on the Hadoop cluster as OS user `bdsq1`. Without multi-user authorization configured, queries will run as the user `oracle`.

The implication is that the Big Data SQL user must be authorized to access the underlying Hadoop data – either by using Sentry privileges and/or HDFS authorizations.

Connecting to Query Server in a Multi-User Mode (Kerberos-enabled Hadoop Clusters)

Query Server supports Kerberos-enabled Hadoop clusters. You use this mode when there are multiple externally identified user accounts, corresponding to Kerberos principals. Connected users' identities (Kerberos principals) will be used for authorization on the Hadoop cluster when the `impersonation_enabled` parameter is set to `true` in the `bds-config.json` configuration file. If this parameter is set to `false`, then authorization on the Hadoop cluster will be performed as user `oracle`. Sentry is used on Cloudera clusters. HDFS authorization is used for Hortonworks clusters. (Oracle Big Data SQL does not yet make use of Apache Ranger on Hortonworks HDP.)

Note:

See The Multi-User Authorization Model to learn how to use Hadoop Secure Impersonation to direct the `oracle` account to execute tasks on behalf of other designated users.

Hadoop queries executing on the cluster on behalf of a user will appear to the Hadoop nodes as the authenticated user and will have the corresponding permissions. The same is true for Hive queries and for statistics gathering operations. Before you can connect to Query Server, you must be authenticated with Kerberos using `kinit`. When you install or reconfigure Big Data Query Server on a secure cluster, Jaguar collects all principals from the Key Distribution Center (KDC) running on nodes where Hadoop DataNodes are also installed. For each principal, an externally identified user will be created on Big Data SQL Query Server. This install-time behavior is controlled by the `syncPrincipals` parameter in the `bds-config.json` configuration file. This operation can also be invoked by running the following command (notice that the spelling of the Jaguar operation is different):

```
jaguar sync_principals
```

You can also use the `DBMS_BDSQS_ADMIN` package which contains procedures to add and drop Query Server users. These Query Servers users are the same Kerberos principals that will be accessing your Hadoop cluster.

```
DBMS_BDSQS_ADMIN.ADD_KERBEROS_PRINCIPALS(principals_list varchar2,  
op_semantics varchar2 DEFAULT 'STOP_ON_FIRST_ERROR')
```

```
DBMS_BDSQS_ADMIN.DROP_KERBEROS_PRINCIPALS(principals_list varchar2,  
op_semantics varchar2 DEFAULT 'STOP_ON_FIRST_ERROR')
```

 **Note:**

Before you can run the procedures in the `DBMS_BDSQS_ADMIN` package, you must connect to Oracle Big Data SQL Query Server as user `sys` using OS authentication. For example, you can login to SQL*Plus as OS user `oracle`. See [Oracle Big Data SQL Reference](#)

Users in a multi-user mode can then connect to SQL*Plus without providing a password as follows:

```
[user_name@cluster_name ~]$ kinit user_name
Password for user_name@cluster_name.US.ORACLE.COM:
[user_name@cluster_name ~]$ sqlplus /@BDSQLUSR

SQL*Plus: Release 18.0.0.0.0 - Production on Tue Oct 2 13:54:39 2018
Version 18.3.0.0.0

Copyright (c) 1982, 2018, Oracle. All rights reserved.

Last Successful login time: Tue Oct 02 2018 13:54:20 -05:00

Connected to:
Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.3.0.0.0

SQL>
```

 **Note:**

In the above example, `user_name` and `cluster_name` reflects your actual username and cluster name.

6

Oracle Big Data SQL Reference

Find reference information for Oracle Big Data SQL here:

- [CREATE TABLE ACCESS PARAMETERS Clause](#)
- [Static Data Dictionary Views for Hive](#)
- [DBMS_BDSQL PL/SQL Package](#)
- [DBMS_BDSQS_ADMIN PL/SQL Package](#)
- [DBMS_HADOOP PL/SQL Package](#)

6.1 CREATE TABLE ACCESS PARAMETERS Clause

This section describes the properties that you use when creating an external table that uses the `ORACLE_HDFS`, `ORACLE_HIVE`, or `ORACLE_BIGDATA` access drivers. In a `CREATE TABLE ORGANIZATION EXTERNAL` statement, specify the parameters in the `opaque_format_spec` clause of `ACCESS PARAMETERS`.

This section contains the following topics:

- [Syntax Rules for Specifying Properties](#)
- [ORACLE_HDFS Access Parameters](#)
- [ORACLE_HIVE Access Parameters](#)
- [Full List of Access Parameters for ORACLE_HDFS and ORACLE_HIVE](#)
- [ORACLE_BIGDATA Access Parameters](#)

6.1.1 Syntax Rules for Specifying Properties

The properties are set using keyword-value pairs in the SQL `CREATE TABLE ACCESS PARAMETERS` clause and in the configuration files.

The syntax must obey these rules:

- The format of each keyword-value pair is a *keyword*, a colon or equal sign, and a *value*. The following are valid keyword-value pairs:

```
keyword=value  
keyword:value
```

The value is everything from the first non-whitespace character after the separator to the end of the line. Whitespace between the separator and the value is ignored. Trailing whitespace for the value is retained.

- A property definition can be on one line or multiple lines.
- A line terminator is a line feed, a carriage return, or a carriage return followed by line feeds.

- When a property definition spans multiple lines, then precede the line terminators with a backslash (escape character), except on the last line. In this example, the value of the `Keyword1` property is `Value part 1 Value part 2 Value part 3`.

```
Keyword1= Value part 1 \
          Value part 2 \
          Value part 3
```

- You can create a *logical line* by stripping each physical line of leading whitespace and concatenating the lines. The parser extracts the property names and values from the logical line.
- You can embed special characters in a property name or property value by preceding a character with a backslash (escape character), indicating the substitution. [Table 6-1](#) describes the special characters.

Table 6-1 Special Characters in Properties

Escape Sequence	Character
<code>\b</code>	Backspace (<code>\u0008</code>)
<code>\t</code>	Horizontal tab (<code>\u0009</code>)
<code>\n</code>	Line feed (<code>\u000a</code>)
<code>\f</code>	Form feed (<code>\u000c</code>)
<code>\r</code>	Carriage return (<code>\u000d</code>)
<code>\"</code>	Double quote (<code>\u0022</code>)
<code>\'</code>	Single quote (<code>\u0027</code>)
<code>\\</code>	Backslash (<code>\u005c</code>) When multiple backslashes are at the end of the line, the parser continues the value to the next line only for an odd number of backslashes.
<code>\uxxxx</code>	2-byte, big-endian, Unicode code point. When a character requires two code points (4 bytes), the parser expects <code>\u</code> for the second code point.

6.1.2 ORACLE_HDFS Access Parameters

The access parameters for the `ORACLE_HDFS` access driver provide the metadata needed to locate the data in HDFS and generate a Hive table over it.

6.1.2.1 Default Parameter Settings for ORACLE_HDFS

Describes default parameter settings for `ORACLE_HDFS`.

If you omit all access parameters from the `CREATE TABLE` statement, then `ORACLE_HDFS` uses the following default values:

```
com.oracle.bigdata.rowformat=DELIMITED
com.oracle.bigdata.fileformat=TEXTFILE
com.oracle.bigdata.overflow={"action":"error"}
com.oracle.bigdata.erroropt={"action":"setnull"}
```

6.1.2.2 Optional Parameter Settings for ORACLE_HDFS

ORACLE_HDFS supports the following optional `com.oracle.bigdata` parameters, which you can specify in the `opaque_format_spec` clause:

- `com.oracle.bigdata.colmap`
- `com.oracle.bigdata.erroropt`
- `com.oracle.bigdata.fields`
- `com.oracle.bigdata.fileformat`
- `com.oracle.bigdata.log.exec`
- `com.oracle.bigdata.log.qc`
- `com.oracle.bigdata.overflow`
- `com.oracle.bigdata.rowformat`

Example 6-1 shows a `CREATE TABLE` statement in which multiple access parameters are set.

Example 6-1 Setting Multiple Access Parameters for ORACLE_HDFS

```
CREATE TABLE ORDER (CUST_NUM VARCHAR2(10),
                    ORDER_NUM VARCHAR2(20),
                    ORDER_DATE DATE,
                    ITEM_CNT NUMBER,
                    DESCRIPTION VARCHAR2(100),
                    ORDER_TOTAL (NUMBER8,2)) ORGANIZATION EXTERNAL
    (TYPE ORACLE_HDFS
    ACCESS PARAMETERS (
    com.oracle.bigdata.fields: (CUST_NUM,          \
                                ORDER_NUM,        \
                                ORDER_DATE,        \
                                ORDER_LINE_ITEM_COUNT, \
                                DESCRIPTION,        \
                                ORDER_TOTAL)
    com.oracle.bigdata.colMap:      {"col": "item_cnt", \
                                    "field": "order_line_item_count"}
    com.oracle.bigdata.overflow:   {"action": "TRUNCATE", \
                                    "col": "DESCRIPTION"}
    com.oracle.bigdata.errorOpt:  [{"action": "replace", \
                                    "value": "INVALID NUM", \
                                    "col": ["CUST_NUM", "ORDER_NUM"]} , \
                                    {"action": "reject", \
                                    "col": "ORDER_TOTAL"}]
    )
    LOCATION ("hdfs:/usr/cust/summary/*"));
```

6.1.3 ORACLE_HIVE Access Parameters

ORACLE_HIVE retrieves metadata about external data sources from the Hive catalog.

The default mapping of Hive data to columns in the external table are usually appropriate. However, some circumstances require special parameter settings, or you might want to override the default values for reasons of your own.

6.1.3.1 Default Parameter Settings for ORACLE_HIVE

Describes the default parameter settings for ORACLE_HIVE.

If you omit all access parameters from the CREATE TABLE statement, then ORACLE_HIVE uses the following default values:

```
com.oracle.bigdata.tablename=name of external table
com.oracle.bigdata.overflow={"action":"error"}
com.oracle.bigdata.erroropt={"action":"setnull"}
```

6.1.3.2 Optional Parameter Values for ORACLE_HIVE

ORACLE_HIVE supports the following optional com.oracle.bigdata parameters, which you can specify in the opaque_format_spec clause:

- [com.oracle.bigdata.colmap](#)
- [com.oracle.bigdata.erroropt](#)
- [com.oracle.bigdata.log.exec](#)
- [com.oracle.bigdata.log.qc](#)
- [com.oracle.bigdata.overflow](#)
- [com.oracle.bigdata.tablename](#)

Example 6-2 shows a CREATE TABLE statement in which multiple access parameters are set.

Example 6-2 Setting Multiple Access Parameters for ORACLE_HIVE

```
CREATE TABLE ORDER (cust_num VARCHAR2(10),
                    order_num VARCHAR2(20),
                    order_date DATE,
                    item_cnt NUMBER,
                    description VARCHAR2(100),
                    order_total (NUMBER8,2)) ORGANIZATION EXTERNAL
(TYPE oracle_hive
 ACCESS PARAMETERS (
  com.oracle.bigdata.tableName: order_db.order_summary
  com.oracle.bigdata.colMap:    {"col":"ITEM_CNT", \
                                "field":"order_line_item_count"}
  com.oracle.bigdata.overflow: {"action":"ERROR", \
                                "col":"DESCRIPTION"}
  com.oracle.bigdata.errorOpt: [{"action":"replace", \
                                "value":"INV_NUM" , \
                                "col":["CUST_NUM", "ORDER_NUM"]} , \
                                {"action":"reject", \
                                "col":"ORDER_TOTAL"}]
 ));
```

6.1.4 Full List of Access Parameters for ORACLE_HDFS and ORACLE_HIVE

6.1.4.1 com.oracle.bigdata.buffersize

Sets the buffer size in kilobytes for large record reads. Set this value if you need to read records that are greater than the default buffer size.

Default Value

1000 KB

Syntax

com.oracle.bigdata.buffersize: n

Example

The following example sets the buffer size to 100 MB:

com.oracle.bigdata.buffersize: 100000

6.1.4.2 com.oracle.bigdata.datamode

Specifies the method that SmartScan uses to scan a Hadoop data source. The method can make a significant difference in performance.

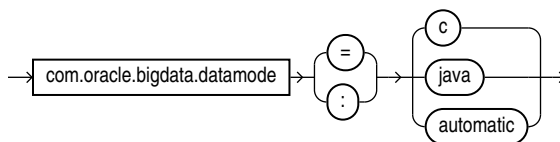
Default Value

automatic

Syntax

A JSON document with the keyword-value pairs shown in the following diagram:

datamode:



Semantics

automatic

Automatically selects the appropriate mode, based on the metadata. It selects *c* mode if possible, or *java* mode if the data contains formats that are not supported by *c* mode.

c

Uses Java to read the file buffers, but C code to process the data and convert it to Oracle format. Specify this mode for delimited data.

If the data contains formats that the C code does not support, then it returns an error.

java

Uses the Java SerDes and InputFormats to process the data and convert it to Oracle format. Specify this mode for Parquet, RCFile, and other data formats that require a SerDe.

6.1.4.3 com.oracle.bigdata.colmap

Maps a column in the source data to a column in the Oracle external table. You can define one or multiple pairs of column mappings. Use this property when the source field names exceed the maximum length of Oracle column names, or when you want to use different column names in the external table.

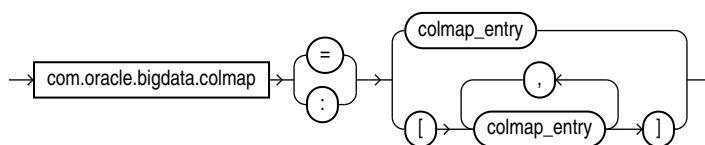
Default Value

A column in the external table with the same name as the Hive column

Syntax

A JSON document with the keyword-value pairs is shown in the following diagram:

colmap:



colmap_entry:



Semantics

"col":*name*

"col": The keyword must be lowercase and enclosed in quotation marks.

name: The name of a column in the Oracle external table. It is case sensitive and must be enclosed in quotation marks.

"field":*name*

"field": The keyword must be lowercase and enclosed in quotation marks.

name: The name of a field in the data source. It is not case sensitive, but it must be enclosed in quotation marks. See [Syntax Rules for Specifying Properties](#).

Examples

This example maps a Hive column named ORDER_LINE_ITEM_COUNT to an Oracle column named ITEM_CNT:

```
com.oracle.bigdata.colMap={"col":"ITEM_CNT", \
    "field":"order_line_item_count"}
```

The following example shows the mapping of multiple columns.

```
com.oracle.bigdata.colmap:[{"col":"KOL1", "field":"PROJECT_NAME"},
{"col":"KOL2","field":"wsdl_name"}, {"col":"KOL3", "field":"method"}]
```

6.1.4.4 com.oracle.bigdata.erroropt

Describes how to handle errors that occur while the value of a column is calculated.

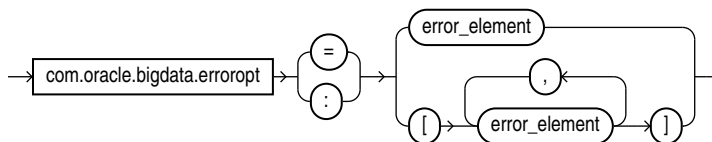
Default Value

```
{"action":"setnull"}
```

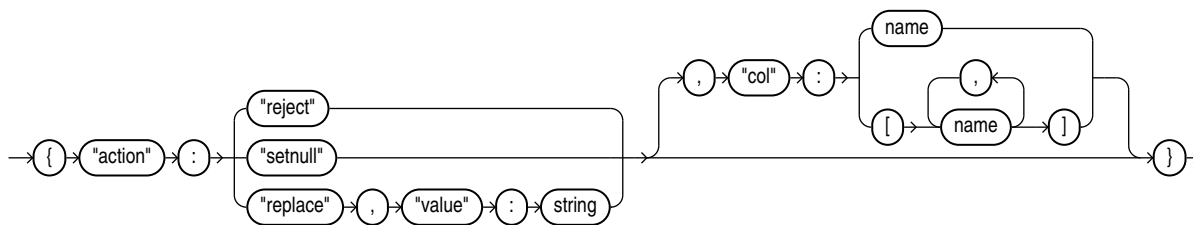
Syntax

A JSON document with the keyword-value pairs is shown in the following diagram:

erroropt:



error_element:



Semantics

The "action", "reject", "setnull", "replace", "value", and "col" keywords must be lowercase and enclosed in quotation marks. See [Syntax Rules for Specifying Properties](#).

"action":value

value: One of these keywords:

- "reject": Does not load any rows.
- "setnull": Sets the column to NULL.
- "replace": Sets the column to the specified value.

"value":string

string: Replaces a bad value in the external table. It must be enclosed in quotation marks.

"col":*name*

name: Identifies a column in an external table. The column name is case sensitive, must be enclosed in quotation marks, and can be listed only once.

Example

This example sets the value of the `CUST_NUM` or `ORDER_NUM` columns to `INVALID` if the Hive value causes an error. For any other columns, an error just causes the Hive value to be rejected.

```
com.oracle.bigdata.errorOpt: { "action": "replace", \
                              "value": "INVALID", \
                              "col": [ "CUST_NUM", "ORDER_NUM" ] }
```

6.1.4.5 com.oracle.bigdata.fields

Lists the field names and data types of the data source.

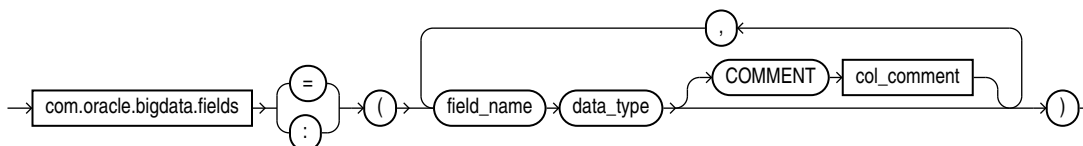
Default Value

Not defined

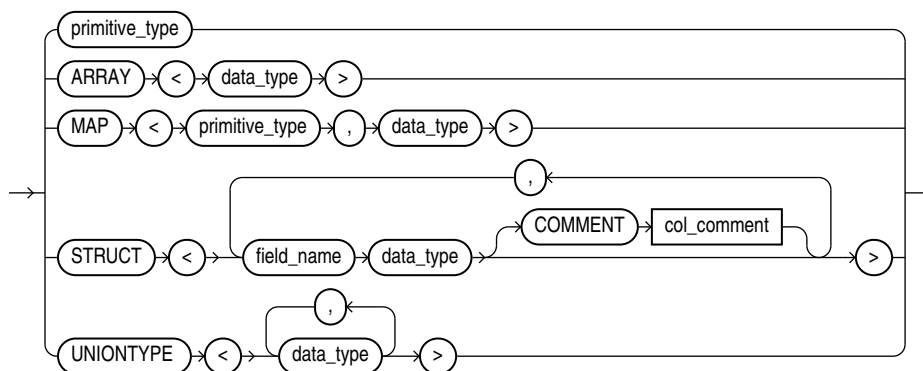
Syntax

A JSON document with the keyword-value pairs is shown in the following diagram:

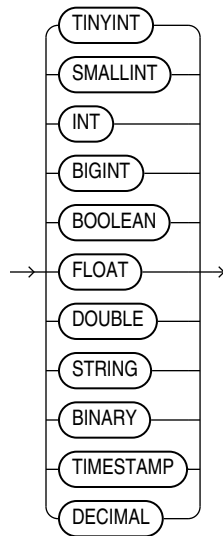
fields:



data_type:



primitive_type:



Semantics

The syntax is the same as a field list for a Hive table. If you split the field list across multiple lines, you must use a backslash to escape the new line characters.

`field_name`

The name of the Hive field. Use only alphanumeric characters and underscores (`_`). The maximum length is 128 characters. Field names are case-insensitive.

`data_type`

The data type of the Hive field. Optional; the default is `STRING`. The character set must be UTF8.

The data type can be complex or primitive:

Hive Complex Data Types

- `ARRAY`: Indexable list
- `MAP`: Key-value tuples
- `STRUCT`: List of elements
- `UNIONTYPE`: Multiple data types

Hive Primitive Data Types

- `INT`: 4 byte integer
- `BIGINT`: 8 byte integer
- `SMALLINT`: 2 byte integer
- `TINYINT`: 1 byte integer
- `BOOLEAN`: TRUE OR FALSE
- `FLOAT`: single precision
- `DOUBLE`: double precision
- `STRING`: character sequence

 **See Also:**

"Data Types" in the *Apache Hive Language Manual* at
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types>

COMMENT *col_comment*

A string literal enclosed in single quotation marks, which is stored as metadata for the Hive table (comment property of TBLPROPERTIES).

6.1.4.6 com.oracle.bigdata.fileformat

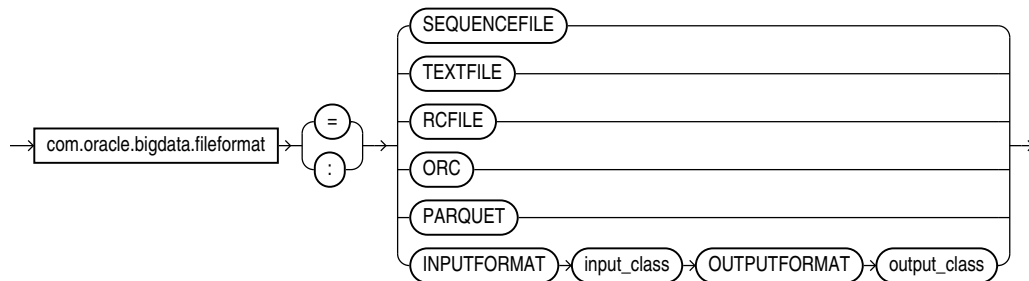
Describes the row format of the data source, based on the ROW FORMAT clause for a Hive table generated by ORACLE_HDFS.

Default Value

TEXTFILE

Syntax

A JSON document with the keyword-value pairs is shown in the following diagram.
fileformat:



Semantics

ORC

Optimized row columnar file format

PARQUET

Column-oriented, binary file format

RCFILE

Record columnar file format

SEQUENCEFILE

Compressed file format

TEXTFILE

Plain text file format

INPUTFORMAT

Identifies a Java class that can extract records from the data file.

OUTPUTFORMAT

Identifies a Java class that can format the output records in the desired format

6.1.4.7 com.oracle.bigdata.log.exec

Specifies how the access driver generates log files generated by the C code for a query, when it is running as parallel processes on CDH.

The access driver does not create or write log files when executing on a Hadoop cluster node; the parallel query processes write them. The log files from the Java code are controlled by `log4j` properties, which are specified in the configuration file or the access parameters. See "[bigdata-log4j.properties](#)".

Default Value

Not defined (no logging)

Syntax

```
[directory_object:]file_name_template
```

Semantics

directory_object

The Oracle directory object for the HDFS path on the Hadoop cluster where the log file is created.

file_name_template

A string used to generate file names. This table describes the optional variables that you can use in the template.

Table 6-2 Variables for com.oracle.bigdata.log.exec

Variable	Value
%p	Operating system process identifier (PID)
%a	A number that uniquely identifies the process.
%%	A percent sign (%)

Example

The following example generates log file names that include the PID and a unique number, such as `xtlogp_hive14_3413_57`:

```
com.oracle.bigdata.log.exec= xtlogp_hive14_%p_%a
```

6.1.4.8 com.oracle.bigdata.log.qc

Specifies how the access driver generates log files for a query.

Default Value

Not defined (no logging)

Syntax

```
[directory_object:]file_name_template
```

Semantics

directory_object

Name of an Oracle directory object that points to the path where the log files are written. If this value is omitted, then the logs are written to the default directory for the external table.

file_name_template

A string used to generate file names. [Table 6-3](#) describes the optional variables that you can use in the string.

Table 6-3 Variables for com.oracle.bigdata.log.qc

Variable	Value
%p	Operating system process identifier (PID)
%%	A percent sign (%)

Example

This example creates log file names that include the PID and a percent sign, such as xtlogp_hive213459_%:

```
com.oracle.bigdata.log.qc= xtlogp_hive21%p_%%
```

6.1.4.9 com.oracle.bigdata.overflow

Describes how to handle string data that is too long for the columns in the external table. The data source can be character or binary. For Hive, the data source can also be STRUCT, UNIONTYPES, MAP, or ARRAY.

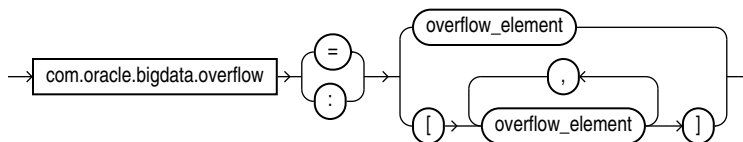
Default Value

```
{"action": "error" }
```

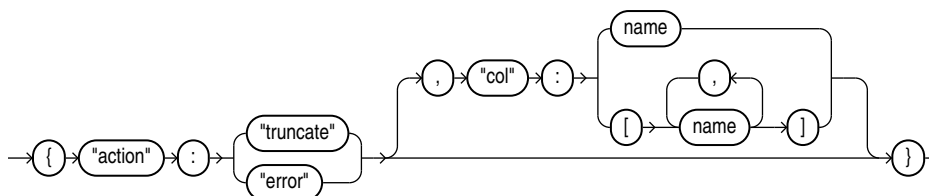
Syntax

A JSON document with the keyword-value pairs is shown in the following diagram:

```
overflow ::=
```



overflow_element ::=



Semantics

The "action", "truncate", "error", and "col" tags must be lowercase and enclosed in quotation marks. See [Syntax Rules for Specifying Properties](#).

"action":value

The value of "action" can be one of the following keywords:

- truncate: Shortens the data to fit the column.
- error: Throws an error. The [com.oracle.bigdata.erroropt](#) property controls the result of the error.

"col":name

name: Identifies a column in the external table. The name is case sensitive and must be enclosed in quotation marks.

Example

This example truncates the source data for the DESCRIPTION column, if it exceeds the column width:

```
com.oracle.bigdata.overflow={"action":"truncate", \
                             "col":"DESCRIPTION"}
```

6.1.4.10 com.oracle.bigdata.rowformat

Provides the information the access driver needs to extract fields from the records in a file.

! Important:

The `com.oracle.bigdata.rowformat` is unrelated to the access parameter syntax of traditional external tables that use "type `ORACLE_LOADER`." There are keywords such as `FIELDS`, `TERMINATED`, and others that appear in both clauses, but the commonality in naming is coincidental and does not imply common functionality. The `com.oracle.bigdata.rowformat` access parameter is passed without change to the default Hive serde. The Hive serde to extract columns from rows is deliberately limited. Complex cases are handled by specialized serdes.

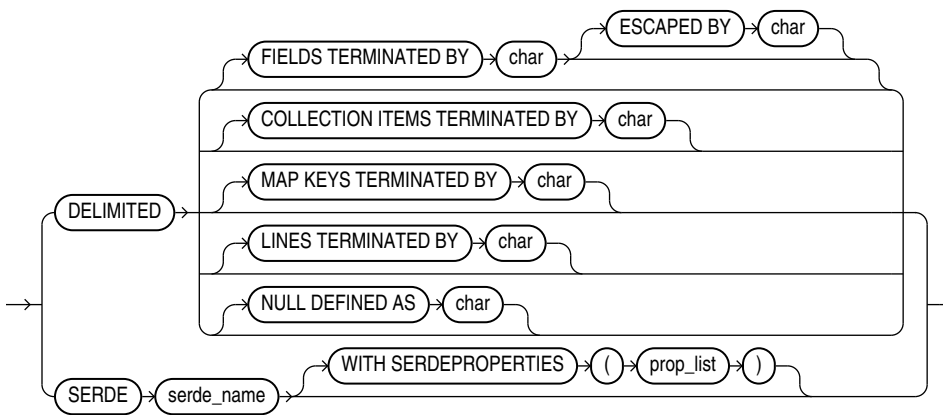
Default Value

DELIMITED

Syntax

A JSON document with the keyword-value pairs is shown in the following diagram.

rowformat:



Semantics

DELIMITED

Describes the characters used to delimit the fields in a record:

- `FIELDS TERMINATED BY`: The character that delimits every field in the record. The optional `ESCAPED BY` character precedes the delimit character when it appears within a field value.
- `COLLECTION ITEMS TERMINATED BY`: The character that marks the end of an array element. Used when a column is a collection or a nested record. In this case the resulting value will be a JSON array.
- `MAP KEYS TERMINATED BY`: The character that marks the end of an entry in a MAP field. Used when a column is a collection or a nested record. The resulting value is a JSON object.
- `LINES TERMINATED BY`: The character that marks the end of a record.

- `NULL DEFINED AS`: The character that indicates a null value.

SERDE

Identifies a SerDe that can parse the data and any properties of the SerDe that the access driver might need.

Example

This example specifies a SerDe for an Avro container file:

```
com.oracle.bigdata.rowformat:  
  SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
```

The next example specifies a SerDe for a file containing regular expressions:

```
com.oracle.bigdata.rowformat=  
  SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' \  
  WITH SERDEPROPERTIES \  
    ("input.regex" = "(\\d{6}) (\\d{5}) (.{29}) .*")
```

6.1.4.11 `com.oracle.bigdata.tablename`

The Hive parameter `com.oracle.bigdata.tablename` identifies the Hive table that contains the source data.

Default Value

DEFAULT.external_table_name

Syntax

[hive_database_name.]table_name

Semantics

The maximum length of *hive_database_name* and *table_name* is 128 UTF-8 characters (512 bytes).

hive_database_name: The Hive database where the source data resides. `DEFAULT` is the name of the initial Hive database.

table_name: The Hive table with the data. If you omit *table_name*, then `ORACLE_HIVE` searches for a Hive table with the same name as the external table. Table names are case-insensitive.

Example

This setting indicates that the source data is in a table named `ORDER_SUMMARY` in the Hive `ORDER_DB` database:

```
com.oracle.bigdata.tablename ORDER_DB.ORDER_SUMMARY
```

6.1.5 ORACLE_BIGDATA Access Parameters

There is a set of access parameters that are common to all file formats. There are also parameters that are unique to a specific file format.

Common Access Parameters

The following table lists parameters that are common to all file formats accessed through `ORACLE_BIGDATA`. The first column identifies each access parameter common to all data file types. The second column describes each parameter.

Table 6-4 Common Access Parameters

Common Access Parameter	Description
<code>com.oracle.bigdata.credential.name</code>	<p>Specifies the credential object to use when accessing data files in an object store.</p> <p>This access parameter is required for object store access. It is not needed for access to files through a directory object or for data stored in public buckets.</p> <p>The name specified for the credential must be the name of a credential object in the same schema as the owner of the table. Granting a user <code>SELECT</code> or <code>READ</code> access to this table means that credential will be used to access the table.</p> <p>Use <code>DBMS_CREDENTIAL.CREATE_CREDENTIAL</code> in the <code>DBMS_CREDENTIAL</code> PL/SQL package to create the credential object:</p> <pre>exec dbms_credential.create_credential(credential_name => 'MY_CRED',username =>'<username>', password => '<password>');</pre> <p>In the <code>CREATE TABLE</code> statement, set the value of the credential parameter to the name of the credential object.</p> <pre>com.oracle.bigdata.credential.name=MY_CRED</pre>
<code>com.oracle.bigdata.fileformat</code>	<p>Specifies the format of the file. The value of this parameter identifies the reader that will process the file. Each reader can support additional access parameters that may or may not be supported by other readers.</p> <p>Valid values: <code>parquet</code>, <code>orc</code>, <code>textfile</code>, <code>avro</code>, <code>csv</code></p> <p>Default: <code>PARQUET</code></p>
<code>com.oracle.bigdata.logopt</code>	<p>Specifies whether log messages should be written to a log file. When <code>none</code> is specified, then no logfile is created. If the value is <code>normal</code>, then log file is created when the file reader decides to write a message. It is up to the file reader to decide what is written to the log file.</p> <p>Valid values: <code>normal</code>, <code>none</code></p> <p>Default: <code>none</code>.</p>

Table 6-4 (Cont.) Common Access Parameters

Common Access Parameter	Description
com.oracle.bigdata.log.qc	Specifies the name of the log file created by the parallel query coordinator. This parameter is used only when <code>com.oracle.bigdata.log.opt</code> is set to <code>normal</code> . The valid values are the same as specified for <code>com.oracle.bigdata.log.qc</code> in ORACLE_HIVE and ORACLE_HDFS.
com.oracle.bigdata.log.exec	Specifies the name of the log file created during query execution. This value is used (and is required) only when <code>com.oracle.bigdata.log.opt</code> is set to <code>normal</code> . The valid values are the same as specified for <code>com.oracle.bigdata.log.exec</code> in ORACLE_HIVE and ORACLE_HDFS. Valid values: <code>normal</code> , <code>none</code> Default: <code>none</code> .

Avro Specific Access Parameters

In addition to common access parameters, there are some that are only valid for the Avro file format. The first column in this table identifies the access parameters specific to the Avro file format and the second column describes the parameter. There is only one Avro-specific parameter at this time.

Table 6-5 Avro Specific Access Parameters

Avro Specific Parameter	Description
com.oracle.bigdata.avro.decimaltpc	Specifies the representation of a decimal stored in the byte array. Valid values: <code>int</code> , <code>integer</code> , <code>str</code> , <code>string</code> Default: If this parameter is not used, an Avro decimal column is read assuming byte arrays store the numerical representation of the values (that is default to <code>int</code>) as the Avro specification defines.

Parquet Specific Access Parameters

Some access parameters are only valid for the Parquet file format. The first column in this table identifies the access parameters specific to the Parquet file format and the second column describes the parameter.

Table 6-6 Parquet Specific Access Parameters

Parquet Specific Access Parameter	Description
com.oracle.bigdata.prq.binary_as_string	This is a boolean property that specifies if binary is stored as a string. Valid values: <code>true</code> , <code>t</code> , <code>yes</code> , <code>y</code> , <code>l</code> , <code>false</code> , <code>f</code> , <code>no</code> , <code>n</code> , <code>0</code> Default: <code>true</code>

Table 6-6 (Cont.) Parquet Specific Access Parameters

Parquet Specific Access Parameter	Description
com.oracle.bigdata.prq.int96_as_timestamp	This is a boolean property that specifies if int96 represents a timestamp. Valid values: true, t, yes, y, l, false, f, no, n, 0 Default: true

Textfile and CSV Specific Access Parameters

The text file and csv file formats are similar to the hive text file format. It reads text and csv data from delimited files. Big Data SQL automatically detects the line terminator (either \n, \r, or \r\n). By default, it assumes the fields in the file are separated by commas and the order of the fields in the file match the order of the columns in the external table.

Example 1: CSV Data File:

This is a simple csv example. The data file has comma separated values with optional enclosing quotes.

```

-----Source csv data in t.dat
t.dat:

1,"abc",
2,xyx,

-----Create an external table over the csv source data in t.dat
CREATE TABLE t
(
  c0 number,
  c1 varchar2(20)
)
ORGANIZATION external
(
  TYPE oracle_bigdata
  DEFAULT DIRECTORY DMPDIR
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.fileformat=csv
  )
  location
  (
    't.dat'
  )
)REJECT LIMIT 1
;

-----Select data from external table
select c0, c1 from t;

```

```

C0      C1
-----
1       abc
2       xyz

```

Example 2: CSV Data File :

This example shows how to create an external table over a csv data source which has '|' as the field separator, the data file compressed with gzip, blanks as null and a date format.

```

-----The source csv data in t.dat
t.dat:

```

```

1|  |
2|Apr-99-30|

```

```

-----Create an external table over the csv data source in t.dat

```

```

CREATE TABLE t(
  c0 number,
  c1 date
)
ORGANIZATION external
(
  TYPE oracle_bigdata
  DEFAULT DIRECTORY DMPDIR
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.fileformat=textfile
    com.oracle.bigdata.compressiontype=gzip
    com.oracle.bigdata.csv.rowformat.separatorcharacter='|'
    com.oracle.bigdata.blankasnull=true
    com.oracle.bigdata.dateformat="MON-RR-DD HH:MI:SS"
  )
  location
  (
    't.dat.gz'
  )
)REJECT LIMIT 1
;

```

```

--Select csv data from external table
QL> select c0, c1 from t;

```

```

C0      C1
-----
1
2       30-APR-99

```

Example 3: Json Data File:

This is a JSON file where each row is a JSON document. The external table reaches each row. Queries use Oracle SQL JSON functions to parse the data.

```
-----Source Json docs. One for station 72 and the other for station 79.
{"station_id":"72","name":"W 52 St & 11
Ave","short_name":"6926.01","lat":40.76727216,"lon":-73.99392888,"region
_id":71,"rental_methods":
["CREDITCARD","KEY"],"capacity":39,"rental_url":"http://app.citibikenyc.com/
S6Lr/IBV092JufD?station_id=72","eightd_has_key_dispenser":false}
{"station_id":"79","name":"Franklin St & W
Broadway","short_name":"5430.08","lat":40.71911552,"lon":-74.00666661,"r
egion_id":71,"rental_methods":
["CREDITCARD","KEY"],"capacity":33,"rental_url":"http://app.citibikenyc.com/
S6Lr/IBV092JufD?station_id=79","eightd_has_key_dispenser":false}
```

```
- Create the external table over Json source
CREATE TABLE stations_ext (
  doc varchar2(4000)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS(
    com.oracle.bigdata.credential.name=MY_CRED
    com.oracle.bigdata.fileformat=textfile
    com.oracle.bigdata.csv.rowformat.fields.terminator='\n'    <-
notice the
    delimiter is a new line (i.e. not a comma). So, it will read to
the end of the line and get
    the whole document.
  )
  LOCATION ('https://swftobjectstorage.us-phoenix-1.oraclecloud.com/v1/
mybucket/stations.json')
)
REJECT LIMIT UNLIMITED;
```

```
----Select data from external table
select s.doc.station_id,
       s.doc.name,
       s.doc.rental_methods[0]    ---notice we're getting the first item in
the array of possible payments
from stations_ext;
```

Station_id	Name	rental_method (1st item in the array)
72	W52 St & 11 Ave	CREDITCARD
79	Franklin St & W Broadway	CREDITCARD

Table 6-7 Textfile and CSV Specific Access Parameters

Textfile-Specific Access Parameter	Description
<code>com.oracle.bigdata.buffersize</code>	Specifies the size of the I/O buffer used for reading the file. The value is the size of the buffer in kilobytes. Note that the buffer size is also the largest size that a record can be. If a format reader encounters a record larger than this value, it will return an error. Default: 1024
<code>com.oracle.bigdata.blankasnull</code>	When set to <code>true</code> , loads fields consisting of spaces as null. Valid values: <code>true</code> , <code>false</code> Default: <code>false</code> Example: <code>com.oracle.bigdata.blankasnull=true</code>
<code>com.oracle.bigdata.charset</code>	Specifies the charset of source files. Valid values: UTF-8 Default: UTF-8 Example: <code>com.oracle.bigdata.charset=UTF-8</code>
<code>com.oracle.bigdata.compressiontype</code>	If this parameter is specified then the code tries to decompress the data according to the compression scheme specified. Valid values: <code>gzip</code> , <code>bzip2</code> , <code>zlib</code> , <code>detect</code> Default: no compression If <code>detect</code> is specified, the format reader tries to determine which of the supported compression methods was used to compress the file.
<code>com.oracle.bigdata.conversionerrors</code>	If a row has data type conversion errors, the related columns are stored as null or the row is rejected. Valid values: <code>reject_record</code> , <code>store_null</code> Default: <code>store_null</code> Example: <code>com.oracle.bigdata.conversionerrors=reject_record</code>
<code>com.oracle.bigdata.csv.rowformat.nulldefinedas</code>	Specifies the character used to indicate the value of a field is NULL. If the parameter is not specified, then there is no value.
<code>com.oracle.bigdata.csv.rowformat.fields.terminator</code>	Specifies the character used to separate the field values. The character value must be wrapped in single-quotes, as in <code>' '</code> Default: <code>' '</code>
<code>com.oracle.bigdata.csv.rowformat.fields.escapedby</code>	Specifies the character used to escape any embedded field terminators or line terminators in the value for fields. The character value must be wrapped in single-quotes, as in <code>'\'</code> .

Table 6-7 (Cont.) Textfile and CSV Specific Access Parameters

Textfile-Specific Access Parameter	Description
<code>com.oracle.bigdata.dateformat</code>	<p>Specifies the date format in the source file. The format option <code>Auto</code> checks for the following formats.</p> <p>J, MM-DD-YYYYBC, MM-DD-YYYY, YYYYMMDD HHMISS, YMMDD HHMISS, YYYY.DDD, YYYY-MM-DD</p> <p>Default: <code>yyyy-mm-dd hh24:mi:ss</code></p> <p>Example: <code>com.oracle.bigdata.dateformat= "MON-RR-DDHH:MI:SS"</code></p>
<code>com.oracle.bigdata.fields</code>	<p>Specifies the order of fields in the data file. The values is the same as for <code>com.oracle.bigdata.fields</code> in <code>ORACLE_HDFS</code> with one exception – in this case, the data type is optional. Because the data file is text, the text file reader ignores the data types for the fields and assumes all fields are text. Since the data type is optional, this parameter can be a list of field names.</p>
<code>com.oracle.bigdata.ignoreblanklines</code>	<p>Blank lines are ignored when set to true.</p> <p>Valid values: <code>true, false</code></p> <p>Default: <code>false</code></p> <p>Example: <code>com.oracle.bigdata.ignoreblanklines=true</code></p>
<code>com.oracle.bigdata.ignoremissingcolumns</code>	<p>Missing columns are stored as null.</p> <p>Valid values: <code>true</code></p> <p>Default: <code>true</code></p> <p>Example: <code>com.oracle.bigdata.ignoremissingcolumns=true</code></p>
<code>com.oracle.bigdata.quote</code>	<p>Specifies the quote character for the fields, the quote characters are removed during loading when specified.</p> <p>Valid values: <code>character</code></p> <p>Default: <code>Null</code> meaning no quote</p> <p>Example: <code>com.oracle.bigdata.csv.rowformat.quotecharacter=' '</code></p>
<code>com.oracle.bigdata.rejectlimit</code>	<p>The operation errors out after specified number of rows are rejected. This only applies when rejecting records due to conversion errors.</p> <p>Valid values: <code>number</code></p> <p>Default: <code>0</code></p> <p>Example: <code>com.oracle.bigdata.rejectlimit=2</code></p>

Table 6-7 (Cont.) Textfile and CSV Specific Access Parameters

Textfile-Specific Access Parameter	Description
com.oracle.bigdata.removequotes	<p>Removes any quotes that are around any field in the source file.</p> <p>Valid values: true, false</p> <p>Default: false</p> <p>Example:com.oracle.bigdata.removequotes=true</p>
com.oracle.bigdata.csv.skip.header	<p>Specifies how many rows should be skipped from the start of the files.</p> <p>Valid values: number</p> <p>Default: 0 if not specified</p> <p>Example: com.oracle.bigdata.csv.skip.header=1</p>
com.oracle.bigdata.timestampformat	<p>Specifies the timestamp format in the source file. The format option AUTO checks for the following formats: YYYY-MM-DD HH:MI:SS.FF, YYYY-MM-DD HH:MI:SS.FF3, MM/DD/YYYY HH:MI:SS.FF3</p> <p>Valid values: auto</p> <p>Default: yyyy-mm-dd hh24:mi:ss.ff</p> <p>Example: com.oracle.bigdata.timestampformat="auto"</p>
com.oracle.bigdata.timestamppltzformat	<p>Specifies the timestamp with local timezone format in the source file. The format option AUTO checks for the following formats: DD Mon YYYY HH:MI:SS.FF TZR, MM/DD/YYYY HH:MI:SS.FF TZR, YYYY-MM-DD HH:MI:SS+/-TZR, YYYY-MM-DD HH:MI:SS.FF3, DD.MM.YYYY HH:MI:SS TZR</p> <p>Valid values: auto</p> <p>Default: yyyy-mm-dd hh24:mi:ss.ff</p> <p>Example: com.oracle.bigdata.timestamppltzformat="auto"</p>
com.oracle.bigdata.timestamptzformat	<p>Specifies the timestamp with timezone format in the source file. The format option AUTO checks for the following formats: DD Mon YYYY HH:MI:SS.FF TZR, MM/DD/YYYY HH:MI:SS.FF TZR, YYYY-MM-DD HH:MI:SS+/-TZR, YYYY-MM-DD HH:MI:SS.FF3, DD.MM.YYYY HH:MI:SS TZR</p> <p>Valid values: auto</p> <p>Default: yyy-mm-dd hh24:mi:ss.ff</p> <p>Example: com.oracle.bigdata.timestamptzformat="auto"</p>

Table 6-7 (Cont.) Textfile and CSV Specific Access Parameters

Textfile-Specific Access Parameter	Description
com.oracle.bigdata.trimspaces	<p>Specifies how the leading and trailing spaces of the fields are trimmed.</p> <p>Valid values: rtrim, ltrim, notrim, ltrim, ldrtrim</p> <p>Default: notrim</p> <p>Example: com.oracle.bigdata.trimspaces=rtrim</p>
com.oracle.bigdata.truncatecol	<p>If the data in the file is too long for a field, then this option truncates the value of the field rather than rejecting the row or setting the field to NULL.</p> <p>Valid values: true, false</p> <p>Default: false</p> <p>Example: com.oracle.bigdata.truncatecol=true</p>

6.2 Static Data Dictionary Views for Hive

The Oracle Database catalog contains several static data dictionary views for Hive tables. You can query these data dictionary views to discover information about the Hive tables that you can access.

For you to access any Hive databases from Oracle Database, you must have read privileges on the `ORACLE_BIGDATA_CONFIG` directory object.

- [ALL_HIVE_DATABASES](#)
- [ALL_HIVE_TABLES](#)
- [ALL_HIVE_COLUMNS](#)
- [DBA_HIVE_DATABASES](#)
- [DBA_HIVE_TABLES](#)
- [DBA_HIVE_COLUMNS](#)
- [USER_HIVE_DATABASES](#)
- [USER_HIVE_TABLES](#)
- [USER_HIVE_COLUMNS](#)

6.2.1 ALL_HIVE_DATABASES

`ALL_HIVE_DATABASES` describes all databases in the Hive metastore accessible to the current user.

Related Views

- `DBA_HIVE_DATABASES` describes all the databases in the Hive metastore.
- `USER_HIVE_DATABASES` describes the databases in the Hive metastore owned by the current user.

Column	Datatype	NULL	Description
CLUSTER_ID	VARCHAR2(4000)	NOT NULL	Hadoop cluster where the Hive metastore is located
DATABASE_NAME	VARCHAR2(4000)	NOT NULL	Hive database name
DESCRIPTION	VARCHAR2(4000)		Hive database description
DB_LOCATION	VARCHAR2(4000)	NOT NULL	
HIVE_URI	VARCHAR2(4000)		Hive database URI

 **See Also:**

- ["DBA_HIVE_DATABASES"](#)
- ["USER_HIVE_DATABASES"](#)

6.2.2 ALL_HIVE_TABLES

ALL_HIVE_TABLES describes all tables in the Hive metastore accessible to the current user.

The Oracle Big Data SQL configuration must identify the default Hive database for the current user. The current user must also have READ privileges on the ORA_BIGSQL_CONFIG database directory. See ["About the bigdata_config Directory"](#).

Related Views

- DBA_HIVE_TABLES describes all tables in the Hive metastore.
- USER_HIVE_TABLES describes the tables in the database owned by the current user in the Hive metastore.

Column	Datatype	NULL	Description
CLUSTER_ID	VARCHAR2(4000)	NOT NULL	Hadoop cluster where the Hive metastore is located
DATABASE_NAME	VARCHAR2(4000)	NOT NULL	Name of the Hive database
TABLE_NAME	VARCHAR2(4000)	NOT NULL	Name of the Hive table
LOCATION	VARCHAR2(4000)		
NO_OF_COLS	NUMBER		Number of columns in the Hive table
CREATION_TIME	DATE		Time when the table was created
LAST_ACCESSED_TIME	DATE		Time of most recent access
OWNER	VARCHAR2(4000)		Owner of the Hive table
TABLE_TYPE	VARCHAR2(4000)	NOT NULL	Type of Hive table, such as external or managed
PARTITIONED	VARCHAR2(4000)		Whether the table is partitioned (YES) or not (NO)
NO_OF_PART_KEYS	NUMBER		Number of partitions

Column	Datatype	NULL	Description
INPUT_FORMAT	VARCHAR2(4000)		Input format
OUTPUT_FORMAT	VARCHAR2(4000)		Output format
SERIALIZATION	VARCHAR2(4000)		SerDe serialization information
COMPRESSED	NUMBER		Whether the table is compressed (YES) or not (NO)
HIVE_URI	VARCHAR2(4000)		Hive database URI

 **See Also:**

- ["DBA_HIVE_TABLES"](#)
- ["USER_HIVE_TABLES"](#)

6.2.3 ALL_HIVE_COLUMNS

ALL_HIVE_COLUMNS describes the columns of all Hive tables accessible to the current user.

The Oracle Big Data SQL configuration must identify the default Hive database for the current user. The current user must also have READ privileges on the ORA_BIGSQL_CONFIG database directory. See ["About the bigdata_config Directory"](#).

Related Views

- DBA_HIVE_COLUMNS describes the columns of all tables in the Hive metastore.
- USER_HIVE_COLUMNS describes the columns of the tables in the Hive database owned by the current user.

Column	Datatype	NULL	Description
CLUSTER_ID	VARCHAR2(4000)	NOT NULL	Hadoop cluster where the Hive metastore is located
DATABASE_NAME	VARCHAR2(4000)	NOT NULL	Name of the Hive database; if blank, then the default database
TABLE_NAME	VARCHAR2(4000)	NOT NULL	Name of the Hive table
COLUMN_NAME	VARCHAR2(4000)	NOT NULL	Name of the Hive column
HIVE_COLUMN_TYPE	VARCHAR2(4000)	NOT NULL	Data type of the Hive column
ORACLE_COLUMN_TYPE	VARCHAR2(4000)	NOT NULL	Oracle data type equivalent to Hive data type
LOCATION	VARCHAR2(4000)		
OWNER	VARCHAR2(4000)		Owner of the Hive table
CREATION_TIME	DATE		Time when the table was created
HIVE_URI	VARCHAR2(4000)		Hive database URI

 **See Also:**

- ["DBA_HIVE_COLUMNS"](#)
- ["USER_HIVE_COLUMNS"](#)

6.2.4 DBA_HIVE_DATABASES

DBA_HIVE_DATABASES describes all the databases in the Hive metastore. Its columns are the same as those in ALL_HIVE_DATABASES.

 **See Also:**

["ALL_HIVE_DATABASES"](#)

6.2.5 DBA_HIVE_TABLES

DBA_HIVE_TABLES describes all tables in the Hive metastore. Its columns are the same as those in ALL_HIVE_TABLES.

The Oracle Big Data SQL configuration must identify the default Hive database for the current user. See ["About the bigdata_config Directory"](#).

 **See Also:**

["ALL_HIVE_TABLES"](#)

6.2.6 DBA_HIVE_COLUMNS

DBA_HIVE_COLUMNS describes the columns of all tables in the Hive metastore. Its columns are the same as those in ALL_HIVE_COLUMNS.

 **See Also:**

["ALL_HIVE_COLUMNS"](#)

6.2.7 USER_HIVE_DATABASES

USER_HIVE_DATABASES describes the databases in the Hive metastore owned by the current user. Its columns (except for OWNER) are the same as those in ALL_HIVE_DATABASES.

**See Also:**

["ALL_HIVE_DATABASES"](#)

6.2.8 USER_HIVE_TABLES

USER_HIVE_TABLES describes the tables in the database owned by the current user in the Hive metastore. Its columns (except for OWNER) are the same as those in ALL_HIVE_TABLES.

The Oracle Big Data SQL configuration must identify the default Hive database for the current user. The current user must also have READ privileges on the ORA_BIGSQL_CONFIG database directory. See ["About the bigdata_config Directory"](#).

**See Also:**

["ALL_HIVE_TABLES"](#)

6.2.9 USER_HIVE_COLUMNS

USER_HIVE_COLUMNS describes the columns of the tables in the Hive database owned by the current user. Its columns (except for OWNER) are the same as those in ALL_HIVE_COLUMNS.

The Oracle Big Data SQL configuration must identify the default Hive database for the current user. The current user must also have READ privileges on the ORA_BIGSQL_CONFIG database directory. See ["About the bigdata_config Directory"](#).

**See Also:**

["ALL_HIVE_COLUMNS"](#)

6.3 DBMS_BDSQL PL/SQL Package

The DBMS_BDSQL PL/SQL package contains procedures to add and remove a user map.

This appendix contains the following sections:

- [ADD_USER_MAP](#)
- [REMOVE_USER_MAP](#)
- [Multi-User Authorization Security Table](#)

In previous releases of Oracle Big Data SQL, all queries against Hadoop and Hive data are executed as the oracle user and there is no option to change users. Although oracle is still the underlying user in all cases, you can now use Multi-User

Authorization (based on Hadoop Secure Impersonation) to direct the oracle account to execute tasks on behalf of other designated users. This enables HDFS data access based on the user that is currently executing the query, rather than the singular `oracle` user.

The `DBMS_BDSQL` package enables you to provide rules for identifying the currently connected user and to map the connected user to the user that is impersonated. Because there are numerous ways in which users can connect to Oracle Database, this user may be a database user, a user sourced from LDAP, from Kerberos, and so forth. Authorization rules on the files apply for that user and audits will reflect that user as well.

 **Note:**

Grant the new `BDSQL_ADMIN` role to designated administrators in order to allow them to invoke these procedures.

6.3.1 ADD_USER_MAP

Use the `ADD_USER_MAP` procedure to specify the rules for identifying the actual user who is running the query.

At query execution time, the database performs a lookup on the `BDSQL_USER_MAP` table to determine the current database user (`current_database_user`). It then uses the `syscontext_namespace` and `syscontext_parm_hadoop_user` parameters to identify the actual user.

Syntax

```
procedure ADD_USER_MAP (
    cluster_name           IN VARCHAR2 DEFAULT '[DEFAULT]',
    current_database_user  IN VARCHAR2 NOT NULL,
    syscontext_namespace  IN VARCHAR2 DEFAULT NULL,
    syscontext_parm_hadoop_user IN VARCHAR2 NOT NULL
);
```

Table 6-8 `ADD_USER_MAP` Parameters

Parameter	Description
<code>cluster_name</code>	The name of the Hadoop cluster where the map will be applied. <code>[DEFAULT]</code> as cluster name designates the default cluster.
<code>current_database_user</code>	The current effective database user. This is what Oracle uses to check for authority. A value of <code>'*</code> indicates that this row to be used if no other rows fit the criteria. There is no default and the value may not be <code>NULL</code> .

Table 6-8 (Cont.) ADD_USER_MAP Parameters

Parameter	Description
<code>syscontext_namespace</code>	Note that for the Oracle <code>USERENV</code> namespace, the only allowed values are <code>GLOBAL_UID</code> , <code>CLIENT_IDENTIFIER</code> , and <code>AUTHENTICATED_IDENTITY</code> . If your Oracle Database installation uses Kerberos credentials, SSL, Active Directory, or LDAP for authentication, it is likely that your Hadoop system uses the same authentication framework. In that case, <code>AUTHENTICATED_IDENTITY</code> must be specified. This identifier only includes the username. The domain segment of the credential is truncated, as is the cluster name if included. For example, the username <code>dirkrb</code> will be used for authorization on the Hadoop cluster as the authenticated identity of <code>dirkrb@HQ.TEST1.DBSEC2008.COM</code> .
<code>syscontext_parm_hadoop_user</code>	The Hadoop user that will impersonate the current database user.

 **Note:**

The values for `current_database_user` and `syscontext_parm_hadoop_user` can be the single asterisk character (*) or any string that meets the requirements of Oracle `simple_sql_name` assertion:

- The name must begin with an alphabetic character. It may contain alphanumeric characters as well as the characters `_`, `$`, and `#` in the second and subsequent character positions.
- Quoted SQL names are also allowed.
- Quoted names must be enclosed in double quotes.
- Quoted names allow any characters between the quotes.
- Quotes inside the name are represented by two quote characters in a row, for example, "a name with "" inside" is a valid quoted name.
- The input parameter may have any number of leading and/or trailing white space characters.

6.3.2 REMOVE_USER_MAP

Use `REMOVE_USER_MAP` to remove a row from `BDSQL_USER_MAP` table. This disassociates a specific Oracle Database user from specific Hadoop user.

Syntax

```
procedure REMOVE_USER_MAP (
    cluster_name IN VARCHAR2 DEFAULT '[DEFAULT]',
```

```
current_database_user IN VARCHAR2 NOT NULL
);
```

See Also:

The reference page for [ADD_USER_MAP](#) describes the `cluster_name` and `current_database_user` parameters.

6.3.3 Multi-User Authorization Security Table

`SYS.BDSQL_USER_MAP` is the multi-user authorization security table.

Use the procedures `ADD_USER_MAP` and `REMOVE_USER_MAP` to update this table.

The primary key is (`cluster_name`, `current_database_user`).

Table 6-9 `SYS.BDSQL_USER_MAP`

Column	Datatype	Description
<code>cluster_name</code>	<code>varchar2</code>	Name of the Hadoop cluster. The default is [DEFAULT].
<code>current_database_user</code>	<code>varchar2</code>	The current effective database user (no default, not NULL). Oracle uses this column to check for the authorization rule that corresponds to the given Oracle Database user. A value of '*' in a row is a directive to use this row if no other rows fit the criteria.
<code>syscontext_namespace</code>	<code>varchar2</code>	This is the optional specification for the Oracle <code>SYS_CONTEXT</code> namespace. If customer security is set up. Note that for the Oracle <code>USERENV</code> namespace, the only allowed values are: 'GLOBAL_UID', 'CLIENT_IDENTIFIER', 'AUTHENTICATED_IDENTITY'.
<code>syscontext_parameter_hadoop_user</code>	<code>varchar2</code>	This column value has alternate interpretations. <ul style="list-style-type: none"> If <code>syscontext_namespace</code> has a value, then <code>syscontext_parameter_hadoop_user</code> refers to the parameter that is specific to <code>syscontext_namespace</code>. However, when the value is '*', this is a directive to use the value of <code>current_database_user</code> for impersonation. The <code>syscontext_namespace</code> column must be NULL in this case. If <code>syscontext_namespace</code> is NULL, then <code>syscontext_parameter_hadoop_user</code> contains the Hadoop user who is impersonated prior to HDFS files access.

Here a customer is using Active Directory, Kerberos, SSL, or LDAP for logon authentication against the database. `AUTHENTICATED_IDENTITY` is specified in this case because customer uses the same Active Directory framework for Hadoop user management.

The example below is similar to running the following SQL query for the currently connected user:

```
select sys_context('USERENV', 'AUTHENTICATED_IDENTITY') from dual;
```

In this example, only the username (without the “@<domain>” segment) is used for authorization on the Hadoop cluster. There may also be cases where the format of AUTHENTICATED_IDENTITY is <username>/<cluster>@<domain_name>.

cluster_name	current_database_u ser	syscontext_namesp ace	syscontext_parm_ha doop_user
[DEFAULT]	*	USERENV	AUTHENTICATED_ID ENTITY

In this example, “HRAPP” is an HR Application that always connects to the database using the HRAPP database user and then programmatically sets the application user through the DBMS_SESSION.SET_IDENTIFIER procedure. There are number of “lightweight” users who are designated with CLIENT_IDENTIFIER (as in sys_context('USERENV', 'CLIENT_IDENTIFIER') [DEFAULT] * USERENV GLOBAL_UID, which is similar to running select sys_context('USERENV', 'CLIENT_IDENTIFIER') from dual;) .

The current database has other effective users who are enterprise users with logons managed by Oracle Internet Directory for Enterprise User Security. In these cases, the GLOBAL_UID is used for Hadoop impersonation.

cluster_name	current_database_u ser	syscontext_namesp ace	syscontext_parm_ha doop_user
[DEFAULT]	HRAPP	USERENV	CLIENT_IDENTIFIER
[DEFAULT]	*	USERENV	GLOBAL_UID

In this example, BIAPP is a business intelligence application whose own context is its username. For customers using the application, their designated ID is used for Hadoop access. In other words, when the effective current user is 'BIAPP', we use sys_context('BIVPD', 'USERID') for the impersonation. For the rest of the users, we simply designate [DEFAULT] * * in order use their current database username for the impersonation.

cluster_name	current_database_u ser	syscontext_namesp ace	syscontext_parm_ha doop_user
[DEFAULT]	BIAPP	BIVPD	USERID
[DEFAULT]	*		*

In this example, the Oracle username SCOTT is impersonated by the hdpusr1 Hadoop user for HDFS access. The user ADAM is impersonated by hdpusr2 for HDFS access.

All other users have more limited access, so we use a syscontext_namespace value of 'lowprivuser' to designate these users.

cluster_name	current_database_u ser	syscontext_namesp ace	syscontext_parm_ha doop_user
hadoop_cl_1	SCOTT		hdpusr1
hadoop_cl_1	ADAM	lowprivuser	hdpusr2
hadoop_cl_1	*		

6.4 DBMS_BDSQS_ADMIN PL/SQL Package

This package contains procedures to manage database and Kerberos users for the Hadoop cluster.

- [#unique_187](#)
- [#unique_188](#)
- [#unique_189](#)
- [#unique_190](#)

6.5 DBMS_HADOOP PL/SQL Package

The DBMS_HADOOP package contains a function to generate the CREATE EXTERNAL TABLE DDL for a Hive table:

- [CREATE_EXTDDL_FOR_HIVE](#)

6.5.1 CREATE_EXTDDL_FOR_HIVE

This function returns a SQL CREATE TABLE ORGANIZATION EXTERNAL statement for a Hive table. It uses the ORACLE_HIVE access driver.

Syntax

```
DBMS_HADOOP.CREATE_EXTDDL_FOR_HIVE (
  cluster_id      IN   VARCHAR2,
  db_name         IN   VARCHAR2  := NULL,
  hive_table_name IN   VARCHAR2,
  hive_partition IN   BOOLEAN,
  table_name      IN   VARCHAR2  := NULL,
  perform_ddl    IN   BOOLEAN   DEFAULT FALSE,
  text_of_ddl    OUT  VARCHAR2
);
```

Parameters

Table 6-10 CREATE_EXTDDL_FOR_HIVE Function Parameters

Parameter	Description
cluster_id	Hadoop cluster where the Hive metastore is located
db_name	Name of the Hive database
hive_table_name	Name of the Hive table
hive_partition	Whether the table is partitioned (TRUE) or not (FALSE)
table_name	Name of the Oracle external table to be created. It cannot already exist.
perform_ddl	Whether to execute the generated CREATE TABLE statement (TRUE) or just return the text of the command (FALSE). Do not execute the command automatically if you want to review or modify it.

Table 6-10 (Cont.) CREATE_EXTDDL_FOR_HIVE Function Parameters

Parameter	Description
text_of_ddl	The generated CREATE TABLE ORGANIZATION EXTERNAL statement.

Usage Notes

The Oracle Database system must be configured for Oracle Big Data SQL. See "[About Oracle Big Data SQL on the Database Server \(Oracle Exadata Machine or Other\)](#)".

The data type conversions are based on the default mappings between Hive data types and Oracle data types. See "[Hive to Oracle Data Type Conversions](#)".

6.5.1.1 Example

The following query returns the CREATE EXTERNAL TABLE DDL for my_hive_table from the default Hive database. The connection to Hive is established using the configuration files in the ORACLE_BIGDATA_CONFIG directory, which identify the location of the HADOOP1 cluster.

```
DECLARE
  DDLtxt VARCHAR2(4000);
BEGIN
  dbms_hadoop.create_extddl_for_hive(
    CLUSTER_ID=>'hadoop1',
    DB_NAME=>'default',
    HIVE_TABLE_NAME=>'my_hive_table',
    HIVE_PARTITION=>FALSE,
    TABLE_NAME=>'my_xt_oracle',
    PERFORM_DDL=>FALSE,
    TEXT_OF_DDL=>DDLtxt
  );
  dbms_output.put_line(DDLtxt);
END;
/
```

The query returns the text of the following SQL command:

```
CREATE TABLE my_xt_oracle
(
  c0 VARCHAR2(4000),
  c1 VARCHAR2(4000),
  c2 VARCHAR2(4000),
  c3 VARCHAR2(4000)
  ORGANIZATION EXTERNAL
    (TYPE ORACLE_HIVE
      DEFAULT DIRECTORY DEFAULT_DIR
      ACCESS PARAMETERS (
        com.oracle.bigdata.cluster=hadoop1
        com.oracle.bigdata.tablename=default.my_hive_table
      )
    )
  PARALLEL 2 REJECT LIMIT UNLIMITED
```

Appendices

This appendix includes appendices relevant for Oracle Big Data SQL.

- [Manual Steps for Using Copy to Hadoop for Staged Copies](#)
- [Using Copy to Hadoop With Direct Copy](#)
- [Using mtactl to Manage the MTA extproc](#)
- [Diagnostic Tips and Details](#)
- [Oracle Big Data SQL Software Accessibility Recommendations](#)

A

Manual Steps for Using Copy to Hadoop for Staged Copies

To manually copy data from Oracle Database to Hadoop using Copy to Hadoop, take the following steps:

1. On the Oracle Database server, connect to Oracle Database and generate Data Pump format files containing the table data and metadata.
See "[Generating the Data Pump Files](#)".
2. Copy the files to HDFS on the Hadoop cluster.
See "[Copying the Files to HDFS](#)".
3. Connect to Apache Hive and create an external table from the files.
See "[Creating a Hive Table](#)".
4. Query this Hive table the same as you would any other Hive table.

A.1 Generating the Data Pump Files

The SQL `CREATE TABLE` statement has a clause specifically for creating external tables, in which you specify the `ORACLE_DATAPUMP` access driver. The information that you provide in this clause enables the access driver to generate a Data Pump format file that contains the data and metadata from the Oracle database table.

This section contains the following topics:

- [About Data Pump Format Files](#)
- [Identifying the Target Directory](#)
- [About the CREATE TABLE Syntax](#)

A.1.1 About Data Pump Format Files

Data Pump files are typically used to move data and metadata from one database to another. Copy to Hadoop uses this file format to copy data from an Oracle database to HDFS.

To generate Data Pump format files, you create an external table from an existing Oracle table. An **external table** in Oracle Database is an object that identifies and describes the location of data outside of a database. External tables use **access drivers** to parse and format the data. For Copy to Hadoop, you use the `ORACLE_DATAPUMP` access driver. It copies the data and metadata from internal Oracle tables and populates the Data Pump format files of the external table.

A.1.2 Identifying the Target Directory

You must have read and write access to a database directory in Oracle Database. Only Oracle Database users with the `CREATE ANY DIRECTORY` system privilege can create directories.

This example creates a database directory named `EXPORTDIR` that points to the `/exportdir` directory on the Oracle Database server (Oracle Exadata Database Machine or other):

```
SQL> CREATE DIRECTORY exportdir AS '/exportdir';
```

A.1.3 About the CREATE TABLE Syntax

The following is the basic syntax of the `CREATE TABLE` statement for Data Pump format files:

```
CREATE TABLE table_name
  ORGANIZATION EXTERNAL (
    TYPE oracle_datapump
    DEFAULT DIRECTORY database_directory
    LOCATION ('filename1.dmp', 'filename2.dmp' ...)
  ) PARALLEL n
  AS SELECT * FROM tablename;
```

DEFAULT DIRECTORY

Identifies the database directory that you created for this purpose. See "[Identifying the Target Directory](#)".

LOCATION

Lists the names of the Data Pump files to be created. The number of names should match the degree of parallelism (DOP) specified by the `PARALLEL` clause. Otherwise, the DOP drops to the number of files.

The number of files and the degree of parallelism affect the performance of Oracle Database when generating the Data Pump format files. They do not affect querying performance in Hive.

PARALLEL

Sets the degree of parallelism (DOP). Use the maximum number that your Oracle DBA permits you to use. By default the DOP is 1, which is serial processing. Larger numbers enable parallel processing.

AS SELECT

Use the full SQL `SELECT` syntax for this clause. It is not restricted. The *tablename* identifies the Oracle table to be copied to HDFS.



See Also:

For descriptions of these parameters:

- [Oracle Database SQL Language Reference](#)
- [Oracle Database Utilities](#)

A.2 Copying the Files to HDFS

The Oracle Big Data SQL installation installs Hadoop client files on the Oracle Database server (Oracle Exadata Database Machine or other). The Hadoop client installation enables you to use Hadoop commands to copy the Data Pump files to HDFS. You must have write privileges on the HDFS directory.

To copy the `dmp` files into HDFS, use the `hadoop fs -put` command. This example copies the files into the HDFS `customers` directory owned by the `oracle` user:

```
$ hadoop fs -put customers*.dmp /user/oracle/customers
```

A.3 Creating a Hive Table

To provide access to the data in the Data Pump files, you create a Hive external table over the Data Pump files. Copy to Hadoop provides SerDes that enable Hive to read the files. These SerDes are read only, so you cannot use them to write to the files.

See Also:

Apache Hive Language Manual DDL at

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create/Drop/TruncateTable>

A.3.1 About Hive External Tables

For external tables, Hive loads the table metadata into its metastore. The data remains in its original location, which you identify in the `LOCATION` clause. If you drop an external table using a HiveQL `DROP TABLE` statement, then only the metadata is discarded, while the external data remains unchanged. In this respect, Hive handles external tables in fundamentally the same way as Oracle Database.

External tables support data sources that are shared by multiple programs. In this case, you use Oracle Database to update the data and then generate a new file. You can replace the old HDFS files with the updated files, while leaving the Hive metadata intact.

The following is the basic syntax of a Hive `CREATE TABLE` statement for creating a Hive external table for use with a Data Pump format file:

```
CREATE EXTERNAL TABLE tablename
ROW FORMAT
  SERDE 'oracle.hadoop.hive.datapump.DPSerde'
STORED AS
  INPUTFORMAT 'oracle.hadoop.hive.datapump.DPInputFormat'
  OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 'hdfs_directory'
```

A.4 Example Using the Sample Schemas

This example shows all steps in the process of creating a Hive table from an Oracle table using Copy to Hadoop.

A.4.1 About the Sample Data

The Oracle tables are from the Sales History (SH) sample schema. The `CUSTOMERS` table provides extensive information about individual customers, including names, addresses, telephone numbers, birth dates, and credit limits. The `COUNTRIES` table provides a list of countries, and identifies regions and subregions.

This query shows a small selection of data in the `CUSTOMERS` table:

```
SELECT cust_first_name first_name,  
       cust_last_name last_name,  
       cust_gender gender,  
       cust_year_of_birth birth  
FROM customers  
ORDER BY cust_city, last_name  
FETCH FIRST 10 ROWS ONLY;
```

The query returns the following rows:

FIRST_NAME	LAST_NAME	GENDER	BIRTH
Lise	Abbey	F	1963
Lotus	Alden	M	1958
Emmanuel	Aubrey	M	1933
Phil	Ball	M	1956
Valentina	Bardwell	F	1965
Lolita	Barkley	F	1966
Heloise	Barnes	M	1980
Royden	Barrett	M	1937
Gilbert	Braun	M	1984
Portia	Capp	F	1948

To reproduce this example, install the sample schemas in Oracle Database and connect as the `SH` user.

A.4.2 Creating the EXPDIR Database Directory

These SQL statements create a local database directory named `EXPDIR` and grant access to the `SH` user:

```
SQL> CREATE DIRECTORY expdir AS '/expdir';  
Directory created.  
SQL> GRANT READ, WRITE ON DIRECTORY expdir TO SH;  
Grant succeeded.
```

A.4.3 Creating Data Pump Format Files for Customer Data

The following examples show how to create the Data Pump files and check their contents.

Copy to Hadoop supports only the syntax shown in the examples. Data pump files created with the Export utility or Oracle Data Pump are not compatible.

A.4.3.1 CREATE TABLE Example With a Simple SELECT Statement

This example shows a very simple SQL command for creating a Data Pump format file from the CUSTOMERS table. It selects the entire table and generates a single output file named customers.dmp in the local /expdir directory.

```
CREATE TABLE export_customers
  ORGANIZATION EXTERNAL
  (
    TYPE oracle_datapump
    DEFAULT DIRECTORY expdir
    LOCATION('customers.dmp')
  )
AS SELECT * FROM customers;
```

A.4.3.2 CREATE TABLE Example With a More Complex SQL SELECT Statement

The next example shows more complexity in the syntax. It joins the CUSTOMERS and COUNTRIES tables on the COUNTRY_ID columns to provide the country names. It also limits the rows to customers in the Americas. The command generates two output files in parallel, named americas1.dmp and americas2.dmp, in the local /expdir directory.

```
CREATE TABLE export_americas
  ORGANIZATION EXTERNAL
  (
    TYPE oracle_datapump
    DEFAULT DIRECTORY expdir
    LOCATION('americas1.dmp', 'americas2.dmp')
  )
  PARALLEL 2
AS SELECT a.cust_first_name first_name,
  a.cust_last_name last_name,
  a.cust_gender gender,
  a.cust_year_of_birth birth,
  a.cust_email email,
  a.cust_postal_code postal_code,
  b.country_name country
FROM customers a,
  countries b
WHERE a.country_id=b.country_id AND
  b.country_region='Americas'
ORDER BY a.country_id, a.cust_postal_code;
```

A.4.4 Verifying the Contents of the Data Files

You can check the content of the output data files before copying them to Hadoop. The previous CREATE TABLE statement created an external table named EXPORT_AMERICAS, which you can describe and query the same as any other table.

The DESCRIBE statement shows the selection of columns and the modified names:

```
SQL> DESCRIBE export_americas;
Name                               Null?      Type
-----
```

```

FIRST_NAME          NOT NULL VARCHAR2(20)
LAST_NAME           NOT NULL VARCHAR2(40)
GENDER              NOT NULL CHAR(1)
BIRTH               NOT NULL NUMBER(4)
EMAIL               VARCHAR2(50)
POSTAL_CODE         NOT NULL VARCHAR2(10)
COUNTRY             NOT NULL VARCHAR2(40)

```

A **SELECT** statement like the following shows a sample of the data:

```

SELECT first_name, last_name, gender, birth, country
FROM export_americas
WHERE birth > 1985
ORDER BY last_name
FETCH FIRST 5 ROWS ONLY;

```

FIRST_NAME	LAST_NAME	GENDER	BIRTH	COUNTRY
Opal	Aaron	M	1990	United States of America
KaKit	Abeles	M	1986	United States of America
Mitchel	Alambarati	M	1987	Canada
Jade	Anderson	M	1986	United States of America
Roderica	Austin	M	1986	United States of America

A.4.5 Copying the Files into Hadoop

The following commands list the files in the local `expdir` directory, create a Hadoop subdirectory named `customers`, and copy the files to it. The user is connected to the Hadoop cluster (Oracle Big Data Appliance or other) as the `oracle` user.

```

$ cd /expdir
$ ls americas*.dmp
americas1.dmp  americas2.dmp
$ hadoop fs -mkdir customers
$ hadoop fs -put *.dmp customers
$ hadoop fs -ls customers
Found 2 items
-rw-r--r--  1 oracle oracle    798720 2014-10-13 17:04 customers/americas1.dmp
-rw-r--r--  1 oracle oracle    954368 2014-10-13 17:04 customers/americas2.dmp

```

A.4.6 Creating a Hive External Table

This HiveQL statement creates an external table using the Copy to Hadoop SerDes. The `LOCATION` clause identifies the full path to the Hadoop directory containing the Data Pump files:

```

CREATE EXTERNAL TABLE customers
ROW FORMAT SERDE 'oracle.hadoop.hive.datapump.DPSerDe'
STORED AS
INPUTFORMAT 'oracle.hadoop.hive.datapump.DPInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION '/user/oracle/customers';

```

The `DESCRIBE` command shows the columns of the `CUSTOMERS` external table.

```

hive> DESCRIBE customers;
OK
first_name          varchar(20)          from deserializer
last_name           varchar(40)          from deserializer

```

gender	char(1)	from deserializer
birth	int	from deserializer
email	varchar(50)	from deserializer
postal_code	varchar(10)	from deserializer
country	varchar(40)	from deserializer

B

Using Copy to Hadoop With Direct Copy

Copy to Hadoop with the direct copy option copies data from an Oracle Database table directly to Oracle Datapump files stored in HDFS.

Copy to Hadoop simplifies the data copy because it does not require intermediate storage on the database server. The rest of the steps needed to make the data accessible to the Hadoop ecosystem such as creating Hive external table to access the copied data and running Hive queries is common to both copy options. (stage and direct).

The intended audience for this section is power users of Hadoop with specialized requirements . All other users should use the Oracle Shell for Hadoop Loader (OHSH) CLI for Copy to Hadoop operations. See [Using Oracle Shell for Hadoop Loaders With Copy to Hadoop](#).

B.1 Manual Steps for Using Copy to Hadoop for Direct Copies

Follow these steps.

Getting Started

1. First confirm that Copy to Hadoop is installed and configured.
2. Ensure that the user account has sufficient privileges to copy a database table. (See [Table Access Requirements for Copy to Hadoop](#).)
3. Make sure that the table contains supported column types. (See [Column Mappings and Data Type Conversions in Copy to Hadoop](#).)
4. Log in to either a node in the Hadoop cluster or a system set up as a Hadoop client for the cluster.
5. If you are connecting to a secure cluster then run `kinit` to authenticate the user.
6. Run the Copy To Hadoop job using direct copy. See “Running the Copy to Hadoop Job for Direct Copy” below.
7. After the job succeeds, check the `jobReport.log` file in the `_ctoh` subdirectory of the output directory in HDFS. Check that the RowCount listed in the log file is equal to the number of rows in the database table.
8. Connect to Apache Hive and create an external table from the Data Pump files. (See [Creating a Hive Table](#).)

Running the Copy to Hadoop Job (Using Direct Copy)

1. Set the environment variables required by Copy to Hadoop.

Locate the installation directory of Copy to Hadoop and set the `CP2HADOOP_HOME` Bash shell variable. For example:

```
# export CP2HADOOP_HOME="/opt/oracle/orahivedp-3.1.0"
```

Add the Copy to Hadoop JARs to `HADOOP_CLASSPATH`. For example:

```
# export HADOOP_CLASSPATH="${CP2HADOOP_HOME}/jlib/*:$
{HADOOP_CLASSPATH}"
```

 **Tip:**

When using Copy to Hadoop, you should always list `${CP2HADOOP_HOME}/jlib/*` first in `HADOOP_CLASSPATH`. Another way to avoid JAR conflicts is to define an appropriately ordered `HADOOP_CLASSPATH` within a script that uses it.

2. Run the Job.

This is the command syntax:

```
# hadoop jar ${CP2HADOOP_HOME}/jlib/orahivedp.jar
oracle.hadoop.ctoh.CtohDriver \
[-D <configuration-property>=<configuration-value>]+
```

Example 1: Running the Job on a Secure Cluster Using Oracle Wallet

```
# hadoop jar ${CP2HADOOP_HOME}/jlib/orahivedp.jar
oracle.hadoop.ctoh.CtohDriver \
-D oracle.hadoop.ctoh.connection.tnsEntry=<my-oracle-tns-entry> \
-D oracle.hadoop.ctoh.connection.walletLoc=<local-oracle-wallet-dir> \
-D oracle.hadoop.ctoh.connection.tnsAdmin=<local-oracle-wallet-dir> \
-D oracle.hadoop.ctoh.connection.clusterWalletLoc=<oracle-wallet-dir-on-
hadoop-cluster> \
-D oracle.hadoop.ctoh.connection.clusterTnsAdmin=<oracle-wallet-dir-on-
hadoop-cluster> \
-D mapreduce.output.fileoutputformat.outputdir=<mytab-hdfs-output-dir> \
-D oracle.hadoop.ctoh.splitterType="BLOCK_SPLITTER" \
-D oracle.hadoop.ctoh.table=<dbSchema.dbTable> \
-D oracle.hadoop.ctoh.maxSplits=10
```

Example 2: Running the Job on A Unsecured Hadoop Cluster (for Demo Purposes Only)

```
# hadoop jar ${CP2HADOOP_HOME}/jlib/orahivedp.jar
oracle.hadoop.ctoh.CtohDriver \
-D oracle.hadoop.ctoh.jdbc.url="jdbc:oracle:thin:@myhost:1521/
myservice" \
-D oracle.hadoop.ctoh.connection.username="username" \
-D oracle.hadoop.ctoh.connection.password="password" \
-D mapreduce.output.fileoutputformat.outputdir="mytable_output_dir" \
-D oracle.hadoop.ctoh.splitterType="BLOCK_SPLITTER" \
```

```
-D oracle.hadoop.ctoh.table="otherUser.mytable" \  
-D oracle.hadoop.ctoh.maxSplits=10
```

Performance Tuning Tips

You can control the degree of parallelism of the Copy to Hadoop job by specifying the number of map processes using the `oracle.hadoop.ctoh.maxSplits` property. The higher the number of map processes, the higher the parallelism. Note that each process connects to the database, so this value also determines the number of simultaneous connections to the database. Typically, a number such as 64 works well.

Required Configuration Properties

See the [Copy to Hadoop Property Reference](#) for more information on these and other properties.

- `oracle.hadoop.ctoh.table`
- `mapreduce.output.fileoutputformat.outputdir`
- `oracle.hadoop.ctoh.maxSplits`
- `oracle.hadoop.ctoh.splitterType`

Connection Properties for a secure Hadoop cluster using Oracle Wallet:

- `oracle.hadoop.ctoh.connection.walletLoc`
- `oracle.hadoop.ctoh.connection.tnsAdmin`
- `oracle.hadoop.ctoh.connection.tnsEntry`
- The following properties are also required if the Oracle Wallet directory on the Hadoop cluster is different from the directory on the Hadoop client:
 - `oracle.hadoop.ctoh.connection.clusterWalletLoc`
 - `oracle.hadoop.ctoh.connection.clusterTnsAdmin`

Connection Properties for Unsecured Hadoop clusters (for Demo Purposes Only):

For demo purposes, use the following properties in place of the properties used with secured clusters.

- `oracle.hadoop.ctoh.connection.username`
- `oracle.hadoop.ctoh.connection.password`
- `oracle.hadoop.ctoh.jdbc.url`

An Incremental Copy using Copy to Hadoop

To incrementally copy data from the same Oracle table to a pre-existing destination directory in HDFS, the following additional properties are required. (This configuration assumes that a Copy To Hadoop job was run initially to copy data from an Oracle Database table to datapump files in an HDFS directory.)

- `oracle.hadoop.ctoh.whereClause`
- `oracle.hadoop.ctoh.datapump.output`
- `oracle.hadoop.ctoh.datapump.basename`

`oracle.hadoop.ctoh.datapump.output` specifies a preexisting HDFS location that contains the datapump files from a previous run of Copy To Hadoop.

`oracle.hadoop.ctoh.whereClause` identifies the subset of rows to be copied from the Oracle table for the incremental load.

`oracle.hadoop.ctoh.datapump.basename` specifies a unique prefix for the datapump files. This property is used to generate unique datapump file names to prevent file name collisions during an incremental load.

B.2 Copy to Hadoop Property Reference

This reference describes customer-accessible properties of Copy to Hadoop.

Copy to Hadoop Configuration Property Reference (for Direct Copy)

Property	Description
<code>oracle.hadoop.ctoh.home</code>	<p>Type: String</p> <p>Default Value: Value of the <code>CP2HADOOP_HOME</code> environment variable.</p> <p>Description: This configuration property is used to locate jars required for the Copy to Hadoop job.</p>
<code>oracle.hadoop.ctoh.table</code>	<p>Type: String</p> <p>Default Value: None.</p> <p>Description: The name of the database table whose content is copied to Hadoop as datapump files. It can also be schema qualified. For example, to specify the table <code>EMPLOYEE</code> in schema <code>MANAGER</code>, you can use <code>MANAGER.EMPLOYEE</code>.</p>
<code>mapreduce.output.fileoutputformat.outputdir</code>	<p>Type: String</p> <p>Default Value: None.</p> <p>Description: The name of the output directory where datapump files are created by the Hadoop job. The job output logs are also stored in the <code>_ctoh</code> subdirectory.</p>
<code>oracle.hadoop.ctoh.datapump.output</code>	<p>Type: String</p> <p>Default Value: None.</p> <p>Description: Specifies the destination directory for datapump files. If this property is not specified, the datapump files will live in the directory specified by the <code>mapreduce.output.fileoutputformat.outputdir</code> property.</p>
<code>oracle.hadoop.ctoh.datapump.basename</code>	<p>Type: String.</p> <p>Default Value: <code>dppart</code></p> <p>Description: The prefix or base-name of generated data pump files. For example if a user specifies this property as <code>"dp_tbl"</code>, then the generated datapump file is <code>dp_tbl-m-00000.dmp</code>.</p>

Property	Description
<code>oracle.hadoop.ctoh.datapump.extension</code>	<p>Type:</p> <p>Default Value: <code>dmp</code></p> <p>Description: The suffix of generated data pump files. For example if a user specifies this property as “.dp”, then the generated datapump file is <code>dppart-m-00000.dp</code>.</p>
<code>oracle.hadoop.ctoh.maxSplits</code>	<p>Type: Integer.</p> <p>Default Value: None.</p> <p>Description: The number of datapump files that are created by the Hadoop job. This is also the number of mappers that will be produced by the Hadoop job.</p>
<code>oracle.hadoop.ctoh.splitterType</code>	<p>Type:</p> <p>Default Value: None.</p> <p>Description: The type of splitters that will be used to split the table data.</p> <ul style="list-style-type: none"> • <code>BLOCK_SPLITTER</code>: This splitter divides the table into block ranges. • <code>ROW_SPLITTER</code>: The splitter divides the table into row ranges. • <code>PARTITION_SPLITTER</code>: If a table is partitioned, the partition splitter can be used. When this splitter is specified, the number of datapump files created by the Hadoop job is at most equal to the number of partitions in the table.
<code>oracle.hadoop.ctoh.columns</code>	<p>Type:</p> <p>Default Value: None.</p> <p>Description: Specifies the subset of columns to be copied. For example if a user specifies “NAME,MANAGER” then the data for columns <code>NAME</code> and <code>MANAGER</code> are copied. If this property is not specified then all columns are copied (unless filtered by a <code>WHERE</code> clause).</p>
<code>oracle.hadoop.ctoh.whereClause</code>	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: This property is used to copy a subset of rows. For example, to copy employees whose ids are less than 1000 and greater than 500, then specify the following <code>WHERE</code> clause: <code>EMPLOYEE_ID < 1000 AND EMPLOYEE_ID > 500</code>.</p>
<code>oracle.hadoop.ctoh.jdbc.url</code>	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: The JDBC url to connect to the database. This property can used for demo purposes and non-secure clusters. Use Oracle Wallet with Secure Hadoop clusters in production environments.</p>

Property	Description
oracle.hadoop.ctoh.connection.username	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: The name of the Oracle Database user. This property can used for demo purposes and non secure clusters. Use Oracle Wallet with Secure Hadoop clusters in production environments.</p>
oracle.hadoop.ctoh.connection.password	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: The password of the Oracle Database user. This property can used for demo purposes and non secure clusters. Use Oracle Wallet with Secure Hadoop clusters in production environments.</p>
oracle.hadoop.ctoh.connection.walletLoc	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: Location of the Oracle Wallet directory on the Hadoop client.</p> <p>When using an Oracle Wallet, you must also set the following properties:</p> <ul style="list-style-type: none"> • oracle.hadoop.ctoh.connection.tnsAdmin • oracle.hadoop.ctoh.connection.tnsEntry
oracle.hadoop.ctoh.connection.clusterWalletLoc	<p>Type:</p> <p>Default Value: Value of oracle.hadoop.ctoh.connection.walletLoc.</p> <p>Description: Location of the Oracle wallet directory on the Hadoop cluster. NOTE: This directory must be visible to all nodes in the Hadoop cluster. When using this property, you must also set the following properties:</p> <ul style="list-style-type: none"> • oracle.hadoop.ctoh.connection.clusterTnsAdmin • oracle.hadoop.ctoh.connection.tnsEntry

Property	Description
<code>oracle.hadoop.ctoh.connection.tnsAdmin</code>	<p>Type: String.</p> <p>Default Value: Not Defined.</p> <p>Description: Location of the directory on the Hadoop client, containing the SQL*Net configuration files such as <code>sqlnet.ora</code> and <code>tnsnames.ora</code>. Set this property so that you can use TNS entry names in the database connection strings. When using Oracle Wallet, this property value can be the same as the value of the <code>oracle.hadoop.ctoh.connection.walletLoc</code> property.</p> <p>You must set this property when using an Oracle Wallet as an external password store. See <code>oracle.hadoop.ctoh.connection.walletLoc</code>.</p>
<code>oracle.hadoop.ctoh.connection.clusterTnsAdmin</code>	<p>Type: String.</p> <p>Default Value: Value of the property <code>oracle.hadoop.ctoh.connection.tnsAdmin</code>.</p> <p>Description: Location of the directory on the Hadoop cluster containing SQL*Net configuration files such as <code>sqlnet.ora</code> and <code>tnsnames.ora</code>. NOTE: This directory must be visible to all nodes in the Hadoop cluster.</p> <p>Set this property so that you can use TNS entry names in database connection strings. When using Oracle Wallet, this property value can be the same as the value of <code>oracle.hadoop.ctoh.connection.clusterWalletLoc</code>.</p> <p>You must set this property when using an Oracle Wallet as an external password store (as Oracle recommends). See <code>oracle.hadoop.ctoh.connection.clusterWalletLoc</code>.</p>
<code>oracle.hadoop.ctoh.connection.tnsEntry</code>	<p>Type: String.</p> <p>Default Value: None.</p> <p>Description: The TNS entry name defined in the <code>tnsnames.ora</code> file. Use this property with <code>oracle.hadoop.ctoh.connection.tnsAdmin</code>. When using Oracle Wallet, make sure that the <code>tnsEntry</code> name matches your wallet credential.</p>

Property	Description
oracle.hadoop.ctoh.cachePath	Type: String. Default Value: <code>\${mapreduce.output.fileoutputformat.outputdir}/../ctohCache</code> Description: Identifies the full path to an HDFS directory where cCopy to Hadoop can create files that are loaded into the MapReduce distributed cache. The distributed cache is a facility for caching large, application-specific files and distributing them efficiently across the nodes in a cluster.

C

Using mtactl to Manage the MTA extproc

The multithreaded agent control utility (`mtactl`) enables Oracle Big Data SQL users to start, stop, and configure the MTA (Multi-Threaded Agent) extproc in both Oracle Clusterware Ready Service (CRS) and non-CRS Oracle Database environments.

Note:

In non-CRS environments, customers must run `mtactl` in order to start the MTA extproc.

Usage

In this usage description, `mta_sid` is the SID that a given multithreaded extproc agent services.

```
mtactl {start|restart|stop|status|delete|show|bdsq} <mta_sid>
mtactl unset <parameter> <mta_sid>
mtactl set <parameter> <parameter_value> <mta_sid>
mtactl -help
mtactl <command> -help
```

Multithreaded Agent Control Utility Commands

Table C-1 mtactl Commands

Comm and	Full Syntax	Description
start	<code>mtactl start <mta_sid></code>	Start an MTA extproc for this SID, with the existing init parameter values stored in the repository. Use the default values if the repository does not exist.
restart	<code>mtactl restart <mta_sid></code>	Clean up the repository and restart the MTA extproc agent for the SID, with the default values.
stop	<code>mtactl stop <mta_sid></code>	Stop the MTA extproc agent that services the given SID.
status	<code>mtactl status <mta_sid></code>	Display status for the MTA extproc that services the given SID.

 **Note:**

If you used Oracle Big Data SQL 3.1, be aware that the behavior of `restart` and `start` are now reversed from what they were in 3.1 – `start` now uses init values from the repository if available. `restart` always uses the default values.

Table C-1 (Cont.) mtactl Commands

Comm and	Full Syntax	Description
delete	<code>mtactl delete <mta_sid></code>	Clean up the repository for the given SID.
show	<code>mtactl show <mta_sid></code>	Display the init parameters for the MTA extproc that services the given SID.
bdsq	<code>mtactl bdsq <mta_sid></code>	Display additional operations. These are for setting up the MTA extproc for use with Oracle Big Data SQL.
set	<code>mtactl set <init parameter> <value> <mta_sid></code>	Set the init parameters for the MTA extproc that services the given SID. Supported parameters are: max_dispatchers tcp_dispatchers max_task_threads max_sessions listener_address
unset	<code>mtactl unset <init parameter> <mta_sid></code>	Unset the init parameters in the repository for the MTA extproc that services the given SID.

Examples

```
$ mtactl start BDSQL_hadoop_cl_1 //note: using existing or default init parameter values
```

```
$ mtactl delete BDSQL_hadoop_cl_1
$ mtactl set max_sessions 200 BDSQL_hadoop_cl_1
$ mtactl set max_dispatchers 5 BDSQL_hadoop_cl_1
$ mtactl set max_task_threads 5 BDSQL_hadoop_cl_1
$ mtactl set listener_address "(ADDRESS=(PROTOCOL=ipc)(KEY=crs))"
BDSQL_hadoop_cl_1
$ mtactl start BDSQL_hadoop_cl_1 (note: use customized parameter values)
```

```
$ mtactl restart BDSQL_hadoop_cl_1 //note: using default init parameter values
```


D

Diagnostic Tips and Details

The following is a collection of notes that can be useful for troubleshooting and for general understanding of Oracle Big Data SQL.

D.1 Running Diagnostics with `bdschecksw`

On the Oracle Database server, you can use the Big Data SQL Diagnostics Collection tool, `bdschecksw`, to do a simple sanity test of Oracle Big Data SQL. This script gathers and analyzes diagnostic information about the Oracle Big Data SQL installation from both the Oracle Database and the Hadoop cluster sides of the installation. The script is in `$ORACLE_HOME/bin` on the database server.

You can run this diagnostic check manually at any time. At installation time, it is also run by `bds-database-install.sh`, the database-side installer.

Syntax

```
bdschecksw Required_Params [Options]
```

The table below describes the required and optional parameters used with `bdschecksw`.

Table D-1 `bdschecksw` Parameters

Parameter	Description	Required or Optional
<code>-h, --help</code>	Display command help and exit.	Optional
<code>-d, --dbhome ORACLE_HOME</code>	The path to the Oracle installation on the Oracle Database server.	Required only if the <code>ORACLE_HOME</code> environment variable is not defined on the Oracle Database server where <code>bdschecksw</code> is executed.

Table D-1 (Cont.) bdschecksw Parameters

Parameter	Description	Required or Optional
-s, --sid= <i>ORACLE_SID</i>	The SID of the Oracle Database server.	Required only if the <i>ORACLE_SID</i> environment variable is not defined on the Oracle Database server where <i>bdschecksw</i> is executed.
-g, --gihome= <i>Grid_Infrastructure_home</i> <i>Oracle_Database_node_IP_address</i>	The Grid Infrastructure path on the Oracle Database server.	Required only if the <i>GI_HOME</i> environment variable is not defined on the Oracle Database server where <i>bdschecksw</i> is executed.
-y, --giuser <i>Oracle_Database_node_IP_address</i> <i>Grid_Infrastructure_home</i>	<i>GI_HOME</i> administrator name or owner (OS user name) of <i>GI_HOME</i> .	
-q, --sqlplus <i>Oracle_Database_node_IP_address</i> <i>username</i>	SQLPlus username on the Oracle Database server. The user is prompted for the password.	Required.
-c, --cell <i>DNS short name</i> [...n]	The Hadoop cluster cell nodes. Use DNS short names (FQDN minus domain name) with a space as a delimiter. IP addresses are not recommended, because a node may exist on multiple networks.	Required for Hortonworks HDP only.
-u, --uname <i>Hadoop_cluster_node_username</i>	Credentials to run remote commands on the Hadoop cluster from the Oracle Database server. This is usually the <i>oracle</i> user.	The username and password are always required.

Table D-1 (Cont.) bdschecksw Parameters

Parameter	Description	Required or Optional
<code>-p, --pdb=PDB_CONTAINER</code>	The Pluggable Database container name on the Oracle Database server.	Required only if the Pluggable Database (PDB) container is configured on the Oracle Database server where bdschecksw is executed.
<code>-k, --key SSH_identity_key_file</code>	Path to an SSH (Secure Shell) key file.	The optional SSH identity (or key) file is used on top of <code>-u</code> and <code>-p</code> to select a file from which the identity (private key) for public key authentication is read.
<code>-r, --cluster Hadoop_cluster_name</code>	Hadoop cluster name.	Optional.
<code>-t, --trace</code>	Turn on extproc and log4j tracing during test execution.	Optional.
<code>-f, --file=file_path</code>	Redirect output to the file.	Optional.
<code>-i, --interactive</code>	Enter command line arguments interactively (rather than all at once).	Optional.
<code>-x</code>	Extensive mode.	Optional. Requires root privilege.
<code>-v, --verbose</code>	Verbose reporting mode.	Optional. (Recommended for full details in the report.)

Exit Status

The bdschecksw script returns one of the following status codes.

Table D-2 Status Codes for bdschecksw

Status	Description
0	Success

Table D-2 (Cont.) Status Codes for bdschecksw

Status	Description
1	Minor problem (for example, no response in interactive mode).
2	Significant problem (for example, an invalid command line argument).

Example

```
$ ./bdschecksw -d /u03/app/oracle/product/$ORACLE_HOME/dbhome_1 -s orcl -p
pdborcl -g /u03/app/oracle/product/$ORACLE_HOME/grid -q sys -u oracle -v
```

D.2 How to do a Quick Test

Here is an all-around series of basic checks to ensure that Oracle Big Data SQL is working.

1. On the Oracle Database server, source the environment using the `hadoop_<hcluster>.env` file in `$ORACLE_HOME/bigdatasql`.
2. If Kerberos is enabled, `kinit` as the `oracle` Linux user on the Oracle Database server. If possible, also `kinit` on each of the Big Data SQL datanodes as the `oracle` user.

 **Note:**

You can run this test without running `kinit` on the datanodes, but then offloading in the test will not work. You will eventually need to `kinit` on the datanodes in order to verify that offloading is working.

3. Create a text file and add several of lines of random text.
4. Copy the text file into hdfs as `/user/oracle/test.txt`.

```
$ hadoop fs -put test.txt /user/oracle/test.txt
```

5. Define an Oracle external table of type `ORACLE_HDFS`:

```
a. CREATE TABLE bds_test (line VARCHAR2(4000))
   ORGANIZATION EXTERNAL
   ( TYPE ORACLE_HDFS DEFAULT DIRECTORY DEFAULT_DIR LOCATION
     ('/user/oracle/test.txt') )
   REJECT LIMIT UNLIMITED;
```

```
b. Select * from bds_test;
```

```
c. select n.name, s.value /* , s.inst_id, s.sid */ from v$statname
   n, gv$mystat s where n.name like '%XT%' and s.statistic# =
   n.statistic#;
```

6. Define a Hive table:

- a. Connect to Hive via Hue, the Hive/Beeline command line, or using Oracle SQL Developer with a Hive JDBC driver.
 - b. `CREATE TABLE bds_test_hive (line string);`
 - c. `LOAD DATA INPATH '/user/oracle/test.txt' OVERWRITE INTO TABLE bds_test_hive;`
7. Define an external ORACLE_HIVE table:

```
CREATE TABLE bds_test_hive (line VARCHAR2(4000))
  ORGANIZATION EXTERNAL
  ( TYPE ORACLE_HIVE DEFAULT DIRECTORY DEFAULT_DIR
    ACCESS PARAMETERS
      (com.oracle.bigdata.tablename=default.bds_test_hive)
  )
  REJECT LIMIT UNLIMITED;
```

D.3 Oracle Big Data SQL Database Objects

Familiarity with the various Oracle Big Data SQL database objects can be helpful in troubleshooting.

Table D-3 Database Objects in Big Data SQL

Type	Object
Directories	<ul style="list-style-type: none"> • <code>DEFAULT_DIR</code> – points to <code>\$ORACLE_HOME/bigdatasql/databases/<database name>/default_dir</code>. • <code>ORACLE_BIGDATA_CONFIG</code> – points to <code>\$ORACLE_HOME/bigdatasql/databases/<database name>/bigdata_config</code>. • <code>ORA_BIGDATA_CL_<hcluster></code> – expected to have a null value for its path. <p>This is a way of limiting access. There always must be a directory object associated with an external table. Because the directory object is used for privilege checking, this is a requirement even for Hive/HDFS, where the files do not reside under the directory.</p>
Database Links (public) These allow Big Data SQL to reach the MTA (multi-threaded agent)	<ul style="list-style-type: none"> • <code>BDSQL\$_DEFAULT_CLUSTER</code> – the connect string's SID should equal <code>bds_<dbname>_<hcluster></code>. And the <code><hcluster></code> should be the default cluster (as defined by <code>bigdata.cluster.default</code>) in <code>\$ORACLE_HOME/bigdatasql/databases/<database name>/bigdata_config/bigdata.properties</code>. • <code>BDSQL\$_<hcluster></code> – the connect string's SID should equal <code>bds_<dbname>_<hcluster></code>.
Data Dictionary Views	<ul style="list-style-type: none"> • <code>User_hive_tables</code>, <code>all_hive_tables</code>, <code>dba_hive_tables</code> – queries all Hive tables for all Hive databases for all Hadoop clusters. • <code>User_hive_databases</code>, <code>all_hive_databases</code>, <code>dba_hive_databases</code> – queries all Hive databases for all Hadoop clusters. • <code>User_hive_columns</code>, <code>all_hive_columns</code>, <code>dba_hive_columns</code> – queries all hHive tables for all Hive databases for all Hadoop clusters. • <code>V\$cell</code> – the Oracle Big Data SQL server processes running on datanodes will appear here (if properly detected by the diskmon).

Table D-3 (Cont.) Database Objects in Big Data SQL

Type	Object
Functions and Procedures for Hive Data Dictionary See cathive.sql, dbmshadp.sql.	<ul style="list-style-type: none"> • DBMS_HADOOP <ul style="list-style-type: none"> – Create_extddl_for_hive() • GetHiveTable – pipeline function that returns data back from the extproc external procedure. Used by the *_hive_[tables/databases/columns] views and DBMS_HADOOP. <ul style="list-style-type: none"> – HiveMetadata – ODCI framework defining the external procedureGetHiveTable. – SYS.DBMSHADOOLIB (libkubsagt12.so) – C library for the external procedure. – HiveMetadata.jar – java library called by libkubsagt12.so.
Tables	SYS.HIVE_URI\$ – security table for non-DBA users.
Statistics	<ul style="list-style-type: none"> • All statistics have %XT% in the name: <ul style="list-style-type: none"> – cell XT granules requested for predicate offload – cell XT granule bytes requested for predicate offload – cell interconnect bytes returned by XT smart scan – cell XT granule predicate offload retries – cell XT granule IO bytes saved by storage index • Use this query: <pre>select n.name, s.value /* , s.inst_id, s.sid */ from v\$statname n, v\$mystat s where n.name like '%XT%' and s.statistic# = n.statistic# ;</pre> <p>If needed, also use: grant select any catalog to <dbuser>;</p>

D.4 Other Database-Side Artifacts

This section describes directories, files, and external procedure agents on the database side that are relevant to Oracle Big Data SQL.

Table D-4 \$ORACLE_HOME/bigdatasql Directory

Subdirectory or Filename	Description of Contents
clusters directory	<p>Contains settings related to all clusters installed on this ORACLE_HOME. It includes a subdirectory for each cluster, which contains:</p> <ul style="list-style-type: none">• <code>config</code> directory – configuration files downloaded for Cloudera Manager or Ambari.• <code>fuse</code> directory – settings for the FUSE-DFS service for the this cluster.• <code>hadoop</code>, <code>hbase</code>, and <code>hive</code> soft links to the actual client directories. (For example: <code>hadoop-2.6.0-cdh5.12.0</code>, <code>hbase-1.2.0-cdh5.12.0</code>, <code>hive-1.1.0-cdh5.12.0</code>, although the versions installed may differ for your system.)• <code>*.conf</code> – IPsec configuration file.• <code>*.keytab</code> – Kerberos keytab file for the database owner.

Table D-4 (Cont.) \$ORACLE_HOME/bigdatasql Directory

Subdirectory or Filename	Description of Contents
databases directory	<p>Contains settings related to all databases running on this ORACLE_HOME. It includes a subdirectory for each database running on this ORACLE_HOME. Each database subdirectory contains a <code>bigdata_config</code> directory, which includes:</p> <ul style="list-style-type: none"> • <code>bigdata.properties</code> – Defines the location of JAR files. If your Hive tables use non-standard Java libraries, you may need to copy those libraries to the database and update the <code>classpath</code> entries in this file. Also defines the Hadoop cluster list and default cluster. Restart the <code>extproc_bds_<dbname>_<hcluster></code> after changing. • <code>bigdata-log4j.properties</code> - This controls the logging of the Java pieces, such as the metadata discovery phase of querying external tables or the query fetch phase if the cells are unavailable. Change <code>log4j.logger.oracle.hadoop.sql</code> to <code>INFO</code> to log more. Restart the <code>extproc_bds_<dbname>_<hcluster></code> after changing. • <code><hcluster></code> directory – a soft link to <code>\$ORACLE_HOME/bigdatasql/clusters/<cluster name>/config</code>, which contains the client configuration files (like <code>hive-site.xml</code>) copied from the Hadoop cluster.
	<p>Each database subdirectory also includes:</p> <ul style="list-style-type: none"> • <code>default_cluster</code> – soft link to the <code>\$ORACLE_HOME/bigdatasql/clusters/</code> subdirectory that is the default cluster for this database. • <code>default_dir</code> – directory for external tables associated with this database.
jdk	<p>Soft link to the JDK installation. The version installed by Oracle Big Data SQL is <code>jdk1.8.0_141</code>, although a different version may be present.</p>
jlib directory	<p>The Oracle Big Data SQL Java JAR directory</p>

Table D-4 (Cont.) \$ORACLE_HOME/bigdatasql Directory

Subdirectory or Filename	Description of Contents
bigdata_config directory	<ul style="list-style-type: none"> bigdata.properties – Defines the location of JAR files. If your Hive tables use non-standard Java libraries, you may need to copy those libraries to the database and update the classpath entries in this file. Also defines the Hadoop cluster list and default cluster. Restart the extproc_bds_<dbname>_<hcluster> after changing. bigdata-log4j.properties - This controls the logging of the Java pieces, such as the metadata discovery phase of querying external tables or the query fetch phase if the cells are unavailable. Change log4j.logger.oracle.hadoop.sql to INFO to log more. Restart the extproc_bds_<dbname>_<hcluster> after changing. <hcluster> directory – a soft link to \$ORACLE_HOME/bigdatasql/clusters/<cluster name>/config, which contains the client configuration files (like hive-site.xml) copied from the Hadoop cluster.
default_dir directory	This directory is usually empty.
log directory	Contains Java log files from the two type of extprocs. Use the PID that is part of the filename to identify which extproc you are looking at (only one log file will have the PID of the currently running extproc_bds_<dbname>_<hcluster> process).
hadoop_<cluster name>.env	Sets the Hadoop client environment. There is one of these .env files for each cluster installation. You can source this file and then run hadoop fs commands to quickly test Hadoop connectivity
orahivedp	A soft link to the installed cp2hadoop (Copy to Hadoop) toolkit. The version of the toolkit installed by Oracle Big Data SQL 3.2 is orahivedp-3.2.0.

Table D-5 External Procedure Agents

Agent	Description
extproc	<p>Runs the external procedure code used by the *_hive_tables, *_hive_databases views and the DBMS_HADOOP procedure.</p> <p>\$ORACLE_HOME/hs/admin/extproc.ora configures the extproc. You can add TRACE_LEVEL=ON to get more trace (but may need to first comment out SET EXTPROC_DLLS= to fix the "Error in assignment statement" message). The C portion's log files are in \$ORACLE_HOME/hs/log/orcl_agt* , but these are usually not interesting for diagnostic purposes. The JVM portion's log files are written to \$ORACLE_HOME/bigdatasql/log (but you need to set up bigdata-log4j.properties first).</p>
extprocbds_<dbname>_<hcluster>	<p>The BDS Multi-threaded Agent that is used when querying external tables. This is started/stopped by Oracle Clusterware which in turn runs the mtactl utility. This is registered to Oracle Clusterware when bds_database_install.sh runs on the last database server node.</p> <p>If you don't have this extprocbds_<dbname>_<hcluster> running, then you probably didn't run bds_database_install.sh on every database server in your RAC cluster. The C portion's log files are in \$ORACLE_HOME/hs/log (but you need to edit \$ORACLE_HOME/hs/admin/initbds - add TRACE_LEVEL=ON and then restart to see logging). The JVM portion's log files go into \$ORACLE_HOME/bigdatasql/log (but you need to setup bigdata-log4j.properties and restart). This is the recommended way to restart (although the quicker way is to run kill -9 on the process):</p> <pre data-bbox="878 1583 1203 1707"> \$ crsctl stop resource bds_<dbname>_<hcluster> \$ crsctl start resource bds_<dbname>_<hcluster></pre>

 **Note:**

Both of External Procedures in the table above make callbacks to the database, which can be blocked by the use of the Secure External Password Store feature. If you use Secure External Password Store (SQLNET.WALLET_OVERRIDE=TRUE), see Document 2126903.1 in [My Oracle Support](#).

Table D-6 Log Files and Trace Files

Directory or Filename	Description
\$ORACLE_HOME/bigdatasql/log	Contains log files from Java code run via the extprocbds_<dbname>_<hcluster> – one shared file with PID equal to the extprocbds_<dbname>_<hcluster> PID and extproc (one per session if the session uses *_hive_tables or DBMS_HADOOP). Tip: This is good diagnostic information.
\$ORACLE_HOME/hs/log	Contains log files from the C code of the extproc processes (one per session) and the multi-threadedextbds_<dbname>_<hcluster> process. The extproc is usually not interesting for diagnostic purposes. The extprocbds_* has a bit more interesting information (but you need to set TRACE_LEVEL=ON in initbds_*.ora).
Database diag directory	<p>Contains log files from the database session. These can yield good information.</p> <ul style="list-style-type: none"> To identify the exact database session log file location: <pre>select value from v\$diag_info WHERE name = 'Default Trace File';</pre> To turn on external table logging: <pre>alter session set "_xt_trace"="low", "compilation", "execution";</pre> To turn on additional logging: <pre>alter session set events 'trace[KCFIS] disk high, memory high';</pre>

Table D-6 (Cont.) Log Files and Trace Files

Directory or Filename	Description
/u01/oracle/ diag/crs/ <hostname>/crs/ trace/diskmon.trc	<p>Contains diskmon logs and errors. In a commodity Oracle Database server to commodity Hadoop environment (support in Oracle Big Data SQL 3.0 and greater), check this trace file for communication errors or fencing (ENTITY_FENCED). Restart diskmon if needed (use crsctl). In a commodity-to-commodity environment, you can simply kill the diskmon process, but do not do that in an Oracle Exadata Database Machine environment.</p> <p>If you want to get additional diskmon tracing, you can set environment parameters before you invoke the crsctl command to start the cluster. Since the cluster is likely already running, you'll first have to shut the cluster down. Then, set the environment and then start it back up. Here is how you do it in the Oracle Big Data SQL 3.x commodity database server scenario using Oracle Restart. (Note that the crsctl commands will be different if using RAC, ASM, and/or Exadata):</p> <pre>crsctl stop has export CELLCLIENT_TRACE_LEVEL="all,4" export CELLCLIENT_AUTOFLUSH_LEVEL="all,4" crsctl start has</pre>
/etc/oracle/ cell/network- config/ cellinit.ora /etc/oracle/ cell/network- config/ celliniteth.ora	<p>Record the IP address and subnet range for the database server. For Oracle Big Data SQL on commodity servers, this file also includes parameters which switch the protocol away from InfiniBand RDS to TCP/UDP (_skgxp_dynamic_protocol=2). On commodity servers, the database server's diskmon (running out of the Oracle Grid home) communicates with the BDS processes on the data nodes listening on TCP port 5042.</p>
Kerberos files: kinit, klist, etc/ krb5.conf, krb5- workstation*.rpm	<p>If your Hadoop cluster uses Kerberos, you'll need to setup Kerberos on the database and have a mechanism (such as crontab) to keep a valid Kerberos ticket at all times for the oracle Linux user. You will need a similar ticket renewal mechanism on the BDS datanodes as well.</p> <p>The Oracle Big Data SQL installer now provides a directive in the Jaguar configuration file that will automatically set up a cron job for this on both the Hadoop cluster and the Oracle Database system. See the description of the configuration file in the installation guide.</p>

D.5 Hadoop Datanode Artifacts

The table below identifies objects on the Hadoop server that can provide helpful information for troubleshooting Big Data SQL.

Table D-7 Hadoop-side Datanode Artifacts That are Useful for Troubleshooting

Datanode Artifact	Description
bdscli command	<ul style="list-style-type: none"> List quarantine detail Drop quarantine all List alerthistory Drop alerthistory List bdsq detail
Log files	<ul style="list-style-type: none"> <code>/var/log/bigdatasql/DM-installer</code> log files <code>/var/log/bigdatasql/cloudera</code> or <code>/var/log/bigdatasql/ambari</code> – Ambari or CM service log files. <code>/opt/oracle/bigdatasql/bdcell-<cell version>/bigdata.properties</code> <code>/opt/oracle/bigdatasql/bdcell-<cell version>/bigdata-log4j.properties</code> <ul style="list-style-type: none"> This defaults to logging off. Change to <code>log4j.logger.oracle.hadoop.sql=INFO</code> and restart. <code>/opt/oracle/bigdatasql/bdcell-<cell version>/log directory</code> <ul style="list-style-type: none"> <code>bigdata-log4j.log</code> – logs entries from the JVM pieces of Big Data SQL (logging defaults to off, so edit <code>bigdata-log4j.properties</code> first and restart). This can be particularly useful information. <code>/var/log/oracle/diag/bdsq/cell/<hostname>/trace/</code> – general cell trace files for the Management Server, Restart Server, and Monitor Server. The <code>alert.log</code> file will have details about quarantine and de-quarantine events. <code>/var/log/oracle/diag/bdsq/cell/SYS_*/trace/</code> – Oracle Big Data SQL offload server trace files for the C portion. These are not useful for troubleshooting in most cases.
Other datanode artifacts	<code>/opt/oracle/cell/cellsrv/deploy/config/cellinit.ora</code> – records the cell's IP address.

D.6 Step-by-Step Process for Querying an External Table

This section describes the events that occur during a query of an external table.

1. User issues a SELECT query involving an Oracle Big Data SQL external table.
2. Database sees that one of the objects in the SQL is an External table of type ORACLE_HIVE
3. Database identifies the cluster name from the `com.oracle.bigdata.cluster` parameter on the External table definition else uses the default cluster.
4. Database identifies the Hive table name from the `com.oracle.bigdata.tablename` parameter, else assumes the Hive table name is the same as the Oracle table name.
5. Database knows that the ORACLE_HIVE External table implementation uses an external procedure which is invoked through the `extproc_bds_<dbname>_<hcluster>` multi-threaded agent.

 **Note:**

The first phase of the query requires getting the Hive metadata. If you get an error during this first phase, you'll likely see an error that begins as follows. Notice the "OPEN" in ODCIEXTTABLEOPEN)

```
ORA-29913: error in executing ODCIEXTTABLEOPEN callout
```

6. Database uses the public database link `BDSQL$_DEFAULT_CLUSTER` or `BDSQL$_<hcluster>` to find the connect string to ask the listener to connect the database session to a thread of the `extproc_bds_<dbname>_<hcluster>` multi-threaded agent
 - a. `extproc_bds_<dbname>_<hcluster>` was previously started by Oracle Clusterware and is using configuration information from the `$ORACLE_HOME/bigdatasql/databases/<database name>/bigdata_config` directory.
 - b. `extproc_bds_<dbname>_<hcluster>` has spawned a JVM running Hadoop client libraries using the above configuration information. The Hadoop client libraries were copied from the Oracle Big Data Appliance to the Oracle Database server when you ran the `bds-exa-install.sh` script.
7. `extproc_bds_<dbname>_<hcluster>` uses its JVM and the Hive metastore client library to call the Hive metastore (using a URL such as `thrift://hostname:9083`) to get metadata (columns, inputformat, serde, other table properties) for the Hive table.
 - a. At this point, if the Hive metastore is protected by Kerberos authentication, the Hive client libraries running in the `extproc_bds` JVM on the Oracle Database server will try to send the local Kerberos ticket to the Hive server. This will be the ticket owned by the `oracle` Linux user account who is running the database
8. `extproc_bds_<dbname>_<hcluster>` calls the Hive metastore to get a list of input paths that hold the data behind the Hive table.

9. `extprocbds_<dbname>_<hcluster>` converts the list of input paths into a list of splits/blocks using Hadoop MapReduce libraries and logic. Then it asks the HDFS namenode for the location (including replicas) of all of the splits /blocks.
 - a. Again, if HDFS is protected by Kerberos, the Kerberos ticket from the `oracle` Linux user account on the database will be need to be used.
 - b. If compression is used, at this point the JVM might have to load specific compression Java or native libraries. If these are non-standard libraries, you will need to install them on both the Oracle Database server and the Hadoop side. For instance, LZO compression requires an additional install and configuration performed on both the database-side on the Hadoop-side.

At this point, the “description” phase is done and the database knows the structure of the Hive table as well as the location of all of the blocks of data (including replicas). This information is also known as the metadata payload. We now begin the “fetch” phase.

10. The database intelligent storage layer, KCFIS (Kernel Cache File Intelligent Storage), which is also used on Oracle Exadata systems, compares the hostnames of where the blocks of data are stored to a list of active BDSQL server hosts being maintained by the Grid’s diskmon process. (You can see diskmon’s list of BDSQL server hosts in `V$CELL`).

 **Note:**

The second phase of the query requires fetching the data. If you get an error during this second phase, you’ll likely see an error that begins as follows. Notice the “FETCH” in `ODCIEXTTABLEFETCH` :

```
ORA-29913: error in executing ODCIEXTTABLEFETCH callout
```

11. Assuming that the list of datanode hostnames matches the list of BDSQL hostnames, the database sends a list of local blocks (also called Granules) to each of the BDSQL servers. The database also sends the BDSQL servers metadata about the table, columns, and structure it is accessing. It does this in parallel and asynchronously for performance

 **Note:**

The database statistics “cell XT granules requested for predicate offload” and “cell XT granule bytes requested for predicate offload” are updated at this point

12. The BDSQL process running on the data nodes checks the `SQL_ID` against its local list of quarantined `SQL_IDS`. If the `SQL_ID` matches the quarantine list, then the BDSQL process on the datanode will return an error. However, the user should not see this error. Instead, the database will first try another cell, then try to do the work itself. (See Steps 15 and 16).
13. Assuming the `SQL_ID` is not quarantined by the BDSQL process on the datanode, the BDSQL process will do its SmartScan work against the list of blocks/granules sent to it.

 **Tip:**

See the blog entry [Big Data SQL Quick Start. Storage Indexes - Part10](#) in [The Data Warehouse Insider](#) for details about Storage Indexes and other aspects of SmartScan processing.

- a. The BDSQL offload process has previously read its configuration information from `/opt/oracle/bigdatasql/bdcell-<cell version>/bigdata.properties`.
 - b. The BDSQL process has previously loaded a JVM based on the properties defined in the above configuration.
 - c. If the Hive table has special InputFormat or Serde classes, the JVM will load those classes assuming it can find them based on the classpath defined in the above configuration. For some common InputFormats (such as delimited text), Oracle has written C code that can handle those formats faster than regular Java code.
 - d. If Kerberos authentication is used, then the BDSQL's JVM will send its local Kerberos ticket to the HDFS datanode process. This is the Kerberos ticket associated with the `oracle` Linux user on the datanode where BDSQL is running.
 - e. If Sentry authorization is used, the `oracle` Linux user's Kerberos ticket's identity needs to have been granted access to the Hive table and underlying HDFS data.
 - f. The BDSQL server will update statistics like "cell XT granule IO bytes saved by StorageIndex" as it runs.
14. The database kcfis layer will collect results as they are returned from the BDSQL processes on the datanodes and send the next batch of blocks/granules to the BDSQL processes.
- a. The database will update the "cell interconnect bytes returned by XT smart scan" statistic as bytes are returned
15. If there are issues with a BDSQL process for a given block, the database will try to send the work to a different BDSQL process (it will pick a location that has a replica of the block that failed).
- a. The database will update the "cell XT granule predicate offload retries" statistic.
 - b. If there are any retries count and the BDSQL server processes are all active (shown with Cloudera Manager or Ambari UI), then the memory on the BDSQL server process could be an issue. Try `grep 'Failed to create or initialize subheap'` on `/opt/oracle/bd_cell/log/diag/bdsqll/cell/SYS_*/trace/bdsqll*.trc` files in each datanode. If you see any of the messages, the BDSQL server processes are memory constrained. Please increase the cgroup setting from Cloudera Manager or add more memory to the particular datanodes.
You could also turn on kcfis trace (refer to [Key Administration Tasks for Oracle Big Data SQL](#) on how to turn on the kcfis trace). If you have memory issue on BDSQL cell processes, you will see 'Cell failed to allocate memory' in the kcfis trace.

Please note that there are two BDSQL offload server processes on each datanode. On any datanode, you could issue 'bdscli -e list OFFLOADGROUP' to find out their names. One is to serve Oracle database 12.1 (SYS_121*), and the other for Oracle database 12.2 and above (SYS_122*). If you do not need both, you could shutdown one of them with ' bdscli -e 'alter OFFLOADGROUP SYSxxxxx shutdown' (where SYSxxxxx is the name displayed with 'bdscli -e list offloadgroup' command. You could use dcli (if you have BDA) to issue the command to apply to all datanodes. Otherwise, you will need to issue the command in each datanode.

16. If the database is unable to get the BDSQL processes to successfully offload a block even after retrying, then the database will “fallback” and have the JVM in the `extproc_bds_<db>_<cluster>` do the work.
 - a. This will be slower as the raw data will need to be moved to the database server for processing.
 - b. If the Hive table has any special InputFormats or Serdes, the `extproc_bds_<db>_<cluster>`'s JVM will need to load them based on the classpath configuration defined on the database's `bigdata.properties` file.
17. The results from the external table data source continue to be collected until all input paths/blocks/granules are handled.

D.7 Step-by-Step for a Hive Data Dictionary Query

This section describes the events that occur in a query over a Hive Data Dictionary.

1. User queries one of the Oracle Big Data SQL data dictionary views, such as `all_hive_tables`.

In Oracle Big Data SQL 2.0 and earlier, if this was the `user_hive_*` view and the user was not a DBA, then the user needed to be listed in the `SYS.HIVE_URI$` table. Oracle Big Data SQL 3.0 removed the `HIVE_URI$` check.

2. The view accesses the `GetHiveTable` pl/sql pipelined table function.
3. The `GetHiveTable` function is implemented by the `HiveMetadata` type which is implemented as an external procedure using the `SYS.DBMSHADOOLIB` library.
4. The Oracle Database spawns a new instance of the “extproc” for this database session. The extproc reads the `$ORACLE_HOME/hs/admin/extproc.ora` file for settings.

You can set `TRACE_LEVEL=ON` for tracing of the C code. Log file will be written to `$ORACLE_HOME/hs/log`.

By default, there may be an error in the `extproc.ora`, causing an “Error in assignment statement” message in the log . The statement “`SET EXTPROC_DLLS=`” (with no value after the equal sign) is not valid. Comment this line out if you want to use `TRACE_LEVEL=ON` .

5. The extproc attaches the `libkubsagt.so` library (as in `SYS.DBMSHADOOLIB`).
6. `Libkubsagt12.so` initiates a JVM and loads the `HiveMetadata.jar`.
 - a. The JVM uses the configuration information in `$ORACLE_HOME/bigdatasql/bigdata_config/` to identify the list of clusters and their Hive metastore connection information.

- b. Logging for the JVM is based on `$ORACLE_HOME/bigdatasql/bigdata_config/bigdata-log4j.properties`. Log files will be written to `$ORACLE_HOME/bigdatasql/log`. There will be a new log file for each database session.
- 7. The Java code in `HiveMetadata.jar` uses the Hive metastore client libraries to connect to the Hive metastore to retrieve data about all of the databases and all of the tables.
 - a. If the Hive metastore is protected by Kerberos, the JVM will try to send the Kerberos ticket of the `oracle` Linux user who is running the database
- 8. The Java code returns the request data back to the database.

D.8 Key Administration Tasks for Oracle Big Data SQL

These are notes about some important administrative tasks.

- Restarting the `extprocbds_<db>_<hcluster>`:

```
$ crsctl stop res bds_<dbname>_<hcluster>
```

Quick way, but not the best way: kill the `extprocbds_*` process and wait for it to come back

- Restarting the `extproc`.

This begins a new database session.

- Restarting the Oracle Big Data SQL software on the datanodes:

- Use Cloudera Manager or the Ambari Web UI.
- Quick way, but not the best way: kill the `bdsqloflsrv` process and wait for it to come back.
- Command line method on an Oracle Big Data Appliance (logged on as `root` on `node1`):

```
$ bdacli stop big_data_sql_cluster
$ bdacli start big_data_sql_cluster
```

- Checking for Oracle Big Data SQL quarantines on a single datanode:

```
$ bdscli -e list quarantine detail
```

To check for quarantines on all datanodes:

```
$ dcli -g cells.lst bdscli -e list quarantine detail
```

- Clearing Oracle Big Data SQL quarantines on a single datanode:

```
$ bdscli -e drop quarantine all
```

To clear quarantines on all DataNodes:

```
$ dcli -g cells.lst bdscli -e drop quarantine all
```

- Checking statistics for proper offloading:
 - Use the Sql Monitor hint: /*+ MONITOR*/.
 - Query XT statistics. Ensure that “retries” is zero and “bytes returned” is greater than zero.
- Looking for log files on the datanodes:
 1. Clear quarantines on all datanodes
 2. Set Log property in /opt/oracle/bigdatasql/bdcell-12.1/bigdata-log4j.properties on datanodes.
 3. Restart bdsqlofflsrv on datanodes.
 4. Cd to the log file directory: /opt/oracle/bigdatasql/bdcell-12.1/log.
 5. tail -f bigdata-log4j.log
 6. Ensure that your query has data on the node you are looking at (i.e. your query should need to access files with many blocks. If you only touch a small number of blocks, the result may be that your datanode is not be asked to do any work)
 7. Make a new database session (to reset XT statistics) and Run query.



Tip:

Use the /*+MONITOR*/ hint if you want to be sure to see it in SQL Monitor.

You should see new entries in the datanode’s bigdata-log4j.log.

8. On the Oracle Database server, query XT statistics and check that retries=0 and bytes returned>0.
- Looking for log files on the database:
 1. Clear quarantines on all data nodes
 2. Make a new database session (to reset XT statistics)
 3. Find out what instance your session is connected to (in case you got load-balanced to a different database server than the one you logged on to):


```
select host_name from v$instance;
```
 4. Log in to that instance’s database server at the Linux level.
 5. Set log properties in \$ORACLE_HOME/bigdatasql/bigdata-log4j.properties.
 6. Restart extproc_bds_<db>_<hcluster> on that instance to pick up the log property changes
 7. Turn on XT tracing:

This command turns on external table logging:

```
alter session set "_xt_trace"="low", "compilation", "execution";
```

This command adds additional tracing:

```
alter session set events 'trace[KCFIS] disk high, memory high';
```

8. Run the query.



Tip:

Use the `/*+ MONITOR */` hint if you want to be sure to see it in SQL Monitor.

9. Query XT statistics and see if `retries=0` and `bytes returned>0`.

```
select n.name, s.value /* , s.inst_id, s.sid */ from v$statname  
n, gv$mystat s where n.name like '%XT%' and s.statistic# =  
n.statistic# ;
```

10. Look at JVM log file: `$ORACLE_HOME/bigdatasql`. (Look for the one with the same PID as the `extprocbds_*` process.)

11. Look at database `trace_file`:

```
select value from v$diag_info WHERE name = 'Default Trace File';
```

D.9 Additional Java Diagnostics

The following are some additional Java diagnostics.

- You can add JVM properties to the `bigdata.properties` file as shown below. This can be good for hard-to-spot low-level Kerberos issues.

```
java.options=-Dsun.security.krb5.debug=true
```

- The `extproc` and `extprocbds_<dbname>_<hcluster>` processes run the JVMs on the database and the `bdsqloflsrv` process runs the JVM on the datanode. You can see this by running the “jps” command:

```
$ORACLE_HOME/bigdatasql/jdk*/bin/jps
```

- If you are very comfortable with your Java skills, you can also use Oracle JVisualVM or Oracle JConsole to connect to the JVMs.

D.10 Checking for Correct Oracle Big Data SQL Patches

Patch and Datapatch errors can have a number of different causes and effects. One thing you can do is check to ensure that the expected patches are loaded.

If you see "wrong number or types of arguments in a call to 'FETCH_OPEN' in the error stack

Here is an example of an error stack that may warrant a query of `DBA_REGISTRY_SQLPATCH` to determine if the correct patches are loaded:

```
ORA-29913: error in executing ODCIEXTTABLEFETCH callout
ORA-29400: data cartridge error
ORA-06550: line 1, column 25:
PLS-00306: wrong number or types of arguments in call to 'FETCH_OPEN'
ORA-06550: line 1, column 14:PL/SQL: Statement ignored
```

This may indicate one of several problems.

- A Bundle Patch was not applied correctly
- Datapatch did not run and therefore a patch required by the installed version of Oracle Big Data SQL is not installed

In this case, run the following query to determine if patches identified as requirements for this installation in the *Oracle Big Data SQL Master Compatibility Matrix* (Doc ID 2119369.1 in [My Oracle Support](#)) have been applied.

```
select PATCH_ID, PATCH_UID, VERSION, STATUS, DESCRIPTION from
DBA_REGISTRY_SQLPATCH order by BUNDLE_SERIES
```

For example, for Oracle Big Data SQL 3.0.1 with BP 12.1.0.2.160419 (22806133), the query should return these results.

PATCH_ID	PATCH_UID	VERSION	STATUS	DESCRIPTION
22806133	19983161	12.1.0.2	SUCCESS	DATABASE BUNDLE PATCH: 12.1.0.2.160419 (22806133)

If the query fails or the correct patch for the installed bundle is not found, see [2335899.2](#) in My Oracle Support for more information about troubleshooting Datapatch.

D.11 Debugging SQL.NET Issues

The following suggestions help to solve possible SQL.NET issues.

Misconfiguration of SQL.NET can result in blocked external procedure calls. If execution of these calls return ORA errors such as "ORA-28579: network error

during callback from external procedure agent," then check the following My Oracle Support note and confirm that your configuration is correct.

[Setting up Oracle Big Data SQL and Oracle Secure External Password Store \(2126903.1\)](#)

My Oracle Support note [1598247.1](#) describes the symptoms of the problem.

E

Oracle Big Data SQL Software Accessibility Recommendations

Oracle Big Data SQL includes tools such as `bdscli` and `ohsh` that you run from the command line. This section provides some tips on using screen readers and screen magnifiers with these tools.

E.1 Tips on Using Screen Readers and Braille Displays

These tips may help you make better use of screen readers and braille displays with Oracle Big Data SQL.

- Use a character mode based terminal such as Putty or Cygwin. Do not use an X-Windows-based VNC.
- For screen reader users, we recommend installing "screen" in order to get multiple session support. The Linux based screen program allows for multiple sessions in different windows. You can access each session with keyboard based commands, for example, `Ctrl-a`. Screen allows you to detach or re-attach to a given window session. Like VNC, if you get disconnected when running a session, you can re-attach to and resume that session.
- In the settings of the terminal software, set the cursor type to "block" cursor, not blinking or flashing.
- The output of the commands can generate a significant amount of information and might spill off the terminal window, and the virtual window or braille display. For example, the following command can generate a long alert history output:

```
bdscli list alerthistory
```

To display the output one screen-full at a time, pipe the output through the `more` command, as in the following:

```
bdscli list alerthistory | more
```

You can then use the space bar key to page through the output.

- A few recommended screen reader settings include the following (JAWS is used here just as an example):
 - Set the JAWS cursor to "All". Use the key combination of `Insert + s` until you hear "All".
 - You may need to turn off virtual cursor. If you are using JAWS, you can do this using the key combination of `Insert + z`.
 - Use the virtual window to capture text. If you are using JAWS, you can do this using the key combination of `Insert + Alt + w`.

E.2 Tips on Using Screen Magnifiers

- Screen magnifiers can support both character-based terminals and X-Window-based VNC.
- If you are using the screen reader function of a screen magnifier, then you should use a character-based terminal as described above.
- If you are using a VNC, decide your preference for a window display, for example, TWM or ICE. A display setting for ICE can be done with the following:

```
vncserver -geometry 1600x950 :2
```

1600x950 specifies the display size, and :2 specifies the VNC display number.

Index

A

access drivers, [1-2](#), [A-1](#)
ACCESS PARAMETERS clause
 special characters, [6-1](#)
 syntax rules, [6-1](#)
ACCESS PARAMETERS Clause
 syntax, [6-1](#)
ALL_HIVE_COLUMNS view, [6-26](#)
ALL_HIVE_DATABASES view, [6-24](#)
ALL_HIVE_TABLES view, [2-4](#), [6-25](#)
array overflows, [6-12](#)

B

bigdata_config directory, [2-38](#)
binary overflows, [6-12](#)

C

catalog views, [6-24](#)
Cell XT, [1-9](#)
character overflows, [6-12](#)
column mapping, [6-6](#)
com.oracle.bigdata.bufferize, [6-5](#)
com.oracle.bigdata.colmap, [6-6](#)
com.oracle.bigdata.datamode, [6-5](#)
com.oracle.bigdata.erroropt, [6-7](#)
com.oracle.bigdata.fields, [6-8](#)
com.oracle.bigdata.fileformat, [6-10](#)
com.oracle.bigdata.log.exec, [6-11](#)
com.oracle.bigdata.log.qc, [6-12](#)
com.oracle.bigdata.overflow, [6-12](#)
com.oracle.bigdata.rowformat, [6-13](#)
com.oracle.bigdata.tablename, [6-15](#)
CREATE TABLE ORGANIZATION EXTERNAL
 syntax, [2-1](#), [A-2](#)
CREATE TABLE statement
 generating automatically for Hive, [6-33](#)
CREATE_EXTDDL_FOR_HIVE function
 syntax, [6-33](#)

D

data dictionary views, [6-24](#)

data mode, [6-5](#)
data source name, [6-15](#)
data type conversion (Big Data SQL), [2-10](#)
data types (HDFS), [6-8](#)
DBA_HIVE_COLUMNS view, [6-27](#)
DBA_HIVE_DATABASES view, [6-27](#)
DBA_HIVE_TABLES view, [6-27](#)
DBMS_HADOOP package, [6-33](#)
DBMS_OUTPUT package, [2-5](#)
delimited text files, [6-13](#)

E

error handling, [6-7](#)
error handling (Big Data SQL), [2-36](#)
external tables
 about, [1-2](#), [A-1](#)

F

field extraction, [6-13](#)
field names, [6-8](#)

H

Hadoop log files, [6-5](#), [6-11](#)
Hive columns, [6-26](#)
Hive data
 access from Oracle Database, [2-3](#)
Hive databases, [6-24](#)
Hive table sources, [6-15](#)
Hive tables, [6-25](#)
Hive views, [6-24](#)

L

log files, [6-12](#)

O

Oracle Big Data SQL
 access drivers, [1-2](#)
 data type conversion, [2-10](#)
 general description, [1-1](#)

Oracle Big Data SQL (*continued*)
 installation changes on the Oracle Database
 server ., [2-38](#)

Oracle Database
 access to Hive data, [2-3](#)
 Data Modeler, [2-3](#)
 DBMS_HADOOP, [2-3](#)
 SQL Developer, [2-3](#)
 Use in Oracle Big Data SQL, [2-3](#)

Oracle Exadata Machine
 Big Data SQL installation changes, [2-38](#)

ORACLE_HDFS access driver, [2-18](#)

ORACLE_HIVE
 access parameters, [6-3](#)

ORC files, [6-10](#)

overflow handling, [6-12](#)

P

Parquet files, [6-10](#)

parsing HDFS files, [6-13](#)

PL/SQL packages, [6-33](#)

PUT_LINE function, [2-5](#)

R

RC files, [6-10](#)

row format description, [6-10](#)

row formats, [6-13](#)

S

sequence files, [6-10](#)

SerDe parsing, [6-13](#)

Smart Scan, [1-2](#)

SmartScan mode, [6-5](#)

source name, [6-15](#)

static data dictionary views, [6-24](#)

Statistics, [1-9](#)

struct overflows, [6-12](#)

T

text files, [6-10](#)

text overflows, [6-12](#)

U

union overflows, [6-12](#)

user access from Oracle Database, [2-36](#)

USER_HIVE_COLUMNS view, [6-28](#)

USER_HIVE_DATABASES view, [6-27](#)

USER_HIVE_TABLES view, [6-28](#)