

Oracle® Big Data SQL

User's Guide



Release 3.2.1.1

F38505-01

January 2021

ORACLE®

Copyright © 2012, 2021, Oracle and/or its affiliates.

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	viii
Related Documents	viii
Conventions	viii
Backus-Naur Form Syntax	ix
Changes in Oracle Big Data SQL 3.2.1.1	ix

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 HDFS	1-2
1.1.4 About Storage Indexes	1-3
1.1.5 About Predicate Push Down	1-5
1.1.6 About Oracle Big Data SQL Statistics	1-6
1.2 Installation	1-8

2 Using Oracle Big Data SQL for Data Access

2.1 Creating an Oracle External Table for Hive Data	2-1
2.1.1 Obtaining Information About a Hive Table	2-2
2.1.2 Using the CREATE_EXTDDL_FOR_HIVE Function	2-3
2.1.3 Using Oracle SQL Developer to Connect to Hive	2-4
2.1.4 Developing a CREATE TABLE Statement for ORACLE_HIVE	2-7
2.1.4.1 Using the Default ORACLE_HIVE Settings	2-7
2.1.4.2 Overriding the Default ORACLE_HIVE Settings	2-7
2.2 Creating an Oracle External Table for Oracle NoSQL Database	2-8
2.2.1 Creating a Hive External Table for Oracle NoSQL Database	2-8
2.2.2 Creating the Oracle Database Table for Oracle NoSQL Data	2-9
2.2.3 About Column Data Type Mappings	2-10
2.2.4 Example of Accessing Data in Oracle NoSQL Database	2-11
2.2.4.1 Creating the Oracle NoSQL Database Example Table	2-11

2.2.4.2	Creating the Example Hive Table for vehicleTable	2-12
2.2.4.3	Creating the Oracle Table for VEHICLES	2-13
2.3	Creating an Oracle External Table for Apache HBase	2-14
2.3.1	Creating a Hive External Table for HBase	2-14
2.3.2	Creating the Oracle Database Table for HBase	2-15
2.4	Creating an Oracle External Table for HDFS Files	2-15
2.4.1	Using the Default Access Parameters with ORACLE_HDFS	2-15
2.4.2	Overriding the Default ORACLE_HDFS Settings	2-16
2.4.2.1	Accessing a Delimited Text File	2-16
2.4.2.2	Accessing Avro Container Files	2-16
2.5	Creating an Oracle External Table for Kafka Topics	2-17
2.5.1	Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics	2-17
2.5.2	Creating an Oracle Big Data SQL Table for Kafka Topics	2-20
2.6	Using the Custom Parquet Reader for Oracle Big Data SQL	2-21
2.7	About the SQL CREATE TABLE Statement	2-22
2.7.1	Basic Syntax	2-23
2.7.2	About the External Table Clause	2-23
2.7.2.1	TYPE Clause	2-23
2.7.2.2	DEFAULT DIRECTORY Clause	2-23
2.7.2.3	LOCATION Clause	2-24
2.7.2.4	ORACLE_HDFS LOCATION Clause	2-24
2.7.2.5	ORACLE_HIVE LOCATION Clause	2-24
2.7.2.6	REJECT LIMIT Clause	2-24
2.7.2.7	ACCESS PARAMETERS Clause	2-25
2.8	About Data Type Conversions	2-25
2.9	Querying External Tables	2-26
2.9.1	Granting User Access	2-26
2.9.2	About Error Handling	2-26
2.9.3	About the Log Files	2-27
2.10	About Oracle Big Data SQL on the Database Server (Oracle Exadata Machine or Other)	2-27
2.10.1	About the bigdata_config Directory	2-27
2.10.2	Common Configuration Properties	2-27
2.10.2.1	bigdata.properties	2-27
2.10.2.2	bigdata-log4j.properties	2-29
2.10.3	About the Cluster Directory	2-30
2.10.4	About Permissions	2-30

3 Storing Oracle Data in Hadoop

3.1	Using Copy to Hadoop	3-1
-----	----------------------	-----

3.1.1	What Is Copy to Hadoop?	3-1
3.1.2	Getting Started Using Copy to Hadoop	3-2
3.1.2.1	Table Access Requirements for Copy to Hadoop	3-3
3.1.3	Using Oracle Shell for Hadoop Loaders With Copy to Hadoop	3-3
3.1.3.1	Introducing Oracle Shell for Hadoop Loaders	3-3
3.1.4	Using Copy to Hadoop to do Direct Copies	3-3
3.1.5	Using Copy to Hadoop to do Staged Copies	3-4
3.1.6	Querying the Data in Hive	3-5
3.1.7	About Column Mappings and Data Type Conversions	3-5
3.1.7.1	About Column Mappings	3-5
3.1.7.2	About Data Type Conversions	3-5
3.1.8	Working With Spark	3-6
3.1.9	Using Oracle SQL Developer with Copy to Hadoop	3-7
3.2	Storing Oracle Tablespaces in HDFS	3-7
3.2.1	Advantages and Limitations of Tablespaces in HDFS	3-8
3.2.2	About Tablespaces in HDFS and Data Encryption	3-9
3.2.3	Moving Tablespaces to HDFS	3-9
3.2.3.1	Using bds-copy-tbs-to-hdfs	3-10
3.2.3.2	Manually Moving Tablespaces to HDFS	3-14
3.2.4	Smart Scan for TableSpaces in HDFS	3-17

4 Oracle Big Data SQL Reference

4.1.1	DBMS_HADOOP PL/SQL Package	4-1
4.1.1.1	CREATE_EXTDDL_FOR_HIVE	4-1
4.1.1.1.1	Example	4-2
4.1.2	DBMS_BDSQL PL/SQL Package	4-3
4.1.2.1	ADD_USER_MAP	4-3
4.1.2.2	REMOVE_USER_MAP	4-5
4.1.2.3	Multi-User Authorization Security Table	4-5
4.1.3	CREATE TABLE ACCESS PARAMETERS Clause	4-7
4.1.3.1	Syntax Rules for Specifying Properties	4-7
4.1.3.2	ORACLE_HDFS Access Parameters	4-8
4.1.3.2.1	Default Parameter Settings for ORACLE_HDFS	4-8
4.1.3.2.2	Optional Parameter Settings for ORACLE_HDFS	4-8
4.1.3.3	ORACLE_HIVE Access Parameters	4-9
4.1.3.3.1	Default Parameter Settings for ORACLE_HIVE	4-9
4.1.3.3.2	Optional Parameter Values for ORACLE_HIVE	4-9
4.1.3.4	com.oracle.bigdata.bufferSize	4-10
4.1.3.5	com.oracle.bigdata.colmap	4-10
4.1.3.6	com.oracle.bigdata.datamode	4-11

4.1.3.7	com.oracle.bigdata.erroropt	4-12
4.1.3.8	com.oracle.bigdata.fields	4-13
4.1.3.9	com.oracle.bigdata.fileformat	4-15
4.1.3.10	com.oracle.bigdata.log.exec	4-16
4.1.3.11	com.oracle.bigdata.log.qc	4-17
4.1.3.12	com.oracle.bigdata.overflow	4-18
4.1.3.13	com.oracle.bigdata.rowformat	4-19
4.1.3.14	com.oracle.bigdata.tablename	4-20
4.1.4	Static Data Dictionary Views for Hive	4-21
4.1.4.1	ALL_HIVE_DATABASES	4-22
4.1.4.2	ALL_HIVE_TABLES	4-22
4.1.4.3	ALL_HIVE_COLUMNS	4-23
4.1.4.4	DBA_HIVE_DATABASES	4-24
4.1.4.5	DBA_HIVE_TABLES	4-24
4.1.4.6	DBA_HIVE_COLUMNS	4-24
4.1.4.7	USER_HIVE_DATABASES	4-25
4.1.4.8	USER_HIVE_TABLES	4-25
4.1.4.9	USER_HIVE_COLUMNS	4-25

Part I Appendices

C Using mtactl to Manage the MTA extproc

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

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-12
D.6	Step-by-Step Process for Querying an External Table	D-13
D.7	Step-by-Step for a Hive Data Dictionary Query	D-16
D.8	Key Administration Tasks for Oracle Big Data SQL	D-17
D.9	Additional Java Diagnostics	D-20
D.10	Checking for Correct Oracle Big Data SQL Patches	D-20

E Licensing Information

E.1	Oracle Big Data SQL Licensing	E-1
E.1.1	ANTLR 4.7	E-1
E.1.2	Apache Commons Exec 1.3	E-2
E.1.3	Apache Licensed Code	E-2
E.1.4	Apache License	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 sufficient background knowledge about the database server and the particular Hadoop platform which will host the software in order to follow the instructions successfully.

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 Oracle Big Data SQL 3.2.1.1

Oracle Big Data SQL Release 3.2.1.1 includes the following changes.

New Database Compatibility Parameter for the Jaguar Installer

The configuration file that is passed to the Jaguar installer now accepts an optional `database_compatibility` parameter:

This parameter gives you a way to save on system resources (particularly memory) if all databases connected to a cluster are of the same version of Oracle Database. Oracle Big Data SQL provides a different offloader process for each supported version of Oracle Database. In this release, there are offloader process for Oracle Database 12.1 and 12.2. By default, both are running. If you only need Oracle 12.1 connections or 12.2 connections, but not both, then you can use `database_compatibility` to prevent start up of the offloader process that you do not need.

This parameter is within the `cluster` section of the configuration file. We set compatibility to 12.1 in this example:

```
"cluster" : {
  "database_compatibility" : [ "12.1" ]
}
```

Like any other configuration parameter, you can change this later by running a `jaguar reconfigure` operation.

Installation Differences on Exadata Database Machine RAC Systems and non-Exadata Database Machine RAC Systems



Note:

The following apply *only* to Oracle Big Data SQL 3.2.1.1 and only for installations on RAC Systems running on commodity hardware (RAC systems that are not Exadata Database Machines).

- Only Ethernet is supported for connections between Oracle Database 12.2 and the Hadoop cluster.
- There are special steps specific to Oracle Big Data SQL 3.2.1.1 that you must follow the first time that you install the database side of the software on the database nodes. The details are provided in Steps for Installing on Oracle Database Nodes in the installation guide.

Known Limitations Reported in Release 3.2.1 That are Resolved in 3.2.1.1

- JSON CLOB predicate pushdown is working in Release 3.2.1.1
- In Release 3.2.1, storage indexes for an Oracle Big Data SQL table may not be utilized where the following DDL operation is performed:

```
ALTER TABLE table_name PROJECT COLUMN REFERENCED;
```

This is also fixed in Release 3.2.1.1.

1

Introducing Oracle Big Data SQL

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, Oracle NoSQL Database, 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.

Since data in the Hadoop HDFS file system 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, notably `InputFormat` and `SerDe` Java classes, optionally through Hive metadata definitions. The Oracle Big Data SQL processing cells on the DataNodes are a layer on top of this generic Hadoop infrastructure. Three key features provided by the cells are *Smart Scan* and *Storage Indexes*, which are described in this chapter.

See Also:

The Oracle Big Data SQL Installation Guide describes how to install and configure the software on the two sides of an Oracle Big Data SQL configuration.

The following sections of this guide provide details on using Oracle Big Data SQL:

- [About Oracle External Tables](#)
- [About the Access Drivers for Oracle Big Data SQL](#)

- [About Smart Scan for HDFS](#)
- [About Storage Indexes](#)
- [About Predicate Push Down](#)
- [About Oracle Big Data SQL Statistics](#)

1.1.1 About Oracle External Tables

Oracle Big Data SQL provides external tables with next generation 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.

External tables use **access drivers** to parse the data outside the database. Each type of external data requires a unique access driver. Oracle Big Data SQL includes two access drivers for big data: one for data that has metadata defined in Apache Hive, and the other for accessing data stored in the Hadoop Distributed File System, with metadata specified only by an Oracle administrator.

1.1.2 About the Access Drivers for Oracle Big Data SQL

By querying external tables, you can access data stored in HDFS and Hive tables as if that data was stored in tables in an Oracle database. Oracle Database accesses the data by using the metadata provided when the external table was created.

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.
- `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 Big Data SQL uses these access drivers to optimize query performance.

1.1.3 About Smart Scan for HDFS

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.

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 [Storing 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

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 the 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 the external tables that use StorageHandlers, such as Apache HBase and Oracle NoSQL.

 **Note:**

In Release 3.2.1.1, Storage Indexes for an Oracle Big Data SQL table may not be utilized where the following DDL operation is performed:

```
ALTER TABLE table_name PROJECT COLUMN REFERENCED;
```

To correct this problem, recreate the external table.

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.

Table				Index	
A	B	C	D		
	1			}	Min B = 1 Max B = 5
	3				
	5				
	5			}	Min B = 3 Max B = 8
	8				
	3				

I/O eliminated by using storage index

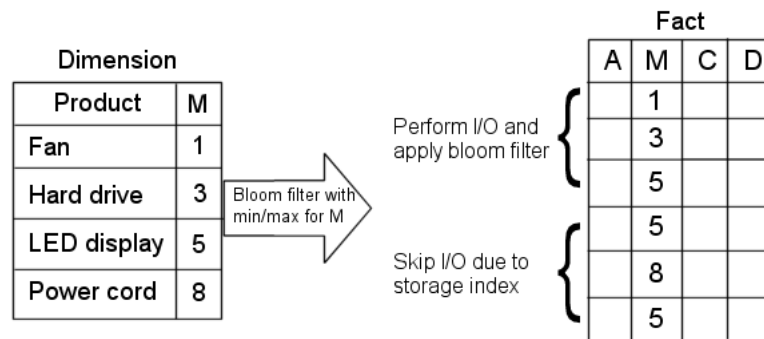
For a query such as the one below, only the first set of rows match. Disk I/O is eliminated because the minimum and maximum of the second set of rows do not match the WHERE clause of the query.

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

Example 1-2 Improved Join Performance Using Storage Indexes

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.

- Queries against Oracle NoSQL Database or Apache HBase use SARGable predicates to drive subscans 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 \geq m
VARCHAR(m)	CHAR(n), VARCHAR2(n) where n is \geq 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 \geq 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 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
```

 **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:

- [Oracle Big Data SQL Installation Guide](#)

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

This chapter describes how to use Oracle Big Data SQL to create external tables and access data from Hadoop data sources as well as Oracle NoSQL Database.

It also describes some of the changes that Oracle Big Data SQL makes on the Oracle Database server.

- [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](#)
- [About the SQL CREATE TABLE Statement](#)
- [About Data Type Conversions](#)
- [Querying External Tables](#)
- [About Oracle Big Data SQL on the Database Server \(Oracle Exadata Machine or Other\)](#)

2.1 Creating an Oracle External Table for Hive Data

Oracle Big Data SQL enables you to query Hive 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 Hive data, leverage robust Oracle Database security features, and take advantage of advanced SQL capabilities like analytic functions, JSON handling, and others.

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.

- **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.
- **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](#)

2.1.1 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 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](#)"

2.1.3 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. Release 4.2 is recommended because it includes support 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 JDBC 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. Navigate to the folder where you downloaded and extracted Oracle SQL Developer.
2. Click the `sqldeveloper` subfolder. Then, click `sqldeveloper.exe` in this folder.
3. In the SQL Developer menu bar, select **Tools>Preferences**.
4. In the file explorer of the **Preferences** window, expand **Database** and then click **Third Party JDBC Drivers**.
5. Click **Add Entry**.
6. 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**.
7. 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 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.

2.1.4 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.

2.1.4.1 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.

2.1.4.2 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`.

2.2 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 Column Data Type Mappings](#)
- [Example of Accessing Data in Oracle NoSQL Database](#)

2.2.1 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 2-1](#) 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>

2.2.2 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](#)".

2.2.3 About Column Data 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 2-1](#) identifies the supported data types Oracle NoSQL Database table API and their mappings to Hive.

Table 2-1 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:**

[About 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](#).

2.2.4 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](#)

2.2.4.1 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 2-2 Fields in the vehicleTable Example

Field Name	Data Type
type	STRING
make	STRING
model	STRING
class	STRING
color	STRING
price	DOUBLE
count	INTEGER

2.2.4.2 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,some1node04.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
```

2.2.4.3 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

2.3 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](#)".

2.3.1 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>

2.3.2 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"](#).

2.4 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.

2.4.1 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.

2.4.2 Overriding the Default ORACLE_HDFS Settings

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

2.4.2.1 Accessing a Delimited Text File

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

```
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.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/*');
```

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

- com.oracle.bigdata.colmap: Handles differences in column names. ORDER_LINE_ITEM_COUNT in the HDFS files matches 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.
- com.oracle.bigdata.erroropt: Replaces bad data. Errors in the data for CUST_NUM or ORDER_NUM set the value to INVALID_NUM.

2.4.2.2 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. The files are not in delimited text format.
- `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.

2.5 Creating an Oracle External Table for Kafka Topics

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

The ORACLE_HIVE access driver can access Kafka data topics. You first create a Hive external table that accesses the Kafka topics and then create an Oracle Big Data SQL table over it.

2.5.1 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 partitionid=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 2-3 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	<p>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".</p> <p>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.</p>
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 2-3 (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 <code>oracle.kafka.table.key.writer.schema</code> 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 <code>oracle.kafka.table.key.writer.schema</code> .
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 <code>oracle.kafka.table.key.schema</code> .
oracle.kafka.table.extra.columns	Optional, default to "true"	A boolean flag to control whether to include extra Kafka columns: <code>partitionid, offset, timestamptype</code> .
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 <code>oracle.kafka.chop.partition</code> 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.

2.5.2 Creating an Oracle Big Data SQL Table for Kafka Topics

Big Data SQL can use the ORACLE_HIVE access driver to query data stored in Hive Tables.

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 default ORACLE_HIVE settings can be overridden in the same way as with other Hive tables. This is how to query the Hive external table that was created using the Hive storage handler for Kafka in the previous section.

```
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](#)

2.6 Using the Custom Parquet Reader for Oracle Big Data SQL

For reading parquet files, you have the option of using the custom Parquet reader for Oracle Big Data SQL.

The custom reader includes an optimization that reduces I/O – lazy retrieval and materialization of non-filter predicate columns. When evaluating a filter for a row-group (about 128 MB of data), the custom reader first retrieves only the columns for the filter predicate. The filter is applied and if there is no match then the reader moves to the next row group. If some row matches then the non-filter columns are read. Filter evaluation builds a data structure to efficiently de-serialize (materialize) only the values

for the matching column indexes. This differs from the Hive implementation which reads AND de-serializes (materializes) all the columns in the select list, then evaluates the filter for the predicate columns, then finally assembles rows from the matching values. This optimization reduces I/O whenever there are no matching rows for the filter in the row group. In this case, the non-predicate columns are not read at all.

The custom reader also reduces CPU consumption in two ways:

- Skips decompression of individual Parquet pages (the minimal read unit) when those pages contain no values that satisfy the filter. This applies to primitive column types only, not to nested column types.
- Skips the de-serialization of individual column values if they do not belong to a matching row index.

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 customer 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 use the parquet tools JAR to dump information about the file:

```
# hadoop jar parquet-tools-1.5.0-cdh5.12.0.jar meta <filename>.parq
```

On a CDH Hadoop distro, the `parquet-tools` command may also be configured in your path.

2.7 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.

2.7.1 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](#)".

2.7.2 About the External Table Clause

CREATE TABLE ORGANIZATION EXTERNAL takes the *external_table_clause* as its argument. It has the following subclauses:

- [TYPE Clause](#)
- [DEFAULT DIRECTORY Clause](#)
- [LOCATION Clause](#)
- [REJECT LIMIT Clause](#)
- [ORACLE_HIVE Access Parameters](#)

2.7.2.1 TYPE Clause

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.

 **Note:**

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

2.7.2.2 DEFAULT DIRECTORY Clause

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

2.7.2.3 LOCATION Clause

The `LOCATION` clause identifies the data source.

2.7.2.4 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 {a, b, c}
<code>[a-b]</code>	Matches one character in the range {a...b}. The character must be less than or equal to b.
<code>[^a]</code>	Matches one character that is not in the character set or range {a}. The carat (^) must immediately follow the left bracket, with no spaces.
<code>\c</code>	Removes any special meaning of c. The backslash is the escape character.
<code>{ab\,cd}</code>	Matches a string from the set {ab, cd}. The escape character (\) removes the meaning of the comma as a path separator.
<code>{ab\,c{de\,fh}</code>	Matches a string from the set {ab, cde, cfh}. The escape character (\) removes the meaning of the comma as a path separator.

2.7.2.5 ORACLE_HIVE LOCATION Clause

Do not specify the `LOCATION` clause for `ORACLE_HIVE`; it raises an error. The data is stored in Hive, and the access parameters and the metadata store provide the necessary information.

2.7.2.6 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.

2.7.2.7 ACCESS PARAMETERS Clause

The `ACCESS PARAMETERS` clause provides information that the access driver needs to load the data correctly into the external table. See "[CREATE TABLE ACCESS PARAMETERS Clause](#)".

2.8 About 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 2-5 Supported Hive to Oracle Data Type Conversions

Hive Data Type	VARCHAR2, CHAR, NCHAR2, NCHAR, CLOB	NUMBER, FLOAT, BINARY_INTEGER, BINARY_FLOAT	BLOB	RAW	DATE, TIMESTAMP, TIMESTAMP WITH TZ, TIMESTAMP WITH LOCAL TZ	INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
INT	yes	yes	yes	yes	no	no
SMALLINT						
TINYINT						
BIGINT						

Table 2-5 (Cont.) 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
DOUBLE	yes	yes	yes	yes	no	no
FLOAT						
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	yes	no	no	no	no	no
ARRAY						
UNIONTYPE						
MAP						

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

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

2.9 Querying External Tables

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

2.9.1 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](#)".

2.9.2 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.

2.9.3 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](#)

2.10 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](#)

2.10.1 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`.

2.10.2 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.

2.10.2.1 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.
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>

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.

Property	Description
LD_LIBRARY_PATH	A colon separated (:) list of directory paths to search for the Hadoop native libraries. Recommended. If you set this option, then do not set <i>java.library</i> path in java.options .

[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/*:$ORACLE_HOME/hadoop/jlib/
hver-2/*:$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
```

2.10.2.2 `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>

2.10.3 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.

2.10.4 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.

3

Storing Oracle Data in Hadoop

Copy to Hadoop and Oracle Database Tablespaces in HDFS are two Oracle Big Data SQL resources for off-loading Oracle Database tables to the HDFS file system on a Hadoop cluster.

The table below compares these two tools.

Table 3-1 Comparison of Copy to Hadoop and Oracle Tablespaces in HDFS

Copy to Hadoop	Oracle Tablespaces in HDFS
Copies Oracle Database tables to Oracle Data Pump files stored in HDFS.	Oracle Database tables or partitions are stored within the tablespace in HDFS in their original Oracle-internal format.
Access is through a Hive external table and from the database with Oracle Big Data SQL.	Access is directly through the original Oracle Database tables. External tables are not needed.
Data is available (through Hive) to other processes in the Hadoop ecosystem and to Oracle Database (through Oracle Big Data SQL).	Data is directly available to Oracle Database only. Data is not accessible to other processes in Hadoop.

3.1 Using Copy to Hadoop

This section describes 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](#)
- [Using Copy to Hadoop to do Direct Copies](#)
- [Using Copy to Hadoop to do Staged Copies](#)
- [Querying the Data in Hive](#)
- [About Column Mappings and Data Type Conversions](#)
- [Working With Spark](#)

3.1.1 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.

3.1.2 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` or `TIMESTAMP` 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.

3.1.2.1 Table Access Requirements for Copy to Hadoop

To copy 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 has one of the following privileges to access a table in another schema:
 - The `select_catalog_role` privilege (which provides `SELECT` privileges on data dictionary views).
 - The `SELECT` privilege on the table.

3.1.3 Using Oracle Shell for Hadoop Loaders With Copy to Hadoop

3.1.3.1 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 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:

- To set up OHS, follow the instructions in the Oracle Big Data SQL Installation Guide.
- The examples directory in the OHS kit contains many examples that define resources and load data using Oracle Shell for Hadoop Loaders. See `<OHS_KIT>/examples/README.txt` for a description of the examples and instructions on how to run OHS load methods.

3.1.4 Using Copy to Hadoop to do Direct Copies

This example shows how to use Oracle Shell for Hadoop Loaders (OHSB) to do a direct, one step copy from Oracle Database to Hadoop.

The example assumes OHSB and Oracle Big Data SQL have been installed and configured, and that the examples have been configured by following the instructions in the `README.txt` file found in the `examples` directory of the OHSB installation. The example scripts can be found in the `examples` directory of the OHSB installation.

Example 3-1 `createreplace_directcopy.ohsh`

This script uses `create` or `replace` to create a Hive table called `cp2hadoop_fivdti` from the Oracle table `FIVDTI` and then loads the Hive table with 10000 rows. It uses the `directcopy` command to run a map job on Hadoop and split the Oracle table into input splits. It then creates Data Pump format files in HDFS that include all the splits, and creates a Hive external table that maps to the Data Pump format files.

```
create or replace hive table hive0:cp2hadoop_fivdti \  
from oracle table olhp:fivdti using directcopy
```

The `load_directcopy.ohsh` script shows how to load the Hive table with an additional 30 rows using the `directcopy` command.

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

Optionally, the data can be converted to Parquet or ORC formats from OHSB:

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

3.1.5 Using Copy to Hadoop to do Staged Copies

This example shows how to use Oracle Shell for Hadoop Loaders (OHSB) to do a staged, two-step copy from Oracle Database to Hadoop.

This example assumes OHSB and Big Data SQL 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 OHSB installation. The scripts below and many others are also available in the `examples` directory.

Example 3-2 `createreplace_stage.ohsh`

This script uses `create` or `replace` to create a Hive table called `cp2hadoop_fivdti` from the Oracle table `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 olhp:fivdti using stage
```

Example 3-3 load_stage.ohsh

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

```
load hive table hive0:cp2hadoop_fivdti from oracle table olhp: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](#).

3.1.6 Querying the Data in Hive

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

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

3.1.7 About Column Mappings and Data Type Conversions

3.1.7.1 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.

3.1.7.2 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

Table 3-2 (Cont.) Oracle to Hive Data Type Conversions

Oracle Data Type	Hive Data Type
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	

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.

3.1.8 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,orail8n.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>'")
```

3.1.9 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 to do the basic installation.

3.2 Storing 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.

3.2.1 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.

3.2.2 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.

3.2.3 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-3.2.1.1/<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.

3.2.3.1 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.

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

Parameter List	Description
--install	Installs the FUSE-DFS service. No action is taken if the service is already installed.
--uninstall	Uninstalls the FUSE-DFS service and removes the mountpoint.
--grid-home	Specifies the Oracle Grid home directory.
--base-mountpoint	By default, the mountpoint is under /mnt. However, on some systems access to this directory is restricted. This parameter lets you specify an alternate location.
--aux-run-mode	<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 bds-copy-tbs-to-hdfs.sh is paused. The --aux-run-mode parameter specifies a mode for running these auxiliary scripts.</p> <p>--aux-run-mode=<mode></p> <p>Mode options are:</p> <ul style="list-style-type: none"> • session — through a spawned session. • su — as a substitute user. • sudo — through sudo. • ssh — through secure shell.

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

Parameter List	Description
<code>--force-uninstall-script</code>	This option creates a secondary script that runs as root and forces the FUSE-DFS uninstall.
<div>  Caution: </div> <p>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.</p>	
<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=mytablespace --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 workaround, you can use the `--staging-dir` parameter, which that 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.

3.2.3.2 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.

6. 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.

7. 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;
```

3.2.4 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;
```

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.

Oracle Big Data SQL Reference

This chapter contains reference information for Oracle Big Data SQL:

- [DBMS_HADOOP PL/SQL Package](#)
- [DBMS_BDSQL PL/SQL Package](#)
- [CREATE TABLE ACCESS PARAMETERS Clause](#)
- [Static Data Dictionary Views for Hive](#)

4.1.1 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](#)

4.1.1.1 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 4-1 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.

Table 4-1 (Cont.) CREATE_EXTDDL_FOR_HIVE Function Parameters

Parameter	Description
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.
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 "[About Data Type Conversions](#)".

4.1.1.1.1 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
```

4.1.2 DBMS_BDSQL PL/SQL Package

The DBMS_BDSQL PL/SQL package contains procedures to add and remove a user map.

- [ADD_USER_MAP](#)
- [REMOVE_USER_MAP](#)

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.

4.1.2.1 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 4-2 ADD_USER_MAP Parameters

Parameter	Description
cluster_name	The name of the Hadoop cluster where the map will be applied. [DEFAULT] as cluster name designates the default cluster.
current_database_user	The current effective database user. This is what Oracle uses to check for authority. A value of '*' indicates that this row to be used if no other rows fit the criteria. There is no default and the value may not be NULL.
syscontext_namespace	<p>Note that for the Oracle USERENV namespace, the only allowed values are GLOBAL_UID, CLIENT_IDENTIFIER, and AUTHENTICATED_IDENTITY.</p> <p>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, AUTHENTICATED_IDENTITY 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 dirkrb will be used for authorization on the Hadoop cluster as the authenticated identity of dirkrb@HQ.TEST1.DBSEC2008.COM.</p>
syscontext_parm_hadoop_user	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.

4.1.2.2 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.

4.1.2.3 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 4-3 SYS.BDSQL_USER_MAP

Column	Datatype	Description
cluster_name	varchar2	Name of the Hadoop cluster. The default is [DEFAULT].
current_database_user	varchar2	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.
syscontext_namespace	varchar2	This is the optional specification for the Oracle SYS_CONTEXT namespace. if customer security is set up. Note that for the Oracle USERENV namespace, the only allowed values are: 'GLOBAL_UID', 'CLIENT_IDENTIFIER', 'AUTHENTICATED_IDENTITY'.
syscontext_parameter_hadoop_user	varchar2	<p>This column value has alternate interpretations.</p> <ul style="list-style-type: none"> If syscontext_namespace has a value, then syscontext_parameter_hadoop_user refers to the parameter that is specific to syscontext_namespace. However, when the value is '*', this is a directive to use the value of current_database_user for impersonation. The syscontext_namespace column must be NULL in this case. If syscontext_namespace is NULL, then syscontext_parameter_hadoop_user 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_user	syscontext_namespace	syscontext_param_hadoop_user
[DEFAULT]	*	USERENV	AUTHENTICATED_IDENTITY

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_user	syscontext_namespace	syscontext_param_hadoop_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_user	syscontext_namespace	syscontext_param_hadoop_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_user	syscontext_namespace	syscontext_parameter_hadoop_user
hadoop_cl_1	SCOTT		hdpsr1
hadoop_cl_1	ADAM	lowprivuser	hdpsr2
hadoop_cl_1	*		

4.1.3 CREATE TABLE ACCESS PARAMETERS Clause

This section describes the properties that you use when creating an external table that uses the `ORACLE_HDFS` or `ORACLE_HIVE` 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](#)
- Alphabetical list of properties

4.1.3.1 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 4-4](#) describes the special characters.

Table 4-4 Special Characters in Properties

Escape Sequence	Character
\b	Backspace (\u0008)
\t	Horizontal tab (\u0009)
\n	Line feed (\u000a)
\f	Form feed (\u000c)
\r	Carriage return (\u000d)
\"	Double quote (\u0022)
\'	Single quote (\u0027)
\\	Backslash (\u005c) 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.
\uxxxx	2-byte, big-endian, Unicode code point. When a character requires two code points (4 bytes), the parser expects \u for the second code point.

4.1.3.2 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.

4.1.3.2.1 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"}
```

4.1.3.2.2 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 4-1 shows a CREATE TABLE statement in which multiple access parameters are set.

Example 4-1 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/*'));
```

4.1.3.3 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.

4.1.3.3.1 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"}
```

4.1.3.3.2 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 4-2 shows a `CREATE TABLE` statement in which multiple access parameters are set.

Example 4-2 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"}]
 ));
```

4.1.3.4 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
```

4.1.3.5 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.

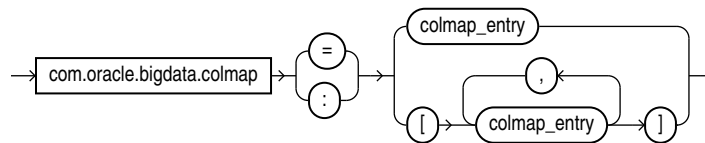
Default Value

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"}]
```

4.1.3.6 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.

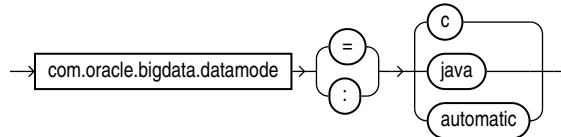
Default Value

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.

4.1.3.7 com.oracle.bigdata.erroropt

Describes how to handle errors that occur while the value of a column is calculated.

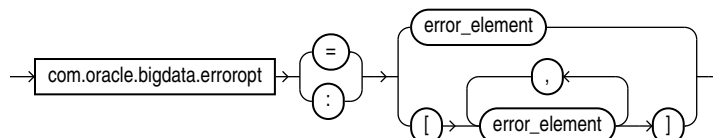
Default Value

```
{"action": "setnull"}
```

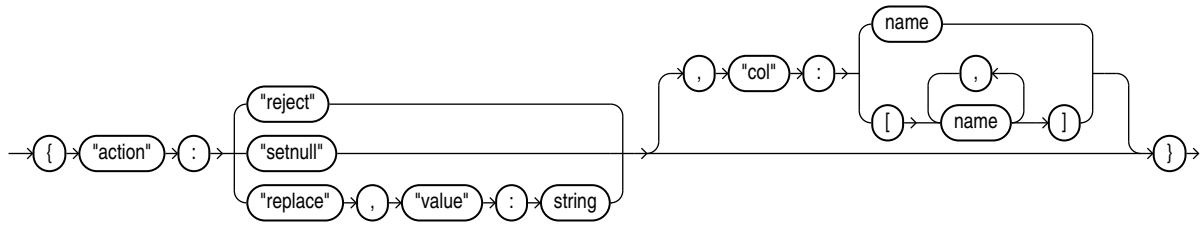
Syntax

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" ] }
```

4.1.3.8 com.oracle.bigdata.fields

Lists the field names and data types of the data source.

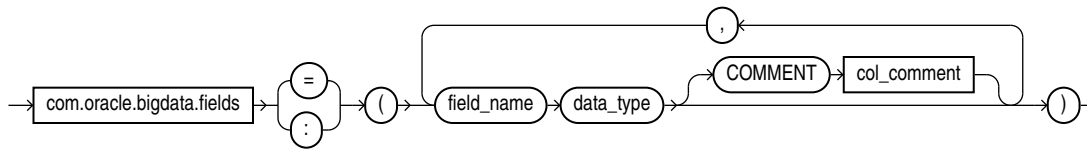
Default Value

Not defined

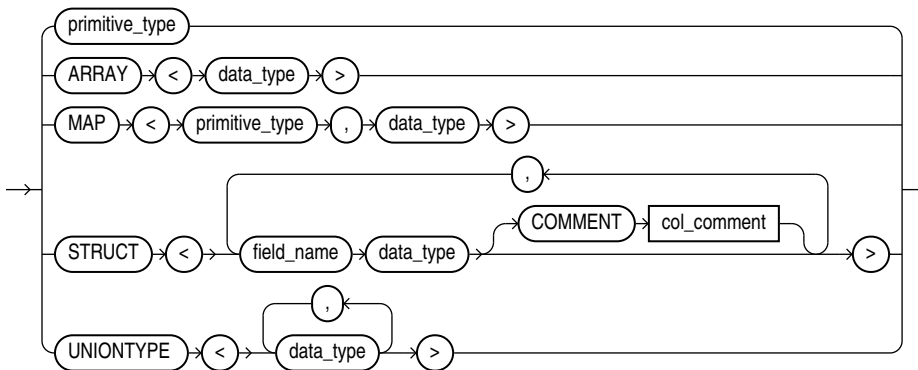
Syntax

A JSON document with the keyword-value pairs is shown in the following diagram:

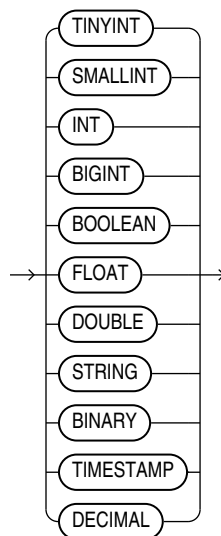
fields:



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`).

4.1.3.9 `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`.

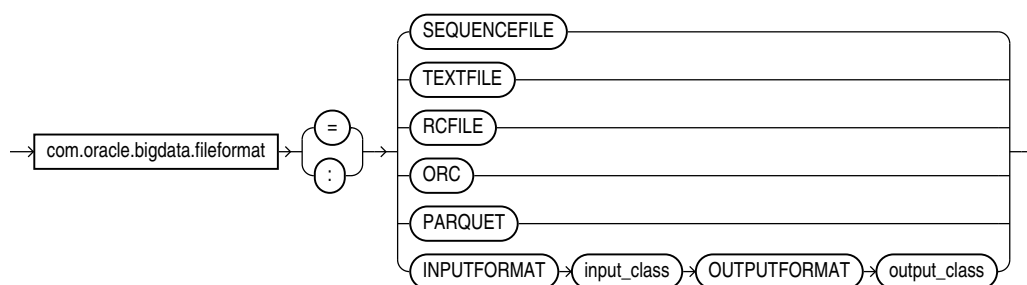
Default Value

`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

4.1.3.10 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. [Table 4-4](#) describes the optional variables that you can use in the template.

Table 4-5 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
```

4.1.3.11 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 4-6](#) describes the optional variables that you can use in the string.

Table 4-6 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_%%
```

4.1.3.12 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.

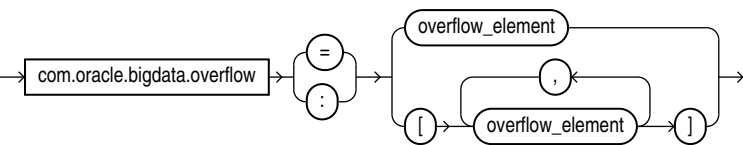
Default Value

```
{"action": "error"}
```

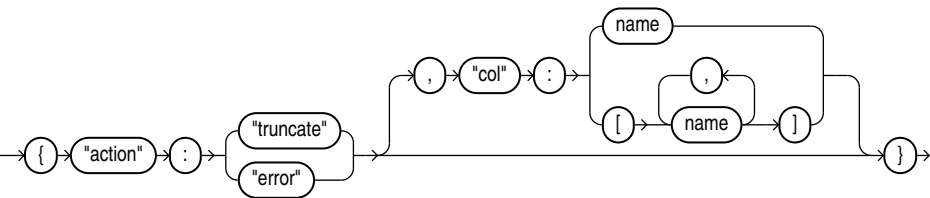
Syntax

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"}
```

4.1.3.13 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.

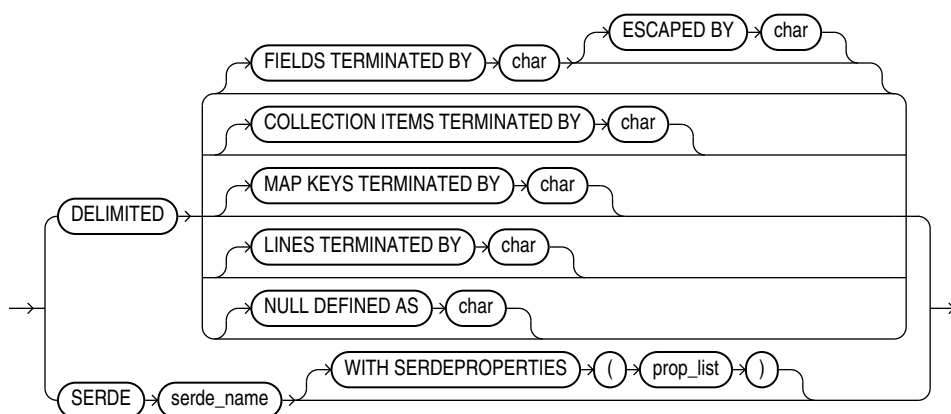
Default Value

`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" = "(\{6\}) (\{5\}) (\{29\}) .*)"
```

4.1.3.14 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
```

4.1.4 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](#)

4.1.4.1 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"](#)

4.1.4.2 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)		

Column	Datatype	NULL	Description
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
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"](#)

4.1.4.3 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	Datatype	NULL	Description
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"](#)

4.1.4.4 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"](#)

4.1.4.5 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"](#)

4.1.4.6 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"](#)

4.1.4.7 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"](#)

4.1.4.8 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"](#)

4.1.4.9 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"](#)

Appendices

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 start <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
```

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](#)
- [Copying the Files to HDFS](#)

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 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 are power users of Hadoop with specialized requirements . All other users should use the Oracle Shell for Hadoop Loader (OHS) 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 [About Column Mappings and Data Type Conversions](#).)
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="myuser" \  
-D oracle.hadoop.ctoh.connection.password="mypassword" \  
-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 Section “Configuration Properties 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 <code>".dp"</code>, 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 <code>"NAME,MANAGER"</code> 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"> • <code>oracle.hadoop.ctoh.connection.tnsAdmin</code> • <code>oracle.hadoop.ctoh.connection.tnsEntry</code>
oracle.hadoop.ctoh.connection.clusterWalletLoc	<p>Type:</p> <p>Default Value: Value of <code>oracle.hadoop.ctoh.connection.walletLoc</code>.</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"> • <code>oracle.hadoop.ctoh.connection.clusterTnsAdmin</code> • <code>oracle.hadoop.ctoh.connection.tnsEntry</code>

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	<p>Type: String.</p> <p>Default Value: <code>\${mapreduce.output.fileoutputformat.outputdir}../ctohCache</code></p> <p>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.</p>

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
<code>-s, --sid=[ORACLE_SID]</code>	The SID of the Oracle Database server.	Required only if the ORACLE_SID environment variable is not defined on the Oracle Database server where bdschecksw is executed.
<code>-g, --gihome=Grid_Infrastructure_home Oracle_Database_node_IP_address</code>	The Grid Infrastructure path on the Oracle Database server.	Required only if the GI_HOME environment variable is not defined on the Oracle Database server where bdschecksw is executed.
<code>-y, --giuser Oracle_Database_node_IP_address Grid_Infrastructure_home</code>	GI_HOME administrator name or owner (OS user name) of GI_HOME.	
<code>-q, --sqlplus Oracle_Database_node_IP_address username</code>	SQLPlus username on the Oracle Database server. The user is prompted for the password.	Required.
<code>-c, --cell DNS short name [...n]</code>	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.
<code>-u, --uname Hadoop_cluster_node_username</code>	Credentials to run remote commands on the Hadoop cluster from the Oracle Database server. This is usually the oracle 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

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

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">• config directory – configuration files downloaded for Cloudera Manager or Ambari.• fuse directory – settings for the FUSE-DFS service for the this cluster.• hadoop, hbase , and hive soft links to the actual client directories. (For example: hadoop-2.6.0-cdh5.12.0,hbase-1.2.0-cdh5.12.0, hive-1.1.0-cdh5.12.0, although the versions installed may different for your system.)• *.conf – IPsec configuration file.• *.keytab – 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 <code>hadoop fs</code> 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 3.2.1.1 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>\$ 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 bdsql detail
Log files	<ul style="list-style-type: none"> /var/log/bigdatasql/DM – installer log files /var/log/bigdatasql/cloudera or /var/log/bigdatasql/ambari – Ambari or CM service log files. /opt/oracle/bigdatasql/bdcell- <cell version>/bigdata.properties /opt/oracle/bigdatasql/bdcell- <cell version>/bigdata- log4j.properties <ul style="list-style-type: none"> This defaults to logging off. Change tolog4j.logger.oracle.hadoop.sql=INFO and restart. /opt/oracle/bigdatasql/bdcell- <cell version>/log directory <ul style="list-style-type: none"> bigdata-log4j.log – logs entries from the JVM pieces of Big Data SQL (logging defaults to off, so edit bigdata-log4j.properties first and restart). This can be particularly useful information. /var/log/oracle/diag/bdsql/cell/ <hostname>/trace/ – general cell trace files for the Management Server, Restart Server, and Monitor Server. The alert.log file will have details about quarantine and de-quarantine events. /var/log/oracle/diag/bdsql/ cell/SYS_*/trace/ – Oracle Big Data SQL offload server trace files for the C portion. These are not useful for troubleshooting in most cases.
Other datanode artifacts	/opt/oracle/cell/cellsrv/ deploy/config/cellinit.ora – records the cell's IP address.

D.6 Step-by-Step Process for Querying 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. `extproc_bds_<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.
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

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

- 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`):

```
$ bdaccli stop big_data_sql_cluster
$ bdaccli 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 `extproc_bds_*` process.)
11. Look at database `trace_file`:

```
select value from v$diag_info WHERE name = 'Default Trace File';
```

D.9 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 `extproc_bds_<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 [1609718.1](#) in My Oracle Support for more information about issues with Datapatch.

E

Licensing Information

E.1 Oracle Big Data SQL Licensing

The licensing for Oracle Big Data SQL is separate from the licensing for other Oracle products.

When you purchase licensing for Oracle Big Data SQL, note the following:

- A separate license must be procured for each Hadoop cluster.
- All data nodes within the Hadoop cluster must be licensed. Partial licensing within a node is not available.
- All disks accessed by Oracle Big Data SQL (either within or outside of the Hadoop cluster) must be licensed. This includes disks used by Kafka or NoSQL stores when Oracle Big Data SQL accesses those sources.
- There is no additional license required for the Oracle Database server side.
- Oracle Copy to Hadoop licensing is included.

Third Party Licensing for Oracle Shell for Hadoop Loaders

Oracle Shell for Hadoop Loaders is included in the Oracle Big Data SQL installation bundle. The following are third-party projects contained in Oracle Shell for Hadoop Loaders.

- ANTLR 4.7
- Apache Commons Exec 1.3

Unless otherwise specifically noted, or as required under the terms of the third party license (e.g., LGPL), the licenses and statements herein, including all statements regarding Apache-licensed code, are intended as notices only.

E.1.1 ANTLR 4.7

Copyright (c) 2015 Terence Parr, Sam Harwell

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

E.1.2 Apache Commons Exec 1.3

Include the following License ONLY ONCE in the documentation even if there are multiple products licensed under the license.

The following applies to all products licensed under the Apache 2.0 License:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License").

You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. A copy of the license is also reproduced below.

E.1.3 Apache Licensed Code

The following is included as a notice in compliance with the terms of the Apache 2.0 License, and applies to all programs licensed under the Apache 2.0 license:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

A copy of the license is also reproduced below.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

E.1.4 Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-

charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - b. You must cause any modified files to carry prominent notices stating that You changed the files; and
 - c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions)

on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>) (listed below):

Index

A

access drivers, [1-2](#), [A-1](#)
ACCESS PARAMETERS clause, [2-25](#)
 special characters, [4-8](#)
 syntax rules, [4-7](#)
ACCESS PARAMETERS Clause
 syntax, [4-7](#)
ALL_HIVE_COLUMNS view, [4-23](#)
ALL_HIVE_DATABASES view, [4-22](#)
ALL_HIVE_TABLES view, [2-2](#), [4-22](#)
array overflows, [4-18](#)

B

bigdata_config directory, [2-27](#)
binary overflows, [4-18](#)

C

catalog views, [4-21](#)
Cell XT, [1-6](#)
character overflows, [4-18](#)
column mapping, [4-10](#)
com.oracle.bigdata.bufferize, [4-10](#)
com.oracle.bigdata.colmap, [4-10](#)
com.oracle.bigdata.datamode, [4-11](#)
com.oracle.bigdata.erroropt, [4-12](#)
com.oracle.bigdata.fields, [4-13](#)
com.oracle.bigdata.fileformat, [4-15](#)
com.oracle.bigdata.log.exec, [4-16](#)
com.oracle.bigdata.log.qc, [4-17](#)
com.oracle.bigdata.overflow, [4-18](#)
com.oracle.bigdata.rowformat, [4-19](#)
com.oracle.bigdata.tablename, [4-20](#)
CREATE TABLE ORGANIZATION EXTERNAL
 syntax, [2-23](#), [A-2](#)
CREATE TABLE statement
 generating automatically for Hive, [4-1](#)
CREATE_EXTDDL_FOR_HIVE function
 syntax, [4-1](#)

D

data dictionary views, [4-21](#)

data mode, [4-11](#)
data source name, [4-20](#)
data type conversion (Big Data SQL), [2-25](#)
data types (HDFS), [4-13](#)
DBA_HIVE_COLUMNS view, [4-24](#)
DBA_HIVE_DATABASES view, [4-24](#)
DBA_HIVE_TABLES view, [4-24](#)
DBMS_HADOOP package, [4-1](#)
DBMS_OUTPUT package, [2-3](#)
DEFAULT DIRECTORY clause, [2-23](#)
delimited text files, [4-19](#)

E

error handling, [4-12](#)
error handling (Big Data SQL), [2-26](#)
external tables
 about, [1-2](#), [A-1](#)

F

field extraction, [4-19](#)
field names, [4-13](#)

H

Hadoop log files, [4-10](#), [4-16](#)
Hive columns, [4-23](#)
Hive data
 access from Oracle Database, [2-1](#)
Hive databases, [4-22](#)
Hive table sources, [4-20](#)
Hive tables, [4-22](#)
Hive views, [4-21](#)

L

licenses, third-party, [E-1](#)
LOCATION clause, [2-24](#)
log files, [4-17](#)

O

Oracle Big Data SQL
 access drivers, [1-2](#)
 data type conversion, [2-25](#)
 general description, [1-1](#)
 installation changes on the Oracle Database
 server ., [2-27](#)
Oracle Database
 access to Hive data, [2-1](#)
 Data Modeler, [2-1](#)
 DBMS_HADOOP, [2-1](#)
 SQL Developer, [2-1](#)
 Use in Oracle Big Data SQL, [2-1](#)
Oracle Exadata Machine
 Big Data SQL installation changes, [2-27](#)
ORACLE_HDFS access driver, [2-15](#)
ORACLE_HIVE
 access parameters, [4-9](#)
ORACLE_HIVE examples, [2-7](#)
ORC files, [4-15](#)
overflow handling, [4-18](#)

P

Parquet files, [4-15](#)
parsing HDFS files, [4-19](#)
PL/SQL packages, [4-1](#)
PUT_LINE function, [2-3](#)

R

RC files, [4-15](#)

REJECT LIMIT clause, [2-24](#)
row format description, [4-15](#)
row formats, [4-19](#)

S

sequence files, [4-15](#)
SerDe parsing, [4-19](#)
Smart Scan, [1-2](#)
SmartScan mode, [4-11](#)
source name, [4-20](#)
static data dictionary views, [4-21](#)
Statistics, [1-6](#)
struct overflows, [4-18](#)

T

text files, [4-15](#)
text overflows, [4-18](#)
third-party licenses, [E-1](#)
TYPE clause, [2-23](#)

U

union overflows, [4-18](#)
user access from Oracle Database, [2-26](#)
USER_HIVE_COLUMNS view, [4-25](#)
USER_HIVE_DATABASES view, [4-25](#)
USER_HIVE_TABLES view, [4-25](#)