Supported features for Apache Hive/Impala in ORAAH 2.8.0

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Oracle R Database Transparency Layer

Oracle R Advanced Analytics for Hadoop transparency layer leverages a subset of the same interface supported by Oracle R Enterprise. The transparency layer supports executing a select set of R functions on data.frame proxy objects that correspond to tables in Oracle Database, Apache Hive, and Apache Impala. The overloaded R functions transparently translate the function request to Oracle Database SQL or Apache Hive/Impala HQL/Impala SQL respectively. Functionality includes data exploration, data preparation, and data analysis, typically prior to applying machine learning algorithms.

With the transparency layer, users avoid shifting programming paradigm or environment, can operate on data as though they were R objects using R syntax. More importantly, the transparency layer provides scalability and performance for big data since data does not need to be pulled to R client memory. Moreover, the user automatically takes advantage of query optimization, tables indexes, deferred evaluation, distributed and parallel execution.

2 Apache Hive

Apache Hive is an open source Hadoop application for data warehousing, analysis and querying of large data systems. Hive query (HiveQL query) is a SQL-like interface that is used extensively to query the contents of databases. Apache Hive has the advantage of deploying high speed data reads and writes within the data warehouses while managing large data sets that are distributed across multiple nodes.

HiveServer2 is a server interface that enables remote clients to submit queries to Hive and retrieve the results. It replaces HiveServer1, which has been deprecated and will be removed in a future release of CDH. HiveServer2 is a container for the Hive execution engine. Hive also has metastore that keeps track of all metadata of database, tables, columns and data types.

3 Apache Impala

Apache Impala is a distributed, lighting fast SQL query engine for huge data stored in Apache Hadoop cluster. It is a massively parallel and distributed query engine that lets you analyse, transform and combine data from a variety of data sources. It is used when there is need of low latency result. Unlike Apache Hive, it does not convert Impala SQL queries to MapReduce which has the problem of cold start, while Impala can return the results in seconds. Impala being a real time query engine is best suited for analytics and for data scientists to perform analytics on data stored in the Hadoop File System. However, not all SQL queries are supported in Impala. In short, Impala SQL is a subset of HiveQL and might have a few syntactical changes.

4 Apache Hive/Impala with ORAAH

We will walk through Apache Hive/Impala using Oracle R Advanced Analytics for Hadoop (ORAAH). R users will not be required to perform any Apache Hive/Impala query in HQL/Impala SQL and instead can perform all operations using R.

In order to work with Apache Hive/Impala tables, we need to load the ORCH library.

```
# Load the library ORCH.
R> library(ORCH)
Loading required package: OREbase
Loading required package: OREcommon

Attaching package: 'OREbase'

The following objects are masked from package:base:

    cbind, data.frame, eval, interaction, order, paste, pmax, pmin,
    rbind, table

Loading required package: OREstats
Loading required package: ORCHcore
```

```
Loading required package: rJava
Loading required package: RJDBC
Loading required package: DBI
Oracle R Connector for Hadoop 2.8.0
Info: using native C base64 encoding implementation
Info: loaded ORCH core Java library "orch-core-2.8.0-mr1.jar"
Loading required package: ORCHstats
Loading required package: ORCHmpi
```

Once the ORCH library is loaded you have to create connection with Hive using `ore.connect()` and can verify the connection using `ore.is.connected()` API.

Note
When connecting to either Apache Hive or Apache Impala, any previous open connection will be automatically closed.

```
# Verify if Hive is connected
R> ore.is.connected(type="HIVE")
[1] FALSE

# Connect with Hive
R> ore.connect(host=Sys.getenv("HIVE_SERVER"), port=Sys.getenv("HIVE_PORT"),
    user=Sys.getenv("HIVE_USER"), password=Sys.getenv("HIVE_PASSWORD"),
    schema=Sys.getenv("HIVE_DATABASE"), type="HIVE", all = TRUE)

# Verify again if HIVE got connected now.
R> ore.is.connected(type="HIVE")
[1] TRUE

# connect with Apache Impala and automatically disconnect from HIVE
R> ore.connect(host=Sys.getenv("HIVE_SERVER"), port=21050,
    user=Sys.getenv("HIVE_USER"), password=Sys.getenv("HIVE_PASSWORD"),
    schema=Sys.getenv("HIVE_DATABASE"), type="IMPALA", all = TRUE)

# Verify if Apache Impala is connected
R> ore.is.connected(type="IMPALA")
[1] TRUE

# Verification for Apache Hive must fail
R> ore.is.connected(type="HIVE")
[1] FALSE
```

As illustrated above, Apache Hive and Apache Impala have a similar syntax.

Once connected to Apache Hive, we can create a table in Hive using R.

```
# Drop a table called "iris_hive" if it exists
R> ore.drop("iris_hive")

# List the existing Hive tables. The table "iris_hive" should not be listed
R> ore.ls()
character(0)

# Creating an Apache Hive table iris_hive from a local R data.frame called iris.
# Because the local data.frame contains variable names with "." they are
# automatically renamed to make them Apache Hive compatible.
R> ore.create(iris, table = "iris_hive")
Warning message:
In ore.create(iris, table = "iris_hive") :
  column names modified by "ore.make.names" function
R> ore.ls()
```

```
[1] "iris_hive"

# Will return TRUE if Apache Hive table exists
R> ore.exists("iris_hive")
[1] TRUE

# iris_hive must be a ore.frame
R> class(iris_hive)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
```

We can also create Apache Impala tables in a similar way. Since, Apache Hive and Apache Impala share the same Metastore, "iris_hive" table can also be accessed through an Apache Impala connection.

Once Apache Hive/Impala table is created we can perform multiple operations on it.

```
# To get details on all columns of the HIVE table
R> summary(iris_hive)
  sepal_length    sepal_width     petal_length    petal_width      species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   Length:150
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   Class :ore.character
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   Mode  :character
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500

# Get number of rows present in HIVE table
R> nrow(iris_hive)
[1] 150
# Get number of columns present in HIVE table
R> ncol(iris_hive)
[1] 5

# Using head() to check the top 6 records of HIVE table
R> head(iris_hive)
  sepal_length sepal_width petal_length petal_width species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Important
If you have an Apache Hive table to be created from a huge R `data.frame` object using `ore.create()` API, then your HiveServer2 must be present on the same machine. A workaround can be to load the data using *beeline* to a Apache Hive table first, outside of ORAAH.

5 MLlib with Apache Hive/Impala table

You can also run Spark-based machine learning algorithms from ORAAH against data stored in Apache Hive/Impala, and the ML computation is fast because of the in-memory nature of Spark. A Spark connection using `spark.connect()` API is required for the processing to work.

```
# Connect to Spark on YARN in client mode.
```

```
R> spark.connect(master='yarn', spark.submit.deployMode='client')
Info: memory is set to 2G
Info: active HDFS namenode is cfclbv3870.us2.oraclecloud.com
Info: loaded configuration from 'spark-default.conf'

# Verify the Spark connection.
R> spark.connected()
[1] TRUE

# We will use a Dataset "Kyphosis" from the R package "rpart".
R> library(rpart)
R> ore.drop("data")

# Creating a Hive or Impala table.
R> ore.create(kyphosis, "data")

# Running a Spark MLlib Random Forest algorithm on Apache Hive/Impala table.
# We are trying to estimate Absence or Presence of Kyphosis based on 2 other variables.
# Running an ORAAH Spark-based Logistic Regression algorithm on Hive/Impala table.
R> model1 <- orch.glm2(formula = Number ~ Age, data = data, verbose = FALSE)

# Running a Spark MLlib Random Forest algorithm on Hive/Impala table.
R> model2 <- orch.ml.random.forest(formula = Kyphosis ~ Number + Age, data = data,
    nTrees=20, type='classification', verbose = FALSE)

# Write new predictions, using the same "data" as new data for this example.
R> pred <- predict(model1, newdata = data, supplemental = c("Kyphosis", "Age"),
    verbose = FALSE)
R> pred_glm2_in_hdfs <- hdfs.write(pred, outPath = "pred_glm2", overwrite = TRUE)
R>
R> pred2 <- predict(model2, newdata = data, supplemental = c("Kyphosis", "Age"),
    verbose = FALSE)
R> pred_rf_in_hdfs <- hdfs.write(pred2, outPath = "pred_rf", overwrite = TRUE)
```

6 MPI based analytics with Apache Hive/Impala table

In ORAAH 2.8.0 we have introduced a new package `ORCHmpi`. This new package has distributed MPI-backed algorithms which run over the Apache Spark framework.

`ORCHmpi` needs MPI libraries made available to ORAAH either by making MPI available system-wide, or by setting ORCH related environment variables on the client node. For more information on setting up MPI, check `help(ORCH_MPI_LIBS)` and `help(ORCH_MPI_MPIEXEC)`. If MPI is configured properly, `orch.mpiAvailable()` and `orch.scalapackAvailable()` functions will return `TRUE`.

MPI has a message-passing programming model. That means distributed jobs must be able to start all MPI workers and load all their data before they can start exchanging messages, and have synchronous lifecycle. For the complete list of new functions made available in `ORCHmpi` see "Change List" document.

ORAAH's new MPI-based algorithms can run on top of Apache Hive/Impala tables as well.

```
# Create table 'data' from iris data set.
R> ore.create(iris, "data")
R> ore.sync()
R> ore.attach()

# Create orch.elm model from Apache Hive/Impala table.
R> model <- orch.elm(formula = Species ~ . - 1, data = data, zScoreX = TRUE,
    l = 10, lambda = 1e-12)
R> summary(model)
R> cfs <- coef(model)
```

```
R> names(cfs)

# Collect coefficient matrix in R.
R> as.matrix(cfs$Beta)

# Predict new observations.
R> predOut <- predict(model, newdata = data, supplemental = "Species")
R> predOut$show()
+----------+----------+
|   Species|   predict|
+----------+----------+
| virginica| virginica|
|    setosa|    setosa|
|versicolor|versicolor|
|versicolor|versicolor|
|    setosa|    setosa|
| virginica|versicolor|
|    setosa|    setosa|
| virginica| virginica|
|versicolor|versicolor|
|versicolor| virginica|
|    setosa|    setosa|
|versicolor|versicolor|
|versicolor|versicolor|
|versicolor|versicolor|
| virginica| virginica|
|    setosa|    setosa|
|    setosa|    setosa|
|versicolor|versicolor|
|    setosa|    setosa|
|    setosa|    setosa|
+----------+----------+
only showing top 20 rows
```

7 orch.summary() with Apache Hive

New for ORAAH in release 2.8.0 is a function that generates descriptive statistics for `ore.frame` objects within flexible row aggregations. Currently, this is only supported for Apache Hive. To check the complete list of available statistical functions, check `help(orch.summary)`.

```
# Create Apache Hive table.
R> ore.create(cars, "cars1")

# Run orch.summary statistics.
R> orch.summary(cars1, c("speed"))
  freq  n mean min max
1   50 50 15.4   4  25
Warning message:
ORE object has no unique key - using random order

R> orch.summary(cars1, c("speed"), stat= c("CSS"))
  freq  css
1   50 1370
Warning message:
ORE object has no unique key - using random order

R> orch.summary(cars1, c("speed"), stat= c("USS"))
  freq   uss
1   50 13228
```

```
Warning message:
ORE object has no unique key - using random order

R> orch.summary(cars1, c("speed"), stat= c("MAX"))
  freq max
1   50  25
Warning message:
ORE object has no unique key - using random order

R> orch.summary(cars1, c("speed"), stat= c("MIN"))
  freq min
1   50   4
Warning message:
ORE object has no unique key - using random order

R> orch.summary(cars1, c("speed"), stat= c("AVG"))
  freq  avg
1   50 15.4
Warning message:
ORE object has no unique key - using random order
```

8 Apache Hive with Kerberos Authentication

If you are connecting to Apache Hiveserver2 on a Hadoop cluster with Kerberos authentication enabled, then use the parameter *principal* to specify the Kerberos server principal for the host where Hiveserver2 is running. In addition, if Apache Hiveserver2 has SSL mode enabled, then use parameters such as `ssl ="true"` and `sslTrustStore` to specify the path to the client's truststore file. Use `trustStorePassword` to specify the password for the truststore.

There are many other configuration parameters of Apache HiveServer2 and Apache Impala server. See the Apache Hive documentation for the various modes and the parameters needed to connect in those modes.

Note that you need to get the *sslTrustStore*, *trustStorePassword*, *principal* and *ssl* parameter values from your cluster administrator.

```
# Apache Hive with Kerberos authentication
R> ore.connect(user="user", password="password", host="localhost", port=10000,
    type="HIVE", schema="default", principal="hive/LOCALHOST@DEV.EXAMPLE.COM")
R> ore.ls()
R> ore.disconnect()

# Apache Hive with Kerberos authentication with SSL enabled
R> ore.connect(user="user", password="password", host="hive_server_host_name",
    port=10000, type="HIVE", schema="default", ssl="true",
    sslTrustStore="path_for_secure_shared_truststore_file",
    trustStorePassword="truststore_password",
    principal="hive/LOCALHOST@DEV.EXAMPLE.COM")
R> ore.ls()
R> ore.disconnect()
```

9 Apache Impala with Kerberos Authentication

If you are connecting to Impala with Kerberos authentication enabled, then you need to use a different set of parameters specific to Apache Impala. Using similar parameters as Apache Hive connection with Kerberos will hinder a successful connection.

Apache Impala with Kerberos authentication, requires to use parameters such as `AuthMech ="1"` to indicate Kerberos authentication. Use `KrbRealm ="realm.example.com"` and `KrbHostFQDN ="Kerberos_host_name"` to specify

the realm and Kerberos Host FQDN (Fully Qualified Domain Name). Use `KrbServiceName ="impala"` to specify the Kerberos service name for Apache Impala.

Similar to Apache Hive, you need to get the *KrbRealm*, *KrbHostFQDN*, *KrbServiceName* and *ssl* parameter values from your administrator.

```
# Apache Impala with Kerberos authentication
R> ore.connect(host = "localhost", port = 21050, AuthMech = "1",
    KrbRealm="DEV.EXAMPLE.COM",
    KrbHostFQDN="krb_host_name",
    KrbServiceName="impala", type="IMPALA")
R> ore.ls()
R> ore.disconnect()

# Apache Impala with Kerberos authentication with SSL enabled
R> ore.connect(host="localhost", port=21050, AuthMech="1",
    KrbRealm="DEV.EXAMPLE.COM",
    KrbHostFQDN="krb_host_name",
    KrbServiceName="impala", ssl="1",
    sslTrustStore="path_for_secure_shared_truststore_file",
    trustStorePassword="truststore_password",
    type="IMPALA")
R> ore.ls()
R> ore.disconnect()
```

10 Differences between Apache Hive and Apache Impala

1. Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax developed by Facebook while Apache Impala is an open source, native analytics database for Apache Hadoop developed by Cloudera.

2. Apache Hive is a batch execution engine based Hadoop MapReduce by default. Apache Impala works more like a Massively Parallel Processing(MPP) database. Apache Hive translates the SQL query to Map reduce jobs. Apache Impala responds quickly through in-memory massive parallel processing.

3. MapReduce is a batch processing engine so by design Apache Hive, which relies on MapReduce is a heavy engine based on a high-latency execution framework. It is possible to run HIVE on Spark, although it might not be recommended for extremely large Jobs. MapReduce Jobs have overheads and are typically slow. Apache Impala on the other hand does not translate a SQL query into another processing framework like map/shuffle/reduce operations, so it does not suffer from latency issues. Apache Impala is designed for SQL query execution and not as a general purpose distributed processing system like MapReduce. Given that, in many cases it is able to deliver much better performance for the same SQL query.

4. Apache Hive is excellent for long running ETL jobs for which fault tolerance is required. For example, if any of your Data Nodes fails while your query is running, the output will still be produced since the MapReduce framework is fault tolerant. That's not the case with Apache Impala, in which a failed query needs to be rerun.

5. Unlike Apache Hive, Apache Impala does not have an extensive support for the SQL language. ORAAH's support for Apache Impala APIs mimic exactly the ones for for Apache Hive, so from the R user's perspective most commands would be the same. You can refer to "Unsupported functions in Hive/Impala" column under "Supported/Unsupported APIs in Apache Hive/Impala" section.

6. If you are running Spark MlLib-based algorithm or ORAAH's MPI-based models, it is recommended to use Apache Hive connection to pass the data to the algorithms instead of Apache Impala, since this interface is proven to work better for the machine learning algorithms when they need to bring the data into Spark from disk.

7. The `orch.summary()` function has a better performance than a regular `summary()` function call, and currently it is only supported with the Apache Hive transparency layer.

11 Performance comparison between Apache Hive and Apache Impala

We performed identical tests on the airline data set known as *ONTIME* data with a range of 100k to 100m records on both Apache Hive and Apache Impala. These tests were performed on a kerberized BDCS cluster having 5 Nodes, with 32 OCPUs, 48 TB of Storage and 256 GB of RAM on each Node. It is easy to notice the improvement of Apache Impala over Apache Hive. It is also easy to see that the MapReduce engine has a large latency, since it takes almost the same time to count the small *iris* data set as it takes a 100k or 1mi record table, explained by the overhead of the engine to get any query started.

```
# Connect to Apache Hive
R> ore.connect(type='HIVE',host='xxxxxxxx',
    ssl='true', sslTrustStore='/opt/cloudera/security/jks/testbdcs.truststore',
    trustStorePassword='xxxxxxxxxxx',
    principal='hive/xxxxxxx@BDACLOUDSERVICE.ORACLE.COM',
    all=TRUE )
R> ore.ls()
[1] "iris_hive" "ontime_100k" "ontime_100m" "ontime_10m" "ontime_1m"

R> system.time({siz <- dim(iris_hive);print(siz) })
[1] 150 5
user system elapsed
0.025 0.000 24.356

R> system.time({siz <- dim(ontime_100k);print(siz) })
[1] 100000 29
user system elapsed
0.033 0.000 26.405

R> system.time({siz <- dim(ontime_1m);print(siz) })
[1] 1000000 29
user system elapsed
0.036 0.000 26.419

R> system.time({siz <- dim(ontime_10m);print(siz) })
[1] 1.0e+07 2.9e+01
user system elapsed
0.034 0.002 29.468

R> system.time({siz <- dim(ontime_100m);print(siz) })
[1] 1.0e+08 2.9e+01
user system elapsed
0.031 0.008 34.575

# Connect to Apache Impala
R> ore.connect(type='IMPALA', port='21050',
    AuthMech='1', KrbRealm='BDACLOUDSERVICE.ORACLE.COM',
    KrbHostFQDN='xxxxxxxx', KrbServiceName='impala',
    all=TRUE)
R> ore.ls()
[1] "iris_hive" "ontime_100k" "ontime_100m" "ontime_10m" "ontime_1m"

R> system.time({siz <- dim(iris_hive);print(siz) })
[1] 150 5
user system elapsed
0.099 0.016 0.198

R> system.time({siz <- dim(ontime_100k);print(siz) })
[1] 100000 29
user system elapsed
0.030 0.008 0.102

R> system.time({siz <- dim(ontime_1m);print(siz) })
```

```
[1] 1000000 29
user system elapsed
0.042 0.005 0.248

R> system.time({siz <- dim(ontime_10m);print(siz) })
[1] 1.0e+07 2.9e+01
user system elapsed
0.052 0.010 0.657

R> system.time({siz <- dim(ontime_100m);print(siz) })
[1] 1.0e+08 2.9e+01
user system elapsed
0.063 0.019 1.461
```

12 Known issues and limitations

As of ORAAH release 2.8.0, the following known issues and limitations apply:

1. If `ore.connect()` to an Apache Impala Service is returning a "ClassNotFound" exception, you might need to set *IMPALA_HOME* and *IMPALA_JAR* environment variables, usually done in the configuration file `/usr/lib64/R/etc/Renviron.site` or `/usr/lib64/R/etc/RBDAprofiles/Renviron.site` (for BDA/BDCS/BDCC).
2. Connecting to an Apache Impala service under kerberos requires different options than the kerberos options required by Apache Hive, as illustrated above in the connections of the "Performance Comparison" section.
3. On a kerberized cluster, passing an Apache Impala table as input into any of the Spark-based machine learning algorithms might fail. In this case, a simple workaround is to switch to an Apache Hive connection and use the same table as input to the Spark-based machine learning algorithms. This is possible since both Apache Hive and Apache Impala share a common Metastore.
4. Using HDFS data after invoking `hdfs.fromHive()` on an Apache Hive table, might cause Spark based analytics to fail if the NA identifier in the table data files is not "NA". Such failure can be resolved by setting the correct value for "na.strings" in the HDFS metadata manually by using `hdfs.meta()`. For example, if the table *iris_tab* has "NULL" as its NA identifer value then you can set the metadata as: `hdfs.meta(iris_hdfs, na.strings ="NULL")`, where *iris_hdfs* is an `hdfs.id` object created by `iris_hdfs <- hdfs.fromHive(iris_tab)`. After setting the correct metadata you should have no problems using data from HDFS with Spark based analytics.

12.1 More known issues

12.1.1 Connection refused

When connecting to Hive/Impala from R in ORAAH using the connection string (or similar):

```
R> ore.connect(host = Sys.getenv("HIVE_SERVER"), port = Sys.getenv("HIVE_PORT"),
    user = Sys.getenv("HIVE_USER"), password = Sys.getenv("HIVE_PASSWORD"),
    schema = Sys.getenv("HIVE_DATABASE"), type = "HIVE", all = TRUE)
```

You might encounter the following error:

```
Could not open client transport with JDBC Uri: jdbc:hive2://localhost:656/default:
java.net.ConnectException: Connection refused (Connection refused)
```

HiveServer2 service should be running and listening to the specified port, if not you will hit this issue as ORAAH will try to connect to HiveServer2 port via the Hive JDBC protocol. There are three possible problems:

1. HiveServer2 might not be running at all on the target server. Try to check if it's there by running `$ ps -ef | grep "Hive"` command.

2. HiveServer2 might be running on a different port. Normally, you should see the port, which is used by the HiveServer2 in "ps" command output (see above) and it will be a part of HiveServer2 startup command line, if not, then it's most likely the default port `10000`.
3. HiveServer2 port might be blocked by a firewall. Check your firewall settings and make sure it's not blocking connections to Apache Hive. You might temporary disable your firewall if order to test connectivity and make sure it's indeed the firewall that causing the issue.
4. For a secured (kerberized) cluster, there is a need for a *principal* parameter in `ore.connect()` to connect to hive server like `principal="hive/HiveServer2Host@YOUR-REALM.COM"`.

12.1.2 Hive queries are too slow

If you notice that Apache Hive connection and simple Hive queries take long durations to finish remember that *HiveServer2* allocates threads for each established connection. Hence, there is a potential performance issue resulting from a large number of threads due to a large number of concurrent connections to Apache Hive. You can try to mitigate this problem in the following ways:

1. Close the R sessions, which are idle and have been connected to hive to increase the performance.
2. Try to restart *HiveServer2* service completely.

12.1.3 Metastore exception on ore.connect()

When connecting to Hive/Impala using `ore.connect()` from R in ORAAH you might see a *Metastore* related exception:

```
java.sql.SQLException: Unable to open a test connection to the given database.
JDBC url = jdbc:mysql://10.0.1.31:3306/metastore?createDatabaseIfNotExist=true,
username = hive. Terminating connection pool (set lazyInit to true if you expect
to start your database after your app).
Original Exception: ------ com.mysql.jdbc.exceptions.jdbc4.CommunicationsException:
Communications link failure
Possible cause ...
```

Any exceptions related to metastore connection, requires a first check that *metastore* service is running on the specified port, with `$ ps -ef | grep "HiveMetaStore"`.

13 Supported/Unsupported API in Apache Hive/Impala

The following table lists the set of Apache Hive/Impala functions supported and not supported in ORAAH 2.8.0.

Table 1: List of APIs supported/unsupported in Apache Hive/Impala

Sr. no.	Category	Supported Functions in Hive	Unsupported Functions in Hive	Supported Functions in Impala	Unsupported Functions in Impala	Supported ORE function (Oracle)
1	`ore.character`	nchar, tolower, toupper, casefold, chartr, gsub, substr, substring	grepl, sub	nchar, tolower, toupper, casefold, chartr, gsub, substr, substring	grepl, sub	nchar, tolower, toupper, casefold, chartr, gsub, substr, substring, grepl, sub
2	`ore.logical`	<, >, ==, <=, >=, !, xor, ifelse, and, or	Nil	<, >, ==, <=, >=, !, xor, ifelse, and, or	Nil	logic, !, xor, ifelse

Table 1: (continued)

3	Apache Hive/Impala Specific	ore.hiveOptions, ore.showHiveOptions	Nil	ore.impalaOptions, ore.showImpalaOptions	Nil	Not ORE specific
4	`ore.factor`	is.factor, as.factor, as.vector, levels, nlevels	summary	is.factor, as.factor, as.vector, levels, nlevels,	summary	levels, nlevels, is.factor, as.factor, as.vector, summary
5	`ore.date`, `ore.datetime`, `ore.difftime`	ore.mday, ore.month, ore.year, ore.hour, ore.minute, ore.second	Arith, summary, trunc, coerce	ore.mday, ore.month, ore.year, ore.hour, ore.minute, ore.second	summary	Arith, trunc, coerce, as.vector, as.character, as.ore.character, ore.year, ore.month, ore.mday, ore.hour, ore.minute, ore.second, summary
6	`ore.vector`	show, length, c, is.vector, as.vector, as.character, as.numeric, as.numeric, as.integer, as.logical, [, [<−, head, tail, I, Compare, ore.recode, is.na, %in%, unique, sort, table, paste, by, tapply	coerce, summary, pmin, pmax, ore.hash, split, rank, order, interaction	show, length, c, is.vector, as.vector, as.character, as.numeric, as.numeric, as.integer, as.logical, [, [<−, head, tail, I, Compare, ore.recode, is.na, %in%, unique, sort, table, paste, by, tapply	coerce, Summary, pmin, pmax, ore.hash, split, rank, order, interaction	show, length, c, is.vector, as.vector, as.character, as.numeric, as.integer, as.logical, coerce, [, [<−, head, tail, I, Compare, Summary, pmin, pmax, ore.recode, ore.hash, is.na, %in%, unique, split, sort, rank, order, table, paste, interaction, tapply, by
7	`ore.number`	+, -, *, ^, %%, %/% , /, is.finite, is.infinite, is.nan, Math, log, round, zapsmall, logb, summary, mean, log10, log2, log1p, acos, asin, atan, exp, expm1, cos, sin, tan, abs, sign, sqrt, ceiling, floor, trunc	cut, diff, pmin, pmax, atan2, factorial, lfactorial, tabulate, besselI, besselK, besselJ, besselY, cosh, sinh, tanh	+, -, *, ^, %%, %/%, /, is.finite, is.infinite, is.nan, Math, log, round, zapsmall, logb, summary, mean, log10, log2, log1p, acos, asin, atan, exp, expm1, cos, sin, tan, abs, sign, sqrt, ceiling, floor, trunc, cosh, sinh, tanh	summary, cut, diff, pmin, pmax,atan2, factorial, lfactorial, tabulate, besselI, besselK, besselJ, besselY	Arith, cut, diff, is.finite, is.infinite, is.nan, pmin, pmax, Math, log, round, zapsmall, atan2, logb, factorial, lfactorial, summary, mean, tabulate, besselI, besselK, besselJ, besselY

Table 1: (continued)

8	Methods	ore.is.connected, ore.connect, ore.disconnect, ore.sync, ore.attach, ore.detach, ore.ls, ore.exists, ore.get, ore.rm, ore.exec, ore.drop, ore.pull, ore.push, ore.const, is.ore.vector, is.ore.logical, is.ore.integer, is.ore.numeric, is.ore.character, is.ore.factor, is.ore.date, is.ore.datetime, is.ore.frame, is.ore.matrix,is.ore, as.ore.vector	ore.make.names, is.ore.difftime	ore.is.connected, ore.connect, ore.disconnect, ore.sync, ore.attach, ore.detach, ore.ls, ore.exists, ore.get, ore.rm, ore.exec, ore.drop, ore.pull, ore.push, ore.const, is.ore.vector, is.ore.logical, is.ore.integer, is.ore.numeric, is.ore.character, is.ore.factor, is.ore.date, is.ore.datetime, is.ore.frame, is.ore.matrix,is.ore, as.ore.vector	ore.attach for partitioned tables is not supported, is.ore.difftime	ore.is.connected, ore.connect, ore.disconnect, ore.sync, ore.attach, ore.detach, ore.ls, ore.exists, ore.get, ore.rm, ore.exec, ore.make.names, ore.drop, ore.pull, ore.push, ore.const, is.ore.vector, is.ore.logical, is.ore.integer, is.ore.numeric, is.ore.character, is.ore.factor, is.ore.date, is.ore.datetime, is.ore.difftime, is.ore.frame, is.ore.matrix, is.ore, as.ore.vector
9	More Methods	as.ore.logical, as.ore.integer, as.ore.numeric, as.ore.character, as.ore.factor, as.ore.date, as.ore.datetime, as.ore.frame, as.ore.matrix, ore.create	as.ore.difftime	as.ore.logical, as.ore.integer, as.ore.numeric, as.ore.character, as.ore.factor, as.ore.date, as.ore.datetime, as.ore.frame, as.ore.matrix, ore.create	as.ore.difftime	as.ore.logical, as.ore.integer, as.ore.numeric, as.ore.character, as.ore.factor, as.ore.date, as.ore.datetime, as.ore.difftime, as.ore.frame, as.ore.matrix, as.ore, ore.create

Table 1: (continued)

10	`ore.frame`	show, attach, [, $, $<−, [[, [[<−, head, tail, length, nrow, ncol, NROW, NCOL, dim, names, names <−, colnames, colnames <−, merge, as.list, unlist, summary, rbind, cbind, data.frame, as.data.frame, as.env, eval, Arith, Compare, Logic, !, xor, is.na, is.finite, is.infinite, is.nan, Math, log, round, logb, rownSums, colSums, rowMeans, colMeans, unique, by	dimnames, row.names, row.names <−, subset, with, within, transform, scale, max.cols, interaction, split	show, attach, [, $, $<−, [[, [[<−, head, tail, length, nrow, ncol, NROW, NCOL, dim, names, names <−, colnames, colnames <−, merge, as.list, unlist, summary, rbind, cbind, data.frame, as.data.frame, as.env, eval, Arith, Compare, Logic, !, xor, is.na, is.finite, is.infinite, is.nan, Math, log, round, logb, colSums, rowMeans, colMeans, unique, by	ore.frame for a partitioned table, with, dimnames, row.names, row.names <−, subset within, transform, scale, max.cols, interaction, split	show, attach, [, $, $<−, [[, [[<−, head, tail, length, nrow, ncol, NROW, NCOL, dim, names, names <−, colnames, colnames <−, dimnames, row.names, row.names <−, merge, as.list, unlist, summary, rbind, cbind, data.frame, as.data.frame, as.env, eval, subset, with, within, transform, Arith, Compare, Logic, !, xor, is.na, is.finite, is.infinite, is.nan, Math, log, round, logb, summary, rowSums, colSums, rowMeans, colMeans, scale, max.col, interaction, split, unique, by

Table 1: (continued)

11	ore.matrix	Nil	show, is.matrix, as.matrix, [, nrow, ncol, NROW, NCOL, dim, rownames, rownames <−, colnames, colnames <−, dimnames, dimnames <−, t, Arith, Math, log, round, atan2, logb, summary, mean, tabulate, besselI, besselK, besselJ, besselY, %*%, crossprod, tcrossprod, rowSums, colSums, rowMeans, colMeans, scale, max.col, solve, backsolve, forwardsolve	Nil	show, is.matrix, as.matrix, [, nrow, ncol, NROW, NCOL, dim, rownames, rownames <−, colnames, colnames <−, dimnames, dimnames <−, t, Arith, Math, log, round, atan2, logb, Summary, mean, tabulate, besselI, besselK, besselJ, besselY, %*%, crossprod, tcrossprod, rowSums, colSums, rowMeans, colMeans, scale, max.col, solve, backsolve, forwardsolve	show, is.matrix, as.matrix, [, nrow, ncol, NROW, NCOL, dim, rownames, rownames <−, colnames, colnames <−, dimnames, dimnames <−, t, Arith, Math, log, round, atan2, logb, Summary, mean, tabulate, besselI, besselK, besselJ, besselY, %*%, crossprod, tcrossprod, rowSums, colSums, rowMeans, colMeans, scale, max.col, solve, backsolve, forwardsolve
12	OREstats	Nil	aggregate, ave, binom.test, chisq.test, complete.cases, cor, cov, dbeta, dbinom, dcauchy, dchisq, dexp, df, dgamma, dgeom, dlnorm, dlogis, dnbinom, dnorm, dpois, dsignrank, dt, dunif, dweibull, factanal, fivenum, get_all_vars, IQR, ks.test, mad, median, model.frame, model.matrix, na.omit, pbeta, pbinom, pcauchy, pchisq, pexp, pf, pgamma	Nil	aggregate, ave, binom.test, chisq.test, complete.cases, cor, cov, dbeta, dbinom, dcauchy, dchisq, dexp, df, dgamma, dgeom, dlnorm, dlogis, dnbinom, dnorm, dpois, dsignrank, dt, dunif, dweibull, factanal, fivenum, get_all_vars, IQR, ks.test, mad, median, model.frame, model.matrix, na.omit, pbeta, pbinom, pcauchy, pchisq, pexp, pf, pgamma	aggregate, ave, binom.test, chisq.test, complete.cases, cor, cov, dbeta, dbinom, dcauchy, dchisq, dexp, df, dgamma, dgeom, dlnorm, dlogis, dnbinom, dnorm, dpois, dsignrank, dt, dunif, dweibull, factanal, fivenum, get_all_vars, IQR, ks.test, mad, median, model.frame, model.matrix, na.omit, pbeta, pbinom, pcauchy, pchisq, pexp, pf, pgamma

Table 1: (continued)

13	`OREstats` contd.	Nil	pgeom, plnorm, plogis, pnbinom, pnorm, ppois, psignrank, pt, punif, pweibull, qbeta, qbinom, qcauchy, qchisq, qexp, qf, qgamma, qgeom, qlnorm, qlogis, qnbinom, qnorm, qpois, qsignrank, qt, qunif, qweibull, prcomp, princomp, prop.test, quantile, reordersd, svd, t.test, terms, var, var.test, wilcox.test, ore.rollmax, ore.rollmin, ore.rollsum, ore.rollmean, ore.rollsd, ore.rollvar, complete.cases, ore.getXlevels, ore.getXnlevels	Nil	pgeom, plnorm, plogis, pnbinom, pnorm, ppois, psignrank, pt, punif, pweibull, qbeta, qbinom, qcauchy, qchisq, qexp, qf, qgamma, qgeom, qlnorm, qlogis, qnbinom, qnorm, qpois, qsignrank, qt, qunif, qweibull, prcomp, princomp, prop.test, quantile, reordersd, svd, t.test, terms, var, var.test, wilcox.test, ore.rollmax, ore.rollmin, ore.rollsum, ore.rollmean, ore.rollsd, ore.rollvar, complete.cases, ore.getXlevels, ore.getXnlevels	pgeom, plnorm, plogis, pnbinom, pnorm, ppois, psignrank, pt, punif, pweibull, qbeta, qbinom, qcauchy, qchisq, qexp, qf, qgamma, qgeom, qlnorm, qlogis, qnbinom, qnorm, qpois, qsignrank, qt, qunif, qweibull, prcomp, princomp, prop.test, quantile, reordersd, svd, t.test, terms, var, var.test, wilcox.test, ore.rollmax, ore.rollmin, ore.rollsum, ore.rollmean, ore.rollsd, ore.rollvar, complete.cases, ore.getXlevels, ore.getXnlevels

Table 1: (continued)

14	`orch.summary`	N, FREQ, COUNT, CNT, NMISS, MEAN, AVG, MIN, MAX, CSS, USS, CV, SUM, SUMWGT, RANGE, STDDEV, STD, STDERR, STDMEAN, VARIANCE, VAR, KURTOSIS, KURT, SKEWNESS, SKEW, LOCCOUNT<, LOC<, LOCCOUNT>, LOC>, LOCCOUNT!, LOC!, LOCCOUNT, LOC, P0, P1, P5, P10, P25, Q1, P50, Q2, Median, P75, Q3, P90, P95, P99, P100, QRANGE, IQR, MODE, LCLM, RCLM, CLM, T, PROBT, PRT	CVMP, CVMT	NA	Not supported for Apache Impala	Not ORE specific
15	Partitioned table	Partitioned table can be created in Apache Hive	NA	NA	Not supported in Apache Impala	Not ORE specific
16	`hdfs.toHive`, `hdfs.fromHive`	Supported	NA	Supported	NA	Not ORE specific
17	Unsupported data type		Binary, Array, Map, Struct, uniontype		Binary, Array, Map, Struct, uniontype	

14 Copyright Notice