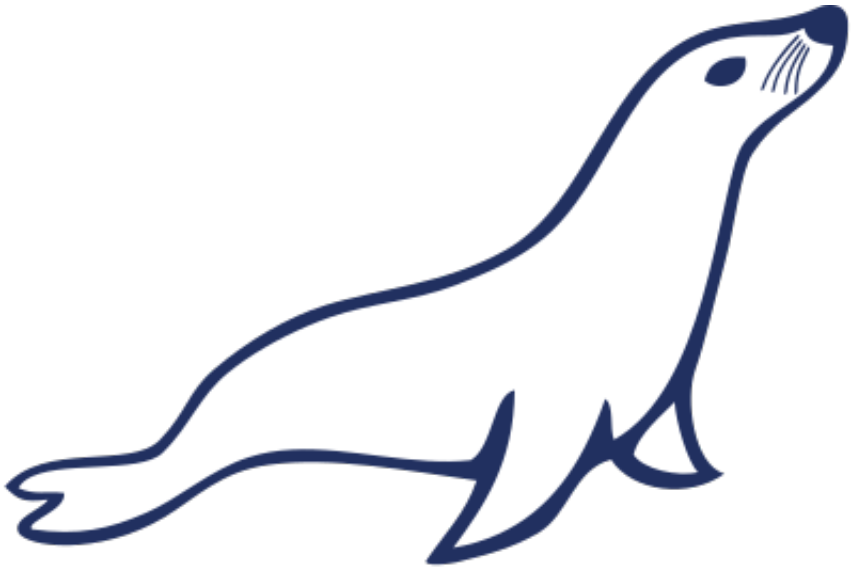

MariaDB ColumnStore Java API Usage Documentation

Release 1.2.0-3e88a57

MariaDB Corporation

Oct 10, 2018

CONTENTS

CHAPTER

ONE

LICENSING

1.1 Documentation Content

The javamcsapi documentation is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

1.2 MariaDB ColumnStore Java API

The MariaDB ColumnStore Java API (javamcsapi) is licensed under the GNU Lesser General Public License, version 2.1.

CHAPTER

TWO

VERSION HISTORY

This is a version history of Java API interface changes. It does not include internal fixes and changes.

Version	Changes
1.1.6	• Java documentation added • Added *ColumnStoreDriver.getJavaMcsapiVersion* to return version information about the java wrapper library javamcsapi.jar • Windows support added (Alpha)
1.1.5	• Added overloaded functions of *ColumnStoreBulkInsert.setColumn* and *ColumnStoreBulkInsert.setNull* to return additional status information
1.1.4	• Make *ColumnStoreSystemCatalog.getTable* and *ColumnStoreSystemCatalogTable.getColumn* case insensitive • Add *ColumnStoreDriver.setDebug* to enable debugging output to stderr
1.1.1	• Add *ColumnStoreBulkInsert.isActive* • Make *ColumnStoreBulkInsert.rollback* fail without error • Add *ColumnStoreBulkInsert.resetRow* • *ColumnStoreDateTime* now uses uint32_t for every parameter • *ColumnStoreSystemCatalog* now uses const for the sub-class strings
1.1.0β	• First beta release

CHAPTER

THREE

USING JAVAMCSAPI

3.1 Usage Introduction

The Java bulk insert API (javamcsapi) is a wrapper around the C++ bulk insert API (mcsapi) generated by SWIG. As a result Java programmers can utilize the same functions to insert data into ColumnStore tables as C++ developers can do. javamcsapi is available for Java 8 and upwards.

3.2 Basic Bulk Insert

In this example we will insert 1000 rows of two integer values into table `test.t1`. The full code for this can be found in the `example/Basic_bulk_insert.java` file in the mcsapi codebase.

You will need the following table in the test database to execute this:

Listing 1: example/basic_bulk_insert.sql

```
CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL
) ENGINE=Columnstore;
```

Listing 2: example/Basic_bulk_insert.java

```
import com.mariadb.columnstore.api.*;
```

We need to import all classes from the package *com.mariadb.columnstore.api* to use mcsapi.

Listing 3: example/Basic_bulk_insert.java

```
public class Basic_bulk_insert {

    public static void main(String[] args) {
        try {
            ColumnStoreDriver d = new ColumnStoreDriver();
```

A new instance of *ColumnStoreDriver* is created which will attempt to find the `Columnstore.xml` configuration file by first searching for the environment variable `COLUMNSTORE_INSTALL_DIR` and then the default path of `/usr/local/mariadb/columnstore/etc/Columnstore.xml`. Alternatively we could provide a path as a parameter to *ColumnStoreDriver*.

Listing 4: example/Basic_bulk_insert.java

```
30              ColumnStoreBulkInsert b = d.createBulkInsert("test", "t1", (short)0, 0);
```

Once we have the ColumnStore installation's configuration in the driver we use this to initiate a bulk insert using *ColumnStoreDriver.createBulkInsert*. We are using the `test` database and the `t1` table. The remaining two parameters are unused for now and set to `0`.

Listing 5: example/Basic_bulk_insert.java

```
31              for (int i=0; i<1000; i++){
32                  b.setColumn(0, i);
33                  b.setColumn(1, 1000-i);
34                  b.writeRow();
35              }
```

A "for" loop is used to loop over 1000 arbitrary inserts in this example. We use *ColumnStoreBulkInsert.setColumn* to specify that column `0` (column `a`) should be set to the integer from the "for" loop and column `1` (column `b`) is set to `1000` minus the integer from the "for" loop.

When we have added something to every column *ColumnStoreBulkInsert.writeRow* is used to indicate we are finished with the row. The library won't necessarily write the row at this stage, it buffers up to 100,000 rows by default.

Listing 6: example/Basic_bulk_insert.java

```
36              b.commit();
37          }
```

At the end of the loop we execute *ColumnStoreBulkInsert.commit* which will send any final rows and initiate the commit of the data. If we do not do this the transaction will be implicitly rolled back instead.

Listing 7: example/Basic_bulk_insert.java

```
38          catch (ColumnStoreException e) {
39              System.err.println("Error caught: " + e.getMessage());
40          }
41      }
```

If anything fails then we should catch *ColumnStoreException* to handle it.

3.3 Advanced Bulk Insert

In this example we will insert 2 rows in a more complex table. This will demonstrate using different kinds of data types, chanined methods and getting the summary information at the end of a transaction.

You will need the following table in the test database to execute this:

Listing 8: example/advanced_bulk_insert.sql

```
1  CREATE TABLE `t2` (
2    `id` int(11) DEFAULT NULL,
3    `name` varchar(40) DEFAULT NULL,
4    `dob` date DEFAULT NULL,
5    `added` datetime DEFAULT NULL,
```

(continues on next page)

(continued from previous page)

```
  `salary` decimal(9,2) DEFAULT NULL
) ENGINE=Columnstore;
```

Listing 9: example/Advanced_bulk_insert.java

```
import com.mariadb.columnstore.api.*;

public class Advanced_bulk_insert {

    public static void main(String[] args) {
        try {
            ColumnStoreDriver d = new ColumnStoreDriver();
            ColumnStoreBulkInsert b = d.createBulkInsert("test", "t2", (short)0, 0);
```

As with the basic example we create an instance of the driver and use it to create a bulk insert instance.

Listing 10: example/Advanced_bulk_insert.java

```
            b.setColumn(0, 1);
            b.setColumn(1, "Andrew");
            b.setColumn(2, "1936-12-24");
            b.setColumn(3, "2017-07-07 15:14:12");
            b.setColumn(4, "15239.45");
            b.writeRow();
```

This demonstrates setting several different data types using strings of data. The second column (column `1`) is a `VARCHAR(40)` and is set to "Andrew". The third column is a `DATE` column and the API will automatically convert this into a binary date format before transmitting it to ColumnStore. The fourth is a `DATETIME` and the fifth a `DECIMAL` column which again the API will convert from the strings into the binary format.

Listing 11: example/Advanced_bulk_insert.java

```
            b.setColumn(0, 2).setColumn(1, "David").setColumn(2, new␣
↪ColumnStoreDateTime("1972-05-23", "%Y-%m-%d")).setColumn(3, new ColumnStoreDateTime(
↪"2017-07-07 15:20:18", "%Y-%m-%d %H:%M:%S")).setColumn(4, new␣
↪ColumnStoreDecimal(2347623, (short)2)).writeRow();
```

Many of the *ColumnStoreBulkInsert* methods return a pointer to the class and a return status which means multiple calls can be chained together in a similar way to ORM APIs. Here we use additional datatypes *ColumnStoreDateTime* and *ColumnStoreDecimal*.

ColumnStoreDateTime is used to create an entry for a `DATE` or `DATETIME` column. It can be used to define custom formats for dates and times using the strptime format.

A decimal is created using the *ColumnStoreDecimal* class. It can be set using a string, `double` or a pair of integers. The first integer is the precision and the second integer is the scale. So this number becomes `23476.23`.

Listing 12: example/Advanced_bulk_insert.java

```
            b.commit();
            ColumnStoreSummary summary = b.getSummary();
            System.out.println("Execution time: " + summary.getExecutionTime());
            System.out.println("Rows inserted: " + summary.getRowsInsertedCount());
            System.out.println("Truncation count: " + summary.getTruncationCount());
            System.out.println("Saturated count: " + summary.getSaturatedCount());
            System.out.println("Invalid count: " + summary.getInvalidCount());
        }
```

After a commit or rollback we can obtain summary information from the bulk insert class. This is done using the *ColumnStoreBulkInsert.getSummary* method which will return a reference *ColumnStoreSummary* class. In this example we get the number of rows inserted (or would be inserted if there was a rollback) and the execution time from the moment the bulk insert class is created until the commit or rollback is complete.

Listing 13: example/Advanced_bulk_insert.java

```
46          catch (ColumnStoreException e) {
47              System.err.println("Error caught: " + e.getMessage());
48          }
49      }
50  }
```

At the end we clean up in the same was as the basic bulk insert example.

CHAPTER

FOUR

COMPILING WITH JAVAMCSAPI

4.1 Pre-requisites

To link javamcsapi to your application and to execute it you first need install the following mcsapi pre-requisites:

4.1.1 Ubuntu

```
sudo apt-get install libuv1
```

4.1.2 CentOS 7

```
sudo yum install epel-release
sudo yum install libuv
```

4.2 Compiling

To compile a Java program from command line you have to explicitly state where to find javamcsapi.jar. Its path can differ depending on your operating system.

4.2.1 Ubuntu

```
javac -classpath ".:/usr/lib/javamcsapi.jar" Basic_bulk_insert.java
java -classpath ".:/usr/lib/javamcsapi.jar" Basic_bulk_insert
```

4.2.2 CentOS 7

```
javac -classpath ".:/usr/lib64/javamcsapi.jar" Basic_bulk_insert.java
java -classpath ".:/usr/lib64/javamcsapi.jar" Basic_bulk_insert
```

4.2.3 Windows 10 (x64)

```
javac -classpath ".;%mcsapiInstallDir%\lib\java\javamcsapi-1.1.7.jar" Basic_bulk_
↪insert.java
java -classpath ".;%mcsapiInstallDir%\lib\java\javamcsapi-1.1.7.jar" -Djava.library.
↪path="%mcsapiInstallDir%\lib" Basic_bulk_insert
```

The variable `%mcsapiInstallDir%` represents the base installation directory of the Bulk Write SDK. (e.g. `C:\Program Files\MariaDB\ColumnStore Bulk Write SDK`)

If you don't want to change the `java.library.path` you can copy javamcapi's DLLs `libiconv.dll`, `libuv.dll`, `libxml2.dll`, `mcsapi.dll` and `javamcsapi.dll` from `%mcsapiInstallDir%\lib` to the directory of the Java class to execute. Another option is to just add `%mcsapiInstallDir%\lib` to your `PATH` environment variable, which is the default setting when you install the Bulk Write SDK.

CHAPTER

FIVE

JAVAMCSAPI API REFERENCE

5.1 ColumnStoreBulkInsert

public class **`ColumnStoreBulkInsert`**

5.1.1 Fields

swigCMemOwn

protected transient boolean **`swigCMemOwn`**

5.1.2 Constructors

ColumnStoreBulkInsert

protected **`ColumnStoreBulkInsert`** (long *cPtr*, boolean *cMemoryOwn*)

5.1.3 Methods

addReference

protected void **`addReference`** (*ColumnStoreDriver columnStoreDriver*)

commit

public void **`commit`** ()

delete

public synchronized void **`delete`** ()

finalize

protected void **`finalize`** ()

getCPtr

protected static long **getCPtr** (*ColumnStoreBulkInsert obj*)

getColumnCount

public int **getColumnCount** ()

getSummary

public *ColumnStoreSummary* **getSummary** ()

isActive

public boolean **isActive** ()

resetRow

public *ColumnStoreBulkInsert* **resetRow** ()

rollback

public void **rollback** ()

setBatchSize

public void **setBatchSize** (long *batchSize*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, String *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, String *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, java.math.BigInteger *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, java.math.BigInteger *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, long *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, long *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, int *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, int *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, short *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, short *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, byte *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, byte *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, double *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, double *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn**(int *columnNumber*, *ColumnStoreDateTime* *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, *ColumnStoreDateTime* *value*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, *ColumnStoreDecimal* *value*, int[] *status*)

setColumn

public *ColumnStoreBulkInsert* **setColumn** (int *columnNumber*, *ColumnStoreDecimal* *value*)

setNull

public *ColumnStoreBulkInsert* **setNull** (int *columnNumber*, int[] *status*)

setNull

public *ColumnStoreBulkInsert* **setNull** (int *columnNumber*)

setTruncateIsError

public void **setTruncateIsError** (boolean *set*)

writeRow

public *ColumnStoreBulkInsert* **writeRow** ()

5.2 ColumnStoreDateTime

public class **ColumnStoreDateTime**

5.2.1 Fields

swigCMemOwn

protected transient boolean **swigCMemOwn**

5.2.2 Constructors

ColumnStoreDateTime

protected **ColumnStoreDateTime** (long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreDateTime

public **ColumnStoreDateTime**()

ColumnStoreDateTime

public **ColumnStoreDateTime**(SWIGTYPE_p_tm *time*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(long *year*, long *month*, long *day*, long *hour*, long *minute*, long *second*, long *microsecond*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(long *year*, long *month*, long *day*, long *hour*, long *minute*, long *second*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(long *year*, long *month*, long *day*, long *hour*, long *minute*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(long *year*, long *month*, long *day*, long *hour*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(long *year*, long *month*, long *day*)

ColumnStoreDateTime

public **ColumnStoreDateTime**(String *dateTime*, String *format*)

5.2.3 Methods

delete

public synchronized void **delete**()

finalize

protected void **finalize**()

getCPtr

protected static long **getCPtr**(*ColumnStoreDateTime obj*)

set

public boolean **set**(SWIGTYPE_p_tm *time*)

set

public boolean **set**(String *dateTime*, String *format*)

5.3 ColumnStoreDecimal

public class **ColumnStoreDecimal**

5.3.1 Fields

swigCMemOwn

protected transient boolean **swigCMemOwn**

5.3.2 Constructors

ColumnStoreDecimal

protected **ColumnStoreDecimal**(long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreDecimal

public **ColumnStoreDecimal**()

ColumnStoreDecimal

public **ColumnStoreDecimal**(long *value*)

ColumnStoreDecimal

public **ColumnStoreDecimal**(String *value*)

ColumnStoreDecimal

public **ColumnStoreDecimal**(double *value*)

ColumnStoreDecimal

public **ColumnStoreDecimal**(long *number*, short *scale*)

5.3.3 Methods

delete

public synchronized void **delete**()

finalize

protected void **finalize**()

getCPtr

protected static long **getCPtr**(*ColumnStoreDecimal obj*)

set

public boolean **set**(long *value*)

set

public boolean **set**(String *value*)

set

public boolean **set**(double *value*)

set

public boolean **set**(long *number*, short *scale*)

5.4 ColumnStoreDriver

public class **ColumnStoreDriver**

5.4.1 Fields

swigCMemOwn

protected transient boolean **swigCMemOwn**

5.4.2 Constructors

ColumnStoreDriver

protected **ColumnStoreDriver**(long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreDriver

public **ColumnStoreDriver**(String *path*)

ColumnStoreDriver

public **ColumnStoreDriver**()

5.4.3 Methods

createBulkInsert

public *ColumnStoreBulkInsert* **createBulkInsert**(String *db*, String *table*, short *mode*, int *pm*)

delete

public synchronized void **delete**()

finalize

protected void **finalize**()

getCPtr

protected static long **getCPtr**(*ColumnStoreDriver obj*)

getJavaMcsapiVersion

public String **getJavaMcsapiVersion**()

getSystemCatalog

public *ColumnStoreSystemCatalog* **getSystemCatalog**()

getVersion

public String **getVersion**()

setDebug

public void **setDebug**(boolean *enabled*)

5.5 ColumnStoreException

public class **ColumnStoreException** extends java.lang.RuntimeException

5.5.1 Constructors

ColumnStoreException

public **`ColumnStoreException`**()

ColumnStoreException

public **`ColumnStoreException`**(String *message*)

ColumnStoreException

public **`ColumnStoreException`**(String *message*, Throwable *cause*)

ColumnStoreException

public **`ColumnStoreException`**(Throwable *cause*)

5.6 ColumnStoreSummary

public class **`ColumnStoreSummary`**

5.6.1 Fields

swigCMemOwn

protected transient boolean **`swigCMemOwn`**

5.6.2 Constructors

ColumnStoreSummary

protected **`ColumnStoreSummary`**(long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreSummary

public **`ColumnStoreSummary`**()

ColumnStoreSummary

public **`ColumnStoreSummary`**(*ColumnStoreSummary* *summary*)

5.6.3 Methods

delete

public synchronized void **delete**()

finalize

protected void **finalize**()

getCPtr

protected static long **getCPtr**(*ColumnStoreSummary obj*)

getExecutionTime

public double **getExecutionTime**()

getInvalidCount

public java.math.BigInteger **getInvalidCount**()

getRowsInsertedCount

public java.math.BigInteger **getRowsInsertedCount**()

getSaturatedCount

public java.math.BigInteger **getSaturatedCount**()

getTruncationCount

public java.math.BigInteger **getTruncationCount**()

5.7 ColumnStoreSystemCatalog

public class **ColumnStoreSystemCatalog**

5.7.1 Fields

swigCMemOwn

protected transient boolean **swigCMemOwn**

5.7.2 Constructors

ColumnStoreSystemCatalog

protected **`ColumnStoreSystemCatalog`** (long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreSystemCatalog

public **`ColumnStoreSystemCatalog`** ()

ColumnStoreSystemCatalog

public **`ColumnStoreSystemCatalog`** (*ColumnStoreSystemCatalog* *obj*)

5.7.3 Methods

addReference

protected void **`addReference`** (*ColumnStoreDriver* *columnStoreDriver*)

delete

public synchronized void **`delete`** ()

finalize

protected void **`finalize`** ()

getCPtr

protected static long **`getCPtr`** (*ColumnStoreSystemCatalog* *obj*)

getTable

public *ColumnStoreSystemCatalogTable* **`getTable`** (String *schemaName*, String *tableName*)

5.8 ColumnStoreSystemCatalogColumn

public class **`ColumnStoreSystemCatalogColumn`**

5.8.1 Fields

swigCMemOwn

protected transient boolean **`swigCMemOwn`**

5.8.2 Constructors

ColumnStoreSystemCatalogColumn

protected **ColumnStoreSystemCatalogColumn** (long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreSystemCatalogColumn

public **ColumnStoreSystemCatalogColumn** ()

ColumnStoreSystemCatalogColumn

public **ColumnStoreSystemCatalogColumn** (*ColumnStoreSystemCatalogColumn obj*)

5.8.3 Methods

compressionType

public short **compressionType** ()

delete

public synchronized void **delete** ()

finalize

protected void **finalize** ()

getCPtr

protected static long **getCPtr** (*ColumnStoreSystemCatalogColumn obj*)

getColumnName

public String **getColumnName** ()

getDefaultValue

public String **getDefaultValue** ()

getDictionaryOID

public long **getDictionaryOID** ()

getOID

public long **getOID** ()

getPosition

public long **getPosition**()

getPrecision

public long **getPrecision**()

getScale

public long **getScale**()

getType

public *columnstore_data_types_t* **getType**()

getWidth

public long **getWidth**()

isAutoincrement

public boolean **isAutoincrement**()

isNullable

public boolean **isNullable**()

5.9 ColumnStoreSystemCatalogTable

public class **ColumnStoreSystemCatalogTable**

5.9.1 Fields

swigCMemOwn

protected transient boolean **swigCMemOwn**

5.9.2 Constructors

ColumnStoreSystemCatalogTable

protected **ColumnStoreSystemCatalogTable**(long *cPtr*, boolean *cMemoryOwn*)

ColumnStoreSystemCatalogTable

public **ColumnStoreSystemCatalogTable**()

ColumnStoreSystemCatalogTable

public **ColumnStoreSystemCatalogTable**(*ColumnStoreSystemCatalogTable obj*)

5.9.3 Methods

delete

public synchronized void **delete**()

finalize

protected void **finalize**()

getCPtr

protected static long **getCPtr**(*ColumnStoreSystemCatalogTable obj*)

getColumn

public *ColumnStoreSystemCatalogColumn* **getColumn**(String *columnName*)

getColumn

public *ColumnStoreSystemCatalogColumn* **getColumn**(int *columnNumber*)

getColumnCount

public int **getColumnCount**()

getOID

public long **getOID**()

getSchemaName

public String **getSchemaName**()

getTableName

public String **getTableName**()

5.10 columnstore_data_convert_status_t

public enum **columnstore_data_convert_status_t**

5.10.1 Enum Constants

CONVERT_STATUS_INVALID

public static final *columnstore_data_convert_status_t* **CONVERT_STATUS_INVALID**

CONVERT_STATUS_NONE

public static final *columnstore_data_convert_status_t* **CONVERT_STATUS_NONE**

CONVERT_STATUS_SATURATED

public static final *columnstore_data_convert_status_t* **CONVERT_STATUS_SATURATED**

CONVERT_STATUS_TRUNCATED

public static final *columnstore_data_convert_status_t* **CONVERT_STATUS_TRUNCATED**

5.11 columnstore_data_types_t

public enum **columnstore_data_types_t**

5.11.1 Enum Constants

DATA_TYPE_BIGINT

public static final *columnstore_data_types_t* **DATA_TYPE_BIGINT**

DATA_TYPE_BIT

public static final *columnstore_data_types_t* **DATA_TYPE_BIT**

DATA_TYPE_BLOB

public static final *columnstore_data_types_t* **DATA_TYPE_BLOB**

DATA_TYPE_CHAR

public static final *columnstore_data_types_t* **DATA_TYPE_CHAR**

DATA_TYPE_CLOB

public static final *columnstore_data_types_t* **DATA_TYPE_CLOB**

DATA_TYPE_DATE

public static final *columnstore_data_types_t* **DATA_TYPE_DATE**

DATA_TYPE_DATETIME

public static final *columnstore_data_types_t* **DATA_TYPE_DATETIME**

DATA_TYPE_DECIMAL

public static final *columnstore_data_types_t* **DATA_TYPE_DECIMAL**

DATA_TYPE_DOUBLE

public static final *columnstore_data_types_t* **DATA_TYPE_DOUBLE**

DATA_TYPE_FLOAT

public static final *columnstore_data_types_t* **DATA_TYPE_FLOAT**

DATA_TYPE_INT

public static final *columnstore_data_types_t* **DATA_TYPE_INT**

DATA_TYPE_MEDINT

public static final *columnstore_data_types_t* **DATA_TYPE_MEDINT**

DATA_TYPE_SMALLINT

public static final *columnstore_data_types_t* **DATA_TYPE_SMALLINT**

DATA_TYPE_TEXT

public static final *columnstore_data_types_t* **DATA_TYPE_TEXT**

DATA_TYPE_TINYINT

public static final *columnstore_data_types_t* **DATA_TYPE_TINYINT**

DATA_TYPE_UBIGINT

public static final *columnstore_data_types_t* **DATA_TYPE_UBIGINT**

DATA_TYPE_UDECIMAL

public static final *columnstore_data_types_t* **DATA_TYPE_UDECIMAL**

DATA_TYPE_UDOUBLE

public static final *columnstore_data_types_t* **DATA_TYPE_UDOUBLE**

DATA_TYPE_UFLOAT

public static final *columnstore_data_types_t* **DATA_TYPE_UFLOAT**

DATA_TYPE_UINT

public static final *columnstore_data_types_t* **DATA_TYPE_UINT**

DATA_TYPE_UMEDINT

public static final *columnstore_data_types_t* **DATA_TYPE_UMEDINT**

DATA_TYPE_USMALLINT

public static final *columnstore_data_types_t* **DATA_TYPE_USMALLINT**

DATA_TYPE_UTINYINT

public static final *columnstore_data_types_t* **DATA_TYPE_UTINYINT**

DATA_TYPE_VARBINARY

public static final *columnstore_data_types_t* **DATA_TYPE_VARBINARY**

DATA_TYPE_VARCHAR

public static final *columnstore_data_types_t* **DATA_TYPE_VARCHAR**

INDEX

E

F

G

I

R

S

W