HPCC JDBC Driver

Boca Raton Documentation Team

HPCC JDBC Driver

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com>

Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2025 Version 9.14.8-1

Introduction

Java Database Connectivity (**JDBC**) is a standard Java API that enables Java applications or client tools that support JDBC to access data from a *presumably* SQL-compliant data source via the SQL language.

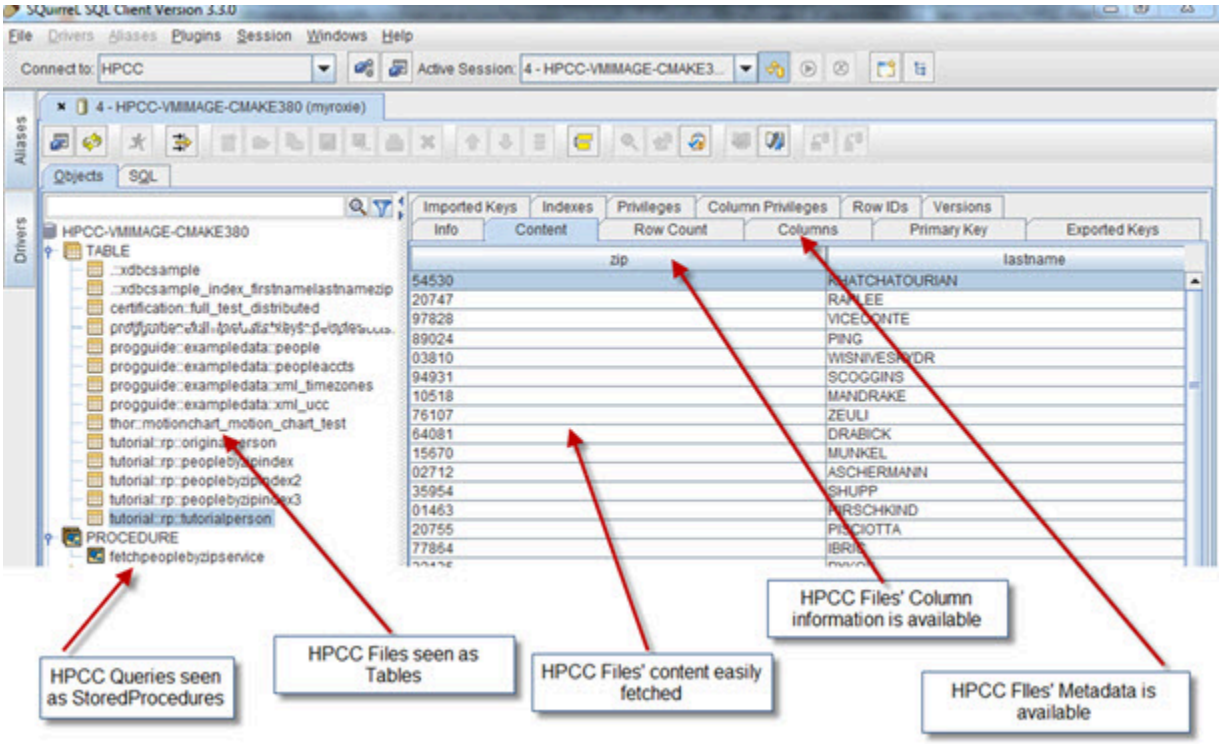
JDBC makes it possible to write a single database application that can run on different platforms and interact with different database management systems.

Currently there are JDBC Drivers available for interaction with many popular data sources; now the HPCC platform is available as a data source.

The HPCC JDBC Driver exposes HPCC logical files as RDB tables.

- HPCC Logical File <-> RDB Table
- HPCC Record Definition Fields <-> RDB Table Columns
- HPCC Published query <-> RDB Stored Procedure
- Provides access to HPCC system data and RDB metadata
- Supports subset of SQL syntax
- Read-only operations supported
- Non-transactional
- Provides means for utilizing HPCC index files for faster reads.

Figure 1. An example SQL Client interface connected to an HPCC Platform with the JDBC driver



Installation

The HPCC JDBC driver is distributed in a self-contained JAVA jar file.

Follow the instructions for your SQL client for installation.

To utilize your HPCC platform, use the configuration settings in the next section. The manner in which you define these settings is dependent on your SQL client.

The driver's full class path is:

org.hpccsystems.jdbcdriver.HPCCDriver

Note: WsSQL must be installed on the target HPCC Platform to utilize a JDBC connection.

See https://hpccsystems.com/download/free-modules/wssql for details.

Configuration

The HPCC JDBC driver supports the following configuration attributes:

Property	Description	Default Value	Req.
ServerAddress	Target HPCC ESP Address (used to contact WsSQL).	“localhost”	Yes
WsSQLPort:	WsSQL port (WsSQL is a web service which must be installed with the HPCC platform. It typically runs on port 8015).	8015	Yes
WsECLWatchAddress	Target HPCC WsECLWatch address	ServerAddress Value	No
WsECLWatchPort	Target HPCC WsECLWatch port	8010	No
username	User name on Target HPCC, if needed	“”	No
password	Password on Target HPCC, if needed	“”	No
PageSize	Max Number of HPCC files or HPCC published queries reported as result of GetTables, or GetProcs	100	No
ConnectTimeoutMilli	Timeout value to establish connection to HPCC (in milliseconds)	1000	No
ReadTimeoutMilli	HPCC Connection **read** timeout value (in milliseconds)	1500	No
EclResultLimit	Max result records returned (use ALL to return all records)	100	No
LazyLoad	Fetch HPCC file and query metadata on-demand (not at connect time)	“true”	No
TargetCluster	ECLDirect target cluster	“hthor”	No
QuerySet	Target published query (stored procedure) QuerySet	“hthor”	No
TraceToFile	When true, tracing is directed to file **./HPCCJDBC.log**, otherwise trace is sent to standard output (stdout)	"false"	No
TraceLevel	Trace Logging level, as defined in java.util.logging.level. Valid values: ALL, SEVERE, WARNING, INFO, FINEST, OFF	INFO	No

These configuration settings are used when creating an HPCC JDBC connection.

They can be passed to the JDBC driver using the JDBC connection URL or as part of the connection properties object. When passed through the connection URL, the values must be URL-encoded.

The connection URL syntax is as follows:

```
jdbc:hpcc[;urlencodedkey=urlencodedvalue]*
```

Note: The **jdbc:hpcc** prefix is required.

Using HPCC as a JDBC data source

Once connected, the HPCC JDBC driver will process submitted SQL statements and generate dynamic ECL code. The code is submitted to and executed by your HPCC Platform. The resultset is returned to your application or SQL client.

Note: The HPCC JDBC driver **only supports files which contain the record definition in the logical file's metadata.** Sprayed files do not contain this metadata. This metadata exists on any file or index which is written to the HPCC Distributed File System. Sprayed data files typically undergo some processing and an OUTPUT of the transformed data to disk before use, so this should not interfere with the driver's usefulness.

In addition, you can utilize indexes on the HPCC in one of two ways:

1. Provide SQL hints to tell driver to use a specific index for your query.

   For example:

   ```
   USEINDEX(TutorialPersonByZipIndex)
   ```

2. Specify the related indexes in the HPCC logical file description.

Index Annotations

The JDBC driver attempts to perform index based reads whenever possible. However, in order to take advantage of index reads, the target HPCC files need to be annotated with the pertinent index file names. This is accomplished by adding the following key/value entry on the file's description using ECL Watch.

From a logical file's details page, enter the information in the Description entry box, then press the **Save Description** button.

This information is used by the driver to decide if an index fetch is possible for a query on the base file.

On source file:

XDBC:RelIndexes= [*fullLogicalFilename1*; fullLogicalFilename2]

Example:

```
XDBC:RelIndexes=[tutorial::yn::peoplebyzipindex;
                tutorial::yn::peoplebyzipindex2;
                tutorial::yn::peoplebyzipindex3]
```

In this example, the source file has three indexes available.

On the index file:

XDBC:PosField=[*indexPositionFieldName***]**

Example:

```
XDBC:PosField=[fpos]
```

The FilePosition field (fpos) can have any name, so it must be specified in the metadata so the driver knows which field is the fileposition.

Simply enter the information in the description entry box, then press the **Save Description** button.

Note: You should enter this information BEFORE publishing any query using the data file or indexes. Published queries lock the file and would prevent editing the metadata.

Supported SQL

CALL

Call *queryname* **([***param list***])**

queryName	The published query name or alias
paramList	The parameters exposed by the published query (comma-separated)

Call executes a published ECL query as if it were a stored procedure.

Example:

```
Call SearchPeopleByZipService (‘33024’)
```

SELECT

select [distinct**]** *columnList* **from** *tableList* **[**USE INDEX*(indexFileName* | 0 *)***]**

[where *logicalExpression***] [group by** *columnList*[1]**] [having** *logicalExpression*[2]**]**

[order by *columnList*[1] [asc | desc]] **[LIMIT** *limitNumber***]**

NOTE: Identifiers can be unquoted or within double quotes, literal string values must be single quoted.

columnList	columnreference1[,columnreference2,columnreference3,...,columnreferen-ce*n*] The column(s) to return (comma-separated list). In addition, these aggregate functions are supported : COUNT, SUM, MIN, MAX, and AVG. These work in a similar manner as their ECL counterparts.
columnreference	[tablename.]columnname[[AS] alias]
distinct	[distinct] col1, col2,... col*n* The result set will only contain distinct (unique) values.
tableList	tableref1[,tableref2,tableref3,...,tableref*n*] One or more tables, separated by commas. **NOTE:** A table list with multiple tables creates an (one or more) implicit inner join using the where clause logical expression as the join condition which must contain an equality condition.
tableref	tableName[[AS] alias] The Name of the table as referenced, optionally defining its alias.
alias	The alias used to refer to the corresponding table or field reference.
logicalExpression	Logical expression based on standard SQL filtering syntax. Compound operators such as NOT IN or IS NULL must be in ALL CAPS or all lower case and can have one and only one space between the words. (NOT IN, not in, IS NULL, is null, etc.) BOOLEAN Only supports *True* or *False*, do not use Y, N, 0, or 1. **Valid operators:** = Equal (e.g., age=33) <> Not equal (e.g., age <>33) > Greater than (e.g., age >55) < Less than (e.g., age < 18) >= Greater than or equal (e.g., age >=21) <= Less than or equal (e.g., age <=21)

	IN(value1,value2,...,value*n*) where values are comma separated homogeneous types. NOT IN(value1,value2,...,value*n*) where values are comma separated homogeneous types. IS NULL
limitNumber	The number of rows to return. This overrides the driver's configuration attribute (EclResultLimit) but cannot be set to ALL.

[1]Aliasing not supported

[2]Can only contain references to aggregate functions if used with *having* clause.

Aggregate functions can only be expressed in logicalExpressions by using *Group by* and *having*

Examples:

Select * from tableList where Sum(F1 > 100) /* is NOT SUPPORTED */

Select * from tableList Group byF1 Haveing Sum (F1 > 100) /* IS SUPPORTED */

Example:

```
Select fname, lname, state from TutorialPerson where
   state='FL' OR (lname='Smith' and fname='Joe')
//returns data that looks like this:
John Doe FL
Jim Smith FL
Jane Row FL
Joe Smith CA

Select fname, lname, state from TutorialPerson where state='FL' AND lname <> 'Smith'
//returns data that looks like this:
John Doe FL
Jane Row FL
```

The driver supports SQL index hints, which gives the SQL user the option to specify the most appropriate HPCC index for the current SQL query. This also allows you to disable the use of an index.

select *columnList* **from** *tableName* **USE INDEX(***hpcc::index::file::name***) where** *logicalExprssions*

USE INDEX(0) forces the system to avoid seeking an index for the current query.

Example:

```
Select fname, lname, zip, state from TutorialPerson
USEINDEX(TutorialPersonByZipIndex)where zip='33024'

//returns data that looks like this:
John Doe FL 33024
Jim Smith FL 33024
Jane Row FL 33024
```

SELECT JOIN

select *columnList* **from** *tableName* ***[as*** *alias**]***

[<outer | inner > **JOIN** *join TableName* **[as** *alias***] on** *joinCondition***]**

[USE INDEX*(*indexFileName | 0 *)*]

[**where** *logicalExpression]* [**group by** *fieldName*]

[**order by** *columnNames* [asc | desc]] [**LIMIT** *limitNumber*]

columnList	columnreference1[,columnreference2,columnreference3,...,columnreference*n*] The column(s) to return (comma-separated list). In addition, these aggregate functions are supported : COUNT, SUM, MIN, MAX, and AVG. These work in a similar manner as their ECL counterparts.
columnreference	[tablename.]columnname[[AS] alias]
distinct	[distinct] col1, col2,... col*n* The result set will only contain distinct (unique) values.
alias	The alias used to refer to the corresponding table or field reference.
outer | inner	The type of JOIN to use.
joinTableName	The JOIN file to use.
joinCondition	Specifies the relationship between columns in the joined tables using logical expression.
logicalExpression	Logical expression based on standard SQL filtering syntax. Compound operators such as NOT IN or IS NULL must be in ALL CAPS or all lower case and can have one and only one space between the words. (NOT IN, not in, IS NULL, is null, etc.) BOOLEAN Only supports *True* or *False*, do not use Y, N, 0, or 1. **Valid operators:** = Equal (e.g., age=33) <> Not equal (e.g., age <>33) > Greater than (e.g., age >55) < Less than (e.g., age < 18) >= Greater than or equal (e.g., age >=21) <= Less than or equal (e.g., age <=21)

	IN(value1,value2,...,value*n*) where values are comma separated homogeneous types. NOT IN(value1,value2,...,value*n*) where values are comma separated homogeneous types. IS NULL
limitNumber	Optional. The number of rows to return. This overrides the driver's configuration attribute (`EclResultLimit`) but cannot be set to ALL.

[1]Aliasing not supported

[2]Can only contain references to aggregate functions if used with *having* clause.

Aggregate functions can only be expressed in logicalExpressions by using *Group by* and *having*

Examples:

Select * from tableList where Sum(F1 > 100) /* is NOT SUPPORTED */

Select * from tableList Group byF1 Having Sum (F1 > 100) /* IS SUPPORTED */

Example:

```
Select t1.personname, t2.address
       from persontable as t1 inner join addresstable as t2
       on (t1.personid = t2.personid AND
           (t1.firstname = 'jim' AND
            t1.lastname  = 'smith' ))
```

The JDBC driver does not convert parameter list or column list values to string literals.

String values should be single quote encapsulated. Field identifier can be left unquoted or double quoted.

For example, the table **persons** has columns Firstname(String) and Zip (numeric)

```
Select Firstname from persons where Firstname = ‘Jim’ and zip > 33445    /* works  */

Select Firstname from persons where Firstname = ‘Jim’ and "zip" > 33445    /* also works  */

Select Firstname from persons where Firstname = Jim and zip > 33445     /* doesn’t work */

Select Firstname from persons where Firstname = ‘Jim’ and zip > ‘33445’   /* doesn’t work */
```

Supported Aggregate Functions

COUNT([DISTINCT]*columnName*)

DISTINCT(*columnName*)

SUM(*columnName*)

MIN(*columnName*)

MAX(*columnName*)

AVG(*columnName*)

These aggregate functions are supported. They behave as their ECL counterparts. See the **ECL Language Reference** for details.

COUNT	Counts the occurrences of columnName in the result, always an integer.
DISTINCT	Returns only distinct values of columnName in the result, output type is dependent on input type.
SUM	Returns the sum of the values of columnName in the result, output type is dependent on input type.
MIN	Returns the minimum value for of columnName in the result, output type is dependent on input type.
MAX	Returns the minimum value for of columnName in the result, output type is dependent on input type.
AVG	Returns the average of the values of columnName in the result, always a real number.
columnName	The column to aggregate.

Example:

```
Select fname, lname, state, COUNT(zip) from TutorialPerson where zip='33024'
```

Supported String Modifiers

UPPER(*columnName*)

LOWER(*columnName*)

UPPER	Returns with all lower case characters converted to upper case.
LOWER	Returns with all upper case characters converted to lower case.
columnName	The column to aggregate

Java Example

```
/* Obtain instance of JDBC Driver */
Driver jdbcdriver = DriverManager.getDriver("jdbc:hpcc");

/* Establish Connection */

HPCCConnection connection = null;
 try
 {

/*populate JAVA properties object with pertinent connection options */
   Properties connprops = new Properties();
   connprops.put("ServerAddress", "192.168.124.128");

/*or create JDBC connection url string with pertinent connection options*/
/* the connection values must be URL-encoded */
   String jdbcurl = "jdbc:hpcc;ServerAddress=HTTP%3A%2F%2F192.168.124.128";

/*provide all necessary connection properties either by URL, or props object */
    connection = (HPCCConnection) driver.connect(jdbcurl, connprops);
  }

   catch (Exception e) { System.out.println("Error");}
/* create HPCCStatement object for single use SQL query execution */

   HPCCStatement stmt = (HPCCStatement) connection.createStatement();

/* Create your SQL query */
   String mysql = "select * from tablename as mytab limit 10";

/* Execute your SQL query */
   HPCCResultSet res1 = (HPCCResultSet) stmt.executeQuery(mysql);

/*Do something with your results */
   printOutResultSet(res1);

/* Or create a prepared statement for multiple execution and parameterization */
   String myprepsql = "select * from persons_table persons where zip= ? limit 100";
   HPCCPreparedStatement prepstmt =
(HPCCPreparedStatement)createPrepStatement(connection, myprepsql);

/* provide parameter values and execute */
   for (int i = 33445; i < 33448; i++)
   {
    prepstmt.clearParameters();
    prepstmt.setString(1, "'" + Integer.toString(i, 10) + "'");
    HPCCResultSet qrs = (HPCCResultSet) ((HPCCPreparedStatement) prepstmt).executeQuery();

/*Do something with your results */
    printOutResultSet(qrs);
   }
```

More code samples are available from:

https://github.com/hpcc-systems/hpcc-jdbc/tree/master/src/test/java/org/hpccsystems/jdbcdriver/tests