# JDBC Storage Handler



## Syntax

JdbcStorageHandler supports reading from jdbc data source in Hive. Currently writing to a jdbc data source is not supported. To use JdbcStorageHandler, you need to create an external table using JdbcStorageHandler. Here is a simple example:

```
CREATE EXTERNAL TABLE student_jdbc
(
  name string,
  age int,
  gpa double
)
STORED BY 'org.apache.hive.storage.jdbc.JdbcStorageHandler'
TBLPROPERTIES (
    "hive.sql.database.type" = "MYSQL",
    "hive.sql.jdbc.driver" = "com.mysql.jdbc.Driver",
    "hive.sql.jdbc.url" = "jdbc:mysql://localhost/sample",
    "hive.sql.dbcp.username" = "hive",
    "hive.sql.dbcp.password" = "hive",
    "hive.sql.table" = "STUDENT",
    "hive.sql.dbcp.maxActive" = "1"
);
```

You can also alter table properties of the jdbc external table using alter table statement, just like other non-native Hive table:

```
ALTER TABLE student_jdbc SET TBLPROPERTIES ("hive.sql.dbcp.password" = "passwd");
```

### Table Properties

In the create table statement, you are required to specify the following table properties:

- hive.sql.database.type: MYSQL, POSTGRES, ORACLE, DERBY, DB2
- hive.sql.jdbc.url: jdbc connection string
- hive.sql.jdbc.driver: jdbc driver class
- hive.sql.dbcp.username: jdbc user name
- hive.sql.dbcp.password: jdbc password in clear text, this parameter is strongly discouraged. The recommended way is to store it in a keystore. See the section “securing password” for detail
- hive.sql.table / hive.sql.query: You will need to specify either “hive.sql.table” or “hive.sql.query” to tell how to get data from jdbc database. “hive.sql.table” denotes a single table, and “hive.sql.query” denotes an arbitrary sql query.

Besides the above required properties, you can also specify optional parameters to tune the connection details and performance:

- hive.sql.catalog: jdbc catalog name (only valid if “hive.sql.table“ is specified)
- hive.sql.schema: jdbc schema name (only valid if “hive.sql.table“ is specified)
- hive.sql.jdbc.fetch.size: number of rows to fetch in a batch
- hive.sql.dbcp.xxx: all dbcp parameters will pass to commons-dbcp. See https://commons.apache.org/proper/commons-dbcp/configuration.html for definition of the parameters. For example, if you specify hive.sql.dbcp.maxActive=1 in table property, Hive will pass maxActive=1 to commons-dbcp

### Supported Data Type

The column data type for a Hive JdbcStorageHandler table can be:

- Numeric data type: byte, short, int, long, float, double
- Decimal with scale and precision
- String date type: string, char, varchar
- Date
- Timestamp

Note complex data type: struct, map, array are not supported

## Column/Type Mapping

hive.sql.table / hive.sql.query defines a tabular data with a schema. The schema definition has to be the same as the table schema definition. For example, the following create table statement will fail:

```
CREATE EXTERNAL TABLE student_jdbc
(
  name string,
  age int,
  gpa double
)
STORED BY 'org.apache.hive.storage.jdbc.JdbcStorageHandler'
TBLPROPERTIES (
    . . . . . .
    "hive.sql.query" = "SELECT name, age, gpa, gender FROM STUDENT",
);
```

However, column name and column type of hive.sql.table / hive.sql.query schema may be different than the table schema. In this case, database column maps to hive column by position. If data type is different, Hive will try to convert it according to Hive table schema. For example:

```
CREATE EXTERNAL TABLE student_jdbc
(
  sname string,
  age int,
  effective_gpa decimal(4,3)
)
STORED BY 'org.apache.hive.storage.jdbc.JdbcStorageHandler'
TBLPROPERTIES (
    . . . . . .
    "hive.sql.query" = "SELECT name, age, gpa FROM STUDENT",
);
```

Hive will try to convert the double “gpa” of underlining table STUDENT to decimal(4,3) as the effective_gpa field of the student_jdbc table. In case the conversion is not possible, Hive will produce null for the field.

# Auto Shipping

JdbcStorageHandler will ship required jars to MR/Tez/LLAP backend automatically if JdbcStorageHandler is used in the query. User don’t need to add jar manually. JdbcStorageHandler will also ship required jdbc driver jar to the backend if it detects any jdbc driver jar in classpath (include mysql, postgres, oracle and mssql). However, user are still required to copy jdbc driver jar to hive classpath (usually, lib directory in hive).

# Securing Password

In most cases, we don’t want to store jdbc password in clear text in table property "hive.sql.dbcp.password". Instead, user can store password in a Java keystore file on HDFS using the following command:

```
hadoop credential create host1.password -provider jceks://hdfs/user/foo/test.jceks -v passwd1
hadoop credential create host2.password -provider jceks://hdfs/user/foo/test.jceks -v passwd2
```

This will create a keystore file located on hdfs://user/foo/test.jceks which contains two keys: host1.password and host2.password. When creating table in Hive, you will need to specify “hive.sql.dbcp.password.keystore” and “hive.sql.dbcp.password.key” instead of “hive.sql.dbcp.password” in create table statement:

```
CREATE EXTERNAL TABLE student_jdbc
(
  name string,
  age int,
  gpa double
)
STORED BY 'org.apache.hive.storage.jdbc.JdbcStorageHandler'
TBLPROPERTIES (
    . . . . . .
    "hive.sql.dbcp.password.keystore" = "jceks://hdfs/user/foo/test.jceks",
    "hive.sql.dbcp.password.key" = "host1.password",
    . . . . . .
);
```

You will need to protect the keystore file by only authorize targeted user to read this file using authorizer (such as ranger). Hive will check the permission of the keystore file to make sure user has read permission of it when creating/altering table.

# Partitioning

Hive is able to split the jdbc data source and process each split in parallel. User can use the following table property to decide whether or not to split and how many splits to split into:

- hive.sql.numPartitions: how many split to generate for the data source, 1 if no split
- hive.sql.partitionColumn: which column to split on. If this is specified, Hive will split the column into hive.sql.numPartitions equal intervals from hive.sql.lowerBound to hive.sql.upperBound. If partitionColumn is not defined but numPartitions > 1, Hive will split the data source using offset. However, offset is not always reliable for some databases. It is highly recommended to define a partitionColumn if you want to split the data source. The partitionColumn must exist in the schema “hive.sql.table”/”hive.sql.query” produces.
- hive.sql.lowerBound / hive.sql.upperBound: lower/upper bound of the partitionColumn used to calculate the intervals. Both properties are optional. If undefined, Hive will do a MIN/MAX query against the data source to get the lower/upper bound. Note both hive.sql.lowerBound and hive.sql. upperBound cannot be null. The first and last split are open ended. And all null value for the column will go to the first split.

For example:

```
TBLPROPERTIES (
    . . . . . .
    "hive.sql.table" = "DEMO",
    "hive.sql.partitionColumn" = "num",
    "hive.sql.numPartitions" = "3",
    "hive.sql.lowerBound" = "1",
    "hive.sql.upperBound" = "10",
    . . . . . .
);
```

This table will create 3 splits: num<4 or num is null, 4<=num<7, num>=7

```
TBLPROPERTIES (
    . . . . . .
    "hive.sql.query" = "SELECT name, age, gpa/5.0*100 AS percentage FROM STUDENT",
    "hive.sql.partitionColumn" = "percentage",
    "hive.sql.numPartitions" = "4",
    . . . . . .
);
```

Hive will do a jdbc query to get the MIN/MAX of the percentage column of the query, which is 60, 100. Then table will create 4 splits: (,70),[70,80),[80,90),[90,). The first split also include null value.

To see the splits generated by JdbcStorageHandler, looking for the following messages in hiveserver2 log or Tez AM log:

```
jdbc.JdbcInputFormat: Num input splits created 4
jdbc.JdbcInputFormat: split:interval:ikey[,70)
jdbc.JdbcInputFormat: split:interval:ikey[70,80)
jdbc.JdbcInputFormat: split:interval:ikey[80,90)
jdbc.JdbcInputFormat: split:interval:ikey[90,)
```

# Computation Pushdown

Hive will pushdown computation to jdbc table aggressively, so we can make best usage of the native capacity of jdbc data source.

For example, if we have another table voter_jdbc:

```
EATE EXTERNAL TABLE voter_jdbc
(
  name string,
  age int,
  registration string,
  contribution decimal(10,2)
)
STORED BY 'org.apache.hive.storage.jdbc.JdbcStorageHandler'
TBLPROPERTIES (
    "hive.sql.database.type" = "MYSQL",
    "hive.sql.jdbc.driver" = "com.mysql.jdbc.Driver",
    "hive.sql.jdbc.url" = "jdbc:mysql://localhost/sample",
    "hive.sql.dbcp.username" = "hive",
    "hive.sql.dbcp.password" = "hive",
    "hive.sql.table" = "VOTER"
);
```

Then the following join operation will push down to mysql:

```
select * from student_jdbc join voter_jdbc on student_jdbc.name=voter_jdbc.name;
```

This can be manifest by explain:

```
explain select * from student_jdbc join voter_jdbc on student_jdbc.name=voter_jdbc.name;
        . . . . . .
        TableScan
          alias: student_jdbc
          properties:
            hive.sql.query SELECT `t`.`name`, `t`.`age`, `t`.`gpa`, `t0`.`name` AS `name0`, `t0`.`age` AS
`age0`, `t0`.`registration`, `t0`.`contribution`
FROM (SELECT *
FROM `STUDENT`
WHERE `name` IS NOT NULL) AS `t`
INNER JOIN (SELECT *
FROM `VOTER`
WHERE `name` IS NOT NULL) AS `t0` ON `t`.`name` = `t0`.`name`
        . . . . . .
```

Computation pushdown will only happen when the jdbc table is defined by “hive.sql.table”. Hive will rewrite the data source with a “hive.sql.query” property with more computation on top of the table. In the above example, mysql will run the query and retrieve the join result, rather than fetch both tables and do the join in Hive.

The operators can be pushed down include filter, transform, join, union, aggregation and sort.

The derived mysql query can be very complex and in many cases we don’t want to split the data source thus run the complex query multiple times on each split. So if the computation is more then just filter and transform, Hive will not split the query result even if “hive.sql.numPartitions” is more than 1.

# Using a Non-default Schema

The notion of schema differs from DBMS to DBMS, such as Oracle, MSSQL, MySQL, and PostgreSQL. Correct usage of the hive.sql.schema table property can prevent problems with client connections to external JDBC tables. For more information, see Hive-25591. To create external tables based on a user-defined schema in a JDBC-compliant database, follow the examples below for respective databases.

## MariaDB

```
CREATE SCHEMA bob;
CREATE TABLE bob.country
(
    id   int,
    name varchar(20)
);

insert into bob.country
values (1, 'India');
insert into bob.country
values (2, 'Russia');
insert into bob.country
values (3, 'USA');

CREATE SCHEMA alice;
CREATE TABLE alice.country
(
    id   int,
    name varchar(20)
);

insert into alice.country
values (4, 'Italy');
insert into alice.country
values (5, 'Greece');
insert into alice.country
values (6, 'China');
insert into alice.country
values (7, 'Japan');
```

## MS SQL

```
CREATE DATABASE world;
USE world;

CREATE SCHEMA bob;
CREATE TABLE bob.country
(
    id   int,
    name varchar(20)
);

insert into bob.country
values (1, 'India');
insert into bob.country
values (2, 'Russia');
insert into bob.country
values (3, 'USA');

CREATE SCHEMA alice;
CREATE TABLE alice.country
(
    id   int,
    name varchar(20)
);

insert into alice.country
values (4, 'Italy');
insert into alice.country
values (5, 'Greece');
insert into alice.country
values (6, 'China');
insert into alice.country
values (7, 'Japan');
```

Create a user and associate them with a default schema. For example:

```
CREATE LOGIN greg WITH PASSWORD = 'GregPass123!$';
CREATE USER greg FOR LOGIN greg WITH DEFAULT_SCHEMA=bob;
```

Allow the user to connect to the database and run queries. For example:

```
GRANT CONNECT, SELECT TO greg;
```

## Oracle

In Oracle, dividing the tables into different namespaces/schemas is achieved through different users. The CREATE SCHEMA statement exists in Oracle, but has different semantics from those defined by SQL Standard and those adopted in other DBMS.

To create "local" users in Oracle you need to be connected to the Pluggable Database (PDB), not to the Container Database (CDB). The following example was tested in Oracle XE edition, using only PDB XEPDB1.

```
ALTER SESSION SET CONTAINER = XEPDB1;
```

Create the bob schema/user and give appropriate connections to be able to connect to the database. For example:

```
CREATE USER bob IDENTIFIED BY bobpass;
ALTER USER bob QUOTA UNLIMITED ON users;
GRANT CREATE SESSION TO bob;

CREATE TABLE bob.country
(
    id   int,
    name varchar(20)
);

insert into bob.country
values (1, 'India');
insert into bob.country
values (2, 'Russia');
insert into bob.country
values (3, 'USA');
```

Create the alice schema/user and give appropriate connections to be able to connect to the database. For example:

```
CREATE USER alice IDENTIFIED BY alicepass;
ALTER USER alice QUOTA UNLIMITED ON users;

GRANT CREATE SESSION TO alice;
CREATE TABLE alice.country
(
    id   int,
    name varchar(20)
);

insert into alice.country
values (4, 'Italy');
insert into alice.country
values (5, 'Greece');
insert into alice.country
values (6, 'China');
insert into alice.country
values (7, 'Japan');
```

Without the SELECT ANY privilege, a user cannot see the tables/views of another user. When a user connects to the database using a specific user and schema it is not possible to refer to tables in another user/schema
-- namespace. You need to grant the SELECT ANY privilege. For example:

```
GRANT SELECT ANY TABLE TO bob;
GRANT SELECT ANY TABLE TO alice;
```

Allow the users to perform inserts on any table/view in the database, not only those present on their own schema. For example:

```
GRANT INSERT ANY TABLE TO bob;
GRANT INSERT ANY TABLE TO alice;
```

## PostgreSQL

```
CREATE SCHEMA bob;
CREATE TABLE bob.country
(
    id   int,
    name varchar(20)
);

insert into bob.country
values (1, 'India');
insert into bob.country
values (2, 'Russia');
insert into bob.country
values (3, 'USA');

CREATE SCHEMA alice;
CREATE TABLE alice.country
(
    id   int,
    name varchar(20)
);

insert into alice.country
values (4, 'Italy');
insert into alice.country
values (5, 'Greece');
insert into alice.country
values (6, 'China');
insert into alice.country
values (7, 'Japan');
```

Create a user and associate them with a default schema <=> search_path. For example:

```
CREATE ROLE greg WITH LOGIN PASSWORD 'GregPass123!$';
ALTER ROLE greg SET search_path TO bob;
```

Grant the necessary permissions to access the schema. For example:

```
GRANT USAGE ON SCHEMA bob TO greg;
GRANT SELECT ON ALL TABLES IN SCHEMA bob TO greg;
```