Pluggable support for custom RDBMs

Introduction

Hive Metastore currently supports following database types:

1. Mysql
2. Postgres
3. Oracle
4. SQL server
5. Derby (mostly used for unit-testing)

In order to add support for a new RDBMS type, we need to add schema scripts and identify all the places where custom database specific code is being executed and modify/refactor it to work for the new database. This requires a deep understanding of metastore internals. Additionally, the testing and maintenance effort for this new supported database type is non-trivial. It would be great to add a pluggable way to support to additional ANSI compliant databases (eg. IBM DB2) which would unlock new use-cases and deployments without modifying the metastore code and add to its testing burden. This mostly works for legacy use cases on the datanucleus code paths, but newer features like ACID transactions invoke direct SQLs to the RDBMS which breaks abstraction of underlying RDBMS for performance reasons.

Goals

Introduce a way to load a custom jar in metastore which can override database specific *DatabaseProduct* which provides the database specific SQL code execution in Hive Metastore.

Non-goals

1. We do not want to evolve metastore into a Datanuleus (ORM) like software which transparently handles all the different nuances of supporting multiple database types. The pluggable custom instance of the DatabaseProduct must be ANSI SQL compliant since the MetastoreDirectSQL and SqlGenerator assumes that. We would like to keep the changes to MetastoreDirectSQL at minimum and assume that all the supported databases are ANSI SQL compliant.
2. Upgrade and performance testing of custom implementations.
3. Schematool to be able to load this custom jar to execute schema initialization and upgrade scripts. This is not currently in scope of this document.

Design

Most of the database specific SQL uses the class DatabaseProduct. It uses the JDBC call DatabaseMetaData#getDatabaseProductName() to determine the underlying RDBMS type. Currently this is an enum. It exposes a getDbProduct() which is used at various places to execute database specific code in multiple other classes (See appendix for details).

We could convert this enum into a class which is the default implementation for all the supported database types above. It will have protected methods which a subclass can override. This subclass represents the custom DatabaseProduct which is provided by a jar in the classpath of Metastore. We will introduce 2 new configurations:

`metastore.use.custom.database.product`
This configuration will take a boolean argument. The default value will be false. In case this is set to true, metastore should look for additional a custom DatabaseProduct implementation in the classpath.

`metastore.custom.database.product.classname`
This configuration will provide the fully-qualified class name for the implementation of the DatabaseProduct class. The jar which has this class needs to be present in the classpath of the metastore. It will also mean the jar will need to be trusted and run as a privileged user within the server process. The class will be instantiated only when `metastore.use.custom.database.product` is set to `true.`

Additionally, the existing JDO configuration properties like `javax.jdo.option.ConnectionURL`, `javax.jdo.option.ConnectionDriverName` should appropriately be configured correctly to be able to talk to the custom database product.

DatabaseProduct class will implement the Configurable interface and will contain a `setConf(Configuration)` method. This method will be used to pass additional configuration properties via the hive-site.xml to the plugin if required.

Currently, the analysis shows the following methods will be exposed via DatabaseProduct and can be overridden in the custom DatabaseProduct class implementation. We may need to add/refactor some more methods as we start implementing these changes but we expect them to be minimum.

Overridable methods in DatabaseProduct

`protected boolean hasJoinOperationOrderBug()`
We probably should rename it to something better. This is a workaround around an issue reported in DERBY-6358 and also applies to Oracle and Postgres.

`public String toDate(String)`
Given a column value returns a String which casts the column value to date type. This may not be necessary to override since the current code only special cases it for Oracle

`protected prepareTxn(Statement stmt)`
This method will be called when a transaction is opened. It will execute any database specific statements which are needed before running any queries. Currently this is only used by MySQL to set the ANSI_QUOTES

`protected boolean needsBatching()`
Many of the queries which fetch multiple objects (eg. Partitions) use the IN clause in the underlying query. If the RDBMS has limitations (performance or strict), metastore will divide the IN clause into multiple batches. The batch size is controlled by configuration `metastore.direct.sql.batch.size`. In case the database needsBatching and the batch size is fixed to 1000.

`protected String addForUpdateClause(String query)`
This is mostly used to lock the table row (typically for generating event id or transaction id). The custom databaseProduct can override the default implementation as needed.

`protected String addLimitClause(String queryWithoutSelectKeyword, int limit)`
This is used to add a limit clause to a given query.

`protected String lockTable(table_name, boolean shared)`
This is mostly a generic form of `createTxnLockStatement`() in SQLGenerator class which is used to lock the table TXN_LOCK_TBL in the given mode (shared/exclusive). The method returns the SQL query to lock the given table name in either shared/exclusive mode.

`public Map<String, String> initDataSourceProperties(Configuration)`
Currently used to initialize the Datasource properties for HikariCp or DbCp connection pools. Mostly not needed to be overridden by the custom database product. The default method does some initialization for MYSQL and Postgres databases.

`protected String getMillisAfterEpochFn()`

This method returns the database specific function to return milliseconds since epoch as a String which is used in queries used by TxnHandler.java

`protected String getTxnSeedFn(long txnId)`
This method returns the String query to restart transaction id for the given txnId. Unfortunately, this is not very clean since it assumes that the DatabaseProduct implementation knows the schema. In a way this is okay since a custom DatabaseProduct will need to provide initialization and upgrade scripts which assumes they know the schema of the metastore. This method would alter the "TXNS" table such that the txnId is restarted to a given txnId. This is currently only used by `seed_txn_id()` HMS API. By default, this method throws an MetaException if the database product is not among the supported database types so custom implementations can choose to not override this function and document this as a limitation. Currently, this API is used in `msck repair` command on transactional tables (need to investigate this in more detail).

`protected String getDbTime()`
Returns the query to fetch the current timestamp in milliseconds in the database.

`protected String isWithinIntervalFn(String expr, long intervalInSeconds)`
Given an expr (such as a column name) returns a database specific SQL function string which when executed returns if the value of the expr is within the given interval of the current timestamp in the database. See TxnHandler#isWithinCheckInterval() for more information.

`protected boolean isDeadLock(SQLException ex)`
Given a SQLException determine if this is caused due to deadlock error in the database.

Future Tasks

1. Integration with SchemaTool to support schema initialization and upgrades within CDW. Note that any changes to schemaTool needs to be carefully tested since they are used for schema initialization and upgrades.

Appendix

Currently DatabaseProduct is being used in following classes

1. MetastoreStoreDirectSQL
    a. hasJoinOperationOrderBug
        i. Workaround for DERBY-6358 (also applies to Oracle, Postgres)
    b. toDate
        i. Cast a given string to date
    c. prepareTxn
        i. MYSQL specific code here. (set ANSI_QUOTES)

    d. needsBatching
        i. RDBMS has restrictions in IN clause (perf based or explicit)
2. DbNotificationListener
    a. Locks the NOTIFICATION_SEQUENCE table since Derby doesn't have a "select for update". This code path is only for added txn event messages which directly adds the row in the NOTIFICATION_LOG table after generating the next event id.
    b. In case of MYSQL table set ANSI_QUOTES
3. ObjectStore
    a. prepareQuotes
        i. Set ANSI_QUOTES for MYSQL database
        ii. Add select for update clause for NOTIFICATION_SEQUENCE table. This JDO code path for added a notification event to the NOTIFICATION_LOG
        iii. Special handling of postgres db for adding null bit vector in the statistics object
4. Connection pool properties
    a. Dbcp
        i. Has database specific connection properties for MYSQL and Postgres
        ii. Runs a validation query? (set ANSI_QUOTES)
    b. HikariCp
        i. Gets connection init sql (set ANSI_QUOTES)
        ii. Datasource connection properties for MYSQL and Postgres
5. SQLGenerator
    a. addEscapeCharacters
        i. MYSQL specific code here to escape '\\'
        ii. Other than above, it also exposes the dbType which is used in ObjectStore, DbNotificationListener, TxHandler for the same reasons. Set ANSI_QUOTES in MYSQL, locking NOTIFICATION_SEQUENCE table, null bit vectors in stats objects,
6. TxnDbUtil
    a. DB_EPOCH_FN
        i. Database specific function to get millisecs after epoch.
    b. DB_SEED_FN
        i. Used to restart the TXN_ID with a seed value. Used by HMS API seed_txn_id()
    c. prepDb
        i. Used by unit tests to initialize transaction tables.
    d. resetTxnSequence
        i. Used by unit tests to cleanup transaction tables.
    e. truncateTable
        i. Used by unit tests to cleanup transaction tables.
    f. executeQueriesInBatchNoCount
        i. Batching logic for queries. In case Oracle uses an anonymous stored procedure instead of a batch query.

    g. supportsGetGeneratedKeys
        i. The code already seems to handle a fallback if this is not supported.
7. TxnHandler
    a. isDeadlock
        i. DB specific error message parsing to detect deadlocks.
        ii. locking/unlocking the derby lock to serialize operations on Derby
    b. acquireLock
        i. Derby specific serialization to handle select for update.
    c. getDbTime
        i. Gets the current time in the database.
    d. isWithinCheckInterval
        i. Gets whether the given column value is within given seconds of the current timestamp.