Item 279
git_comments:

1. test is not null
2. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
3. * * End to end test for JSON data type for {@link PJson} and {@link PhoenixJson}.
4. test is null
5. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
6. * * Returns the root of the resulting {@link JsonNode} tree.
7. * * @return length of the string represented by the current {@link PhoenixJson}.
8. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
9. * input data has been stored as it is, since some data is lost when json parser runs, for * example if a JSON object within the value contains the same key more than once then only last * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of * keeping user data as it is.
10. Default for unit testing
11. ignore
12. * * Get {@link PhoenixJson} for a given json paths. For example : * <p> * <code> * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' * </code> * <p> * for this source json, if we want to know the json at path {'f4','f6'} it will return * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key * exist more than once. * <p> * If the given path is unreachable then it throws {@link SQLException}. * @param paths {@link String []} of path in the same order as they appear in json. * @return {@link PhoenixJson} for the json against @paths. * @throws SQLException
13. * * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It * should be used to represent the JSON data type and also should be used to parse Json data and * read the value from it. It always conside the last value if same key exist more than once.
14. * * Get {@link PhoenixJson} for a given json paths. For example : * <p> * <code> * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' * </code> * <p> * for this source json, if we want to know the json at path {'f4','f6'} it will return * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key * exist more than once. * <p> * If the given path is unreachable then it return null. * @param paths {@link String []} of path in the same order as they appear in json. * @return {@link PhoenixJson} for the json against @paths.
15. * * Serialize the current {@link PhoenixJson} to String. Its required for * json_extract_path_text(). If we just return node.toString() it will wrap String value in * double quote which is not the expectation, hence avoiding calling toString() on * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If * PhoenixJson just represent a {@link ValueNode} then it should return value returned from * objects toString().

16. * * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and * throws {@link SQLException} if it is invalid with line number and character. * @param jsonData Json data as {@link String}. * @return {@link PhoenixJson}. * @throws SQLException
17. * avoiding the type casting of object to String by calling toString() since String's * toString() returns itself.
18. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
19. * * <p> * A Phoenix data type to represent JSON. The json data type stores an exact copy of the input text, * which processing functions must reparse on each execution. Because the json type stores an exact * copy of the input text, it will preserve semantically-insignificant white space between tokens, * as well as the order of keys within JSON objects. Also, if a JSON object within the value * contains the same key more than once, all the key/value pairs are kept. It stores the data as * string in single column of HBase and it has same data size limit as Phoenix's Varchar. * <p> * JSON data types are for storing JSON (JavaScript Object Notation) data, as specified in RFC 7159. * Such data can also be stored as text, but the JSON data types have the advantage of enforcing * that each stored value is valid according to the JSON rules.
20. * * Unit test for {@link PhoenixJson}.
21. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
22. * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
23. * * Unit test for {@link PJson}.
24. * * @return true if {@link PDataType} can be declared as primary key otherwise false.
25. * * @return true if {@link PDataType} supports equality operators (=,!=,<,>,<=,>=) otherwise * false.

git_commits:

1. **summary:** PHOENIX-628 Support native JSON data type
   message: PHOENIX-628 Support native JSON data type

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Support native JSON data type
   description: MongoDB and PostGres do some interesting things with JSON. We should look at adding similar support. For a detailed description, see JSONB support in Postgres:
   http://www.craigkerstiens.com/2014/03/24/Postgres-9.4-Looking-up
   http://www.depesz.com/2014/03/25/waiting-for-9-4-introduce-jsonb-a-structured-format-for-storing-json/
   http://michael.otacoo.com/postgresql-2/manipulating-jsonb-data-with-key-unique/

jira_issues_comments:

1. Comment:stoens:10/23/13 06:29:49 PM: assigned
2. **body:** Comment:preillyme:11/08/13 09:33:23 PM: :+1:
   label: code-design
3. +1 for this we could extend to xml and yaml as well add supply away to leverage xmlpath or jsonpath when querying
4. **body:** I have an ideal. we can create new lexer and parser for JSON. and parse to classes implement CompilableStatment. then use default compiler and execute. so that we just only rewrite lexer and parser.
   label: code-design
5. I'm Maduranga Siriwardena, 4th year undergraduate from University of Moratuwa, Sri Lanka. This feature seems to be interesting and I'm interested in this project for GSoC 2015. I have experience about JSON related features in MongoDB. Can I get more details about the project? Thank you
6. [~maduranga.siriwardena] - thanks for the interest. I think at this point that PHOENIX-628 has too many unknowns for being a GSoC item. A fair amount of design would be required first and then likely a feature branch in which to do the work. I'd recommend taking a look at PHOENIX-1661 and starting by implementing a good set of JSON built-in functions that'll allow us to get started.
7. GitHub user AakashPradeep opened a pull request: https://github.com/apache/phoenix/pull/76 PHOENIX-628 Support native JSON data type This pull request has following changes: 1. Renames a) PJsonDataType to PJson b) PJsonDataTypeTest to PJsonTest c) PhoenixJsonE2ETest to PhoenixJsonIT 2. Added new SqlException Code INVALID_JSON_DATA 3. Some changes in PhoenixJson 4. Removed overridden method from PJson for which default implementation is fine. a) coerceBytes() b) isCoercibleTo(). @JamesRTaylor and @twdsilva please review the changelist. Note : Some of the classes have space formatting problem, I am trying to fix my eclipse for that. Hopefully it will be resolved in next pull request. You can merge this pull request into a Git repository by running: $ git pull https://github.com/AakashPradeep/phoenix json Alternatively you can review and apply these changes as the patch at: https://github.com/apache/phoenix/pull/76.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #76 ---- commit bab8f40c745b0f3cfd28597d6938574f4ba91ea2 Author: Aakash <aakash.pradeep@salesforce.com> Date: 2015-04-14T01:32:10Z Support native JSON data type and json_extract_path function PHOENIX-1743 and PHOENIX-1710 commit a18f617398e32dd9f5599316a408f401302f3182 Author: Aakash <aakash.pradeep@salesforce.com> Date: 2015-04-24T09:31:55Z PHOENIX-628 Support native JSON data type - fix for first review ----
8. Github user twdsilva commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096277 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and

uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, --- End diff -- minor typo getPhoenixJson

9. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096324 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more

than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, --- End diff -- Nice catch.. I will fix it.

10. Github user twdsilva commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096341 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and

return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { --- End diff -- Since getPhoenixJson and getNullablePhoenixJson are almost the same, maybe have a helper function that takes a nullable argument in order to prevent code duplication.

11. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096442 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { +

jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { --- End diff -- getPhoenixJson() throws Exception if path is not found. It can be used on those case where we want to throw exception to user if path is not correct but in case we just want to return null if path is not valid then I dont want my logic to be catching the exception and then ignore it since Exception are always costly so I provided getNullablePhoenixJson() which return null if path is not valid. I will try to put them in a helper and have a boolean argument to specify whether to throw exception or not.

12. Github user twdsilva commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096496 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + + String jsonDataStr =

Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + return null; + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + return null; + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.node == null || this.node.isNull()) { + return null; + } else if (this.node.isValueNode()) { + + if (this.node.isNumber()) { + return this.node.getNumberValue().toString(); + } else if (this.node.isBoolean()) { + return String.valueOf(this.node.getBooleanValue()); + } else if (this.node.isTextual()) { + return this.node.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.node.isArray()) { + return getJsonAsString(); + } else if (this.node.isContainerNode()) { + return getJsonAsString(); + } + return null; + } + + @Override + public String toString() { + if (this.jsonAsString == null && this.node != null) { --- End diff -- can jsonAsString ever be null, seems like you create a PhoenixJson object you it will always have be from a string

13. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096515 --- Diff: phoenix-

core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new

PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + return null; + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + return null; + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.node == null || this.node.isNull()) { + return null; + } else if (this.node.isValueNode()) { + + if (this.node.isNumber()) { + return this.node.getNumberValue().toString(); + } else if (this.node.isBoolean()) { + return String.valueOf(this.node.getBooleanValue()); + } else if (this.node.isTextual()) { + return this.node.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.node.isArray()) { + return getJsonAsString(); + } else if (this.node.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + if (this.jsonAsString == null && this.node != null) { --- End diff -- when we call getPhoenixJson() from PhoenixJson object after applying path (String [] paths) then it will be null.

14. Github user twdsilva commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096621 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDataType.java --- @@ -1144,17 +1144,18 @@ public static PDataType fromLiteral(Object value) { return null; } for (PDataType type : PDataType.values()) { - if (type.isArrayType()) { - PhoenixArray arr = (PhoenixArray) value; - if ((type.getSqlType() == arr.baseType.sqlType + PDataType.ARRAY_TYPE_BASE) - && type.getJavaClass().isInstance(value)) { - return type; - } - } else { - if (type.getJavaClass().isInstance(value)) { - return type; - } + if (type.isArrayType() && type.getJavaClass().isInstance(value)) { --- End diff -- might be more readable if you just move isInstance outside if (type.getJavaClass().isInstance(value)) { if (type.isArrayType()) { ..... } else { return type; } }
15. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096669 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDataType.java --- @@ -1144,17 +1144,18 @@ public static PDataType fromLiteral(Object value) { return null; } for (PDataType type : PDataType.values()) { - if (type.isArrayType()) { - PhoenixArray arr = (PhoenixArray) value; - if ((type.getSqlType() == arr.baseType.sqlType + PDataType.ARRAY_TYPE_BASE) - && type.getJavaClass().isInstance(value)) { - return type; - } - } else { - if (type.getJavaClass().isInstance(value)) { - return type; - } + if (type.isArrayType() && type.getJavaClass().isInstance(value)) { --- End diff -- I get it (finally :-)). I like Thomas' suggestion, though.
16. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096692 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import

org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { --- End diff -- Typical pattern is to have a method method like getPhoenixJsonOrNull(). Calls call that one if they're ok getting null back. Then you have a getPhoenixJson() which calls getPhoenixJsonOrNull() and throws if it returns null. Will something like that work?

17. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29096924 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,280 @@ +/* + *

Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(byte[] jsonData, int offset, int length) + throws SQLException { + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getPhoenixJson(jsonDataStr); + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getPhoenixJson(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getPhoneixJson(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); + return phoenixJson; + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + private static PhoenixJson getPhoneixJson(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + JsonNode rootNode = objectMapper.readTree(jsonParser); + PhoenixJson phoenixJson = new PhoenixJson(rootNode); + return phoenixJson; + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.node = node; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link PhoenixJsonException} with the message + * having information about not found path. It is caller responsibility to wrap it in + * {@link SQLException} or catch it and return null to client. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws PhoenixJsonException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws PhoenixJsonException { + JsonNode node = this.node; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + try { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } catch (NumberFormatException nfe) { + throw new PhoenixJsonException("path: " + path + " not found", nfe); + } + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + throw new PhoenixJsonException("path: " + path + " not found"); + } + node = nodeTemp; + } + return new PhoenixJson(node); + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + *

<p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getNullablePhoenixJson(String[] paths) { --- End diff -- Nice suggestion that will work. I will change the code.

18. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29097031 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDataType.java --- @@ -1144,17 +1144,18 @@ public static PDataType fromLiteral(Object value) { return null; } for (PDataType type : PDataType.values()) { - if (type.isArrayType()) { - PhoenixArray arr = (PhoenixArray) value; - if ((type.getSqlType() == arr.baseType.sqlType + PDataType.ARRAY_TYPE_BASE) - && type.getJavaClass().isInstance(value)) { - return type; - } - } else { - if (type.getJavaClass().isInstance(value)) { - return type; - } - } + if (type.isArrayType() && type.getJavaClass().isInstance(value)) { --- End diff -- isInstance() is costly it has a native implementation, I will try to avoid it as much as possible.
19. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29097048 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDataType.java --- @@ -1144,17 +1144,18 @@ public static PDataType fromLiteral(Object value) { return null; } for (PDataType type : PDataType.values()) { - if (type.isArrayType()) { - PhoenixArray arr = (PhoenixArray) value; - if ((type.getSqlType() == arr.baseType.sqlType + PDataType.ARRAY_TYPE_BASE) - && type.getJavaClass().isInstance(value)) { - return type; - } - } else { - if (type.getJavaClass().isInstance(value)) { - return type; - } - } + if (type.isArrayType() && type.getJavaClass().isInstance(value)) { --- End diff -- @twdsilva , your placement of isInstance() is good since in current case if it is of Array type but not compatible class then it will call isInstance() twice (once in if () + once in else part) whereas in your case it will be called only once.
20. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102470 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/expression/ExpressionType.java --- @@ -204,7 +204,7 @@ SecondFunction(SecondFunction.class), WeekFunction(WeekFunction.class), HourFunction(HourFunction.class), - NowFunction(NowFunction.class) + NowFunction(NowFunction.class), --- End diff -- No change, please revert.
21. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102507 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,266 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more contributor license + * agreements. See the NOTICE file distributed with this work for additional information regarding + * copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance with the License. You may obtain a + * copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable + * law or agreed to in writing, software distributed under the License is distributed on an "AS IS" + * BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License + * for the specific language governing permissions and limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param data Buffer that contains data to parse + * @param

offset Offset of the first data byte within buffer + * @param length Length of contents to parse within buffer + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(byte[] jsonData, int offset, int length) + throws SQLException { + String jsonDataStr = Bytes.toString(jsonData, offset, length); + return getInstance(jsonDataStr); + } + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + PhoenixJson phoenixJson = getInstanceInternal(jsonParser); + /* + * input data has been stored as it is, since some data is lost when json parser runs, + * for example if a JSON object within the value contains the same key more than once + * then only last one is stored rest all of them are ignored, which will defy the + * contract of PJsonDataType of keeping user data as it is. + */ + phoenixJson.setJsonAsString(jsonData); --- End diff -- Pass the String through the PhoenixJson constructor. PhoenixJson should be immutable.

22. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102541 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PJson.java --- @@ -0,0 +1,223 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.types; + +import java.sql.SQLException; +import java.sql.Types; +import java.text.Format; + +import org.apache.hadoop.hbase.io.ImmutableBytesWritable; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.schema.IllegalDataException; +import org.apache.phoenix.schema.SortOrder; +import org.apache.phoenix.schema.json.PhoenixJson; +import org.apache.phoenix.util.ByteUtil; +import org.apache.phoenix.util.StringUtil; + +import com.google.common.base.Preconditions; + +/** + * <p> + * A Phoenix data type to represent JSON. The json data type stores an exact copy of the input text, + * which processing functions must reparse on each execution. Because the json type stores an exact + * copy of the input text, it will preserve semantically-insignificant white space between tokens, + * as well as the order of keys within JSON objects. Also, if a JSON object within the value + * contains the same key more than once, all the key/value pairs are kept. It stores the data as + * string in single column of HBase and it has same data size limit as Phoenix's Varchar. + * <p> + * JSON data types are for storing JSON (JavaScript Object Notation) data, as specified in RFC 7159. + * Such data can also be stored as text, but the JSON data types have the advantage of enforcing + * that each stored value is valid according to the JSON rules. + */ +public class PJson extends PDataType<String> { + + public static final PJson INSTANCE = new PJson(); + + PJson() { + super("JSON", Types.OTHER, PhoenixJson.class, null, 48); + } + + @Override + public int toBytes(Object object, byte[] bytes, int offset) { + if (object == null) { + return 0; + } + byte[] b = toBytes(object); + System.arraycopy(b, 0, bytes, offset, b.length); + return b.length; + } + + @Override + public byte[] toBytes(Object object) { + if (object == null) { + return ByteUtil.EMPTY_BYTE_ARRAY; + } + PhoenixJson phoenixJson = (PhoenixJson) object; + return PVarchar.INSTANCE.toBytes(phoenixJson.toString()); + } + + @Override + public Object toObject(byte[] bytes, int offset, int length, + @SuppressWarnings("rawtypes") PDataType actualType, SortOrder sortOrder, + Integer maxLength, Integer scale) { + if (!actualType.isCoercibleTo(this)) { + throwConstraintViolationException(actualType, this); + } + if (length == 0) { + return null; + } + return getPhoenixJson(bytes, offset, length); + } + + @Override + public Object toObject(Object object, @SuppressWarnings("rawtypes") PDataType actualType) { + if (object == null) { + return null; + } + if (equalsAny(actualType, PJson.INSTANCE)) { + return object; + } + if (equalsAny(actualType, PVarchar.INSTANCE)) { + return getJsonFromVarchar(object, actualType); + } + return throwConstraintViolationException(actualType, this); + } + + @Override + public boolean isCoercibleTo(@SuppressWarnings("rawtypes") PDataType targetType) { + return equalsAny(targetType, this, PVarchar.INSTANCE); + } + + @Override + public boolean isSizeCompatible(ImmutableBytesWritable ptr, Object value, + @SuppressWarnings("rawtypes") PDataType srcType, Integer maxLength, Integer scale, + Integer desiredMaxLength, Integer desiredScale)

{ + return PVarchar.INSTANCE.isSizeCompatible(ptr, value, srcType, maxLength, scale, + desiredMaxLength, desiredScale); + } + + @Override + public boolean isFixedWidth() { + return false; + } + + @Override + public int estimateByteSize(Object o) { + PhoenixJson phoenixJson = (PhoenixJson) o; + return phoenixJson.estimateByteSize(); + } + + @Override + public Integer getByteSize() { + return null; + } + + @Override + public int compareTo(Object lhs, Object rhs, @SuppressWarnings("rawtypes") PDataType rhsType) { + if (PJson.INSTANCE != rhsType) { + throwConstraintViolationException(rhsType, this); + } + PhoenixJson phoenixJsonLHS = (PhoenixJson) lhs; + PhoenixJson phoenixJsonRHS = (PhoenixJson) rhs; + return phoenixJsonLHS.compareTo(phoenixJsonRHS); + } + + @Override + public Object toObject(String value) { + return getPhoenixJson(value); + } + + @Override + public boolean isBytesComparableWith(@SuppressWarnings("rawtypes") PDataType otherType) { + return otherType == PJson.INSTANCE || otherType == PVarchar.INSTANCE; + } + + @Override + public String toStringLiteral(Object o, Format formatter) { + if (o == null) { + return StringUtil.EMPTY_STRING; + } + PhoenixJson phoenixJson = (PhoenixJson) o; + return PVarchar.INSTANCE.toStringLiteral(phoenixJson.toString(), formatter); + } + + @Override + public Object getSampleValue(Integer maxLength, Integer arrayLength) { + Preconditions.checkArgument(maxLength == null || maxLength >= 0); + + char[] key = new char[4]; + char[] value = new char[4]; + int length = maxLength != null ? maxLength : 1; + if (length > (key.length + value.length)) { + key = new char[length + 2]; + value = new char[length - key.length]; + } + int j = 1; + key[0] = ""; + key[j++] = 'k'; + for (int i = 2; i < key.length - 1; i++) { + key[j++] = (char) ('0' + RANDOM.get().nextInt(Byte.MAX_VALUE) % 10); + } + key[j] = ""; + + int k = 1; + value[0] = ""; + value[k++] = 'v'; + for (int i = 2; i < value.length - 1; i++) { + value[k++] = (char) ('0' + RANDOM.get().nextInt(Byte.MAX_VALUE) % 10); + } + value[k] = ""; + StringBuilder sbr = new StringBuilder(); + sbr.append("{").append(key).append(":").append(value).append("}"); + + byte[] bytes = Bytes.toBytes(sbr.toString()); + return getPhoenixJson(bytes, 0, bytes.length); --- End diff -- Get rid of this method: getPhoenixJson(byte[], offset, length). Instead, just have getPhoenixJson(String), since you're relying on having a String. Converting from a String -> byte[] -> String which is a waste.

23. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102549 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PJson.java --- @@ -0,0 +1,223 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.types; + +import java.sql.SQLException; +import java.sql.Types; +import java.text.Format; + +import org.apache.hadoop.hbase.io.ImmutableBytesWritable; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.schema.IllegalDataException; +import org.apache.phoenix.schema.SortOrder; +import org.apache.phoenix.schema.json.PhoenixJson; +import org.apache.phoenix.util.ByteUtil; +import org.apache.phoenix.util.StringUtil; + +import com.google.common.base.Preconditions; + +/** + * <p> + * A Phoenix data type to represent JSON. The json data type stores an exact copy of the input text, + * which processing functions must reparse on each execution. Because the json type stores an exact + * copy of the input text, it will preserve semantically-insignificant white space between tokens, + * as well as the order of keys within JSON objects. Also, if a JSON object within the value + * contains the same key more than once, all the key/value pairs are kept. It stores the data as + * string in single column of HBase and it has same data size limit as Phoenix's Varchar. + * <p> + * JSON data types are for storing JSON (JavaScript Object Notation) data, as specified in RFC 7159. + * Such data can also be stored as text, but the JSON data types have the advantage of enforcing + * that each stored value is valid according to the JSON rules. + */ +public class PJson extends PDataType<String> { + + public static final PJson INSTANCE = new PJson(); + + PJson() { + super("JSON", Types.OTHER, PhoenixJson.class, null, 48); + } + + @Override + public int toBytes(Object object, byte[] bytes, int offset) { + + if (object == null) { + return 0; + } + byte[] b = toBytes(object); + System.arraycopy(b, 0, bytes, offset, b.length); + return b.length; + + } + + @Override + public byte[] toBytes(Object object) { + if (object == null) { + return ByteUtil.EMPTY_BYTE_ARRAY; + } + PhoenixJson phoenixJson = (PhoenixJson) object; + return

PVarchar.INSTANCE.toBytes(phoenixJson.toString()); --- End diff -- Better if you delegate the toBytes() call to PhoenixJson. That'll give you the flexibility to cache the byte[] if you wanted to.

24. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102581 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PJson.java --- @@ -0,0 +1,223 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.types; + +import java.sql.SQLException; +import java.sql.Types; +import java.text.Format; + +import org.apache.hadoop.hbase.io.ImmutableBytesWritable; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.schema.IllegalDataException; +import org.apache.phoenix.schema.SortOrder; +import org.apache.phoenix.schema.json.PhoenixJson; +import org.apache.phoenix.util.ByteUtil; +import org.apache.phoenix.util.StringUtil; + +import com.google.common.base.Preconditions; + +/** + * <p> + * A Phoenix data type to represent JSON. The json data type stores an exact copy of the input text, + * which processing functions must reparse on each execution. Because the json type stores an exact + * copy of the input text, it will preserve semantically-insignificant white space between tokens, + * as well as the order of keys within JSON objects. Also, if a JSON object within the value + * contains the same key more than once, all the key/value pairs are kept. It stores the data as + * string in single column of HBase and it has same data size limit as Phoenix's Varchar. + * <p> + * JSON data types are for storing JSON (JavaScript Object Notation) data, as specified in RFC 7159. + * Such data can also be stored as text, but the JSON data types have the advantage of enforcing + * that each stored value is valid according to the JSON rules. + */ +public class PJson extends PDataType<String> { + + public static final PJson INSTANCE = new PJson(); + + PJson() { + super("JSON", Types.OTHER, PhoenixJson.class, null, 48); + } + + @Override + public int toBytes(Object object, byte[] bytes, int offset) { + if (object == null) { + return 0; + } + byte[] b = toBytes(object); + System.arraycopy(b, 0, bytes, offset, b.length); + return b.length; + } + + @Override + public byte[] toBytes(Object object) { + if (object == null) { + return ByteUtil.EMPTY_BYTE_ARRAY; + } + PhoenixJson phoenixJson = (PhoenixJson) object; + return PVarchar.INSTANCE.toBytes(phoenixJson.toString()); + } + + @Override + public Object toObject(byte[] bytes, int offset, int length, + @SuppressWarnings("rawtypes") PDataType actualType, SortOrder sortOrder, + Integer maxLength, Integer scale) { + if (!actualType.isCoercibleTo(this)) { + throwConstraintViolationException(actualType, this); + } + if (length == 0) { + return null; + } --- End diff -- Here you want to call through to PVarchar to get a String, then call getPhoenixJson(String). This will take care of the maxLength check and the inversion if necessary based on sortOrder. String jsonStr = PVarChar.toObject(bytes, offset, length, actualType, sortOrder, maxLength, scale); return getPhoenixJson(jsonStr);

25. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29102587 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJsonException.java --- @@ -0,0 +1,32 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +public class PhoenixJsonException extends Exception { --- End diff -- Get rid of this and instead declare your checked exception as SQLException

26. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29108294 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,266 @@ +/* + *

Licensed to the Apache Software Foundation (ASF) under one or more contributor license + * agreements. See the NOTICE file distributed with this work for additional information regarding + * copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance with the License. You may obtain a + * copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable + * law or agreed to in writing, software distributed under the License is distributed on an "AS IS" + * BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License + * for the specific language governing permissions and limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; --- End diff -- Needs suggestions to decide whether json data should be cached as byte [] or String. byte[] ------- If it is byte[] following methods would not involve any String to byte [] or byte [] to String conversion 1. toBytes(Object object, byte[] bytes, int offset) 2. toBytes(Object object) 3. toObject(byte[] bytes, int offset, int length, PDataType actualType, SortOrder sortOrder, Integer maxLength, Integer scale) But for following, we require byte [] -> String 1. toStringLiteral(Object o, Format formatter) And for toObject(Object object, @SuppressWarnings("rawtypes") PDataType actualType) if actualType is Varchar we require String -> byte [] String ----- Where as in case of String, following methods will require String -> byte [] 1. toBytes(Object object, byte[] bytes, int offset) 2. toBytes(Object object) 3. toObject(byte[] bytes, int offset, int length, PDataType actualType, SortOrder sortOrder, Integer maxLength, Integer scale) No conversion required for toStringLiteral(Object o, Format formatter) and toObject(Object object, @SuppressWarnings("rawtypes") PDataType actualType) if actualType is Varchar. byte[] seems to be more efficient here but I am not sure which is most used method and will help in performance or not. @JamesRTaylor @twdsilva

27. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29110600 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,266 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more contributor license + * agreements. See the NOTICE file distributed with this work for additional information regarding + * copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance with the License. You may obtain a + * copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable + * law or agreed to in writing, software distributed under the License is distributed on an "AS IS" + * BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License + * for the specific language governing permissions and limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode node; + private String jsonAsString; --- End diff -- Cache String as you always need a String for parsing and comparison. >
28. Github user AakashPradeep commented on the pull request: https://github.com/apache/phoenix/pull/76#issuecomment-96815390 @twdsilva @JamesRTaylor I have

made changes as per the comments. Please let me know if there is anything more.

29. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29200389 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/types/PVarchar.java --- @@ -30,134 +31,145 @@ public class PVarchar extends PDataType<String> { - public static final PVarchar INSTANCE = new PVarchar(); - - private PVarchar() { - super("VARCHAR", Types.VARCHAR, String.class, null, 0); - } - - @Override - public byte[] toBytes(Object object) { - // TODO: consider using avro UTF8 object instead of String - // so that we get get the size easily - if (object == null) { - return ByteUtil.EMPTY_BYTE_ARRAY; - } - return Bytes.toBytes((String) object); - } - - @Override - public int toBytes(Object object, byte[] bytes, int offset) { - if (object == null) { - return 0; - } - byte[] b = toBytes(object); // TODO: no byte[] allocation: use CharsetEncoder - System.arraycopy(b, 0, bytes, offset, b.length); - return b.length; - } - - @Override - public Object toObject(byte[] bytes, int offset, int length, PDataType actualType, - SortOrder sortOrder, Integer maxLength, Integer scale) { - if (!actualType.isCoercibleTo(this)) { - throwConstraintViolationException(actualType, this); - } - if (length == 0) { - return null; - } - if (sortOrder == SortOrder.DESC) { - bytes = SortOrder.invert(bytes, offset, length); - offset = 0; - } - return Bytes.toString(bytes, offset, length); - } - - @Override - public Object toObject(Object object, PDataType actualType) { - if (equalsAny(actualType, this, PChar.INSTANCE)) { - String s = (String) object; - return s == null || s.length() > 0 ? s : null; - } - return throwConstraintViolationException(actualType, this); - } - - @Override - public boolean isCoercibleTo(PDataType targetType) { - return equalsAny(targetType, this, PChar.INSTANCE, PVarbinary.INSTANCE, PBinary.INSTANCE); - } - - @Override - public boolean isCoercibleTo(PDataType targetType, Object value) { - if (isCoercibleTo(targetType)) { - if (targetType.equals(PChar.INSTANCE)) { - return value != null; - } - return true; - } - return false; - } - - @Override - public boolean isSizeCompatible(ImmutableBytesWritable ptr, Object value, PDataType srcType, - Integer maxLength, Integer scale, Integer desiredMaxLength, - Integer desiredScale) { - if (ptr.getLength() != 0 && maxLength != null && desiredMaxLength != null) { - return maxLength <= desiredMaxLength; - } - return true; - } - - @Override - public boolean isFixedWidth() { - return false; - } - - @Override - public int estimateByteSize(Object o) { - String value = (String) o; - return value == null ? 1 : value.length(); - } - - @Override - public Integer getByteSize() { - return null; - } - - @Override - public int compareTo(Object lhs, Object rhs, PDataType rhsType) { - return ((String) lhs).compareTo((String) rhs); - } - - @Override - public Object toObject(String value) { - return value; - } - - @Override - public boolean isBytesComparableWith(PDataType otherType) { - return super.isBytesComparableWith(otherType) || otherType == PChar.INSTANCE; - } - - @Override - public String toStringLiteral(Object o, Format formatter) { - if (formatter != null) { - return "'" + formatter.format(o) + "'"; - } - return "'" + StringUtil.escapeStringConstant(o.toString()) + "'"; - } - - private char[] sampleChars = new char[1]; - - @Override - public Object getSampleValue(Integer maxLength, Integer arrayLength) { - Preconditions.checkArgument(maxLength == null || maxLength >= 0); - int length = maxLength != null ? maxLength : 1; - if (length != sampleChars.length) { - sampleChars = new char[length]; - } - for (int i = 0; i < length; i++) { - sampleChars[i] = (char) RANDOM.get().nextInt(Byte.MAX_VALUE); - } - return new String(sampleChars); - } + public static final PVarchar INSTANCE = new PVarchar(); --- End diff -- Minor nit: indenting looks off here. Should be no tabs and 4 space chars.

30. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29200934 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException;

+import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + return null; + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if

(this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { + return 0; + } else { + return this.toString().compareTo(o.toString()); + } + } + + private PhoenixJson getPhoenixJsonInternal(String[] paths) { + JsonNode node = this.rootNode; + for (String path : paths) { + JsonNode nodeTemp = null; + if (node.isArray()) { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + return null; + } + node = nodeTemp; + } + return new PhoenixJson(node, null); --- End diff -- Might be better to just call node.toString() here instead of passing in null. Otherwise, the getJsonAsString() and toString() methods will allocate a big string on each call.

31. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29201182 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than

once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2": {"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; --- End diff -- Why do we need to compare both String and the JsonNode here? Seems like we should just compare the JsonNode instead.

32. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29201597 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws

{@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- Checking both JsonNode.equals and doing a string compare on the string representation is not so good. Does JSON have the concept of < and > comparison? Should we always rely on string equality to mean node equality? When would that break down? If that works, we should

perhaps consistently check that for equality and hashcode, etc. If that doesn't work, we may need to introduce the concept of a type that doesn't support comparison, but only = and !=.

33. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29201607 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they

appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; --- End diff -- I think String match will faster since JsonNode.equals() traverse the entire Tree Model of Json.

34. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29202428 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) {

Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- Good that you brought it up. I was not happy with it and about to discuss. Jackson library does not have any Json Comparator. I thought of writing own comparator but there are too many unknowns and did not find any specification around it. I will have to assume too many things which is not good. Probably we should not support '=' and '!=' with Json data type. Postgres also throw an error on using '=' operator with json column # select * from t1 where c1 ='[1,2,3]'; ERROR: operator does not exist: json = unknown LINE 1: select * from t1 where c1 ='[1,2,3]'; ^ HINT: No operator matches the given name and argument type(s). You might need to add explicit type casts. where as this works # select * from t1 where c1::varchar ='[1,2,3]'; c1 --------- [1,2,3] (1 row)

35. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29202881 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS

IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return

this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- Is throwing an exception from compareTo(), will help us? Since postgres does not allow '=' and order by with json.

36. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29203436 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; +import com.google.common.base.Preconditions; +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p>

* for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2": {"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- Or should we do something like PArrayDataType.compareTo() public int compareTo(Object lhs, Object rhs) { PhoenixArray lhsArr = (PhoenixArray) lhs; PhoenixArray rhsArr = (PhoenixArray) rhs; if(lhsArr.equals(rhsArr)) { return 0; } return 1; }

37. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29205463 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException;

+import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + return null; + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if

(this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- That just seems wrong. The compare(Object, Object, PDataType) method is only called in a few places. For example, with COUNT DISTINCT. We rely on the compareTo() method that takes byte[] more. I think we should just do the String comparison as that'll match what the byte[] based compare method does.

38. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29205807 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,255 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson =

getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } else if (this.equals(o)) { --- End diff -- ok. that make sense. I will change it to String comparison.

39. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29262890 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of

PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { + if (other.rootNode != null) return false; + } else if (!this.rootNode.equals(other.rootNode)) return false; + return true; + } + + /** + * @return length of the string represented by the current {@link PhoenixJson}. + */ + public int estimateByteSize() { + String jsonStr = getJsonAsString(); + return jsonStr == null ? 1 : jsonStr.length(); + } + + public byte[] toBytes() { + return Bytes.toBytes(getJsonAsString()); + } + + @Override + public int compareTo(PhoenixJson o) { + if (o == null) { + return 1; + } + return toString().compareTo(o.toString()); + } + + private PhoenixJson getPhoenixJsonInternal(String[] paths) { + JsonNode node = this.rootNode; + for (String path : paths) { + JsonNode nodeTemp = null; + if

(node.isArray()) { + int index = Integer.parseInt(path); + nodeTemp = node.path(index); + } else { + nodeTemp = node.path(path); + } + if (nodeTemp == null || nodeTemp.isMissingNode()) { + return null; + } + node = nodeTemp; + } + return new PhoenixJson(node, node.toString()); + } + + private String getJsonAsString() { + if (this.jsonAsString != null) { --- End diff -- Should just be the following, as it's always passed in: return this.jsonAsString;

40. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29263039 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to

know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- Remove references to rootNode.equals(). We can just rely on the jsonString being equal. We can do more for the JSONB data type, but this will be the most consistent way to go for now.

41. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29263073 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) +

.setMessage(x.getMessage()).build().buildException(); + } + + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); --- End diff -- Change to use this.jsonAsString.hashCode().

42. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29271341 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and

uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + return null; + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff --

String equals wont be able to match this kind of json equality {"k1":"v1","k2":"v2"} against {"k2":"v2","k1":"v1"}.

43. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29273245 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they

appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- If a JSON type is used in the row key (where the comparison is done at the byte level), this won't be considered equals either. Better in my mind to be consistent. We should normalize the JSON on input if order of tags doesn't matter.

44. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29276493 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser)

throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- We have two options: 1) We should not allow json column as primary key as in Postgres. create table json_pk(c1 json primary key); ERROR: data type json has no default operator class for access method "btree" HINT: You must specify an operator class for the index or define a default operator class for the data type 2) We should normalize it. Which one you suggest ?

45. Github user JamesRTaylor commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29279471 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import

```
org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + return null; + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + +
```

@Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- I already suggested it: (2). We don't have a declarative way of doing (1) right now, so (2) is the only viable option.

46. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29280348 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.phoenix.schema.json; +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */ + PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":

{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- I looked into json specification and it says "Json: An object is an unordered set of name/value pairs". So order of key value does not matter. I looked into postgres and this what i found 1) does not support = and != operator 2) not allow to group by or use count distinct with json column basically they do not allow any equality operator on json. Here is the error message in Postgres select count(distinct(c1)) from t1; ERROR: could not identify an equality operator for type json LINE 1: select count(distinct(c1)) from t1; ^ select count(*) from t1 group by c1 ; ERROR: could not identify an equality operator for type json LINE 1: select count(*) from t1 group by c1 ; ^ IMHO we should do the same for JSON in Phoenix. And allow followings: 1. select count(*) from json_table group by json_extract_path_text(json_column, ARRAY['k1']) ; 2. select * from json_table where json_extract_path_text(json_column, ARRAY['k1'])='v1';

47. Github user AakashPradeep commented on a diff in the pull request: https://github.com/apache/phoenix/pull/76#discussion_r29280570 --- Diff: phoenix-core/src/main/java/org/apache/phoenix/schema/json/PhoenixJson.java --- @@ -0,0 +1,252 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one + * or more contributor license agreements. See the NOTICE file + * distributed with this work for additional information + * regarding copyright ownership. The ASF licenses this file + * to you under the Apache License, Version 2.0 (the + * "License"); you may not use this file except in compliance + * with the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +package org.apache.phoenix.schema.json; + +import java.io.IOException; +import java.sql.SQLException; +import java.util.Arrays; + +import org.apache.hadoop.hbase.util.Bytes; +import org.apache.phoenix.exception.SQLExceptionCode; +import org.apache.phoenix.exception.SQLExceptionInfo; +import org.codehaus.jackson.JsonFactory; +import org.codehaus.jackson.JsonNode; +import org.codehaus.jackson.JsonParser; +import org.codehaus.jackson.JsonParser.Feature; +import org.codehaus.jackson.JsonProcessingException; +import org.codehaus.jackson.map.ObjectMapper; +import org.codehaus.jackson.node.ValueNode; + +import com.google.common.base.Preconditions; + +/** + * The {@link PhoenixJson} wraps json and uses Jackson library to parse and traverse the json. It + * should be used to represent the JSON data type and also should be used to parse Json data and + * read the value from it. It always conside the last value if same key exist more than once. + */ +public class PhoenixJson implements Comparable<PhoenixJson> { + private final JsonNode rootNode; + /* + * input data has been stored as it is, since some data is lost when json parser runs, for + * example if a JSON object within the value contains the same key more than once then only last + * one is stored rest all of them are ignored, which will defy the contract of

PJsonDataType of + * keeping user data as it is. + */ + private final String jsonAsString; + + /** + * Static Factory method to get an {@link PhoenixJson} object. It also validates the json and + * throws {@link SQLException} if it is invalid with line number and character. + * @param jsonData Json data as {@link String}. + * @return {@link PhoenixJson}. + * @throws SQLException + */ + public static PhoenixJson getInstance(String jsonData) throws SQLException { + if (jsonData == null) { + return null; + } + try { + JsonFactory jsonFactory = new JsonFactory(); + JsonParser jsonParser = jsonFactory.createJsonParser(jsonData); + JsonNode jsonNode = getRootJsonNode(jsonParser); + return new PhoenixJson(jsonNode, jsonData); + } catch (IOException x) { + throw new SQLExceptionInfo.Builder(SQLExceptionCode.INVALID_JSON_DATA).setRootCause(x) + .setMessage(x.getMessage()).build().buildException(); + } + + } + + /** + * Returns the root of the resulting {@link JsonNode} tree. + */ + private static JsonNode getRootJsonNode(JsonParser jsonParser) throws IOException, + JsonProcessingException { + jsonParser.configure(Feature.ALLOW_COMMENTS, true); + ObjectMapper objectMapper = new ObjectMapper(); + try { + return objectMapper.readTree(jsonParser); + } finally { + jsonParser.close(); + } + } + + /* Default for unit testing */PhoenixJson(final JsonNode node, final String jsonData) { + Preconditions.checkNotNull(node, "root node cannot be null for json"); + this.rootNode = node; + this.jsonAsString = jsonData; + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it throws {@link SQLException}. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + * @throws SQLException + */ + public PhoenixJson getPhoenixJson(String[] paths) throws SQLException { + try { + PhoenixJson phoenixJson = getPhoenixJsonInternal(paths); + if (phoenixJson == null) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found."); + } + return phoenixJson; + } catch (NumberFormatException nfe) { + throw new SQLException("path: " + Arrays.asList(paths) + " not found.", nfe); + } + } + + /** + * Get {@link PhoenixJson} for a given json paths. For example : + * <p> + * <code> + * {"f2":{"f3":1},"f4":{"f5":99,"f6":{"f7":"2"}}}' + * </code> + * <p> + * for this source json, if we want to know the json at path {'f4','f6'} it will return + * {@link PhoenixJson} object for json {"f7":"2"}. It always returns the last key if same key + * exist more than once. + * <p> + * If the given path is unreachable then it return null. + * @param paths {@link String []} of path in the same order as they appear in json. + * @return {@link PhoenixJson} for the json against @paths. + */ + public PhoenixJson getPhoenixJsonOrNull(String[] paths) { + try { + return getPhoenixJsonInternal(paths); + } catch (NumberFormatException nfe) { + // ignore + } + return null; + } + + /** + * Serialize the current {@link PhoenixJson} to String. Its required for + * json_extract_path_text(). If we just return node.toString() it will wrap String value in + * double quote which is not the expectation, hence avoiding calling toString() on + * {@link JsonNode} until PhoenixJson represent a Json Array or container for Json object. If + * PhoenixJson just represent a {@link ValueNode} then it should return value returned from + * objects toString(). + */ + public String serializeToString() { + if (this.rootNode == null || this.rootNode.isNull()) { + return null; + } else if (this.rootNode.isValueNode()) { + + if (this.rootNode.isNumber()) { + return this.rootNode.getNumberValue().toString(); + } else if (this.rootNode.isBoolean()) { + return String.valueOf(this.rootNode.getBooleanValue()); + } else if (this.rootNode.isTextual()) { + return this.rootNode.getTextValue(); + } else { + return getJsonAsString(); + } + } else if (this.rootNode.isArray()) { + return getJsonAsString(); + } else if (this.rootNode.isContainerNode()) { + return getJsonAsString(); + } + + return null; + + } + + @Override + public String toString() { + return getJsonAsString(); + } + + @Override + public int hashCode() { + final int prime = 31; + int result = 1; + result = prime * result + ((this.rootNode == null) ? 0 : this.rootNode.hashCode()); + return result; + } + + @Override + public boolean equals(Object obj) { + if (this == obj) return true; + if (obj == null) return false; + if (getClass() != obj.getClass()) return false; + PhoenixJson other = (PhoenixJson) obj; + if ((this.jsonAsString != null) && (other.jsonAsString != null)) { + if (this.jsonAsString.equals(other.jsonAsString)) return true; + } + if (this.rootNode == null) { --- End diff -- Sorry did not read your last comment. So do you want to normalize it as part of this JIRA ?

48. GitHub user petercdc opened a pull request: https://github.com/apache/phoenix/pull/88 PHOENIX-628 Support native JSON data type This pull request has following changes: 1. JSON Operator : "->" "->>" "#>" "#>>" 2. JSON Boolean Operactor : "?" "?|" "?&" "<@" "@>" For more detail about how to use these operators, Please See http://www.postgresql.org/docs/9.4/static/functions-json.html You can merge this pull request into a Git repository by running: $ git pull https://github.com/petercdc/phoenix json Alternatively you can review and apply these changes as the patch at:

https://github.com/apache/phoenix/pull/88.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #88 ---- commit db4432fa76d4229ad4939c995a0a374004a2a1ff Author: LiChiachi <test@example.com> Date: 2015-04-24T11:05:34Z This patch can support json operation operator '->>' can be run commit 415b5257d6b3b6e4347f72fedae626e8e0364443 Author: Andy <ex2s62026202@gmail.com> Date: 2015-04-30T08:54:04Z Add json operation (first ver) commit 3294fccf2d9a5c53ac51e81722a202fdae7ce977 Author: LiChiachi <test@example.com> Date: 2015-05-19T05:19:23Z Create Json Point Expression commit 960e0e4eb9001ddbdd83a02cffa2ca04ea061cc4 Author: LiChiachi <test@example.com> Date: 2015-05-19T07:03:58Z Append ExpressionType and can be run commit 9c4f02475aa9fa41426f474e320ad34de66d205a Author: LiChiachi <test@example.com> Date: 2015-05-19T07:25:41Z Use Byte String to get or put Json Structure commit 2c214ba9590b81f83454b417892fb2fa9dc80c10 Author: Andy <ex2s62026202@gmail.com> Date: 2015-05-19T10:30:53Z Merge branch 'jsonOperatorFromChiachi' into json Conflicts: phoenix-assembly/src/build/components-major-client.xml phoenix-assembly/src/build/server.xml phoenix-core/pom.xml phoenix-core/src/main/antlr3/PhoenixSQL.g phoenix-core/src/main/java/org/apache/phoenix/compile/ExpressionCompiler.java phoenix-core/src/main/java/org/apache/phoenix/expression/ExpressionType.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/BaseExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/CloneExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/ExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/StatelessTraverseAllExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/StatelessTraverseNoExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/parse/JsonPathAsElementParseNode.java phoenix-core/src/main/java/org/apache/phoenix/parse/JsonPathAsTextParseNode.java phoenix-core/src/main/java/org/apache/phoenix/parse/JsonPointAsElementParseNode.java phoenix-core/src/main/java/org/apache/phoenix/parse/JsonPointAsTextParseNode.java phoenix-core/src/main/java/org/apache/phoenix/parse/JsonSingleKeySearchParseNode.java phoenix-core/src/main/java/org/apache/phoenix/parse/ParseNodeFactory.java commit 5257535902ebc430239c9a07464845ab163ef0ff Author: Andy <ex2s62026202@gmail.com> Date: 2015-05-19T10:35:22Z Add JSON Operation commit 09fae4aad2bf875225a68ac04ec0feb4fb618337 Author: LiChiachi <test@example.com> Date: 2015-05-21T08:48:43Z Merge branch 'jsonOperator' into jsonForGooYoi Conflicts: phoenix-core/src/main/java/org/apache/phoenix/compile/ExpressionCompiler.java phoenix-core/src/main/java/org/apache/phoenix/expression/ExpressionType.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/BaseExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/CloneExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/ExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/StatelessTraverseAllExpressionVisitor.java phoenix-core/src/main/java/org/apache/phoenix/expression/visitor/StatelessTraverseNoExpressionVisitor.java commit d16e731d37649bf562e1b1977b42c75f00555d4b Author: LiChiachi <test@example.com> Date: 2015-05-21T08:52:52Z debug for '->' operator commit efdf95dd14955c5329fb7f631e647387a8ca3e93 Author: LiChiachi <test@example.com> Date: 2015-05-25T02:41:25Z all operation can be executed. commit a24bf459b315c476129fb41734a18ef9e794f84f Author: Andy <ex2s62026202@gmail.com> Date: 2015-05-27T08:35:05Z Add JSON opeation(fix bug) commit 9acfaa3aab8a038b23390505737897acf38c3852 Author: LiChiachi <test@example.com> Date: 2015-05-27T11:35:56Z Merge branch 'jsonToBeMerge' into jsonMain commit 4e500eff05844efac8f209482f03bfa54b4ce9d7 Author: LiChiachi <test@example.com> Date: 2015-05-29T04:43:16Z Create JSON operation JUNIT test. commit af6a2ec0ab7d3c74275fe7ee28a6f33e82ccd1b1 Author: LiChiachi <test@example.com> Date: 2015-05-29T05:56:19Z Update JUNIT test commit 453576bc906374a25418cf408f43b43ba7b2ef10 Author: Andy <ex2s62026202@gmail.com> Date: 2015-06-01T01:07:23Z JSON operation(fix bug) commit c46f0e905cb55f500fc8d1b7a371c93fc68fad60 Author: Andy <ex2s62026202@gmail.com> Date: 2015-06-01T01:58:04Z Merge branch 'jsonToBeMerge' into json commit 194deace60e5fe1144a945e5faac82a6212dec8a Author: Andy <ex2s62026202@gmail.com> Date: 2015-06-12T05:28:05Z fix JSON operation BUG ----

49. Github user twdsilva commented on the pull request: https://github.com/apache/phoenix/pull/88#issuecomment-115383349 @petercdc Can you please rebase

this pull request with the json branch? @AakashPradeep had already implemented the json data type.
50. Github user petercdc closed the pull request at: https://github.com/apache/phoenix/pull/88
51. GitHub user petercdc opened a pull request: https://github.com/apache/phoenix/pull/99 PHOENIX-628 Support native JSON data type Adding following JSON operators: JSON data Operator : "->" "->>" "#>" "#>>" JSON Boolean Operactor : "?" "?|" "?&" "<@" "@>" For more detail about how to use these operators, Please See http://www.postgresql.org/docs/9.4/static/functions-json.html You can merge this pull request into a Git repository by running: $ git pull https://github.com/petercdc/phoenix jsoncdc Alternatively you can review and apply these changes as the patch at: https://github.com/apache/phoenix/pull/99.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #99 ---- commit a131aa3664d18a048a41b0dab9155d90b17103f5 Author: LiChiachi <test@example.com> Date: 2015-07-01T10:55:39Z rebase from json brance (but just recreate new branch base on json branch) fixed: All ParseNode and Visiter , All ExpressionVisiter , PhoenixJson method to be used for Expression method Not fixed yet: Expression method exclusive JsonPonit Operator commit 8babfb5fba20d24a1c65b02fdd13075b0632d4fc Author: LiChiachi <test@example.com> Date: 2015-07-02T08:48:36Z rebase on json branch (but create new branch base on json branch) fixed: All ParseNode and All ExpressionNode exclusive Superset and Subset not fixed yet: Type Checking for As Element Node , Superset and Subset Node commit 077d5f41da0c22bcafd9387df90c60ea0be659ed Author: LiChiachi <test@example.com> Date: 2015-07-06T09:56:47Z rebase on json branch (but recreate new branch based on json branch) fixed: All ParseNode and ExpressionNode are done not fixed yet: TypeChecking for Skipping checking type on JsongetElementExpression commit 5b8a43e09202b04e3371ff8cfd8451b68baa2759 Author: LiChiachi <test@example.com> Date: 2015-07-10T01:52:53Z rebase on json branch (but recreate new branch based on json branch) fix bug wronging with superset and subset node commit 7793e94cf27d4057b633b7c67234f6e451d5577c Author: Andy <ex2s62026202@gmail.com> Date: 2015-07-11T09:37:09Z First Version commit 229939099154042af98b24839cfb483624400fb0 Author: Andy <ex2s62026202@gmail.com> Date: 2015-07-11T11:15:14Z fix type checking bug ----