Item 71
git_comments:

1. All ValueVector types have been handled.
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. false, and bit was previously set
4. * * Allocate a new memory space for this vector. Must be called prior to using the ValueVector. * * @param valueCount * The number of values which can be contained within this vector.
5. * * Get the element at the specified position. * * @param index position of the value * @return value of the element, if not null * @throws NullValueException if the value is null
6. * * Returns the maximum number of values contained within this vector. * @return Vector size
7. * * For testing only -- randomize the buffer contents
8. * * Get the Java Object representation of the element at the specified position * * @param index Index of the value to get
9. * * Repeated${minor.class} implements a vector with multple values per row (e.g. JSON array or * repeated protobuf field). The implementation uses two additional value vectors; one to convert * the index offset to the underlying element offset, and another to store the number of values * in the vector. * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
10. number of repeated elements in each record offsets to start of each record
11. * * Get the size requirement (in bytes) for the given number of values. Takes derived * type specs into account.
12. logger.debug("BIT GET: index: {}, byte: {}, mask: {}, masked byte: {}", index, data.getByte((int)Math.floor(index/8)), (int)Math.pow(2, (index % 8)), data.getByte((int)Math.floor(index/8)) & (int)Math.pow(2, (index % 8)));
13. * * Add an element to the given record index. This is similar to the set() method in other * value vectors, except that it permits setting multiple values for a single record. * * @param index record of the element to add * @param value value to add to the given row
14. * * Get the elements at the given index.
15. ** * 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. **
16. * * Set the element at the given index to the given value. Note that widths smaller than * 32 bits are handled by the ByteBuf interface. * * @param index position of the bit to set * @param value value to set
17. * * Get information about how this field is materialized. * @return
18. * * Release the underlying ByteBuf and reset the ValueVector
19. * * Get the size requirement (in bytes) for the given number of values.
20. * * ${minor.class} implements a vector of variable width values. Elements in the vector * are accessed by position from the logical start of the vector. A fixed width lengthVector * is used to convert an element's position to it's offset from the start of the (0-based) * ByteBuf. Size is inferred by adjacent elements. * The width of each element is ${type.width} byte(s) * The equivilent Java primitive is

'${minor.javaType!type.javaType}' * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.

21. * * Set the bit at the given index to the specified value. * * @param index position of the bit to set * @param value value to set (either 1 or 0)
22. set the end offset of the buffer
23. * * Get a value for the given record. Each element in the repeated field is accessed by * the positionIndex param. * * @param index record containing the repeated field * @param positionIndex position within the repeated field * @return element at the given position in the given record
24. * * Release supporting resources.
25. * * Get the size requirement (in bytes) for the given number of values. Only accurate * for fixed width value vectors.
26. * * Get the byte holding the desired bit, then mask all other bits. Iff the result is 0, the * bit was not set. * * @param index position of the bit in the vector * @return 1 if set, otherwise 0
27. * * Get the explicitly specified size of the allocated buffer, if available. Otherwise * calculate the size based on width and record count.
28. * * Allocate a new memory space for this vector. Must be called prior to using the ValueVector. * * @param valueCount The number of values which may be contained by this vector.
29. * * Return the underlying buffers associated with this vector. Note that this doesn't impact the * reference counts for this buffer so it only should be used for in-context access. Also note * that this buffer changes regularly thus external classes shouldn't hold a reference to * it (unless they change it). * * @return The underlying ByteBuf.
30. * * Allocate a new memory space for this vector. Must be called prior to using the ValueVector. * * @param valueCount The number of values which can be contained within this vector.
31. * * Mutable${minor.class} implements a mutable vector of fixed width values. Elements in the * vector are accessed by position from the logical start of the vector. Values should be pushed * onto the vector sequentially, but may be randomly accessed. * The width of each element is ${type.width} byte(s) * The equivilent Java primitive is '${minor.javaType!type.javaType}' * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
32. * * Allocate a new memory space for this vector. Must be called prior to using the ValueVector. * * @param totalBytes Optional desired size of the underlying buffer. Specifying 0 will * estimate the size based on valueCount. * @param sourceBuffer Optional ByteBuf to use for storage (null will allocate automatically). * @param valueCount Number of values in the vector.
33. * * ValueVectorTypes defines a set of template-generated classes which implement type-specific * value vectors. The template approach was chosen due to the lack of multiple inheritence. It * is also important that all related logic be as efficient as possible.
34. * * Set the variable length element at the specified index to the supplied byte array. * * @param index position of the bit to set * @param bytes array of bytes to write
35. * * Get the number of records allocated for this value vector. * @return number of allocated records
36. * * Nullable${minor.class} implements a vector of values which could be null. Elements in the vector * are first checked against a fixed length vector of boolean values. Then the element is retrieved * from the base class (if not null). * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
37. * * Allocate a new memory space for this vector. Must be called prior to using the ValueVector. * * @param valueCount * The number of elements which can be contained within this vector.
38. * * ValueVector.Base implements common logic for all immutable value vectors.
39. * * Get the metadata for this field. * @return
40. * * MutableBit implements a vector of bit-width values. Elements in the vector are accessed * by position from the logical start of the vector. Values should be pushed onto the vector * sequentially, but may be randomly accessed. * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
41. true
42. * * Bit implements a vector of bit-width values. Elements in the vector are accessed * by position from the logical start of the vector. * The width of each element is 1 bit. * The equivilent Java primitive is an int containing the value '0' or '1'. * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
43. * * Define the number of records that are in this value vector. * @param recordCount Number of records active in this vector.
44. * * Mutable${minor.class} implements a vector of variable width values. Elements in the vector * are accessed by position from the logical start of the vector. A fixed width lengthVector * is used to convert

an element's position to it's offset from the start of the (0-based) * ByteBuf. Size is inferred by adjacent elements. * The width of each element is ${type.width} byte(s) * The equivilent Java primitive is '${minor.javaType!type.javaType}' * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.

45. * * ${minor.class} implements a vector of fixed width values. Elements in the vector are accessed * by position, starting from the logical start of the vector. Values should be pushed onto the * vector sequentially, but may be randomly accessed. * The width of each element is ${type.width} byte(s) * The equivilent Java primitive is '${minor.javaType!type.javaType}' * * NB: this class is automatically generated from ValueVectorTypes.tdd using FreeMarker.
46. Build an optional float field definition
47. Create and set 3 sample strings
48. Put and set a few values
49. Build a required boolean field definition
50. test setting the same value twice
51. Ensure unallocated space returns 0
52. Build a required uint field definition
53. Build an optional varchar field definition
54. Check the sample strings
55. test toggling the values
56. Build an optional uint field definition
57. Ensure null values throw
58. Create a new value vector for 1024 integers
59. Ensure null value throws
60. public static WritableBatch get(ValueVector.Base[] vectors){
61. TODO: ensure the foreman handles the exception

git_commits:

1. **summary:** Create new generated value vectors utilizing fmpp. Includes:
   message: Create new generated value vectors utilizing fmpp. Includes: - First pass; integrate build system and some cursory implementations - starting to split common logic into base class - implement most of varlen value vector functionality, minor cleanup of tdd tags - added nullable derived class - Merge changes from JA, minor format cleanup. - minor fix and cleanup - added bit vector, removed widthInBits which also allowed removal of FixedBase ctor - apply TC's fix for resetAllocation() - added repeated value vectors - Hooked up templated ValueVectors to codebase. Removed old ValueVector classes. Cleanup. - fix repeated get() and add() - added some value vector tests. fixed bugs in VV and some call sites. generated TypeHelper from FMPP template. removed unused VV methods - made base immutable, some debugging - split mutable/immutable basic VV types. minor refactoring - fix several allocation bugs - fix various bugs, only JSONRecordReader test is failing - fix nullable bit value vector - make bit vectors use ints to represent the bit value - remove superfluous logging - fix value vector getters and setter - comments and cleanup - temp disable repeated map JSONReader test - formatting - whitespace cleanups

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

jira_issues_comments: