Item 278
git_comments:

1. * * Returns true if x1 is less than x2, when both values are treated as unsigned.
2. **comment:** * Compare 8 bytes at a time. Benchmarking shows comparing 8 bytes at a time is no slower than comparing * 4 bytes at a time even on 32-bit. On the other hand, it is substantially faster on 64-bit.
   label: code-design
3. * The offset to the first element in a byte array.
4. **comment:** It doesn't matter what we throw; it's swallowed in getBestComparer().
   label: code-design
5. The epilogue to cover the last (minLength % 8) elements.
6. * * Utility code to do optimized byte-array comparison. * This is borrowed from org.apache.hadoop.io.FastByteComparisons * which was borrowed and slightly modified from Guava's {@link UnsignedBytes} * class to be able to compare arrays that start at non-zero offsets. * * The only difference is that we sort a smaller length bytes as *larger* * than longer length bytes when all the bytes are the same.
7. yes, UnsafeComparer does implement Comparer<byte[]>
8. Short circuit equal case
9. Bring WritableComparator code local
10. Use binary search
11. used via reflection
12. * * Provides a lexicographical comparer implementation; either a Java implementation or a faster implementation based * on {@link Unsafe}. * <p> * Uses reflection to gracefully fall back to the Java implementation if {@code Unsafe} isn't available.
13. sanity check - this should never fail
14. * * Returns the Unsafe-using Comparer, or falls back to the pure-Java implementation if unable to do so.
15. ensure we really catch *everything*
16. * * Lexicographically compare two arrays. * * @param buffer1 * left operand * @param buffer2 * right operand * @param offset1 * Where to start comparing in the left buffer * @param offset2 * Where to start comparing in the right buffer * @param length1 * How much to compare from the left buffer * @param length2 * How much to compare from the right buffer * @return 0 if equal, < 0 if left is less than right, etc.
17. * * Lexicographically compare two byte arrays.
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. * 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.
20. If we have a schema where column A is DESC, reverse the sort order and nulls last since this is the order they actually are in.
21. Treat as binary if descending because we've got a separator byte at the end which is not part of the value.
22. Don't return if evaluated to null
23. **comment:** Special hack for PHOENIX-2067 to change the constant array to match the separators used for descending, variable length arrays. Note that there'd already be a coerce expression around the constant to convert it to the right type, so we shouldn't do that here.
    label: code-design

24. Constants are always build with rowKeyOptimizable as true, using the correct separators We only need to do this conversion if we have a table that has not yet been converted.
25. Add cell for ROW_KEY_ORDER_OPTIMIZABLE = true, as we know that new tables conform the correct row key. The exception is for a VIEW, which the client sends over depending on its base physical table.
26. **comment:** FIXME: we should allow the update, but just not propagate it to this view The one exception is PK changes which need to be propagated to diverged views as well
    label: code-design
27. Don't allow view PK to diverge from table PK as our upgrade code does not handle this.
28. Size for worst case - all new columns are PK column
29. Column to track tables that have been upgraded based on PHOENIX-2067
30. We're starting a rebuild of the index, so add our rowKeyOrderOptimizable cell so that the row keys get generated using the new row key format
31. We may be adding a DESC column, so if table is already able to be rowKeyOptimized, it should continue to be so.
32. We may be adding a DESC column, so if index is already able to be rowKeyOptimized, it should continue to be so.
33. **comment:** TODO: Switch this to Region#batchMutate when we want to support indexes on the
    label: requirement
34. Copy existing cell but with new row key
35. for local indexes (prepend region start key)
36. force to use correct separator byte
37. Special case for re-writing DESC ARRAY, as the actual byte value needs to change in this case
38. If Put, point delete old Put
39. **comment:** * * <code>optional bool rowKeyOrderOptimizable = 26;</code>
    label: code-design
40. optional bool rowKeyOrderOptimizable = 26;
41. **comment:** * * <code>optional bool rowKeyOrderOptimizable = 26;</code>
    label: code-design
42. Flip back here based on sort order, as the compiler flips this, but we want to display the original back to the user.
43. Don't remove trailing separator byte unless it's the one for ASC as otherwise we need it to ensure sort order is correct
44. **comment:** FIXME: calling version of coerceBytes that takes into account the separator used by LHS If the RHS does not have the same separator, it'll be coerced to use it. It's unclear if we should do the same for all classes derived from the base class. Coerce RHS to LHS type
    label: code-design
45. **comment:** FIXME: concatArrays will be fine if it's copying the separator bytes, including the terminating bytes.
    label: code-design
46. **comment:** TODO: remove when rowKeyOrderOptimizable hack no longer needed
    label: code-design
47. **comment:** Hack to serialize whether the index row key is optimizable
    label: code-design
48. Write separator byte if variable length unless it's the last field in the schema (but we still need to write it if it's DESC to ensure sort order is correct).
49. **comment:** Hack for PHOENIX-2067 to force raw scan over all KeyValues to fix their row keys
    label: code-design
50. We project *all* KeyValues across all column families as we make a pass over a physical table and we want to make sure we catch all KeyValues that may be dynamic or part of an updatable view.
51. Remove any filter Traverse (and subsequently clone) all KeyValues Pass over PTable so we can re-write rows according to the row key schema
52. * Same as regular comparator, but if all the bytes match and the length is * different, returns the longer length as bigger.
53. * * Returns true if this connection is being used to upgrade the * data due to PHOENIX-2067 and false otherwise. * @return
54. **comment:** Manually transfer the ROW_KEY_ORDER_OPTIMIZABLE_BYTES from parent as we don't want to add this hacky flag to the schema (see PHOENIX-2067).
    label: code-design

55. Set so that we get the table below with the potentially modified rowKeyOrderOptimizable flag set
56. * * Determines whether or not we may optimize out an ORDER BY or do a GROUP BY * in-place when the optimizer tells us it's possible. This is due to PHOENIX-2067 * and only applicable for tables using DESC primary key column(s) which have * not been upgraded. * @return true if optimizations row key order optimizations are possible
57. Need trailing byte for DESC columns
58. after hasDescVarLengthColumns is calculated
59. Last field has no terminator unless it's descending sort order
60. Separator always zero byte if zero length
61. First byte
62. Only applicable for RowKeySchema (and only due to PHOENIX-2067), but added here as this is where serialization is done (and we need to maintain the same serialization shape for b/w compat).
63. assumes stats table columns not DESC assumes stats table columns not DESC
64. Separator for new value Double byte separator
65. Increase of length required to store nulls Length increase incremented by one when there were no nulls at the beginning of array and when there are nulls at the end of array 1 as we need to allocate a byte for separator byte in this case.
66. Creates a byte array to store the concatenated array
67. Write separator explicitly, as it may not be 0
68. offsets for nulls in the middle
69. Writes nulls in the middle of the array. Copies the elements from array 2 beginning from the first non null element.
70. In both case the elements and the value array will be the same but the Array 1 is actually smaller because it has more nulls. Now we should have mechanism to show that we treat arrays with more nulls as lesser. Hence in the above case as For Array 2, by inverting we would get a -ve value. On comparison Array 2 > Array 1. Now for cases where the number of nulls is greater than 255, we would write an those many (byte)1, it is bigger than 255. This would ensure that we don't compare with triple zero which is used as an end byte
71. **comment:** FIXME: remove this duplicate code
    label: code-design
72. offsets for the elements from array 1. Simply copied.
73. Explicitly set separator byte since for DESC it won't be 0.
74. offsets for the elements from the first non null element from array 2
75. Coerce to new max length when only max lengths differ
76. padding
77. Copies all the elements from array 1 to new array
78. serialize nulls at the beginning
79. Subtract one and invert so that more remaining nulls becomes smaller than less remaining nulls and min byte value is always greater than 1, the repeating value The reason we invert is that an array with less null elements has a non
80. Here the int for noofelements, byte for the version, int for the offsetarray position and 2 bytes for the end seperator
81. writes the new offset and changes the previous offsets
82. * * creates array bytes * @param rowKeyOrderOptimizable TODO
83. handle the case where prepended element is null counts the number of nulls which are already at the beginning of the array gets the offset of the first element after nulls at the beginning Calculates the increase in length due to prepending the null There is a length increase only when nRemainingNulls == 1 nRemainingNulls == 1 and nMultiplesOver255 == 0 means there were no nulls at the beginning previously. At that case we need to increase the length by two bytes, one for separator byte and one for null count. ex: initial array - 65 0 66 0 0 0 after prepending null - 0 1(inverted) 65 0 66 0 0 0 nRemainingNulls == 1 and nMultiplesOver255 != 0 means there were null at the beginning previously. In this case due to prepending nMultiplesOver255 is increased by 1. We need to increase the length by one byte to store increased that. ex: initial array - 0 1 65 0 66 0 0 0 after prepending null - 0 1 1(inverted) 65 0 66 0 0 0 nRemainingNulls == 0 case. ex: initial array - 0 254(inverted) 65 0 66 0 0 0 after prepending null - 0 1 65 0 66 0 0 0 nRemainingNulls > 1 case. ex: initial array - 0 45(inverted) 65 0 66 0 0 0 after prepending null - 0 46(inverted) 65 0 66 0 0 0
84. checks whether offset array consists of shorts or integers count nulls at the end of array 1 count nulls at the beginning of the array 2
85. checks whether offset array consists of shorts or integers

86. Assume an offset array that fit into Short.MAX_VALUE. Also not considering nulls that could be > 255 In both of these cases, finally an array copy would happen
87. Writing offset arrays offsets for the elements from array 1. Simply copied.
88. a, b, null, null, c, null would be Follow the above example to understand how this works
89. A match for IS NULL or IS NOT NULL should not have a DESC_SEPARATOR_BYTE as nulls sort first
90. * * Return the separator byte to use based on: * @param rowKeyOrderOptimizable whether or not the table may optimize descending row keys. If the * table has no descending row keys, this will be true. Also, if the table has been upgraded (using * a new -u option for psql.py), then it'll be true * @param isNullValue whether or not the value is null. We use a null byte still if the value is null * regardless of sort order since nulls will always sort first this way. * @param sortOrder whether the value sorts ascending or descending. * @return the byte to use as the separator
91. Write trailing separator if last expression was variable length and descending
92. Must be a table Must be global Must be the physical table
93. Find views to mark as upgraded
94. Open tenant-specific connection when we find a new one
95. Replace trailing , with) to end IN expression
96. Upgrade view indexes
97. Run query
98. Find tables/views for index
99. Add any tables (which will all be physical tables) which have not already been upgraded.
100. * * Upgrade tables and their indexes due to a bug causing descending row keys to have a row key that * prevents them from being sorted correctly (PHOENIX-2067).
101. Find the header rows for tables that have not been upgraded already. We don't care about views, as the row key cannot be different than the table. We need this query to find physical tables which won't have a link row.
102. First query finds column rows of tables that need to be upgraded. We cannot tell if the column is from a table, view, or index however.
103. Mark the table and views as upgraded now
104. * * Upgrade shared indexes by querying for all that are associated with our * physical table. * @return true if any upgrades were performed and false otherwise.
105. Find physical table name from views, splitting on '.' to get schema name and table name
106. Upgrade global indexes
107. * * Identify the tables that need to be upgraded due to PHOENIX-2067

git_commits:

1. **summary:** PHOENIX-2067 Sort order incorrect for variable length DESC columns
   message: PHOENIX-2067 Sort order incorrect for variable length DESC columns Conflicts: phoenix-core/src/main/java/org/apache/phoenix/coprocessor/MetaDataEndpointImpl.java phoenix-core/src/main/java/org/apache/phoenix/coprocessor/UngroupedAggregateRegionObserver.java phoenix-core/src/main/java/org/apache/phoenix/execute/BaseQueryPlan.java phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixConnection.java Conflicts: phoenix-core/src/main/java/org/apache/phoenix/coprocessor/MetaDataEndpointImpl.java phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixConnection.java

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Sort order incorrect for variable length DESC columns

description: Steps to reproduce: 1. Create a table: CREATE TABLE mytable (id BIGINT not null PRIMARY KEY, timestamp BIGINT, log_message varchar) IMMUTABLE_ROWS=true, SALT_BUCKETS=16; 2. Create two indexes: CREATE INDEX mytable_index_search ON mytable(timestamp,id) INCLUDE (log_message) SALT_BUCKETS=16; CREATE INDEX mytable_index_search_desc ON mytable(timestamp DESC,id DESC) INCLUDE (log_message) SALT_BUCKETS=16; 3. Upsert values: UPSERT INTO mytable VALUES(1, 1434983826018, 'message1'); UPSERT INTO mytable VALUES(2, 1434983826100, 'message2'); UPSERT INTO mytable VALUES(3, 1434983826101, 'message3'); UPSERT INTO mytable VALUES(4, 1434983826202, 'message4'); 4. Sort DESC by timestamp: select timestamp,id,log_message from mytable ORDER BY timestamp DESC; Failure: data is sorted incorrectly. In case when we have two longs which are different only by last two digits (e.g. 1434983826155, 1434983826100) and one of the long ends with '00' we receive incorrect order. Sorting result: 1434983826202 1434983826100 1434983826101 1434983826018

jira_issues_comments:

1. [~giacomotaylor], please take a look at this issue.
2. This bug potentially affects any PK column of variable length that is declared as DESC sort order. The particular case is when one value is a subpart of another value. Simplest case would be this: {code} CREATE TABLE t (k VARCHAR DESC PRIMARY KEY); UPSERT INTO t VALUES ('a'); UPSERT INTO t VALUES('ab'); SELECT * FROM t ORDER BY k; {code} The 'ab' row should appear before the 'a' row. This is due to there being no terminator at the end of the row key as well as because we're not inverting the null/zero terminator/separator byte between parts of the row key. The fix is to: - use a 255 byte separator byte instead of a 0 byte separator between row key parts - include a 255 byte terminator at the end of the row key - include a 255 byte for any null value at the end of the row key (without this, row keys with null values in the middle of the row key might sort before a row key with null values at the end of the row key) - disallow a DESC pk column to be added to the PK in an ALTER TABLE <table> ADD <column> as it would require updating existing data. We need a script that users can run to fix their existing data (or at least identify that there's an issue).
3. WIP patch. Unit tests pass along with new ones that demonstrate the issue.
4. Parking partial patch
5. More WIP. With just a few test failures, but no upgrade or conditional optimization for existing data. This is with nulls last when DESC, but there's a problem with this - we'd need to include trailing nulls until the last DESC row key column and you wouldn't be able to add a new DESC row key column without mucking with the data (which is a showstopper). I'm going to instead use a null separator with DESC for null values and otherwise a 0xFF. That way, nulls will sort first for ASC and DESC, but DESC sort order will work for all values.
6. Another WIP patch. All unit tests passing except one.
7. All unit tests passing.
8. Includes upgrade ability and warning message
9. With upgrade
10. Fix OrderedResultIterator for DESC variable length data
11. Working upgrade, but still needs some refinement.
12. Refined upgrade code
13. Working and tested patch
14. [~samarthjain] - please review. Here's an overview: - No change in behavior for existing tables. Queries that have an ORDER BY for a variable length, descending row key will now sort correctly, but at the expense of forcing an ORDER BY (since they aren't sorted correctly in their natural order, we can't optimize out the ORDER BY). - New tables (or indexes) will use the correct separator for DESC variable length row keys, so they won't be hit with the ORDER BY cost. See SchemaUtil.getSeparatorByte() for an overview of the logic to determine the separator byte. - A new utility (psql.py -u option) is available to 1) display the physical tables affected by this bug, and 2) to optionally re-write them so that they sort correctly.
15. As far as testing the upgrade, I did a bunch of manual testing: - Created table with descending, variable length row key, table without but with an index that has one, table without but with a view that has a local and shared index, multi-tenant tables against the same. - Verified correct physical tables identified that need upgrading. - Verified upgrade worked correctly for all of above
16. Fixed test failure due to this change for adding column to base table. Also disallowing VARBINARY DESC which can't really work correctly since we can't control what bytes are used and thus cannot guarantee the sort order is correct. A workaround for users would be to upsert using INVERT which is the

same as what would occur today. Shorter but equal byte values would sort above longer equal byte values, though.
17. Addendum patch for handling DESC ARRAY. Still needs a few more tweaks, but parking here for now.
18. **body:** [~Dumindux] and [~ram_krish] - would you guys mind reviewing this patch? This ensures that descending, variable length arrays sort correctly. The change is the use a 255 byte as the separator for non null values (including the terminators). See the couple of new tests in ArrayIT. Much of the type changes are just formatting and moving a couple of duplicated methods where they belong at the base type. The other changes are to ensure we keep the same separator - for example if a table has not been upgraded, we need to keep using the 0 byte separator. That's where most of the complication comes in. Much appreciated. [~Dumindux] - if you have time perhaps you can write a couple of lower level unit tests to confirm that my isRowKeyOrderOptimized works in all situations and that array_cat, prepend, and append work in that they maintain the same separator byte that the array is already using.
    label: code-design
19. Went through the patch, on a high level ->We will rewrite the array bytes to use the new seperator byte if it is of type DESC. -> for the array_cat - if the existing array to which we will append a new array is of the old type we will coerce it to use the new sepeartor and the new array that we add should automatically use the new seperator (if the overall sort order is DESC) right? ->same with the prepend and append. But one question regarding the other operations where we try to use the SEPERATOR_BYTE to find if we have reached the end of the array - in all such places we should not blindly check with SEPERTOR_BYTE right - instead try to decide it based on the order of the current byte[]?
20. Thanks for the review, [~ram_krish]. bq. should not blindly check with SEPERTOR_BYTE right - instead try to decide it based on the order of the current byte[]? Luckily, for arrays we navigate using the offsets in the header, so this part didn't need to change. The separator bytes are purely to make sure variable length arrays sort correctly relative to each other. I'll check in shortly unless you have other feedback?
21. SUCCESS: Integrated in Phoenix-master #834 (See [https://builds.apache.org/job/Phoenix-master/834/]) PHOENIX-2067 Sort order incorrect for variable length DESC columns (jtaylor: rev 2620a80c1e35c0d214f06a1b16e99da5415a1a2c) * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedTinyintArray.java * phoenix-core/src/main/java/org/apache/phoenix/exception/SQLExceptionCode.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PBooleanArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/DelegateTable.java * phoenix-core/src/main/java/org/apache/phoenix/util/ScanUtil.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedIntArray.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/MetaDataEndpointImpl.java * phoenix-core/src/main/java/org/apache/phoenix/compile/OrderPreservingTracker.java * phoenix-core/src/main/java/org/apache/phoenix/index/IndexMaintainer.java * phoenix-core/src/main/java/org/apache/phoenix/expression/util/regex/JONIPattern.java * phoenix-core/src/main/java/org/apache/phoenix/schema/stats/StatisticsUtil.java * phoenix-core/src/main/java/org/apache/phoenix/iterate/OrderedResultIterator.java * phoenix-core/src/test/java/org/apache/phoenix/compile/QueryCompilerTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PLongArray.java * phoenix-core/src/main/java/org/apache/phoenix/util/SchemaUtil.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PTimeArray.java * phoenix-core/src/main/java/org/apache/phoenix/compile/TupleProjectionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/expression/RowValueConstructorExpression.java * phoenix-core/src/main/java/org/apache/phoenix/iterate/BaseResultIterators.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/generated/PTableProtos.java * phoenix-core/src/main/java/org/apache/phoenix/util/MetaDataUtil.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PFloatArray.java * phoenix-core/src/main/java/org/apache/phoenix/util/PhoenixRuntime.java * phoenix-core/src/main/java/org/apache/phoenix/compile/ScanRanges.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/LpadFunctionIT.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/ReverseScanIT.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/SortOrderIT.java * phoenix-core/src/main/java/org/apache/phoenix/compile/JoinCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/RowKeyValueAccessor.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PArrayDataType.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/RowValueConstructorIT.java * phoenix-core/src/main/java/org/apache/phoenix/util/TupleUtil.java * phoenix-

core/src/main/java/org/apache/phoenix/util/UpgradeUtil.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDoubleArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTable.java * phoenix-core/src/main/java/org/apache/phoenix/util/ByteUtil.java * phoenix-core/src/main/java/org/apache/phoenix/compile/FromCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/filter/SkipScanFilter.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionQueryServicesImpl.java * phoenix-core/src/main/java/org/apache/phoenix/expression/function/LpadFunction.java * phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixConnection.java * phoenix-core/src/main/java/org/apache/phoenix/schema/RowKeySchema.java * phoenix-core/src/main/java/org/apache/phoenix/expression/ArrayConstructorExpression.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedSmallintArray.java * phoenix-core/src/main/java/org/apache/phoenix/compile/UpsertCompiler.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/ArrayIT.java * phoenix-protocol/src/main/PTable.proto * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedDoubleArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PVarbinaryArray.java * phoenix-core/src/main/java/org/apache/phoenix/expression/function/ArrayModifierFunction.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedDateArray.java * phoenix-core/src/main/java/org/apache/phoenix/compile/UnionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/MetaDataClient.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedTimestampArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PTimestampArray.java * phoenix-core/src/main/java/org/apache/phoenix/expression/function/ArrayConcatFunction.java * phoenix-core/src/main/java/org/apache/phoenix/query/KeyRange.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDecimalArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PIntegerArray.java * phoenix-core/src/main/java/org/apache/phoenix/compile/WhereOptimizer.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/IsNullIT.java * phoenix-core/src/main/java/org/apache/phoenix/execute/DescVarLengthFastByteComparisons.java * phoenix-core/src/test/java/org/apache/phoenix/query/OrderByTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PCharArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDataType.java * phoenix-core/src/main/java/org/apache/phoenix/compile/OrderByCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedTimeArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PVarcharArray.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/BaseScannerRegionObserver.java * dev/eclipse_prefs_phoenix.epf * phoenix-core/src/main/java/org/apache/phoenix/schema/ValueSchema.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PBinaryArray.java * phoenix-core/src/it/java/org/apache/phoenix/end2end/SortOrderFIT.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedLongArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PDateArray.java * phoenix-core/src/main/java/org/apache/phoenix/execute/BaseQueryPlan.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryConstants.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTableImpl.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PUnsignedFloatArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PTinyintArray.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PSmallintArray.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/UngroupedAggregateRegionObserver.java * phoenix-core/src/main/java/org/apache/phoenix/expression/OrderByExpression.java * phoenix-core/src/test/java/org/apache/phoenix/schema/types/PDataTypeTest.java

22. SUCCESS: Integrated in Phoenix-master #835 (See [https://builds.apache.org/job/Phoenix-master/835/]) PHOENIX-2067 Sort order incorrect for variable length DESC columns (jtaylor: rev 4b99c632c5e40251451e69fbe6d108f51e549e9e) * phoenix-core/src/main/java/org/apache/phoenix/util/UpgradeUtil.java
23. [~jamestaylor] sorry for the late response. Here are some tests for the built-ins. I have found a small issue. As we are converting double separator bytes at the end also to new separator bytes for DESC arrays, there was a problem in counting trailing nulls in first array(in ARRAY_CAT). I changed the code to consider the new separator byte also.

24. As [~ram_krish] suggested I looked at the other parts of the code also. The trailing null issue was in two other places. In array deserialization and positionAtArrayElement method. This patch has two tests which demonstrate the cases and the fix.
25. +1. Please commit to 4.x and master branch, [~ram_krish] (and you're welcome to review as well too, of course).
26. +1 . Nice tests. Thanks for covering up all the cases [~Dumindux]. [~giacomotaylor] I was trying to suggest this point only in my review.
27. SUCCESS: Integrated in Phoenix-master #841 (See [https://builds.apache.org/job/Phoenix-master/841/]) PHOENIX-2067 Sort order incorrect for variable length DESC columns - ARRAY (ramkrishna: rev 33d60506c5f2d4408a1df79f278d7a45d3401a27) * phoenix-core/src/test/java/org/apache/phoenix/expression/ArrayAppendFunctionTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/types/PArrayDataType.java * phoenix-core/src/test/java/org/apache/phoenix/schema/types/PDataTypeForArraysTest.java * phoenix-core/src/test/java/org/apache/phoenix/expression/ArrayPrependFunctionTest.java * phoenix-core/src/test/java/org/apache/phoenix/expression/ArrayConcatFunctionTest.java
28. [~jamestaylor] Here is the backport patch for 4.4.1. Most of the patch applied manually. I ran all the tests and they have passed. Can you please review it. Will commit if it's ok. Thanks.
29. Patch looks good, [~rajeshbabu]. Thanks for back porting. One thing that needs to be manually tested is converting old tables to the correct row key using "psql.py -u localhost". You can wait to do this until after back porting PHOENIX-2171 so you only have to do it once. You'd want to create various tables with DESC row keys in pre 4.4.1 and then run the script to ensure that the conversion works correctly and you get the expected query plan post conversion.
30. Bulk close of all issues that has been resolved in a released version.
31. I think this patch broke PK constraints for NOT NULL, non-fixed-width VARCHAR columns. {noformat} // If some non null pk values aren't set, then throw if (i < nColumns) { PColumn column = columns.get(i); if (column.getDataType().isFixedWidth() || !column.isNullable()) { throw new ConstraintViolationException(name.getString() + "." + column.getName().getString() + " may not be null"); } } {noformat} The constraint we're managing is that of non-null for PK columns. For VARCHAR, fixed-width meets this criteria because there's no available representation for NULL. I assume variable-length columns can represent null (though I don't know the encoding details off the top of my head), so we must look for the `NOT NULL` constraint added in schema. I think the if condition should be {noformat} if (!column.getDataType.isFixedWidth() || column.isNullable()) { throw... } {noformat} Also, can this be folded into a single {{column.isNullable()}} -- shouldn't that method check the datatype on the caller's behalf? Or is there a scenario where we want to know what additional constraints the schema defined vs. what the datatype offers?
32. Nope, this patch didn't change the logic on that line: https://github.com/apache/phoenix/commit/2620a80c1e35c0d214f06a1b16e99da5415a1a2c#diff-d87ee86bba434603ba73b6a85a139529