Item 183
git_comments:

1. If the ranges here do not qualify all the keys then those keys are unbound
2. * This is already considered inside of ScanUtil.setKey we may want to refactor to pull these out. * range/single boundary bound increment * range inclusive lower no * range inclusive upper yes, at the end if occurs at any slots. * range exclusive lower yes * range exclusive upper no * single inclusive lower no * single inclusive upper yes, at the end if it is the last slots.
3. We have a composite key need the row key from the combination make a copy of ranges as we may add items to fully qualify our rowkey
4. update position This loops through all settings of each of the primary keys by counting from the trailing edge based on the number of settings of that key.
5. Don't send a null range, send an empty range.
6. Product to bound our counting loop for safety
7. * * Produces the list of KeyRanges representing the fully qualified row key by calling into ScanUtil.setKey * repeatedly for every combination of KeyRanges in the ranges field. The bounds will be set according to the * properties of the setKey method. * * @return list of KeyRanges representing the fully qualified rowkey, coalesced
8. If scanRanges.ranges has no information then should be in the scanRanges.scanRange
9. Have to construct the intersection
10. Note ScanUtil.setKey internally handles the upper/lower exclusive from a Scan point of view. It would be good to break it out in the future for rowkey construction vs ScanKey construction To handle differences between hbase 2 and hbase 1 scan boundaries
11. unbound
12. Counting Loop
13. If its scanRanges are everything or nothing, then we short circuit and leave as schema is not filled in
14. Region is lowerInclusive true by definition Region is upperInclusive false by definition Unless it returns HConstants.EMPTY_BYTE_ARRAY which indicates it is unbound
15. Since we don't have the empty key value in MAPPED tables,
16. If gp after stop key, but still in last region, track min ts as fallback
17. Load the region information
18. width.
19. * * @return List of KeyRanges to represent each region * @throws SQLException
20. Map each HRegionLocation to a KeyRange - no Java 8
21. In case of a keyOnlyFilter, we only need to project the
22. Does not handle some edge conditions like empty string
23. * range/single boundary bound increment * range inclusive lower no * range inclusive upper yes, at the end if occurs at any slots. * range exclusive lower yes * range exclusive upper no * single inclusive lower no * single inclusive upper yes, at the end if it is the last slots.

git_commits:

1. **summary:** Merge pull request #463 from dbwong/phoenix-stats
   message: Merge pull request #463 from dbwong/phoenix-stats Phoenix stats Initial Commit

github_issues:

github_issues_comments:

github_pulls:

1. **title:** Phoenix stats Initial Commit
   body: Add the utility functions from my local work that @BinShi-SecularBird needed access to.

github_pulls_comments:

1. Not finish the code review yet. Could you handle these feedbacks first?
2. Review isn't finished yet. Could you handle these feedback first? I still need some time to check whether we have equivalent refactor for local index.

3. @twdsilva Stats related PR. This is going into a Stats integration branch that @karanmehta93 made for us.
4. @dbwong Can you please rebase and squash the commits? I will get this committed.

github_pulls_reviews:

1. Do you want to keep this blank line at line 698? I don't see we have such blank line at the beginning of function body in other functions.
2. Could you add comments (function spec - The algorithm? What is the output? Is the output KeyRange coalesced? Is the output guaranteed to be in "lower bound inclusive and upper bound exclusive" format?) for this function? Need to add test cases for all the cases mentioned in the func spec.
3. Could you add comma after nothing and change As to as?
4. Could you use this.isEverthy() here? It is an existing helper function and seems much cheaper than equals.
5. Could you use this.isDegenerate()? It is an existing helper function seems much cheaper.
6. Delete the blank line at 715?
7. Delete the extra space in "be in"? Could you please generally improve the comments in this change as best as you can?
8. Could you add "else {" to a new line which is also in your change and seems to be standard coding style? Also could you move the comment "//We have a composite key need the row key from the combination" to a new line?
9. Improve this comment?
10. Could you change "if(" to "if ("? The same to other places?
11. Could you change "for(" to "for ("? The same to other places?
12. I prefer to insert a blank line above return.
13. Could you change "for(" to "for ("? Other places too.
14. Should be { byte[] lowerRange = isFirstRegion ? KeyRange.UNBOUND : regionLocations.get(0).getLowerRange; // Even they are both new byte[0], they are different. boolean lowerInclusive = ! isFirstRegion; byte[] upperRange = isLastRegion ? KeyRange.UNBOUND : regionLocations.get(regionLocations.size()-1).getUpperRange; // Even they are both new byte[0], they are different. boolean upperExclusive = false; } I just use the code showed above, as a quick sample code, to explain the problem in your original change.
15. I think we should use Lists.newArrayListWithCapacity(regionLocations.size()), because we know the exact size here. newArrayListWithExpectedSize actually use "given expectsize + 10% more" which we don't need.
16. The original comment says "// TODO: when implementing PHOENIX-4585, we should change this to an assert". Shall we do it now or open a Jira to track it?
17. It is ugly to have a new line here. What do you think?
18. I prefer the following coding style which seems to be used by phoenix code base generally. <type >scanRanges = computePrefixScanRanges(dataPlan.getContext().getScanRanges(), computeColumnsInCommon()); Did you follow the following instructions on https://phoenix.apache.org/develop.html to set phoenix coding style. Get Settings and Preferences Correct Import Settings from eclipse profile File -> Settings -> Editor -> Code Style -> Java Set From… -> Import… -> Eclipse XML Profile -> {repository}/dev/PhoenixCodeTemplate.xml
19. (Coding Style) Should it be "else {"?
20. Ok
21. Ok
22. Ok
23. I don't understand this is standard in this file see lines 105,262 etc. And standard in the formatter. I believe this is the correct line for this comment because of the above.
24. Removed
25. Ok
26. I agree that we can specifically call out the boundaries I dislike this specific code pasted. I have added the refactor to specify unbound directly rather than implicitly.
27. So I looked into this few weeks ago. That JIRA, 4585, has been submitted but when I changed this to an assert I still found cases where we the dataplan was not filled in. So I have left the original code and did not spend time investigating.
28. Ok

29. I agree but then the line is over 100 characters which previously on phoenix commits have complained. I have reverted for now we will see later if there are complaints from the format bot.
30. I thought this was already on but appeared this was a new workspace. I have applied in general.
31. Done
32. Done
33. I have added function level comments. Some basic testing was already included in the unit tests. As this code snippet does nothing, I can not include functional tests.
34. Ok
35. Not sure if this is referring to the same line.
36. Shall we just use List<List<KeyRange>> ranges = new ArrayList<>(this.getRanges())?
37. Shall we just use Lists.newArrayListWithCapacity(rangesPermutations) since we know the count of expandedRanges is bounded by rangesPermutations? newArrayListWithExpectedSize actually allocate 10% more capacity with the given expectedSize.
38. change name to positions?
39. Do you mean count++?
40. From line 752 to 777, I prefer the following code which seems more compact. Thought? This might be just personal preference. // comment int result = ScanUtil.setKey(schema, ranges, slotSpans, position, KeyRange.Bound.LOWER, keyBuffer, 0, 0, slotSpans.length); byte[] lowerBound = result < 0 ? KeyRange.UNBOUND : Arrays.copyOf(keyBuffer, result); // comment result = ScanUtil.setKey(schema, ranges, slotSpans, position, KeyRange.Bound.UPPER, keyBuffer, 0, 0, slotSpans.length); byte[] upperBound = result < 0 ? KeyRange.UNBOUND : Arrays.copyOf(keyBuffer, result);
41. For the above logic, I really suspect (didn't look at code in details) it should be as follows if we want the returned range list is sorted in ascending order. Could you double check? I might be wrong. (Shall we use Bitmap operations to replace the for loop?) int count = 0; while (count < rangesPermutations) { ... for (; i < position.length; i++) { int x = position.length - 1 - i; if (position[x] < ranges.get(x).size - 1) { position[x]++; break; } else { position[x] = 0; } } if (i == position.length) { break; } count++; }
42. here the slotSpans[0...this.getSlotSpans().length-1] is wiped off, right? Should it be: slotSpans = new int[schema.getMaxFields()]; System.arraycopy(this.getSlotSpans(), 0, slotSpans, 0, this.getSlotSpans().length);
43. remove the blank line here?
44. How about change name to countOfRangesPermutations? rangesPermutations sounds like the permutations of ranges instead of a counter.
45. Ok
46. Ok
47. Ok
48. Good catch!, refactoring error.
49. I personally dislike using ternary operator outside of final initializations.
50. Good catch this would have been an issue for non-fully qualified RVCs.
51. Ok
52. Sure changed to rangePermutationCount
53. You are correct that doing this from the trailing edge will provide the order constant easier as opposed to resorting. I'm not sure what you intend by bitmap operations as we are not a strict bitmap. Can you elaborate?
54. From line 605 to line 616, if we prepend each query key ranges with every region start/end key, how do we prune the query key ranges later?
55. Can we have test case for this func spec, i.e., for all these cases in comments, every returned key range's upper boundary is exclusive?
56. Will do as part 2 as discussed.
57. Sure
58. `regionInfo.getEndKey()` or `endKey`?
59. Looks like it should not really matter since `KeyRange.UNBOUND` and `HConstants.EMPTY_BYTE_ARRAY` are the same. Should we just refactor these static constants? Some of this code exists from 2014. You can also remove line 56 if you want.
60. I might be missing some context here but I see this piece of code in this file which can re-used and handles the required corner cases. ``` static final Function<HRegionLocation, KeyRange> TO_KEY_RANGE = new Function<HRegionLocation, KeyRange>() { @Override public KeyRange apply(HRegionLocation region) { return KeyRange.getKeyRange(region.getRegionInfo().getStartKey(), region.getRegionInfo().getEndKey()); } }; ```
61. Can you add some comments here as to how this method will help?

62. nit: use `ranges` directly.
63. @dbwong, is the upper bound of each generated key range is exclusive? If it isn't, can we call nextKey for the upper bound when it is inclusive?
64. Do we really want to generate everything and store in memory for the client? How about using iterator approach instead?
65. Can you elaborate a bit here? What is the difference between `ScanRanges` and `KeyRanges`?
66. Also do we also want to refactor the `SkipScanFilter` class too since it relies on the same logic?
67. Can you explain the usage of this loop?
68. Can you add tests for `EVERYTHING_RANGE` and `EMPTY_RANGE` as well?
69. Done.
70. Added additional comments but this is a counting loop from the trailing edge to the leading edge for all possible settings of the Keys based on the ranges. That is for PK (key1,key2) and Where clause key1 in {a,b,c} and key2 in {x,y,z}, then we need to generate keys ax,ay,az,bx,by,bz,cx,cy,cz. See the setKey method for more details.
71. This was removed as part of the handling of Bin's request for exclusive upper. With this there is no reason to handle pointGets differently.
72. Sure.
73. Changed
74. Good catch actually. endKey is correct a typo.
75. Added comments and removed the TO_KEY_RANGE dead code. This was mostly used by the next set of code to intersect the query ranges with the region ranges to build per region ranges for scan consideration.
76. I think using in memory is fine for a first approach. I discussed this briefly with @BinShi-SecularBird and we felt that even large number of ranges should be okay for a first pass. Some estimates with a 100 byte key, and 100 keys, with 100 settings, is ~1 mb. Is this a current concern for the memory footprint? Second with an iterator approach the next steps of intersecting with region boundaries and with guidepost tree structure may be less efficient so we are trading memory for execution time.
77. Can you be more specific on what you think should be refactored in SkipScanFilter?
78. nit:getRangeKeyExclusiveLowerIncrementedUpperNotIncremented

jira_issues:

jira_issues_comments: