Item 275
git_comments:

1. * 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.
2. Nothing to do as we've already set the position to an empty byte array
3. Impossible
4. Set columnValue at position to any non null value. We always strip the last byte so that we can recognize null as a value with a single byte. Will be overridden during ViewWhereExpressionVisitor
5. Used to track column references and their
6. Set the columnValue at the position of the column to the constant with which it is being compared. We always strip the last byte so that we can recognize null as a value with a single byte.
7. Disallow explicit reference to salting column, tenant ID column, and index ID column
8. Rewrite view constants in updatable views as literals, as they won't be in the schema for an index on the view.
9. Ignore last byte, as it's only there so we can have a way to differentiate null from the absence of a value.
10. In SELECT *, don't include tenant column or index ID column for tenant connection
11. if (iterator.hasNext()) iterator.next();
12. Add unique index ID for shared indexes on views. This ensures that different indexes don't interleave.
13. Add tenant data isolation for tenant-specific tables if (iterator.hasNext()) iterator.next();
14. Disallow deletion of column referenced in WHERE clause of view
15. **comment:** TODO can we determine this won't return anything?
    label: requirement
16. Encode indexedColumns.size() and whether or not there's a viewIndexId
17. Encode nIndexSaltBuckets and isMultiTenant together
18. Fixed length
19. Account for potential view constants which are always bound plan2 is index plan. Ignore the viewIndexId if present
20. Check index state of now potentially updated index table to make sure it's active
21. Query can't possibly return anything so just return this plan.
22. Checking number of columns handles the wildcard cases correctly, as in that case the index must contain all columns from the data table to be able to be used.
23. plan1 is index plan. Ignore the viewIndexId if present
24. This will make the value null if the value passed through represents null for the given type. For example, an empty string is treated as a null.
25. **comment:** Index on view Physical index table created up front for multi tenant TODO: if viewIndexId is Short.MIN_VALUE, then we don't need to attempt to create it
    label: code-design
26. Don't attempt to make any metadata changes for a VIEW
27. Table in cache is newer than client timestamp which shouldn't be the case
28. Ignore, as we may never have created a view index table
29. Create view index table up front for multi tenant tables
30. Get percentage to use from table props first and then fallback to config
31. Success If we're changing MULTI_TENANT to true or false, create or drop the view index table
32. Only use splits if table is salted, otherwise it may not be applicable
33. For views this will ensure that metadata already exists
34. Link metadata (only set on rows linking table to index or view)
35. % of data table max file size for index table
36. Next add index ID column
37. **comment:** Create at parent timestamp as we know that will be earlier than now and earlier than any SCN if one is set. TODO: If sequence already exists, then we don't need to ensure metadata is already created. Set physicalName to null in this case?

label: code-design
38. Don't add columns with constant values from updatable views, as we don't need these in the index
39. Can't set any of these on views or shared indexes on views
40. Tenant ID must be VARCHAR or CHAR and be NOT NULL NOT NULL is a requirement, since otherwise the table key would conflict potentially with the global table definition.
41. Can't set MULTI_TENANT or DEFAULT_COLUMN_FAMILY_NAME on an index
42. For client-side cache, we need to update the column
43. **comment:** Index on view TODO: Can we support a multi-tenant index directly on a multi-tenant table instead of only a view? We don't have anywhere to put the link from the table to the index, though.
    label: code-design
44. We need this in the props so that the correct column family is created
45. **comment:** TODO: consider removing this, as the DROP INDEX done for each DROP VIEW command would have deleted all the rows already
    label: code-design
46. Add tenant ID column as first column in index
47. **comment:** TODO: we need to drop the index data when a view is dropped
    label: code-design
48. Don't add link for mapped view, as it just points back to itself and causes the drop to fail because it looks like there's always a view associated with it.
49. Set physical name of view index table
50. Delete rows in view index if we haven't dropped it already We only need to do this if the multiTenant transitioned to false
51. **comment:** If we're not dropping metadata, then make sure no rows are left in our view index physical table. TODO: remove this, as the DROP INDEX commands run when the DROP VIEW commands are run would remove all rows already.
    label: code-design
52. **comment:** TODO: better to return error code
    label: code-design
53. Ignore view constants for updatable views as we don't need these in the index
54. Disallow setting these props for now on an add column, as that would a bit of a strange thing to do. We can revisit later if need be. We'd need to do the error checking we do in the else if we want to support this.
55. * We need an empty byte array to mean null, since * we have no other representation in the row key * for null.
56. Create global sequence of the form: <prefixed base table name><tenant id> rather than tenant-specific sequence, as it makes it much easier to cleanup when the physical table is dropped, as we can delete all global sequences leading with <prefix> + physical name.

git_commits:

1. **summary:** PHOENIX-21: Support indexes on multi-tenant views. Still needs more testing
   message: PHOENIX-21: Support indexes on multi-tenant views. Still needs more testing
   label: test

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Support indexes on VIEWs including tenant-specific views
   description: Current plan is to create a sibling index table based on the base multi-tenant table, suffixed with _INDEX when the base table is created. If your interested in this feature, best to review the following documents first: http://phoenix.incubator.apache.org/views.html http://phoenix.incubator.apache.org/multi-tenancy.html In addition, we need a multi-tenant sequence,

again based on the multi-tenant table name suffixed with _SEQ. The base columns in this table would be: the first column from the base table (i.e. the tenant ID column) followed by index id SMALLINT. This index ID is required to ensure that data from different indexes on the same multi-tenant table don't intermix with each other (as the rest of the PK is specific to the index being added). The index ID would be based on the next value in the _SEQ (a multi-tenant sequence). For simplicity, make it two bytes, starting with Short.MIN_VALUE. We'll never reuse an ID, so this gives us 64K create/drop indexes per multi-tenant base table. The rest of the PK columns would depend on the index being created, and would include in this order: <any constant columns (think key-prefix here) from your updatable view> <the indexed columns from the create index statement> <the rest of the PK columns from the data table - standard logic here> The changes necessary to support this include: 1. Tracking the constant value referenced on a column for an updatable view in PColumn. Might as well track if a column is referenced by a view as well. Two new columns: IS_REFERENCED_BY_VIEW, CONSTANT_VALUE_IN_VIEW. For the IS NULL case, we can use an empty byte array to differentiate from a null. These values would be passed through the CreateTableCompiler and upserted with the rest of the data. 2. Disallow the removal of any column in a view that is referenced in the view statement. An alternative would be to invalidate the view. 3. Add a new INDEX_ID column to PTable specifically for an INDEX on a VIEW. This will get populated based on the next value in the sequence. 4. Push the CONSTANT_VALUE_IN_VIEW into the IndexMaintainer along with the tenantId and the index ID. These values will be used to form the Put(s) and Delete(s) that get formed when an index is maintained. 5. Modify the code that automatically prepends tenant_id to also prepend the index ID in the case. This will be in WhereOptimizer, UpsertCompiler, and DeleteCompiler.

jira_issues_comments:

1. Awesome! IMHO 64K might be a bit low for base tables that are generic and support a large number of tenant-specific views on top of them.
2. The 64K is per tenant per base table, not across all tenants. Do you still think that's low?
3. I'm considering requiring the user to create a "shared" index from the physical table to "enable" indexing on views. What do you think, [~elilevine]? It would be something like this: {code} CREATE TABLE t (k VARCHAR PRIMARY KEY, c VARCHAR); // Enables indexes to be created on views - it's possible we // could require an additional keyword like SHARED or VIEW // before INDEX, or even use a different syntax altogether. // The advantage is that you could pass properties through // and or split points, as typically indexes are smaller than // tables and you usually want to split them differently. // However, it's difficult to know what to use, as the "real" // index would be created later - we don't know how // many columns it'll have yet. CREATE INDEX i ON t MAX_FILE_SIZE=200000 CREATE VIEW v(c2 VARCHAR) AS SELECT * FROM t WHERE c = 'foo'; // This would use the physical index table i CREATE INDEX iv ON v(c2); {code} The alternative is to auto-magically create a physical index table by tacking on a "_IDX" to the table name. We could do this for multi-tenant tables when they're created. We wouldn't have a facility for passing through properties specific to the index, but I suppose we could guess at the MAX_FILE_SIZE and just have it be some percentage smaller than the tables (50% ?). I suppose this could be tuned after-the-fact directly through HBase too. For views that are not tenant-specific we could just create separate physical index tables when an index is added to them. Or we could auto-magically create a shared physical index table as above when the first view is created off of a table. Thoughts?
4. **body:** Thinking about this a bit more, I think it's best to just create the physical index table under-the-covers. We can provide a new property to define the MAX_FILE_SIZE of the index table as a percentage of the MAX_FILE_SIZE of the data table. One other thing I realized is that I don't need to include the constant values from the view definition in the index at all. I'll just replace their occurrence in the query that gets generated for the index plan with the constant value. That simplifies things a bit and will make the index smaller too which is good.
   label: code-design
5. GitHub user JamesRTaylor opened a pull request: https://github.com/apache/incubator-phoenix/pull/7 PHOENIX-21: Support indexes on multi-tenant views Still needs more testing and a bit more bookkeeping for cleanup, but it's pretty close. You can merge this pull request into a Git repository by running: $ git pull https://github.com/apache/incubator-phoenix tenant-specific-index Alternatively you can review and apply these changes as the patch at: https://github.com/apache/incubator-phoenix/pull/7.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 #7 ---- commit 0524c0bf166b2c6b49f2d85b3d707364d78aecce Author: James Taylor <jamestaylor@apache.org> Date:

2014-02-13T06:36:09Z Added VIEW_CONSTANT column and prevent deletion of referenced coluns for views commit d0ab91315b71a6d239411a6db558da7b2e1f73df Author: James Taylor <jamestaylor@apache.org> Date: 2014-02-13T08:48:24Z Disallow deletion of referenced column for mapped views commit 1a48763be02ab9ef0b0f1065b1553345698cd4bd Author: James Taylor <jamestaylor@apache.org> Date: 2014-02-14T20:05:26Z Add VIEW_INDEX_ID for indexes on multi-tenant views commit afbce642290ef1c5277f3011ab3fe0c750fef401 Author: James Taylor <jamestaylor@apache.org> Date: 2014-02-17T08:22:34Z Tenant specific view indexes ----

6. Awesome! I like the more explicit approach that requires users to create a separate index table, since it allows more flexibility in terms of setting table properties. But I don't feel strongly either way. With respect to non-tenant-specific view indexes, are you implying there would be a table per index for those views?
7. I've made it so that non-tenant specific indexes also use a single, shared physical table, just like with tenant-specific indexes. I automatically manage the creation of the physical index table. For a multi-tenant table, I create the view index table up front, when the multi-tenant base table is created. For a regular table, I create it the first time an index is added to a view. The physical index table is a copy, structure-wise of the data table. There's one property (phoenix.index.maxDataFileSizePerc) you can provide at table creation time to control the percentage of the data table MAX_FILE_SIZE so that you can have smaller region sizes for indexes.
8. SUCCESS: Integrated in Apache Phoenix - Branch:master #55 (See [https://builds.apache.org/job/Phoenix/55/]) PHOENIX-21: Support indexes on multi-tenant views. Still needs more testing (jamestaylor: rev e2bd0ee06b2d2982155ddcb34549fff1249f67fc) * phoenix-core/src/main/java/org/apache/phoenix/compile/ProjectionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/SaltingUtil.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PDataType.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixDatabaseMetaData.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PMetaDataImpl.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryConstants.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/MetaDataEndpointImpl.java * phoenix-core/src/main/java/org/apache/phoenix/execute/BasicQueryPlan.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/QueryDatabaseMetaDataTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/TableAlreadyExistsException.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionQueryServicesImpl.java * phoenix-core/src/main/java/org/apache/phoenix/compile/QueryCompiler.java * phoenix-core/src/test/java/org/apache/phoenix/iterate/AggregateResultScannerTest.java * phoenix-core/src/main/java/org/apache/phoenix/compile/ExpressionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/DelegateColumn.java * phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixStatement.java * phoenix-core/src/main/java/org/apache/phoenix/compile/PostIndexDDLCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/parse/CreateSequenceStatement.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionQueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/execute/HashJoinPlan.java * phoenix-core/src/main/java/org/apache/phoenix/compile/FromCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/index/IndexMaintainer.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PColumn.java * phoenix-core/src/main/java/org/apache/phoenix/parse/LiteralParseNode.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PColumnImpl.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionlessQueryServicesImpl.java * phoenix-core/src/main/java/org/apache/phoenix/query/DelegateConnectionQueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/compile/DeleteCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/exception/SQLExceptionCode.java * phoenix-core/src/main/java/org/apache/phoenix/compile/QueryPlan.java * phoenix-core/src/main/java/org/apache/phoenix/compile/CreateTableCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/compile/WhereOptimizer.java * phoenix-core/src/main/java/org/apache/phoenix/compile/PostDDLCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/optimize/QueryOptimizer.java * phoenix-core/src/main/java/org/apache/phoenix/compile/IndexStatementRewriter.java * phoenix-core/src/main/java/org/apache/phoenix/schema/MetaDataClient.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTableImpl.java * phoenix-

core/src/main/java/org/apache/phoenix/query/QueryServicesOptions.java * phoenix-core/src/main/java/org/apache/phoenix/expression/LiteralExpression.java * phoenix-core/src/main/java/org/apache/phoenix/util/MetaDataUtil.java * phoenix-core/src/main/java/org/apache/phoenix/compile/JoinCompiler.java * phoenix-core/src/test/java/org/apache/phoenix/expression/ColumnExpressionTest.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/TenantSpecificViewIndexTest.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/ViewTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTable.java * phoenix-core/src/main/java/org/apache/phoenix/compile/UpsertCompiler.java

9. Looks good. Thanks!
10. [~giacomotaylor] It seems that function PTableImpl#Init() doesn't set the instance field tenantId using the passed in value. A bug?
11. Yes, that sounds like a bug. Want me fix that for my next check in?
12. Sure. Please go ahead, Thanks. I'm rebasing master branch into 4.0 branch now.
13. FAILURE: Integrated in Apache Phoenix - Branch:4.0 #35 (See [https://builds.apache.org/job/Phoenix-4.0/35/]) PHOENIX-21: Support indexes on multi-tenant views. Still needs more testing (jamestaylor: rev e2bd0ee06b2d2982155ddcb34549fff1249f67fc) * phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixStatement.java * phoenix-core/src/main/java/org/apache/phoenix/optimize/QueryOptimizer.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryServicesOptions.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTableImpl.java * phoenix-core/src/main/java/org/apache/phoenix/compile/ProjectionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/TableAlreadyExistsException.java * phoenix-core/src/main/java/org/apache/phoenix/execute/HashJoinPlan.java * phoenix-core/src/main/java/org/apache/phoenix/compile/QueryCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/compile/FromCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/expression/LiteralExpression.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/QueryDatabaseMetaDataTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/MetaDataClient.java * phoenix-core/src/main/java/org/apache/phoenix/compile/DeleteCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PColumn.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PTable.java * phoenix-core/src/main/java/org/apache/phoenix/execute/BasicQueryPlan.java * phoenix-core/src/main/java/org/apache/phoenix/compile/UpsertCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionQueryServicesImpl.java * phoenix-core/src/main/java/org/apache/phoenix/parse/LiteralParseNode.java * phoenix-core/src/main/java/org/apache/phoenix/jdbc/PhoenixDatabaseMetaData.java * phoenix-core/src/main/java/org/apache/phoenix/compile/PostDDLCompiler.java * phoenix-core/src/test/java/org/apache/phoenix/expression/ColumnExpressionTest.java * phoenix-core/src/main/java/org/apache/phoenix/compile/PostIndexDDLCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/compile/JoinCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PDataType.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/ViewTest.java * phoenix-core/src/test/java/org/apache/phoenix/iterate/AggregateResultScannerTest.java * phoenix-core/src/main/java/org/apache/phoenix/index/IndexMaintainer.java * phoenix-core/src/main/java/org/apache/phoenix/util/MetaDataUtil.java * phoenix-core/src/main/java/org/apache/phoenix/compile/WhereOptimizer.java * phoenix-core/src/main/java/org/apache/phoenix/compile/CreateTableCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PColumnImpl.java * phoenix-core/src/main/java/org/apache/phoenix/schema/DelegateColumn.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryConstants.java * phoenix-core/src/main/java/org/apache/phoenix/coprocessor/MetaDataEndpointImpl.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionQueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/exception/SQLExceptionCode.java * phoenix-core/src/test/java/org/apache/phoenix/end2end/TenantSpecificViewIndexTest.java * phoenix-core/src/main/java/org/apache/phoenix/schema/PMetaDataImpl.java * phoenix-core/src/main/java/org/apache/phoenix/query/DelegateConnectionQueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/compile/QueryPlan.java * phoenix-core/src/main/java/org/apache/phoenix/compile/IndexStatementRewriter.java * phoenix-

core/src/main/java/org/apache/phoenix/compile/ExpressionCompiler.java * phoenix-core/src/main/java/org/apache/phoenix/query/QueryServices.java * phoenix-core/src/main/java/org/apache/phoenix/query/ConnectionlessQueryServicesImpl.java * phoenix-core/src/main/java/org/apache/phoenix/schema/SaltingUtil.java * phoenix-core/src/main/java/org/apache/phoenix/parse/CreateSequenceStatement.java

14. Bulk close of all issues that has been resolved in a released version.