Item 349
git_comments:

git_commits:

1. **summary:** ACCUMULO-1000 was merged into accumulo master branch
   message: ACCUMULO-1000 was merged into accumulo master branch

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** support compare and set
   description: Add support to mutation for compare and set operations. This would allow user to specify that a row must contain certain data for a mutation to be applied.

jira_issues_comments:

1. This is a fine requirement for ticket #1000, for those of us with an affinity for base-10 numbers.
2. Could this be a family of iterators that implement the compare and set semantics, or is this meant to be resolved at insert time (AKA read-modify-write)?
3. Attached a design doc w/ my thoughts on this issue. This was my first time using asciidoc, wanted to try it out since Billie proposed using it for the user manual. The .txt file is the asciidoc source, the html was created from it.
4. The HTML looks pretty. :)
5. **body:** I used the following command from the asciidoc docs to generate the html {noformat} asciidoc -b html5 -a icons -a toc2 -a theme=flask ACCUMULO-1000-proposal-01.txt {noformat}
   label: documentation
6. From the mailing list: {quote} In the design document I suggested a method called putCondition() and if the value was null, thats mean that the user expected it to be absent. I don't really like that API for specifying expected absence. {quote} Could we do this as an additional method instead? relying on null for special API behavior smells like it'll lead to unexpected behavior if users mess up. {code} /** * Specify that a given cell must not exist */ void putConditionAbsent(CharSequence cf, CharSequence cq, ColumnVisibility cv) {} {code} And then putCondition can assert all non-null arguments.
7. **body:** putConditionAbsent() and asserting non-null would be less error prone, I am in favor of that.
   label: code-design
8. Would we want a conditional compare and delete analogues as well?
9. bq. Would we want a conditional compare and delete analogues as well? ContionalMutation will extend mutation, so the following would be supported. {code:java} ConditionMutation cm; cm1.putCondition(X); cm1.putDelete(Y); {code}
10. I wrote a simple client side implementation of ConditionalWriter so I could start writing unit test. During this process I stumbled upon some things I had not considered in the design. Conceptually a conditional writer will read and write. To read, it must use a set of auths. There are a few options : # User must pass a set of auths to conditional writer # User can not pass auths, Conditional writer uses all of users auths to read. # User can optionally pass auths, if they do not then use all of users auths # User can optionally pass auths, if they do not then use empty set of auths I am leaning twoards #1 because its consistent w/ the scanner API. #3 is like the scanner API, but maybe more convenient. #4 seems like it would cause problems. I am thinking if we decided to do something besides #1, that the Scanner API should be made consistent. ACCUMULO-246 is also a consideration. Another issue is that an absence test could pass when data is not really absent, its just that user can not see it. To remedy this I am thinking of making the absence condition fail if the auths supplied are not sufficient to read the column. This would not be a test

against data in Accumulo, only against the col vis supplied in the mutation condition. One last thing, I am thinking of renaming putCondition() and putConditionAbsent() to putEqualityTest() and putAbsenceTest().

11. **body:** We should definitely allow passing of auths, so -1 to #2. #4 would be consistent with what we do for the rest of the Mutation API, which I like, but I agree that it smells of confused users down the road. #1 seems most supportable, since it makes clear that you're only checking what you're allowed to see. As a bonus a change from required -> optional would be non-breaking, but not vice-versa, so we could change to #4 or #3 down the line if operational use finds them compelling.
    label: code-design
12. **body:** Also, we should avoid leaking information about data being present at a level the user can't see. If visibility is really a part of the key, then as far as a user with only access FOOBAR is concerned, a cell with vis CATSDEE doesn't exist and the absence test should reflect that. Unless there's a different API already that lets a user say "Given these Auths, can I read everything in column X.Y.Z?"
    label: code-design
13. **body:** It seems the implementation Keith gave is for compare and swap, not compare and set. This means that my pre-commit check is an equality operation, which may not be enough. One example that's always used is a bank account- when another person makes a charge to a bank account, they don't necessarily care that the account balance is the same as some initial check (ie, you said you have $100 but the bank says you have $90), I only care that the amount covered by the charged exists in the bank account (ie, this gumbo cost $50 and you have $90, you may pass go). I think we could hammer in this functionality by using an iterator that checks the precondition and returns a boolean- have you thought about how to add in that to {{ConditionalMutation}}?
    label: code-design
14. **body:** bq. One last thing, I am thinking of renaming putCondition() and putConditionAbsent() to putEqualityTest() and putAbsenceTest(). I initially found Test to be more confusing than Condition. What about ifExists(cf, cq, cv, val) and ifAbsent(cf, cq, cv)? That would provide a good semantic flow: cm.ifExists, cm.put. An alternative to ifExists could be ifMatches. I confess I don't have a full grasp of the use cases for this; would we ever want to test ifExists(cf, cq, cv)? Regarding the absence test with non-visible keys, maybe we should just let the test pass. Maybe the conditional writer should only allow writing visibilities that can be seen by the set of authorizations used for reading. You could use the same conditions with different authorizations to insert otherwise identical keys with different visibilities.
    label: code-design
15. putEqualityTest() and putAbsenceTest(). Would putInclusionTest() and putExclusionTest() make sense? Or PutExistsTest() and PutNotExists()? For me Equality and Absence are not opposites but the methods perform the opposite function?
16. Nevermind. I think my thinking was about a different topic.
17. I was thinking about the following scenario w/ absence and auths. In the case below, the absence check will always succeed because the user is scanning for data w/ auth "A" and checking that key w/ colvis "B" is absent. The check is pointless, it must be a logic error in the user code. {code:java} Connector conn; Authorizations auths = new Authorizations("A"); //auths used to read data ConditionalWriter cw = conn.createConditionalWriter(auths); ConditionalMutation cm = new ConditinalMutation("row1"); cm.putConditionAbsent("name","last",B"); //this test will always succeed no matter whats in the table cm.put("name","last","B","Doe"); cw.write(cm); {code} Only being able to write data that you can read is a separate case and is covered by a Constraint that checks this case.
18. bq. What about ifExists(cf, cq, cv, val) and ifAbsent(cf, cq, cv) I kinda like this, however to me it does not connotate that you are adding to a list of conditions/tests/checks that must be satisfied before a mutation is applied. I am too excited about any of the names that have been proposed. The thing I like most about ifExists() and ifAbsent() is they are short. bq. This means that my pre-commit check is an equality operation, which may not be enough. thats one thing I was thinking when I proposed putEqualityTest().. that maybe other test would be wanted in the future like putLessThanTest() or putGreaterThanTest()... not something I want do now.
19. Maybe something like the following is an option. {code:java} public class Column { public Column(byte[] colf, byte colq[]){} //setters for optional parts public Column setColVis(ColVis cv){} public Column setTimestamp(long ts){} } public abstract class Test { //package private constructor to limit subtypes Test(){} } public class AbsenceTest extends Test { public AbsenceTest(); } public class EqualityTest extend Test { public EqualityTest(byte[] val); } public class ConditionalMutation extends Mutation { public void putCondition(Column col, Test test); } {code}
20. **body:** Make Test an enum, and that starts to look pretty slick.
    label: test

21. bq. I think we could hammer in this functionality by using an iterator that checks the precondition and returns a boolean- have you thought about how to add in that to ConditionalMutation? My only thoughts about iterators so far is that the check will be done on the server side by reading the data through the iterator stack configured for the table. I suppose the ConditionalWriter could be configured to use iterators just like the scanner can. This made me think that ConditionalWriter could extend ScannerBase, took a look at the API most of the methods make sense except for iterator().
22. bq. Make Test an enum, and that starts to look pretty slick. That would be nice, but what about constructors? Absence() and Equality(byte[] val) are different constructors, suggestions?
23. [~billie.rinaldi] and [~busbey] you both made comments about auths and vis, did the example w/ auth "A" and col vis "B" address your concerns? Want to make sure we are not talking past each other. bq. #1 seems most supportable, since it makes clear that you're only checking what you're allowed to see. As a bonus a change from required -> optional would be non-breaking, but not vice-vers good point. I am going to go w/ that option for now in my prototype. If anyone wants to see my client side prototype let me know. Once the discussion settles for a bit I will post another revision of the design doc.
24. Can you put it up on the gitchubs or bitbucket?
25. [~kturner] Ah, yeah that makes sense now. Presumably an IllegalArgumentException?
26. re: Column and Test Objects I liked the simplicity of vanilla methods to add conditions to the ConditionalMutation. This might just be my bias against Object creation in Java. the unwrapped version has the advantage of looking more like other methods used in Mutation. If you do go down this route: bq. That would be nice, but what about constructors? Absence() and Equality(byte[] val) are different constructors, suggestions? * Change Column to Cell, make val another optional argument. * Make the docs expressly say that not setting an optional argument means "any". * Make Test enums have only noargs constructor. * Make the entire correctness check in the Test instance, so all of them get the entire Cell * Either allow Absence checks that go to the value level or document that it will ignore any value in the Cell. On that last bit, I'm in favor of the former. But maybe then "AbsenceTest" becomes "NonEqualityTest"
27. bq. Ah, yeah that makes sense now. Presumably an IllegalArgumentException? When a batch of conditional mutations is passed to the conditional writer, can not throw an exception. Need to know the status of each one. I am thinking of adding something to the status enum, just not sure what to call the enum. bq. Can you put it up on the gitchubs or bitbucket? https://github.com/keith-turner/ACCUMULO-1000 [~busbey], I like the Cell proposal. I also agree w/ your arguments for not using it. bq. This might just be my bias against Object creation in Java. Thats a good bias to have (if only there were some safe way to allocate objects on the stack). Normally I would agree w/ this, but I am thinking in this case the operation (lock row on server side, read, maybe update) is so expensive that it possibly dwarfs this concern. However in the case where data is cached in the tserver its not as expensive. bq. the unwrapped version has the advantage of looking more like other methods used in Mutation. Agreed, consistency is very important. For me this is the main argument for not using the Cell+Test design. Changing the API for mutation is not really an option. [~bills] I just fully understood the significance of your earlier comment w.r.t API. Users will be able to do any check they like using iterators. Therefore we do not need to anticipate future mutation conditions besides equality and presence.
28. I would like to see the IteratorOptions pushed down to the ConditionalMutations, rather than put on the ConditionalWriter.
29. I would also like to see an explicit addCondition(Condition) or setConditions(List<Condition>) method on the ConditionalMutation, because I think if we don't, we're going to begin to suffer from the method overloading mess that our other APIs have suffered from, as we identify new useful conditions.
30. **body:** bq. I would like to see the IteratorOptions pushed down to the ConditionalMutations, rather than put on the ConditionalWriter. agreed bq. I would also like to see an explicit addCondition(Condition) I think pushing the iterator option down to the condition necessitates a Condition object, otherwise the overloading would be completely insane. So I did this on github. I created a [Condition class|https://github.com/keith-turner/ACCUMULO-1000/blob/2ac9b3aeb6975488c26353a045d37abe92986969/src/main/java/core/data/Condition.java] where visibility, timestamp, value, and iterators are optional. If the value is not set, then its expected to be absent. I replaced all of the putCondition() methods with a single addCondition() method (replacing the implementions of putCondition() w/ addCondition() and then inlining the putCondition methods in eclipse made this so easy, much easier that typing this explanation)
    label: code-design
31. **body:** I like addCondition(Condition). To use a less than condition, for example, the approach would be to have an iterator that drops values if they are greater than a particular value, then to use an existence condition to determine if the mutation is applied? The iterator is almost a simple filter, except that you'd

want to transform all the accepted values into the same thing (probably an empty value) so that they could pass the same equality condition. If this is what we'd be recommending to people, I think we should make sure it is relatively easy to do. I could help out with that, if you'd like. (IMO requiring the user to implement a filter to do less than/greater than would be okay, but not an entire iterator. We could add an option to Filter that would empty out any accepted values.)
label: code-design

32. Commit 1502726 from [~kturner] [https://svn.apache.org/r1502726] ACCUMULO-1000 initial checkin of conditional mutations that does locking on tablet server. The implementation is pretty far along, but still a good bit to do.
33. Commit 9dc244484b4d35859d4d22b27580a47ae7da0e1a in branch refs/heads/master from [~keith_turner] [https://git-wip-us.apache.org/repos/asf?p=accumulo.git;h=9dc2444] ACCUMULO-1000 added conditional mutations to Accumulo
34. Commit 6677a162c0d17a5e69675a5e82e6a1841933984b in branch refs/heads/master from [~ctubbsii] [https://git-wip-us.apache.org/repos/asf?p=accumulo.git;h=6677a16] ACCUMULO-1000 Fix generated thrift and suppress unchecked cast warning (because a comprehensive generics fix would touch far too much code)
35. Implementing tablet server lock checks (as mentioned in ACCUMULO-1152) in the tablet location cache would make invalidating conditional update session more reliable.
36. Commit ffaba0888389b418dd63901f6336ef2f2236ee79 in branch refs/heads/master from [~ecn] [https://git-wip-us.apache.org/repos/asf?p=accumulo.git;h=ffaba08] ACCUMULO-1000 use Assert.fail() vs Assert.assertTrue(false), delete test folder
37. accidently closed
38. Commit 42853c14aa7e46328272d4c91e0afe3ab938a27b in branch refs/heads/1.6.0-SNAPSHOT from [~elserj] [https://git-wip-us.apache.org/repos/asf?p=accumulo.git;h=42853c1] ACCUMULO-1000 Some spelling/grammar corrections.
39. Commit 42853c14aa7e46328272d4c91e0afe3ab938a27b in branch refs/heads/master from [~elserj] [https://git-wip-us.apache.org/repos/asf?p=accumulo.git;h=42853c1] ACCUMULO-1000 Some spelling/grammar corrections.