Cassandra2474

CQL Compound Column Proposals

This page is an attempt to summarize CASSANDRA-2474: CQL support for compound columns, collecting the most recent versions of each proposal, in the hopes that they can be more easily compared.

The names are arbitrary (proposers, feel free to change them), and the ordering is based on when/where each appeared in issue #2474.

Background

Supercolumns

Cassandra has supported limited nesting of data within a row via SuperColumns since its initial release. A supercolumn is a named container of subcolumns, with no other metadata attached to it: unlike data columns, it cannot have a timestamp or TTL associated with it. (The one exception is that supercolumns CAN be deleted as a unit, and thus SuperColumns CAN be tombstones.) Thus, a row using supercolumns looks like this:

key	**supercolumn1**			**supercolumn2**	
	subcolumn1	subcolumn2	**subcolumn3**	subcolumn4	subcolumn5

The most common use case for [SuperColumns] is to represent "materialized views" or "precomputed resultsets": each object in the resultset maps to a single supercolumn. This usually takes advantage of the sorting-by-column-name to give very performant "slice" lookups for this resultset. To use a more concrete example, we could represent the Twitter timeline as a single supercolumn row per user, with the tweets made by that user's friends represented as supercolumns within that row. The supercolumn names will be the posted_at information, so this lets us get "most recent tweets, in [reverse] chronological order" easily:

tjefferson	**1763**			**1790**	**1818**	
	body	posted_by	body	posted_by	body	posted_by
	Democracy will soon degenerate into an anarchy	jadams	To be prepared for war is one of the most effectual means of preserving peace	gwashington	Revolution was effected before the war commenced	jadams
bfranklin	**1781**					
	body	posted_by				
	Every government degenerates when trusted to the rulers of the people alone	tjefferson				

Composite columns

SuperColumns have a number of limitations, most notably

1. there can only ever be a single level of nesting
2. to read any subcolumn from supercolumn X, all of X is read into memory
3. they add a lot of complexity to the Cassandra implementation and cause a fair number of bugs

To address these problems, Cassandra added the CompositeType, which encodes a multi-value column name into a single column – essentially the column name becomes a Tuple, for those with a background in Python. I will use Python tuple representation (x, y, z) to denote a composite column with components x, y, and z.

Composite columns are flexible enough that there are multiple ways to encode the same data. The most natural ways to encode the above timeline data are, first, an encoding where each object becomes a single column, with an empty value:

tjefferson	(1763, 'Democracy will soon degenerate into an anarchy', 'jadams')	(1790, 'To be prepared for war is one of the most effectual means of preserving peace', 'gwashington')	(1818, 'Revolution was effected before the war commenced', 'jadams')

<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="d84b092e-e39c-4a08-a7e2-90a570812aac" ><ac:plain-text-body><![CDATA[		[empty]	[empty]	[empty]	]]></ac:plain-text-body></ac:structured-macro>
bfranklin	(1781, 'Every government degenerates when trusted to the rulers of the people alone', 'tjefferson')				
<ac:structured-macro ac:name="unmigrated-wiki-markup" ac:schema-version="1" ac:macro-id="8415d9be-036f-43bc-a020-6627caca7d94" ><ac:plain-text-body><![CDATA[		[empty]			]]></ac:plain-text-body></ac:structured-macro>

(One could arbitrarily pick any column whose sort order is not important to be the column "value" instead, but I think it's more straightforward to treat things uniformly.)

The main drawback to this representation is that like row keys, column names are necessarily immutable in Cassandra. So there is no way to update an object using this representation other than by deleting the old and adding the new. More subtly, this exposes us to some of the drawbacks of a pure key /value approach that normal Cassandra columns avoid: if one client updates field X in a result, while another client updates field Y, there will be no race when X and Y are distinct columns. But if these fields are stored as part of the same composite column then there is a race.

Another way to encode this data addresses these drawbacks by splitting updateable fields into separate composite columns:

tjefferson	(1763, body)	(1763, posted_by)	(1790, body)	(1790, posted_by)	(1818, body)	(1818, posted_by)
	Democracy will soon degenerate into an anarchy	jadams	To be prepared for war is one of the most effectual means of preserving peace	gwashington	Revolution was effected before the war commenced	jadams
bfranklin	(1781, body)	(1781, posted_by				
	Every government degenerates when trusted to the rulers of the people alone	tjefferson				

For lack of better terms, we have been calling these "dense" and "sparse" composite column encodings.

DynamicCompositeType

DCT has no set type information or field count – each component of the composite column name includes the type name as well (encoded as a utf-8 String). Currently, this allows rows within the same ColumnFamily to have different kinds of data in them. In the future, this will also allow different kinds of data within the same row (https://issues.apache.org/jira/browse/CASSANDRA-3625).

Goals

Primary: provide a CQL syntax for updating and querying composite column families.

Secondary goal: proposed syntax should be implementable by the Hive driver with the minimum of changes from mainline Hive. In particular, changes to the Hive parser are too difficult to maintain long-term and are Right Out. We would prefer to avoid changes to the Hive metastore but this is doable if necessary.

Tertiary goal: it would be nice to support supercolumns as well as composite columns

Non-goals and related tickets

Supporting DynamicCompositeType or other arbitrarily-and-non-uniformly nested "document" data is a non-goal. https://issues.apache.org/jira/browse/CASSANDRA-3647 is created to follow up on this related problem.

Supporting non-utf8 column names is orthogonal to supporting composite columns; will address that in https://issues.apache.org/jira/browse/CASSANDRA-3685.

Alpha

The short-lived first proposal envisioned adding the "prefix" from which to select a resultset to the table name in the FROM clause. Discussion starts Discussion starts here

```
SELECT x, y FROM foo:bar WHERE parent='columnA'
```

```
select a, b FROM foo:bar:columnA where subparent='x'
```

Discussion Summary

Jonathan was thinking in terms of supercolumns for this early proposal. It's not clear how to generalize this to composites where the "subcolumns" are not explicitly named in the CompositeType definition.

This proposal would require a Hive metastore change, but the nail in the coffin is that this means you cannot use WHERE clauses with the "parent" parts of the column. So, no range queries (necessary for map/reduce) or even slices within the same row.

Beta

This proposal suggests the use of a keyword or hint to indicate that a query is transposed. Discussion starts here

The first part of the discussion is where to put the transposition marker:

```
select /*+TRANSPOSED*/ key, column, subcolumn, value from foo;
```

```
select key, column, subcolumn, value from foo TRANSPOSED;
```

```
select transposed(key, column, subcolumn, value) from foo;
```

Settling on "table:transposed" because that requires no Hive changes:

```
select key, column, subcolumn, value from foo:transposed;
```

The second part, starting here, digs into how to deal with destructuring the composite column name:

```
SELECT name AS (tweet_id, username), value AS body
FROM timeline:transposed
WHERE tweet_id = '95a789a' AND user_id = 'cscotta'
```

```
SELECT component1 AS tweet_id, component2 AS username, component3 location, value AS body
FROM timeline:transposed
WHERE user_id = '95a789a'
```

```
UPDATE tweets:transposed SET COMPOUND NAME ('2e1c3308', 'cscotta') = 'My motocycle...' WHERE KEY = <key>;
```

```
UPDATE tweets:transposed SET value = 'my motorcycle' WHERE KEY= <key> AND column = COMPOUND_NAME('2e1c3308',
'cscotta');
```

Discussion Summary

There was general agreement that "FROM foo:transposed" is a reasonable syntax, however, neither the "componentX" syntax (where X is in range(1, number of components in the compositetype) nor the "name AS (x, y)" syntax met with approval: the "name AS" syntax requires patching the Hive parser, and the "componentX" syntax is ugly and repetitive to use. The UPDATE syntaxes were also unsatisfactory.

Gamma

This proposal switches gears to dealing with transposition using DDL instead of

Discussion starts here

Gamma can represent both dense and sparse composite types; fields included in the PRIMARY KEY definition will be represented as part of the composite column "prefix" with a dense encoding:

```
-- the "dense" encoding shown above in the Background section
CREATE TABLE timeline (
    user_id int,
    posted_at uuid,
    body string,
    posted_by string,
    PRIMARY KEY(user_id, posted_at, body, posted_by)
);

-- the "sparse" encoding
CREATE TABLE timeline (
    user_id int,
    posted_at uuid,
    body string,
    posted_by string,
    PRIMARY KEY(user_id, posted_at)
);
```

Examples

SELECT, INSERT, and UPDATE syntax require no changes. Some examples, using the timeline data from the Background section above:

```
INSERT INTO timeline (user_id, posted_at, posted_by, body)
VALUES ('tjefferson', '1818', 'jadams', 'Revolution was effected before the war commenced');

INSERT INTO timeline (user_id, posted_at, posted_by, body)
VALUES ('tjefferson', '1763', 'jadams', 'Democracy will soon degenerate into an anarchy');

INSERT INTO timeline (user_id, posted_at, posted_by, body)
VALUES ('tjefferson', '1790', 'gwashington', 'To be prepared for war is one of the most effectual means of
preserving peace');

INSERT INTO timeline (user_id, posted_at, posted_by, body)
VALUES ('bfranklin', '1781', 'tjefferson', 'Every government degenerates when trusted to the rulers of the
people alone');
```

An example SELECT:

```
SELECT * FROM timeline WHERE user_id = 'tjefferson' AND posted_at > 1770;
```

user_id	posted_at	posted_by	body
tjefferson	1790	gwashington	To be prepared for war is one of the most effectual means of preserving peace
tjefferson	1818	jadams	Revolution was effected before the war commenced

Discussion Summary

Only minimal CQL changes are required. The Hive metastore would need to be updated to understand the TRANSPOSED syntax. Normal SELECTs and UPDATEs are supported, including "SELECT *," a weakness of the Beta proposals.

The PRIMARY KEY syntax allows for specifying both "sparse" and "dense" data layouts, without the SPARSE keyword that some found unappealing. It also improves conceptual integrity with existing C* practice, namely, that row keys are not update-able. So, the tradeoff is straightforward: include a column in the PRIMARY KEY if you want it to be part of the positional CompositeType tuple (and be more space efficient); leave it out if you want to update it.

Originally a TRANSPOSED WITH [options] syntax was proposed but consensus is that this is weaker than just inferring from the composite PRIMARY KEY definitions.

This also allows supporting SuperColumns, should we choose to do so.

https://c.statcounter.com/9397521/0/fe557aad/1/|stats