## 

Product Requirements Document

HBase Cell/Row-level security

Table of Contents

Overview 3

Use Cases 3

P1: No cell/row-level security needs: 3

P1: Clearance levels: 3

P2: Hierarchical classification: 3

P2: Compartmentalization 4

P2: Pluggable flexibility 4

P3: Read/Write permissions: 4

Use Case Notes: 4

Goals 4

P1 (must have): 4

P2 (should have and will eventually need to have): 5

P3 (could have - not needed for initial release but possible subsequent customer asks): 5

Non-goals: 6

Functional Workflows 6

Write: 6

Change visibility: 7

Delete: 7

TODOs 7

Background 7

# Overview

Cell or row-level security provides the ability to control visibility of individual rows and/or cell values accessible to different groups of users.

Table and column family permissions provide access controls across all data (all rows/cells) of a certain type, customers can have the same types of data (i.e. same table/column family structure) that have different levels of sensitivity. Instead of having multiple column families and/or tables based on sensitivity, cell/row level security enables all the data to exist in a single table with different visibility per cell or row. The primary benefits:

* Fine granularity visibility control within a single table
* Less complexity of objects (tables or column families) to manage
* Transparent flexibility of admins to control visibility independent of app changes (no need for separate table/column family per visibility type)
* Simpler client app development (no correlation across tables/column families)

# Use Cases

This section spells out a few different admin scenarios that solutions to cell/row-level security can be used to solve. We should seek to maximize simplicity and flexibility over purpose-built solutions.

## P1: No cell/row-level security needs:

* A customer that has no need for cell/row-level security should expect no noticeable overhead in management or performance compared to current codebase

## P1: Clearance levels:

* Users have ordered security levels within a project (e.g. UNCLASSIFIED, CONFIDENTIAL, TOP SECRET) and can only view data at their classification level or lower. All other data is elided from their view.
* A user can have different levels of access for different projects (eg. a user may have TOP SECRET for project X but only UNCLASSIFIED for project Y and no access to Z)

## P2: Hierarchical classification:

* Users are set in groups where each group has a position in a hierarchy (e.g. management structure)
* Group visibilities cascade down but not up e.g.
  + CEO office has access to entire company
  + COO has access to Sales and Marketing but not Engineering
  + Engineering has access to only to Engineering tagged data
* A piece of data may be visible to multiple groups (e.g. a marketing doc may be given access to Product and Marketing)

## P2: Compartmentalization

* Users may be granted visibility based on whether they are part of a specific group (e.g. project team)

## P2: Pluggable flexibility

* An organization may have a custom cell/row-level security requirement and should be able to implement their desired model with advanced work (e.g. coding co-processor). Examples of things:
  + Read/write permissions (see below)
  + Custom encoding and evaluation of security descriptors
* Complex or nested visibility evaluation
* Expiring visibility
* Different precedence evaluation between table/column familiy ACLs and cell-level security

## P3: Read/Write permissions:

* User may be confined to write to a lower classification level than their maximum read-access
* User may or may not be restricted from writing below a certain level (e.g. user can read TOP SECRET or lower, but can only write confidential)
* Some users may only have read access while others have write access
* Users can have read and write access separately e.g.:
  + HR may be given read access on items but not write access

## Use Case Notes:

* Accumulo doesn’t provide granularity of write and read visibility

# Goals

## P1 (must have):

* Minimize impact to customers that don’t need row/cell-level security
* Security enforcement should occur within admin trust boundary (e.g. cannot rely on client to do enforcement without establishing trust)
* Table and column-level permission model evaluated at higher precedence than row/cell-level visibility (note: custom co-processor should be able to alter this if desired)
* Cell/row-level security turned on and off at column family level
* Either or both individual users or individual groups can be granted visibility to a cell
* Users and groups defined and authenticated through standard Hadoop security
* Many-to-many relationship between users and groups
* Multiple users and/or groups can have visibility on the same object
* App/client can specify visibility tags per write
* No visibility tag means visible to all
* App/client can specify Delete to delete all cells for which they have visibility access
* App/client cannot Delete cells with for which they don’t have visibility access
* Delete can specify a set visibility tags to delete and only delete cells that have that particular tag(s) (cannot delete a cell for which that user doesn’t have visibility access)
* App/client can specify Delete of only cells that have no tags (i.e. visible to all). Note: API needs a way to distinguish between delete no tags versus delete all my tags.
* Get, Scan return all values for which the user has visibility access (i.e. no errors just silently skipped)
* Mutation APIs (e.g. Increment) require visibility access
* Put allows user to write any visibility tags the framework allows the user to apply
* Column qualifier versions are kept per visibility access expression (e.g. repeatedly writing a column qualifier with a different access expression will not cause a higher expression to compact away)
* If a user has no visibility into any cell within a row, the row-name is also not visible to them

**Perf/size trade-offs:**

* Impact without row/cell-security > get/put/scan performance > change visibility > space > turn on visibility > turn off visibility

## P2 (should have and will eventually need to have):

* Super-user visibility tag for read and/or write that gives access to everything
* Ability to codify a specialized cell/row-level security policy (e.g. separate read and write visibility, change the hierarchy by writing custom co-processor or extension, changing precedence of table vs. cell security)
* Ability to specify AND visibility tags (e.g. must be part of group X and Y)
* HBase admin can set config option to control what tags a user can apply:
  + Allow users to apply any tag (default)
  + Require users to only apply tags that evaluate to true for themselves

## P3 (could have - not needed for initial release but possible subsequent customer asks):

* Users and groups can have either read, write, or read-write access to access or apply a tag
* Configuration option to throw access-violation error rather return NULL if a cell has values but only at higher visibility tags than the user can see
* Ability to specify default visibility tags instead of empty if not specified
* Ability for admin to grant a delete permission that enables a user to delete data they don’t have access to either read or write according to visibility level

## Non-goals:

* Purpose-built solution for each of use cases above (i.e. enabling customers to solve is primary; then optimize for simplicity/flexibility over end-to-end solutions)
* Management of relationships between tags (i.e. HBase should enforce which users belong to which tags and policy enforcement of tags; HBase should enable admins but not aid in enforcement of tag hierarchy, levels, and other such associations between tags)
* Fully hiding existence of data or size of data (e.g. latency of response times between no data and no visibility to data)

# Functional Workflows

To illustrate above assume initial table of the following with permissions (C=Classified, S=Secret, TS=Top Secret):

|  |  |  |
| --- | --- | --- |
| **Row Name** | **Person** | **State** |
| 1 | John Smith {TS} | NY {} |
| 2 | Jane Doe {S | TS} | CA {} |

## Write:

Attempt to write Person in Row Name 1 of John with {S} would lead to:

|  |  |  |
| --- | --- | --- |
| **Row Name** | **Person** | **State** |
| 1 | John Smith {TS}  John {S} | NY {} |
| 2 | Jane Doe {S | TS} | CA {} |

A subsequent read with by a user with {S} would only show name “John” for Row Name 1. A read by a user with {TS} would only show name “John Smith” for Row Name 1. While a read from a user with both {TS} and {S} would return both “John Smith” and “John” (regardless of maxVersions setting since they are considered distinct values rather than versions of same value).

## Change visibility:

A subsequent user with {S,C} write visibility could then delete Person for Row Name 1 and write a new value with a Person value “John” for Row Name 1 but with visibility {C | S}:

|  |  |  |
| --- | --- | --- |
| **Row Name** | **Person** | **State** |
| 1 | John Smith {TS}  ~~John {S}~~  John {C | S} | NY {} |
| 2 | Jane Doe {S | TS} | CA {} |

## Delete:

If a user with either of both {C} or {S] tag attempts to delete row name 1 for all his permissions this would leave the following:

|  |  |  |
| --- | --- | --- |
| **Row Name** | **Person** | **State** |
| 1 | John Smith {TS}  ~~John {C | S}~~ | ~~NY {}~~ |
| 2 | Jane Doe {S | TS} | CA {} |

To delete the entire row one of the following would need to occur:

* Original user attempting delete to have had all visibility tags {S} or {C} as well as {TS}
* A subsequent user with {TS} to attempt delete remaining “John Smith” entry
* User has superuser access to write any row

# TODOs

* Complex logic possible for end-user to add in co-processor? Frequency?

# Background

<http://www.databasejournal.com/features/oracle/article.php/3065431/Oracle-Label-Security-Part-1-Overview.htm>

<http://docs.oracle.com/cd/E11882_01/network.112/e10745/toc.htm>

<http://docs.oracle.com/cd/E11882_01/network.112/e10745/accpriv.htm#i1007327>

<http://www.oracle.com/us/products/database/options/label-security/overview/index.html>

<http://www.oracle.com/technetwork/database/security/ds-security-label-security-11gr2-134491.pdf?ssSourceSiteId=ocomen>

<http://www.symantec.com/connect/articles/oracle-row-level-security-part-1>

<http://askanantha.blogspot.com/2007/09/keeping-information-private-with-vpd.html>

<http://www.oracle.com/technetwork/database/security/owp-security-label-security-11gr2-133601.pdf>